Abstract

As more and more sensitive information is entering web based applications, and thus are available through a web browser, securing these systems is of increasing importance. A software system accessible through the web is continuously exposed to threats, and is accessible to anyone who would like to attempt a break-in. These systems can not rely on only external measures like separate network zones and firewalls for security.

Symantecs[1] Internet Security Threat Report [34] is published every six months. Main findings in the last one published prove that there is an increase in threats to confidential information and *more attacks aimed at web applications*. Almost 48 percent of all vulnerabilities documented the last six months of 2004 were vulnerabilities in web applications.

Security principles that one should pay attention to developing web applications do exist. This report have taken a look at existing guidelines, and provided an independent guide to developing secure web applications. These guidelines will be published at the homepage of The Centre for Information Security[2] (SIS), www.norsis.no.

The report also describes how a web application has been developed using the provided security guidelines as reference points. Relevant vulnerabilities and threats were identified and described. Misuse cases have related the various threats to specific system functionality, and a risk analysis ranks the threats in order to see which ones are most urgent.

During the design phase, the application areas exposed to threats with a high rank from the risk analysis, have been at center of attention. This is also the case in the implementation phase, where countermeasures to some of these threats are provided on the Java platform. The implemented solutions can be adapted by others developing applications on this platform.

The report comes to the conclusion, that the use of security guidelines throughout the entire development process is useful when developing a secure system.

[1]Symantec works with information security providing software, appliances and services designed to secure and manage IT infrastructures [33].

[2]The Centre for Information Security (SIS) is responsible for coordinating activities related to Information and Communications Technology (ICT) security in Norway. The centre receives reports about security related incidents from companies and departments, and is working on obtaining an overall impression of threats towards Norwegian ICT systems [30].

Preface

The writing of this report has been a process in which we have learned a lot about not only web application security, but also computer security in general.

We would like to thank some people who have helped us during the making of this report. First of all, we would like to thank our teaching supervisor Lillian Røstad who works on her PhD at The Norwegian University of Science and Technology (NTNU). She is a very capable teaching supervisor, with a great ability to motivate and inspire.

Also, we would like to thank Hallvard Trætteberg, associate professor at NTNU. He was kind enough to read through the implementation chapter, and gave us some ideas on improvement.

Working on this report has been very interesting, and we hope that we get the opportunity to keep on working with information security when we start our occupational career.

Trondheim, 24th June 2005

Julie-Marie Foss Nina Ingvaldsen

Contents

List of Figures

List of Tables

Introduction

A web application is a piece of distributed software that uses the Internet as communication channel. Web applications are exposed to a number of threats, and many considerations have to be made in order to develop secure web systems.

Secode[3] did a survey [24] for their customers over a period of two years, searching for security breaches in web applications. They discovered that as much as 81% of the web applications tested had one or more critical security breaches. The most common vulnerabilities were lack of input validation, lack of encryption, username and passwords stored as plaintext in scripts, and defective logout procedures in the applications.

Vulnerabilities is hard to fix with security patches[4]. Security problems should therefore be identified early in a development process. Ad hoc solutions to security problems, may create new vulnerabilities and new security holes.

This report will describe the development of a web application, where the main focus is on security. This will be done by first defining security guidelines, that will work as reference points during the development process. Then vulnerabilities and threats will be described, along with countermeasures. The application will be developed according to the waterfall model, and will go through a requirement engineering phase, a design phase, and an implementation phase. Testing and maintenance are also important elements of the waterfall model, but are left out, as the focus of this report is on security during design and implementation.

In the requirement engineering phase, we will identify functional- and system requirements, but also security requirements. The security requirements will be modeled by misuse cases, which emphasize how, and where, threats to various functionality may appear, and suggest countermeasures. In addition, the threats that affects the system, will go through a risk analysis, and be ranked in order of the probability of occurring in the application.

In the design phase, we will design the system using our security guidelines to prevent the threats identified in the requirement phase. The security guidelines will hopefully help us to choose a sensible architecture.

The Java technology is well known, and commonly used to develop web based applications. We will be using JavaServer Pages (JSP) to implement our design. The intent of the implementation, is to shed light on vulnerable processes in the application, and provide technology specific solutions.

It will be of great interest for us to find out if our suggested security guidelines will serve as good reference points during the design- and implementation phase. If

[3]Secode is a leading Digital Security Company in the Nordic region [24].

[4]A patch is a piece of object code that is inserted into an executable program, and is a fix to a program bug [44].

this is so, we hope that these guidelines can be adapted as a template to other people developing web applications.

In addition, we hope that the solutions we will be implementing on the Java platform, will be found useful for other developers, and that our solutions may be useful in their work.

Chapter 1

Web applications and security issues

This report concerns web applications. To familiarize with the setting in which they exist, this chapter will provide basic information that will be referenced throughout this report.

The Internet is a massive network of networks. It connects millions of computers together globally, forming a network infrastructure, where all connected computers can communicate with each other.

The World Wide Web (WWW), or simply the web, is a way of accessing information over this medium. It is an information-sharing model that is built on top of the Internet, using the underlying Hyper Text Transfer Protocol[1] (HTTP) to transmit data. Web applications communicate using the HTTP protocol over the web. Users can log on to web applications via their web browser although they might be far away from the server the system is installed on.

This chapter will start off defining what a web application is, and move on to presenting security issues concerning both general computer systems, and web applications. There will be a description of different attacks that can be directed towards these systems, and countermeasures to circumvent them.

1.1 Web applications

In essence, a web application is a client/server software application that interacts with users or systems using HTTP [20]. The client can be a web browser like MS Internet Explorer or Netscape Navigator, where users are able to view pages and to interact by sending requests to and from the system.

Web applications are often built using a three-tiered architecture. This architecture consists, as shown in Figure 1.1, of three logical tiers, namely the presentation tier, application tier and the data tier [20].

The first tier, *the presentation tier*, is responsible for presenting data to end users or systems. This tier include web browsers and web servers. They may also include application components that create the page layout [20].

[1]HTTP defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands [44].

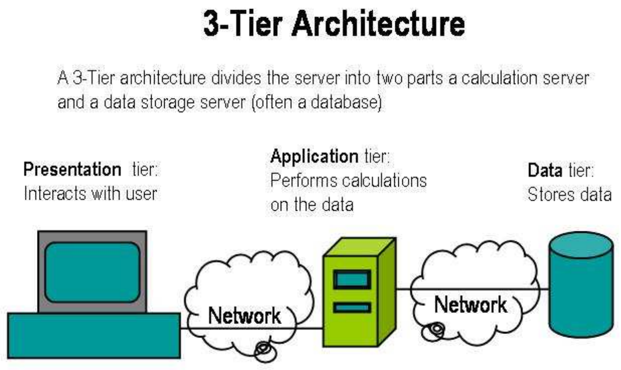

3-Tier Architecture
A 3-Tier architecture divides the server into two parts a calculation server and a data storage server (often a database)
Presentation tier: Interacts with user
Application tier: Performs calculations on the data
Data tier: Stores data
Network
Network

Figure 1.1: Web applications can be made up by three logical tiers as depicted in the figure above. A presentation tier that interacts with the user, an application tier that handles the business logic, and a data tier that stores the data.

The second tier, *the application tier*, is the engine of a web application. It performs the business logic, like processing user input, making decisions, obtaining more data and sends data to the presentation tier which presents it to the user [20].

The third tier is *the data tier*, which is used to store things needed by the application, and acts as a repository for both temporary and permanent data. It is the bank vault of a web application [20].

1.2 Definition of security

Computer security rests on the concept of preserving authentication, confidentiality, integrity, and availability [5]. The overall goal is to maintain these aspects when developing computer systems.

Authentication is the process of uniquely identifying the clients of your applications and services [44]. There are three main factors that can be used to authenticate users. The user can be authenticated on the bases of something the user knows (for instance a password), something the user has (like nonce words or a smartcard[2]), or something that the user is (for example a fingerprint) [41].

Confidentiality, also referred to as privacy, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users or eavesdroppers. Access control mechanisms and encryption[3] is frequently used to enforce confidentiality.

[2]A card that has electronic logic embedded in it. Smartcards are commonly used to authenticate users for access control purposes [46].

[3]Encryption is scrambling of data so that only the authorized recipient can read it [44]. This will be thoroughly described in section 1.4.

Integrity is the guarantee that data is protected from unauthorized modification. Like confidentiality, integrity is a key concern for data passed over networks. Integrity for data in transit is typically provided by hashing techniques and message authentication codes[4] (MAC).

Availability refers to the ability to use the information or resource desired. That is, that the system remains available to legitimate users. The aspect of availability that is relevant to security, is that someone may deliberately arrange to deny access to data or to a service by making it unavailable.

Another aspect of security that should be emphasized, is *authorization*. This is the process that governs the resources and operations that authenticated clients are permitted to access [32].

When introducing web applications and security issues, there are core terms that needs to be defined. These are assets, threats, vulnerabilities, attacks and countermeasures. We have used the same notations as the Open Web Application Security Project[5] (OWASP) to define these terms.

Asset. An asset is a resource of value such as the data in a database, or on the file system, or a system resource.

Threat. A threat is a potential occurrence, malicious or otherwise, that may harm an asset.

Vulnerability. A weakness that make a threat possible.

Attack. An attack is an action taken to harm an asset.

Countermeasures. A safeguard that addresses a threat and mitigates a risk. In this report it is important to notice that the main goal is to prevent attacks on security. Safety issues[6] will not be handled in this report.

Security is about protecting assets, and is more of a path than a destination. When analyzing infrastructure and applications, one may identify potential threats and understand that each threat represents a degree of risk. Security is thus about risk management and implementing effective countermeasures.

When creating software and web applications, we should strive the maintain four fundamental principles of security. There are always those who wants illegitimate access to assets held by the system, or information that flows over the network. They get this access by performing attacks directed at the systems.

1.3 Attacks on security

Security is threatened when attackers tries to break into a system without permission. There are two ways of attacking a system. Either by performing an *active attack* or a *passive attack*. An active attack attempts to alter system resources or affect their operations. A passive attack attempts to learn from, or make use of, information

[4]A Message Authentication Code (MAC) is a one-way hash computed from a message and some secret data. Its purpose is to detect if the message has been altered [44].

[5]The Open Web Application Security Project (OWASP) is an open-source software project organization, which produces many types of materials in a collaborative, open way. OWASP is dedicated to finding and fighting the causes of insecure software [19].

[6]Safety issues are for instance problems concerning network capacity, the servers handling of many simultaneous users and the handling and consequences of down-time of a computer system.

from the system, but does not affect the resources [32]. These two forms of attacks, will be presented next.

Active attacks

Active attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories, namely masquerade, replay, modification of messages, and denial of services [32].

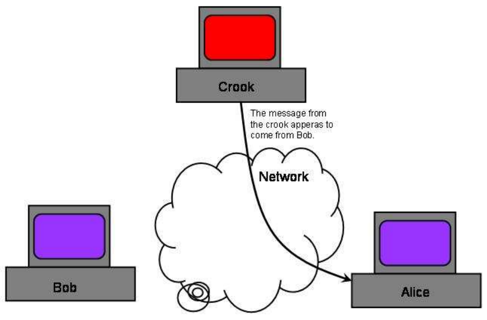


Figure 1.2: Masquerade.

A *masquerade*, shown in Figure 1.2, takes place when one entity in a communication process pretends to be a different entity. Masquerade attacks usually includes other active attacks. For instance, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges, to obtain extra privileges by impersonating an entity that has those privileges.

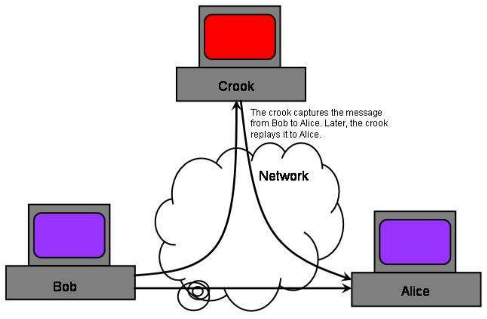

Crook
The crook captures the message from Bob to Alice. Later, the crook replays it to Alice.
Network
Bob
Alice

Figure 1.3: Replay.

Figure 1.3 shows a *replay* attack, that involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect.

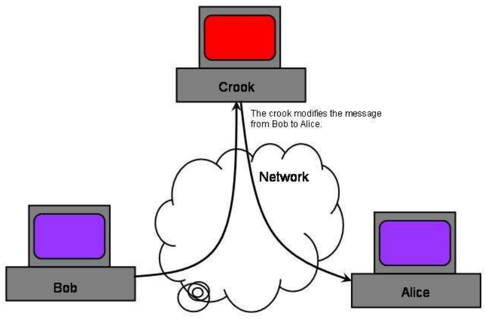


Figure 1.4: Modification of messages.

Modification of messages simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect. The attack is depicted in Figure 1.4.

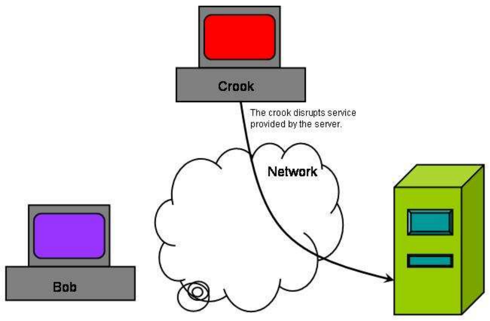


Figure 1.5: Denial of services.

Figure 1.5 shows how a *Denial of Service* (DoS) attack prevents or inhibits the normal use or management of communications facilities. This attack may have a specific target, for example, an entity may suppress all messages directed to a particular destination. Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance.

Passive attacks

Passive attacks are eavesdropping on, or monitoring of, transmissions. The goal of the attacker is to obtain information that is being transmitted. There are two types of passive attacks, namely release of message content and traffic analysis [32].

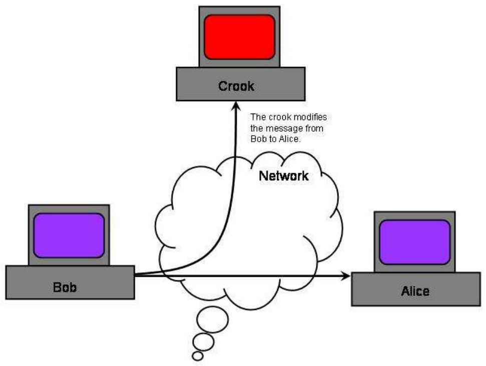


Figure 1.6: Release of message content.

Figure 1.6 shows the passive attack *release of message content*. What happens, is that an attacker gets his or her hands on confidential information. The information can for instance be shared through e-mail, telephone or transferred files. To avoid release of message content, messages can be encrypted.

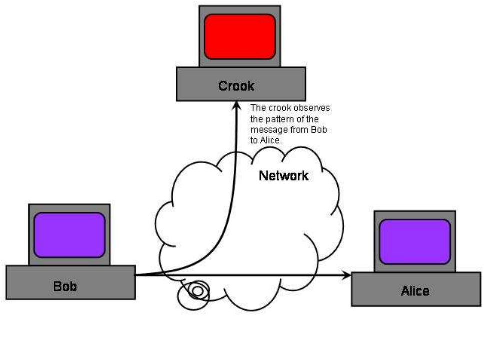

Crook
The crook observes the pattern of the message from Bob to Alice.
Network
Bob
Alice

Figure 1.7: Traffic analysis.

Figure 1.7 describes the *traffic analysis* attack. What happens, is that an attacker analyzes the stream of messages that is sent between users. Then the attacker tries to extract a pattern from the sent messages, to potentially reveal its content.

Passive attacks are difficult to detect because they do not alter any data. It is feasible to protect data by using encryption. The best way of dealing with passive attacks, is prevention rather than detection.

1.4 Countermeasures

Most security products available today are technical solutions that target a specific type of issue, problem or protocol weaknesses. Most of them are products retrofitting security into existing infrastructure, including tools such as application layer firewalls and host/network based Intrusion Detection Systems[7] (IDS).

People argue that the only secure host is the one that is unplugged. Even if that were true, an unplugged host is of no functional use, and is therefore hardly a practical solution to the security problem. But zero risk is not practical, nor achievable either. The goal should be to determine what the appropriate level of security is for the application to function as planned in its environment. This process normally involves accepting some level of risk.

For some risks, the only way to manage them, are to accept them. For instance, if a company has a software system in a closed network, threats only come from within. But if the company finds it necessary for employees to access this system outside their closed network, they have to accept the fact that anyone connected to the Internet can attempt a break-in. This will make the system far more exposed to attacks.

Security will almost always bring about some overhead, either in cost or in performance. It is no use spending a lot of money securing assets that is not very important to secure. Therefore, one should decide which assets one have to secure, and find a way to secure them. To determine what the core assets are, is a key task. This is part of the process of getting to know what needs to be secured.

It is possible to mitigate risks using *countermeasures*. Following is a list with several techniques that can be employed to reduce the risk of security being compromised. Techniques that are emphasized in definitions will be explained immediately in upcoming descriptions.

Encryption. Encryption is the conversion of plaintext or data into unintelligible form by means of a reversible translation, based on a translation table or algorithm. One can encrypt by means of *symmetric encryption* or *asymmetric encryption* [32].

Symmetric and asymmetric encryption. Symmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the same key. Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using two different keys, where one of them is referred to as the *public key* and the other is referred to as the *private key* [32].

Public and private key. The public key is one of the two keys used in an asymmetric encryption system. The public key is made public, to be used in conjunction with a corresponding private key. The private key is the other of the two keys used in an asymmetric encryption system. For secure communication, the private key should only be known to its creator [32].

Secure Socket Layer (SSL). One way to encrypt data is by using the Secure Socket Layer (SSL). SSL is used for authentication and to establish basic parameters for subsequent interaction [32].

[7]An Intrusion Detection Systems is an automated system that can detect a security violation on a system or a network [44].

Digital signature. A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature. The signature guarantees the source and integrity of the message [32]. This can be used when users of a computer system should be able to identify themselves to other users.

Data signing Digital signatures can be created using data signing. This is done by encrypting data using the private key, then anyone who is able to correctly decrypt the data, can easily prove the data origin. Correct decryption means that the data was encrypted by the corresponding public key. As the private key is always kept private and never leaves the possession of the person who generated it, the correct data decryption is proof of their origin [32].

Strong encryption. Strong encryption is plaintext encrypted with ciphers [8] that are essentially unbreakable without the decryption keys [44].

Strong authentication. Authentication is, as mentioned before, the process of uniquely identifying the clients of your applications and services. *Strong authentication* is when the identities of networked users, clients and servers are verified without transmitting passwords over the network [44]. Strong authentication refers to systems that require multiple factors to identify users when they access private networks and applications. A common example of multifactor authentication is your ATM card or bank card. This requires something you have (your card), and something you know (your PIN). This only leaves out the third factor of authentication, namely the part of something you are.

Data hashing. Data hashing is when users produce hash values for accessing data or for security. A hash value (or simply hash), also called a message digest, is a number generated from a string of text. The hash is substantially smaller than the text itself, and is generated by a formula in such a way that it is extremely unlikely that some other text will produce the same hash value. Hashes play a role in security systems where they're used to ensure that transmitted messages have not been tampered with. The sender generates a hash of the message, encrypts it, and sends it with the message itself. The recipient then decrypts both the message and the hash, produces another hash from the received message, and compares the two hashes. If they're the same, there is a very high probability that the message was transmitted intact. If a user wants to make sure that transmitted data has not been tampered with, one can make use of the data hashing technique [32].

Principle of least privilege. The principle of least privilege requires that a user be given no more privilege than necessary to perform a job. Ensuring least privilege requires identifying what the user's job is, determining the minimum set of privileges required to perform that job, and restricting the user to a domain with those privileges and nothing more. By denying subjects transactions that are not necessary for the performance of their duties. The denied privileges can therefore not be exploited by an attacker [42].

Audit trails. An audit trail is a series of records of computer events or user activities. It is generated by an auditing system that monitors system activity [42].

[8] Chipers are systems of concealing a message by substituting individual letters, numbers or symbols with other letters, numbers or symbols. Cipher systems are usually based on a key, or method of encryption [44].

Audit trails can for example prove that actions performed by users actually have taken place in the system.

1.5 Summary

This chapter has defined web applications and the setting in which they operate. Security was defined as preserving the fundamental principles authentication, confidentiality, integrity and availability. This chapter also introduced how security might be compromised by groups of attack, and identified some widespread countermeasures. This is material that will be referenced and elaborated more thoroughly throughout the report.

Chapter 2

Security guidelines

Guidelines are developed to point out important aspects that one should be aware of in different contexts. The guidelines presented in this chapter are high-level security principles concerning software development. They include examples, and have been divided into a general section and a section aimed specifically at developing a sub group of software systems, that is web applications. The general section provides principles that one should pay attention to regardless of the kind of system being developed.

We will be making practical use of both of the groups of guidelines when developing a secure web application in this project. Later on in this report, we will illustrate how they translate, in more specific details, to various parts of the development process. One goal of our project, is to work by these guidelines all through the development process. That is, *to be security conscious from the first point in the development process and onwards*. This is something that we want to emphasize, although being security conscious during the entire process is not a stand-alone guideline. Keeping the focus on security will be referred to throughout the report.

2.1 General security guidelines

This section will present general security guidelines that can be used during development of software systems. They should always be kept in mind, and may provide useful reference points on which one may base design and implementation decisions.

2.1.1 Validate input

When input is referred to, both user- and system input is implied [42]. Although two systems are interconnected and mutually authenticated to each other, there is no guarantee that input coming from one of them hasn't been corrupted during transmission.

Input to and from a system is the route for malicious payloads that may compromise the system. The best strategy for dealing with input and output is to allow only input explicitly defined and drop all other data. Input should therefore be ensured to be both appropriate and expected [20]. Functions that handles the validation must decide if a given input matches expected format. For instance, if input from a user is expected to be an email address, the input should be checked to see if it contains a string, an @, an address, a dot and a top level domain name[1].

If input is not properly validated, an attacker can enter input that will overflow buffers[2] in a computer system. What happens is, that there is put more data into a buffer then the buffer can handle. This can result in that the attacker makes the entire system crash, or that he or she creates a back door leading to system access [42].

2.1.2 Fail securely

Any sufficiently complex system has failure modes. Failure is unavoidable and should be planned for. What in fact is avoidable are security problems related to failure [42]

Any security mechanism should be designed in such a way that when it fails, it fails closed. That is, it should fail to a state that rejects all subsequent security requests rather than allow them. An example would be a user authentication system. If it is not able to process a request to authenticate a user, and the process crashes, the user should not get access to the system [20].

When a system fails, it may be a problem that it reveals critical system information [42]. For instance, error messages from the system might reveal information on how tables and rows are organized in a database. The messages may contain information on weaknesses that can be exploited by an attacker. Detailed error messages should therefore never be sent to the client, but rather to a system administrator, or to a log.

2.1.3 Keep it simple

Complexity increases the risk of problems concerning security. A complex design is never easy to understand, and is therefore more likely to include subtle problems. Complex code tends to be harder to maintain as well. To stay out of this kind of trouble, design and implementation should be as straightforward as possible in a software development process [42].

It might be tempting bo develop an elaborate and complex security control, but this should be avoided, as users will just end up finding a way around these measures. Even though it might be a good idea, theoretically, to assign all users a number of passwords, and then ask for a random one of them upon authentication, it will probably result in the user writing down all the passwords, and thereby make the effort useless. Often the most effective security is the simplest security [19].

It is possible to make a system less complex by leading all critical security operations through a choke point. At this choke point, all traffic passing through will be checked by an interface. The choke point might for instance be a gateway that supervises and controls all traffic, giving control of the information flow.

2.1.4 Use and reuse trusted components

When developing computer systems, it should be considered to reuse components that are believed to be of good quality. A component is of good quality if it is thoroughly

[1] A domain name is the unique name of an Internet site. A top level domain name, is the last part of a domain address. This can be .gov for governmental institutions, .org for organizations, .com for commercial industries and so on [44].

[2] An area of computer memory that is used to temporarily store data[44].

tested by people you trust, and no documents pointing out critical weaknesses have been published [42].

The more successful a component has proven itself to be over time , the better reason to reuse it. Many people developing computer systems have gone through the same kind of problems, and may have invested large amounts of time researching and developing robust solutions to them. In many cases, components have been improved through an iterative process.

Using and reusing trusted components makes sense both from a resource stance and from a security viewpoint. When someone else has proven they got it right it is beneficial to take advantage of it [19].

For example, it is sensible to reuse cryptographic libraries. Well-used libraries are much more likely to be robust than something put together in-house, because people are more likely to have noticed implementation problems [42].

2.1.5 Defense in depth

The idea behind defense in depth is to manage risk with diverse defensive strategies. It is important to make sure that various layers of security in a system works together in harmony to supplement each others functionality, but also to overlap, so that if one should fail, another layer will prevent a total compromise of the system [20].

It is unrealistic to rely on one component to perform its function perfectly at all times. Predicting unexpected events may be hard, but it is certainly necessary. One should take the necessary precautions so that *if* any unexpected event takes place, there is a plan of how to take care of it.

For instance, information can be stored in a database protected with an authentication mechanism where only authenticated users are given access. But if an attacker manages to bypass the security mechanisms, the information stored in the database will be disclosed. Encryption of sensitive data before storage, will preserve the principal of defence in depth.

2.1.6 Secure the weakest link

Security might be looked upon as a chain. And just as a chain is no stronger than the weakest link, a software system is only as secure as its weakest component. Attackers will attack the weakest part of the system because they are the parts most likely to be easily broken [42].

It is probably not possible to make a system perfectly safe of security breaches. One can secure transits between the users browser and the web server, but there might be lack of security elsewhere in the system. It is not enough to secure a system at one point.

For instance, it won't help if a computer system has a well functioning authentication mechanism, if the system has many easily available backdoors[3], leading right into the systems core. Therefore it is important to secure all aspects of the system.

[3]A backdoor in a computer system is a method of bypassing normal authentication or obtaining remote access to a computer, while intended to remain hidden to casual inspection. The backdoor may take the form of an installed program or could be a modification to a legitimate program [44].

2.1.7 Practice the principle of least privilege

This principle states that only the minimum access necessary to perform an operation should be granted, and that access should be granted only for the minimum amount of time necessary [42].

Systems should be designed in such a way that they run with the least amount of system privilege they need to do their job. This is the "need to know" approach. If a user account doesn't need root privileges to operate, don't assign them in the anticipation they may need them [20]. The reason why a system should operate in this fashion is to avoid potential misuse and reduce the damage that can occur should your code be exploited by a malicious user. If your code only runs with basic user privileges, it is difficult for malicious users to do much damage with it.

2.1.8 Practice compartmentalization

The basic idea behind compartmentalization is to minimize the amount of damage that can be done to a system by breaking up the system into as few units as possible, while isolating code that has security privileges [42].

Similarly, compartmentalizing users, processes and data helps contain problems if they do occur. Compartmentalization is an important concept widely adopted in the information security realm.

In a computer system where for instance some information are public, while other is classified, the data can be compartmentalized according to the level of security they need. For instance, the public information can be stored on a server that can be accessed from the Internet, while the classified information may be stored on a server within a secure network zone.

2.1.9 Awareness

People who uses computer systems need to know how to use the system, and how to be conscious about security issues. To communicate security policies and procedures to users of computer systems, and getting their commitment to adopt these methods, is an important way of lowering the risk of loss or damage to a computer system. Users should be aware of data security and protection principles, and what actions that might be a danger to the system security and the data confidentiality. Making legitimate users aware of the risks they may constitute to a system they are working with, may reduce, and even prevent, illegitimate actions and breaches in security. As users of computer system needs privileged access to do their job, an example of user awareness might be that a user should never write down a password, or leave his or her workstation unattended.

2.2 Specific web application security guidelines

Compared to traditional software systems, web applications exist in challenging environments. They are potential victims of attacks launched by anyone connected to the Internet, and we believe they should be provided separate guidelines.

Following are three guidelines aimed directly at the development of secure web applications. They should be applied during the development process, guidelines should be applied when developing web applications, in addition to the general ones.

2.2.1 Validate input on the server side

When developing web applications, it is important to validate input on the server side. All validation that takes place on the client side is exposed of being altered by potential attackers. Input from the client side can therefore compromise the entire system when it arrives at the server [9].

2.2.2 Encrypt communication lines

In web applications, a client communicates with one or more distributed systems. If the information exchanged between these systems are considered sensitive, the communication lines should be encrypted. This should be done to prevent attackers from getting their hands on the information being sent [9].

2.2.3 Limit cache usage

Cache is local storage of remote data designed to reduce network transfers [44]. In web applications, where data is likely to change frequently, the value of information being up to date is great compared to more static systems.

It is important that users are provided information that is correct. For instance in a web site publishing stock exchange information, the rate value needs to be immediately updated in the web page should it change. In such systems, where change is frequent, caching will not do anything good. It is more important to serve users updated information than to save them response time by providing information from cache instead of going all the way to the web server.

Also, web pages that contain sensitive information should not be cached. For example, if a doctor have access to check his or her patients health records over the Internet, and the doctor uses a computer that others also have access to, the cached information can go devious ways. Therefore it is important not to store web pages that contains sensitive information in cache.

2.3 Summary

This chapter has identified a total of twelve security guidelines. Together they provide an overview of what to take into consideration when developing secure software and applications. Most of the guidelines introduced are applicable to all software systems, whereas the last three represents web applications more specifically. This report is on security in web applications, and the following chapters will focus on this subject in particular.

In succeeding chapters, we will describe vulnerabilities that are often discovered in web applications, and the threats that are directed towards them. Subsequent chapters will describe the development of a web application through requirements engineering and design. Then parts of the designed application will be implemented, where the goal is to see how some of the security issues pointed out in this report, can be solved on the Java platform.

Throughout the entire development of the application, we will make use of the security guidelines introduced in this chapter. It will be interesting to figure out to what extent these guidelines may help us focus on security through the development process, and wether this approach will contribute to making the system less vulnerable.

Chapter 3

Vulnerabilities, threats and countermeasures

This chapter will introduce the most common vulnerabilities and threats that can be directed towards computer systems today. The overview of threats is constantly changing, and it is important to keep oneself posted on changes that might occur in the overview.

The vulnerabilities described in this chapter, will be dealt with one by one, and countermeasures will be suggested for each one of them. The same goes for the threats. We take a closer look upon vulnerabilities and threats in order to identify where security is threatened. We mainly focus on web applications, but some of the vulnerabilities and threats described, will also apply to software systems in general. In subsequent chapters in this report, we will design a web application, and point out where in the application different security breaches may appear, and relate these to vulnerabilities and attacks presented in this chapter.

3.1 Vulnerabilities

A vulnerability is a weakness that makes a threat possible [20]. There are several vulnerabilities that may arise when developing web applications. A vulnerability might be defined as a weakness in the software and/or hardware design that allows circumvention of the system security.

It is important to identify vulnerabilities that exists, in order to work against them during the development of applications. Therefore, this section, will describe the most widespread vulnerabilities, along with countermeasures to counteract them.

3.1.1 Unvalidated input

This vulnerability comes into play when information from web requests is not validated before it is used by a web application. Attackers can may exploit vulnerabilities of this kind to attack back end components through a web application. An application uses input from HTTP requests to determine how to respond. Attackers may tamper with any part of the HTTP[1] request, namely the URL, query strings, head-

[1]HTTP defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands [44].

ers, cookies, form fields and hidden fields[2]. All this to try to bypass the site's security mechanisms [43].

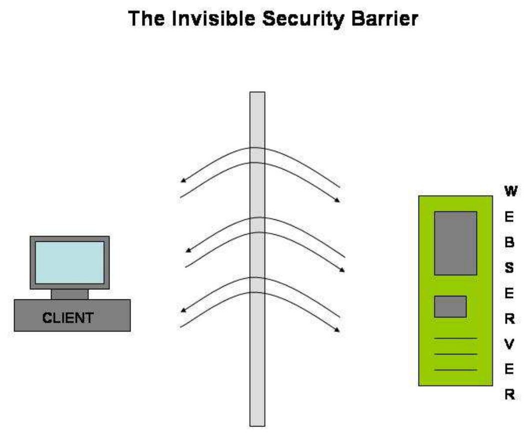


Figure 3.1: The invisible security barrier is a conceptual barrier between server and client. Everything that has crossed this barrier from the client to the server, may be unsafe [9].

All information sent from the server to the client side, and back again, must be validated on return to the server side [9]. This is necessary because just about *anything can happen on the client side*. When data is sent from the server to the client, it crosses **the invisible security barrier** shown in Figure 3.1 [9]. Although the data was correct when it left from the server, there is no guarantee that the information is unchanged on return. Everything that has crossed the invisible security barrier from client to server, may be unsafe. Client side validation is a nice idea for performance and usability, but it will not contribute to the security of a system at all.

Countermeasures

The possible consequences of not validating input should not be taken lightly as back end components may be compromised. A huge number of attacks can be prevented if the developers simply *validate input before using it.* Unless a web application has a strong centralized mechanism for validating *all* input, vulnerabilities based on malicious input is likely to occur. It is important to remember that not only may input come from users, but also from for instance systems or from files.

To complicate the task of possible attackers, it is important that input parameters are converted to their simplest form before they are validated. This will help avoiding that a masked input slips through filters. Another important aspect is that all parameters should be checked against a strict format specifying exactly what is expected input. If a e-mail address is required, one should expect input of a certain format, that would look something like: someone@something.com. The opposite

[2]For definitions of these terms, see "Glossary" in Appendix A.

technique, of filtering certain bad input, for instance quotes, is likely to be difficult to maintain and also unlikely to be a complete coverage of possible bad inputs.

3.1.2 Cross site scripting flaws

One of the most widespread techniques that exploit vulnerabilities of systems where data is not properly validated, is Cross Site Scripting (XSS). XSS is a weakness in an application that give attackers the opportunity to plant incorrect information that to the user seems to come from a serious web site, as shown in Figure 3.2 [9]. This

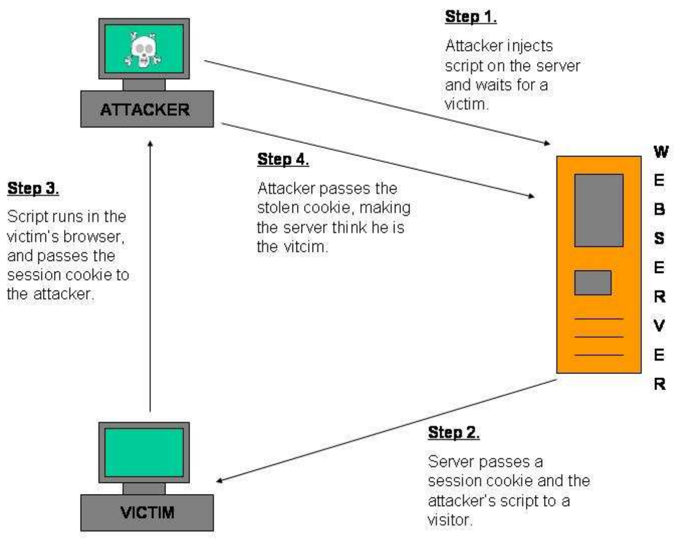

Step 1.
Attacker injects script on the server and waits for a victim.
ATTACKER
Step 4.
Attacker passes the stolen cookie, making the server think he is the vitcim.
Step 3.
Script runs in the victim's browser, and passes the session cookie to the attacker.
WEBSERVER
Step 2.
Server passes a session cookie and the attacker's script to a visitor.
VICTIM

Figure 3.2: Example of cross site scripting.

vulnerability also makes it possible for an attacker to hijack communication sessions and thereby pass his or her self off as someone else, with the intent of executing malicious code. This code might make it possible for the attacker to eavesdrop, destroy and/or tamper with data on random client machines [24].

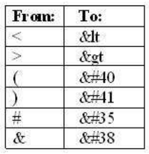
A survey conducted by Secode from 2005 [25], points out that 81% of their test objects were exposed to XSS attacks. The trend is that the use of XSS is decreasing [24]. But still, vulnerabilities of this kind are discovered all the time. For instance, in April 2005 an XSS vulnerability in PHP was reported, which could be exploited by attackers to inject malicious HTML codes [8]. This flaw was due to an input validation error in the "index.php" script when handling a specially crafted parameter, which could be exploited by attackers to cause arbitrary scripting code to be executed by the users browser.

The effects of a XSS attack can vary from simple annoyance to complete account compromise. The most severe XSS attacks involves disclosure of a users session cookie[3] Other damaging attacks include the disclosure of end user files, installation

[3]Temporary cookie stored in a computers memory for remembering preferences during a web site visit, which is destroyed when leaving the site [44], allowing an attacker to hijack the users session and take control of the victims account.

of Trojan horses[4], redirecting a user to some other page or site, and modifying presentation of content.

Countermeasures



From:	To:
<	&lt
>	&gt
(	&#40
)	&#41
#	&#35
&	&#38

Figure 3.3: HTML entity coding.

There exist countermeasures one can make use of to avoid XSS. The best way to prevent a web application from XSS is validating headers, cookies, query strings, form fields and hidden fields against a rigorous specification of what should be allowed. It is also a good idea to convert the characters in Table 3.3, in all generated output, to the appropriate HTML entity coding. The reason why these characters should be converted is because it may prevent inserted scripts to be transmitted to users in executable form, and it might thus provide some protection from javascript based attacks.

3.1.3 SQL injections

The most common type of injection attack, is the *SQL injection*. Secode reports in [24] that 31% of the web applications they tested were vulnerable to SQL injections. This is a result of developers that doesn't manage to filter input good enough when it comes from the client headed for the server.

SQL injection is a technique by which attackers can execute SQL statements of their choice on the back end database by manipulating the input to the application. An example SQL injection might be directed at login functionality, where the user needs to provide a username and a password. Suppose that the user enters the following: Username: Marcus and Password: Safari. This input will then be used to dynamically build a query that looks something like this: SELECT * FROM Users WHERE username = 'Marcus' and password = 'Safari'. This would return to the application a row from the database with the given values. The user is considered authenticated if the database returns one or more rows to the application. Now, suppose that an attacker enters the following input to the login procedure: Username: ' or 1=1–. The resulting query will look like this: SELECT * FROM Users WHERE username = ” or 1=1– and password=” The notation – makes the rest of the line a comment. This means that the query effectively looks like this: SELECT * FROM Users WHERE username=” or 1=1”. This query will search the database for rows where either the username is blank or the condition 1=1 is met. Since the latter always evaluates to true, the query will return all rows of the “Users” table, and the user is authenticated.

The SQL injection is very dangerous, and almost all platforms are vulnerable to this attack. To be able to exploit this flaw, the attacker must find a parameter

[4]An apparently useful and innocent program containing additional hidden code which allows the unauthorized collection, exploitation, falsification, or destruction of data [44].

that the web application passes through to a database. The attacker can then trick the web application to forward malicious commands to the database, by embedding malicious SQL commands into the content of a parameter. Consequences of SQL injection are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents [21].

Countermeasures

To avoid SQL injections certain countermeasures exist. First of all, never pass detailed error messages to the client. Error messages can provide an attacker with valuable system information [9]. Another countermeasure is to validate input in such a way, that every possible meta character to a subsystem is identified. In this way, possible attackers can't make the necessary inputs to the application to compromise the system. It is also possible to use the technique of canonicalization[5].

3.1.4 Buffer overflow

Buffer overflow is a vulnerability that alter the flow of an application by overwriting parts of memory. It is a common software flaw that might result in an error condition. This error condition occurs when data written to memory exceed the allocated size of the buffer. As the buffer is overflowed, adjacent memory addresses are overwritten, and might cause the software to fault or crash. By sending carefully crafted input to a web application an attacker might cause the web application to execute arbitrary code, and this means that the attacker is effectively taking over the machine.

Buffer overflow vulnerabilities have become quite common in the information security industry and have often plagued web servers. However, they have not been commonly seen or exploited at the web application layer itself. The primary reason is that an attacker needs to analyze the application source code or the software binaries. Since the attacker must exploit custom code on a remote system, they would have to perform the attack blindly, making success very difficult.

Countermeasures

The countermeasure used to avoid buffer overflow attacks is validation. One should always check that all input has reasonable length. It doesn't matter if the input comes from a user, from files or from another system; all input should always be checked. In high level languages like Java, Perl, PHP, VBScript and so on, buffer overflows generally cannot occur [42]. The languages have enforced size restrictions on arrays, which means that one cannot store something in the arrays unless it fits. In medium-level languages like C and C++, nothing stops the program from storing bytes past the end of the array. This requires the developers attention to make the code safe from possible attacks [9].

3.1.5 Broken authentication and session management

Authentication and session management includes all aspects of handling user authentication and managing active sessions. Authentication is a critical aspect, but solid

[5]Canonicalization is a way of simplifying encoding. Some sites protect themselves by filtering out malicious input. The problem is that there are many ways of encoding input. Therefore, parameters must always be decoded to the simplest form before they are validated, or malicious input may be masked and can slip past filters [21].

authentication mechanisms may be undermined by flawed credential management functions, including forgot password feature, password change, account update, and other related functions. Because "walk by" attacks, that is attacks where a privileged user leaves his or her workstation unattended and and attacker exploits this to get privileged access, are likely for many web applications, all account management functions should require re-authentication even if the user has a valid session identifier [9].

User authentication in a web application typically involves requiring the user to provide of information on something he or she knows, like a valid username and password. Stronger methods of authentication, like biometrics[6] (something the user is), are commercially available, but in most applications they are cost prohibited.

Web applications must establish sessions to keep track of the stream of requests from each user. HTTP does not provide this capability, so web application must create it themselves. Account credentials and session tokens needs to be properly protected, otherwise authentication and session management can be broken. Attackers may compromise passwords, keys, session cookies, or other tokens to defeat authentication restrictions and assume other user' identities [21].

Unless all authentication credentials and session identifiers are protected with Secure Socket Layer (SSL) at all times and protected against disclosure from other flaws, such as cross site scripting, an attacker can hijack a user's session and assume their identity.

Countermeasures

Be sure to design a robust and secure authentication and session management scheme that is consistently enforced. In addition be careful so that user credentials are properly handled. Some key areas are to control the password storage, they should be stored in either hashed or encrypted form for protection. Hashed form is preferred as it is not reversible. Credentials in transit should be protected, for instance by employing SSL. Concerning the HTTP protocol, authentication and session data should never be submitted as part of a GET[7], POST[8] should always be used instead.

3.1.6 Broken access control

Access control is how a web application grants access to content and functions to some users, and not others. These checks are performed after authentication, and govern what authorized users are allowed to do. Access control sounds easy enough, but is hard to implement correctly [32].

It is easy to underestimate the difficulty of implementing reliable access control mechanisms. Usually the access control rules are inserted different places in the code, and one cannot create an image of how it all is put together, and this makes it almost impossible to understand.

Many of these access control schemes are not difficult to reveal and exploit. Once a flaw is revealed, the consequences of a flawed access control scheme can be devas-

[6]Security and authentication measures which rely on biological characteristics though to be sufficiently unique. For example, physical characteristics such as finger prints, retina structure, voice recognition, or facial recognition [4].

[7]GET is a request used by the HTTP protocol, and should be used when a client asks for information [9].

[8]POST is a request used by the HTTP protocol, that should be used when a client performs actions where something is permanently changed on the server [9].

tating. Attackers can then exploit the system to access other users accounts, view sensitive files, or use unauthorized functions [21].

One specific type of access control problem is that administrative interfaces allow site administrators to manage a site over the Internet. Due to the administrators powers to change user rights, data and content on the site, these interfaces are frequently prime targets for attack by both outsiders and insiders.

Countermeasures

Preventing access control from being broken can be done by thoroughly planning the access control scheme before implementing it. When designing the access control scheme, always keep in mind the *principle of least privilege.* It is also useful to review logs to spot any attempts, or maybe even success', to break the access control scheme.

3.1.7 Improper error handling

A common problem is when detailed internal error messages such as stack trees, database dumps, and error codes are displayed to the user. These messages reveal implementation details that always should be kept secret from outsiders. The messages can give the user important clues of potential flaws in the site and such messages are also disturbing to normal users [9]. The attacker can, through improper error handling, gain detailed system information, deny services, cause security mechanisms to fail, or crash the server [21].

Frequent errors from the web application may be out of memory, null pointer exceptions, system call failure, database unavailable, network timeouts and many more.

Countermeasures

A policy for how to handle errors should be documented, including the types of errors to be handled, and for each of these what information should be reported to the user, and what information should be logged. The goal should be to provide a meaningful error message to the user, diagnostic information to the sites maintainers, and *no useful information to the attacker.* Simple error messages should be produced and logged so that their cause, whether an error in the site or a hacking attempt, can be reviewed. Error handling should not focus solely on input provided by the user, but should also include any errors that can be generated by internal components such as system calls, database queries, or any other internal function.

3.1.8 Insecure storage

Most web applications need to store sensitive information, either in a database or on a file system somewhere. The information might be passwords, credit card numbers, account records, or proprietary information. Frequently, encryption techniques are used to protect this information. Although encryption techniques has become more easier to implement and use, developers still make mistakes while integrating this techniques into web applications. They overestimate the protection gained by using encryption, and forgets to secure other aspects of the site.

Countermeasures

When you are in need of using encryption, use a public library that has been exposed to public scrutiny and make sure that there are no open vulnerabilities. Do not develop a new encryption algorithm. One should try to minimize the use of encryption, and only keep information that is absolutely necessary. For instance, rather than encrypting and storing credit card information, simply ask the user to re-enter the numbers [21].

3.1.9 Insecure configuration management

Web server and application server configurations play a key role in the security of a web application. The servers are responsible for serving content, and invoking applications that generate content. Some application servers also provide other services like data storage, directory services[9], mail and messaging [42].

Problems with the configuration can be detected by an attacker with readily available security scanning tools. Once detected, these problems can easily be exploited and result in total compromise of a web site. Successful attacks can also result in the compromise of back end systems including databases and corporate networks.

Countermeasures

It is critical to have a strong server configuration standard to secure a web application [21]. Keeping the server configuration secure requires that the ones responsible are on the alert. One should pay attention to published security vulnerabilities, and make sure the system is updated. The latest security patches should be applied, and one should make sure that configurations, for instance of SSL, are correct.

3.2 Threats

A threat is a potential occurrence that may harm an asset [20]. These occurrences may be malicious. That is, it might be a deliberate attempt to harm an asset. If such an action occurs it is called an attack.

There are several different types of attacks. The attacks that will be treated here are categorized into seven groups; spoofing, tampering data, tap communication, repudiation, information disclosure, Denial of Services (DoS), and elevation of privileges. These will all be described in following subsections.

3.2.1 Spoofing

Spoofing is an attempt to access a system by using a false identity. There are several ways to do this, for example using stolen user credentials or a false IP address [21].

To get a hold of someones user credentials, one may make use of a social engineering[10] technique called *phishing*.

[9]A directory service organizes content in a directory server into a logical and accessible structure.

[10]In the realm of computers, the act of obtaining or attempting to obtain otherwise secure data by conning an individual into revealing secure information. Social engineering might be successful because its victims innately want to trust other people and are naturally helpful. The victims of social engineering are tricked into releasing information that they do not realize will be used to attack a computer network [44].

Phishing is an expanding problem today, and the kind of phishing that will be presented here, is getting more and more sophisticated as attackers are starting to directly address individuals as opposed to addressing the mass marked [22]. The attackers will send e-mails containing personal information that is already stolen, the idea being that for instance bank customers will be more likely to give up valuable information, like the pin-code to their bank card, when presented correct personal information. The receiver of the e-mail is sent to a page that looks very similar to the real bank page, by pressing a link in the e-mail. The page is in fact placed on a bot net[11], though. At the malicious page, the victim is asked to provide personal information. When this is completed, the user is typically sent to his real Internet bank page and may continue surfing with no worries. The consequence of a phishing attack varies in accordance with the goal of the attack. Often, economical gain is the motivation behind a phishing attack, but in-house company information might also be a target.

Another spoofing technique is *IP spoofing*. IP spoofing is when an attacker forges the IP address in packets going from the attacking box, so that the packets appear to come from an address that should be authenticated when received by the recipient [42].

In most situations, IP spoofing by itself is not good enough for the attacker to make a successful attack. For instance, the attacker needs to make sure that the fake packets are actually routed to the target , and even if packets makes it to the target, responses are routed to *the forged IP address*. In order for the forgery to be useful, the attacker needs *to receive the responses*. The only real way to accomplish this is for the attacker to become interposed somewhere on the route between the spoofed machine and the target.

IP spoofing are, as one can see, difficult to execute. There are tools that largely automate individual IP spoofing attacks, but the attacker must possess some technical insight to perform the attacks using the tools available [42].

Countermeasures

There are different countermeasures that may prevent spoofing attacks from happening. When it comes to phishing, the only way to encounter this problem, is to make the users aware of the fact that attackers exploit users naivety to get their hands on sensitive information. To make users more critical to information received in e-mails, is useful.

As to IP spoofing, one may use countermeasures like strong authentication[12] and encryption to avoid attacks. Simple rules are not to store secrets in plain text, not pass credentials in plain text over the wire, and protect authentication cookies with SSL [36].

3.2.2 Tampering data

Tampering is unauthorized modification of data. It usually takes place when the data is flowing over a network between computers. When data is tampered with, it

[11]A bot (robot) is a machine that has been compromised and is controlled externally. A bot program, that has somehow entered the machine opens a port to receive instructions on. Instructions might be to download new versions of bot software, to send out viruses or spam, or to participate in denial of service attacks against specified sites. At all times, ten thousands of machines over the entire world are active in bot nets without the legitimate owners being aware of it [31].

[12]Strong authentication is described in chapter 1, “Web applications and securiy issues”.

is compromised [32].

Tampering data is known as an active attack. There are several widespread tampering techniques. These techniques either involve some modification of the data stream or the creation of a false stream. The attacks are *the masquerade attack*, *replay attack*, *modification-of-messages attack* and *Denial of Service (DoS) attack*. A masquerade attack takes place when one entity pretends to be a different entity [32]. A replay attack occurs when a message, or part of a message, is repeated to produce an unauthorized effect [32]. Modification of messages occurs when the content of a data transmission is altered without detection and results in an unauthorized effect [32]. An example might be when a message "Allow John Mils to read confidential files" is changed to "Allow Agatha Anderson to read confidential files". Denial of service occurs when an entity fails to perform its proper function [32]. Examples are generation of extra traffic or messages intended to disrupt the operation of the network. Denial of service will be treated separately.

Countermeasures

Active attacks are quite difficult to protect from, as it would involve physically protecting all of the communication facilities and transmission paths at all times. Instead the goal is to detect them and to recover from any disruption or delays caused by them. Because detection has a deterrent effect, it may also contribute to prevention [32]. To prevent attackers from tampering with data, one can use countermeasures like data hashing, data signing, digital signatures, strong authorization [13], or make use of a connection-oriented integrity service[14].

3.2.3 Tap communication

Tapping of communication is unauthorized eavesdropping, or traffic analysis, of messages flowing over a network. When data is tapped, it looses integrity [32].

There are several techniques an attacker can make us of for tapping communication flowing over a network. These techniques are known as passive attacks, more exactly release of message content or traffic analysis.

Release of message content is when an attacker get his or her hands on information flowing over the network. *Traffic analysis* is when an attacker analyzes the stream of messages sent between users. The attacker tries to extract a pattern from the sent messages.

Countermeasures

Passive attacks are very difficult to detect because they do not involve any alteration of data. However, it is feasible to prevent the success of these attacks, usually by means of encryption. Thus, the emphasis when dealing with passive attacks is on prevention rather than detection.

[13]Data hashing, data signing, digital signatures and strong authorization is described in chapter 1, "Web applications and security issues".

[14]A connection-oriented integrity service assures that messages are received as sent, with no duplication, insertion modification, reordering or replays. The destruction of data is also covered under this service. Thus, the connection-oriented integrity service addresses both message stream modification and denial of services [44].

3.2.4 Repudiation

Repudiation is the user, legitimate or not, denying that he or she performed specific actions or transactions. To prevent repudiation, one should require non-repudiation, which prevents either sender or receiver from denying a transmitted message [32]. Nonrepudiation is a method of proving either that a user performed an action (such as enrolling in a stock plan or applying for a car loan), or that the user sent or received some information at a particular time. This prevents the individual from fraudulently reneging on a transaction. By comparison, if you purchase an item, you might have to sign for the item upon receipt. If you decide to renege on the deal, the vendor can simply show you the signed receipt.

Countermeasures

There are three techniques that are useful when implementing nonrepudiation. *Digital signatures* can function as a unique identifier of an individual, much like a written signature, and therefore prevent repudiation. *Confirmation services* avoids repudiation because the message transfer agent creates digital receipts to indicate that messages were sent and/or received. *Time stamps* contain the date and time a document was composed and proves that a document existed at a certain time, and therefore avoids repudiation.

3.2.5 Information disclosure

Information disclosure is unwanted exposure of private data. A user may, for instance, view the content of a table or file he or she isn't supposed to open, or monitor data passed in plain text over a network [21].

A simple method to disclose information is, according to [9], to rely on the error supporting system. In some cases, an attacker may provoke an error by inserting invalid SQL[15], and be rewarded with a wealth of information. The information can be names of tables and columns from the database used by the application. The attacker can make use of this information to compromise information stored in the database.

Another way an attacker may proceed to attack a system, is to look for "old" files. These files might contain revealing information about the system that can give the attacker unwanted access to private data. An example of this, is if the script file is called main.asp. Then the attacker can try out "old" file names like for instance main.bak, main.old or main.asp˜ and other previous-version-type extensions.

Countermeasures

Countermeasures that will help prevent information of being disclosed, is to use strong authorization, strong encryption, secure communication links with protocols that provide message confidentiality. Also, avoid storing secrets, like passwords, in plain text, and don't let "old" files be accessible through the Internet [9].

[15]Short for Standard Query Language. SQL is a standardized query language for requesting information from a database. More on SQL on http://www.sql.org.

3.2.6 Denial of service

During a Denial of Service (DoS) attack, an attacker attempts to stop legitimate users from accessing a service, or information. By attacking the users machine, or a web site the user is trying to access, the attacker might be able to prevent the user from his or her legitimate access [21].

The most regular form of DoS attack, is to *flood the network* with useless traffic. When a user enters an URL into his or her web browser, a request is sent to the inquired web site to view the page. A web site host machine can only handle a certain amount of requests simultaneously, and if an attacker successfully overloads the host machine with requests, the host can't handle the users requests anymore, and thus it breaks down [43].

Another attack, is when an attacker *locks out a legitimate user* by sending invalid credentials until the system locks the account. This is possible in systems where a user is only provided a limited number of attempts to enter a valid username/password combination. If an attacker provides a valid username in combination with an invalid password a number of times, the account might be locked, and thus the legitimate user is denied access on his or her return. The attacker could also request a new password for a user, forcing the legitimate user to access his or her email account to regain access [21].

An attacker could also *flood the system* by registering many new users through a registry form. If the attacker runs a script that generates random input to a registration form, and this input is accepted and stored in the database, the attacker might be able to occupy system resources. The registry form, and in worst case the entire system, will then be unavailable to legitimate users.

Corresponding, an attacker can *flood your e-mail account* with spam-mail[16]. There is always a limit on how many e-mail a user can keep in their mailbox. If an attacker sends many, or huge, e-mails to a users mailbox, it will prevent the user from receiving legitimate e-mails [30].

A Distributed Denial of Service (DDoS) is when an attacker uses a legitimate users computer to attack another computer. By exploiting security gaps or vulnerabilities, an attacker can take control of legitimate users machine. The attacker will then be able to use the legitimate users computer to send great amounts of data to a web site, or to spam particular e-mail addresses. The attack is said to be distributed since the attacker uses several computers to execute the attack [30]. As a DDoS attack involves multiple machines cooperating on an attack at a specified target, a bot net will be a excellent tool conducting it. If the bot net is built from machines located at many different locations this is an advantage for the attacker [31].

Countermeasures

Unfortunately there are no effective countermeasures that will prevent DoS- or DDoS attacks from happening. But there are ways that will reduce the probability of an attacker managing to take control over your computer and use it to attack other computers. Some of these countermeasures are enlisted below, and comes from [30], [33] and [43].

- Use bandwidth throttling techniques and validate and filter input.

[16]Spam is electronic junk mail [44].

- Install and maintain antivirus software and a firewall. The firewall should be configured to only allow limited amounts of traffic to and from the computer.
- Keep e-mail address' safe, by not posting it on the Internet in its correct form. Do not open suspicious e-mails either, and use a spam-filter to filter out spam-mail.
- If a user forgets his or her password, and requests a new one, the system should require the user to enter some personal information.
- Use time delays instead of locking users out of their accounts when their password is entered wrong multiple times.
- Use load balancing techniques to make a potential attack more difficult to perform. Still it is possible to perform DoS attacks, and especially if sessions are tied to a particular server. This is why one should make an applications session data as small as possible and make it somewhat difficult to start a new session.

3.2.7 Elevation of privileges

Elevation of privileges occurs when an attacker assumes the identity of a more privileged user to gain privileged access to an application. For instance, an attacker might elevate his or her privilege level to compromise and take control of a highly privileged and trusted process or account [32].

In security terms, privilege means "permission to do something, but only if there is some verification of identity" [6]. If a user have the credential, access is granted. If the user don't have the credential, access is denied.

Elevation of privilege is not a class of attack as much as it is the process of any attack. Virtually, all attacks are attempts on performing actions the attacker is not privileged to do. He or she wants to somehow leverage limited privilege, and turn it into more elevated privilege.

Most commonly, attackers guess or otherwise obtain the security tokens of a user who has higher privileges than the attacker. Thus, you'll often hear the term "elevation of privileges" sloppily used to mean gaining someone else's user ID. But that is just one of the ways to elevate privileges.

The attacker can gain elevated privileges by exploiting physical premises, which can be done as in the following scenario: A user with privileges walks away from his or her on-logged computer for a brief moment. An attacker approaches this computer, uses it, and therefore gains elevated privileges through this machine [42].

Another way the attacker can elevate his or her privileges, is by making use of a social engineering technique and by this make use of a well-meaning person to perform the attack [42].

The attacker can also run code on the victim's machine borrowing the privileges of one of his system-level applications. The attacker finds a process that is running with greater privileges than he or she has, and tries to mess it up. As it crashes, the attacker does something that makes it give its privileges to him or her. Attackers typically use a buffer overflow[17] to accomplish this.

[17]A buffer overflow attack interrupts the program as it executes, and makes it run additional code supplied by the attacker. A successful buffer overflow can take an attacker from zero privileges to total control.

Countermeasures

To prevent the problem of elevated privileges one should secure physical premises and make users aware of software engineering (awareness). Other countermeasures are to follow the principle of least privilege, and use least privileged service accounts to run processes and access resources. One should also review logs for abnormal activity, keep up-to-date on patches, have strong passwords, manage settings aggressively and deploy server security software[18].

3.3 Summary

In this chapter vulnerabilities and threats relevant to web applications have been presented. Vulnerabilities are weaknesses that make a threat possible, and threats are potential occurrences that may harm an asset. Both groups are presented with possible countermeasures that will either help avoiding the problem, or decrease the possibilities of a successful attack. It is important to be aware that there are many different vulnerabilities and threats to consider when developing a web application. Even if the application that is developed is a small one, the threat overview it is exposed to might be just as massive. Another important aspect is the possible consequences of an attack. They have also been introduced in this chapter. When developing a computerized system one should try to rate which attacks are most relevant. The intent of doing this is to be capable of protecting the assets of the system in the best possible way by employing appropriate countermeasures to avoid the highest ranking threats.

The next chapter represents the start of our development process, and will begin by identifying the requirements to the application.

[18]For definition of Server Security Software, see Appendix A, “Glossary”

Chapter 4

Requirements engineering

This report is on web application security. The best way of showing how to include security in an application, is by making one. That is what this project will do. The first step of making a secure web application is to make sure that you design for security. And before you design anything, you should identify the requirements that you need your application to fulfill. During this project we will be working according to the waterfall model, as shown in Figure 4.1 [40]. The waterfall model particulary expresses the interaction between subsequent phases in a development process. It is important to be aware that before moving on to a subsequent phase, one should compare the results obtained to those that are required. In all phases, quality should be assessed and controlled. The reason why we chose to work by the waterfall model, as opposed to various alternatives, is that it maps clear cut tasks to each phase. We have adapted the waterfall model to our project by including a risk analysis in the requirements engineering part.

This has been done because we wanted to be able to identify vulnerable processes, and highlight those most relevant in the design. It is thus possible to specifically design to avoid the most relevant threats.

In this chapter we will be presenting our work with requirements engineering. The next chapter will contain the design of our application, and after that the implementation will be presented. We will not be reaching the testing phase of the waterfall model, as this would require a full-grown system. Maintenance is naturally not treated either. Rather the focus will be set on the design of a secure application and highlight security vulnerable processes.

This chapter will identify our requirements. First they will be presented in tables, and then this chapter will use misuse cases to model potential threats to the application. Misuse cases will shed light on potential attacks and relate them to functionality in our system.

4.1 System introduction

The application that will be made, will serve the purpose of illustrating vulnerable processes in a system, and indicate exactly what threats are faced by specific functionality. Important security issues will be highlighted inn all phases of the development process.

This project will make a web application that is an online ticket ordering system.

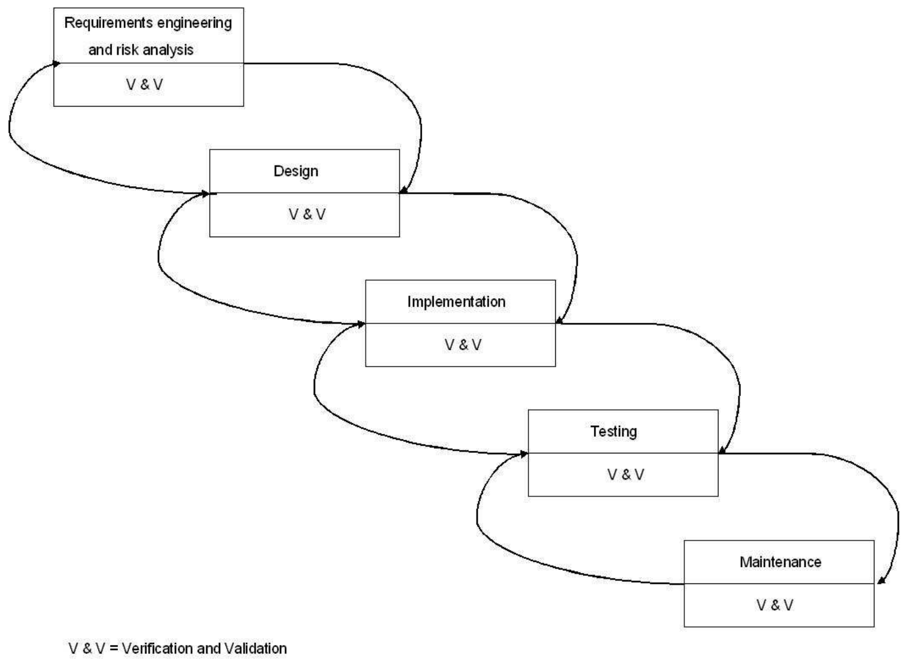

Requirements engineering and risk analysis
V & V
Design
V & V
Implementation
V & V
Testing
V & V
Maintenance
V & V
V & V = Verification and Validation

Figure 4.1: The waterfall model shows the systems development process.

There will be tickets to different kinds of arrangements, namely concerts, theaters and musicals. Customers will be able to order and pay by VISA online, and tickets will be sent by post. For customers to be able to order tickets they need to be registered in our customer database, and be logged in to the application. Each order is limited to a maximum of ten tickets. While logged in the customers may also edit their personal information and change their password. If you are a new user, it is possible to register and create a user account. Should a returning customer have forgotten his or her password, the system will provide a mechanism to reestablish the system identity of the user. The system has three different kinds of user roles, namely *regular users*, who are customers, *administrator users*, who have extended rights in the system, including editing events, and *super users*. Super users have complete editing rights in the system, including the possibility of creating and deleting administrator accounts.

4.2 Requirements

All requirements to the application have a unique ID, so that one may check each one explicitly if it is satisfied when the implementation is done. In addition they will be used as references from the misuse cases in the succeeding section. That is, they will be used to identify which requirements a misuse case models, and thereby link requirements to potential security threats. Requirements are organized in tables. The first column holds user actions and elaboration of requirements, while the second column is an illustration that we will be able to tick off requirements after implementing the functionality. For now, the second column is left open. As the focus will be set on security throughout the development process, requirements relevant to security have been marked with an asterisk.

The requirements are logically organized in tables identifying system functionality. The functionality available is dependent on two factors, if the user is logged in or not, and what role the user has in our system. The first table represents requirements to what functionality, independent of user role, that should be available before login. That is, what opportunities a users that browses the page has. The next three tables identifies functionality that needs to be available to regular users, administrator users and super users respectively. After user functionality is treated, database and system requirements will be presented. Network requirements are not presented, as this is not a relevant topic to this report.

The tables are slightly differently structured depending on their contents. The first table is in fact a composition of what could have been four tables containing various functionality before login. These are put together to form Table 4.1, "Before login", in which what would have been the headings to each table is presented in bold letters. Simple user actions are, as in all user functionality tables, presented in italic writing. Requirements to each of these user actions are attached to them, and ordered alphabetically. User requirements tables contains a bold header, and user actions as specified for the compound table with functionality before login. Database requirements and system requirements do not contain user actions, but rather explicit requirements the database and the system needs to fulfill. These requirements are presented in italic letters, and some requirements in the system requirements table have the same kind of elaboration as user actions in user functionality tables.

4.2.1 Before login

In Table 4.1, "Before login", four different user actions have been identified. Browse the page is what any user initially does, and requirement BP identifies what kind of information should be available. In addition to being able to see information, a user should be able to register a new user account from this page. This is specified in requirement RUA, which also consists of five sub actions a user needs to take to complete registration. RUA1a) identifies what kind of personal information the application will require to register. This information needs to be kept in a customer database. As security guideline 2.2.1, "Validate input on the server side" explains, nothing from the client should be trusted, and input needs to be validated on the server side. This is expressed in both requirement RUA1b), SLb) and FP1b). These requirements are marked with an asterisk indicating that they are security critical. Requirement RUA2a) states that a username should unique. This implies that the username needs to be checked against other usernames in the database. A user also needs to select a password, and the system will require that it is according to requirement RUA3a) and b). Before completion of the registration, requirement RUA4 and RUA5 requires the user to sign an electronic personal information protection statement and identify scrambled letters. The latter is a countermeasure to avoid Denial of Service (DoS) attacks and is marked by an asterisk. This is further treated in the misuse case part of this chapter. The login functionality has a set of requirements that have been extracted from the user actions. They are listed before user actions are presented, and contains general requirements concerning login that are all security critical. The login functionality will be using a time delay technique. This is treated in more detail in the misuse case part of this chapter. The administrators and super users should log in using a login page with a specified URL, that is hidden from regular users (SLe)). The password should not be transferred in clear text, and

Browse the page (BP)	**Ok?**
a) Site information, contact information and a list of upcoming events should be available.	
Register a user account (RUA)	
RUA1. User states personal information. a) First name, last name, address, postal code, post office, phone number and email should be provided by the user. *b) Input should be validated on the server side.	
RUA2. User selects a username. a) The username should be unique.	
RUA3. User selects a password. a) Password should contain eight characters. b) Password should contain small and big characters.	
RUA4. User should read and agree to a personal information protection statement.	
**RUA5. User should read and state scrambled letters to the application.*	
Standard login (SL)	
*a) A new session should be created. *b) Input should be validated on the server side. *c) The user should be taken to another, secure page. *d) Time delays should be used when a user provides wrong *e) Administrators and super users should log in from a hidden URL. credentials.	
SL1. User states username.	
SL2. User states password. *a) Password should never be transferred in clear text.	
Forgot password feature (FP)	
FP1. User clicks the "Forgot password?"-link. *a) Ask the user to supply username and postal code. *b) Input should be validated on the server side. *c) Send mail to the users registered email address with link to a page for resetting the password and a one-time password.	

Table 4.1: Requirement table - Before login.

this is expressed in SL2a) and is a security requirement. The forgot password feature is the last available functionality identified in this table. This procedure is further treated in the misuse case part of this chapter.

4.2.2 Regular users

After the user is logged in, different actions are possible depending on the type of user we are dealing with. Requirements are listed in three tables, Table 4.2 “Regular users”, Table 4.3 “Administrator users” and Table 4.4 “Super users”. First we will concern ourselves with regular users. These are the users that have no special rights in the system. All users will be granted these rights when registering in our database.

Regular users (RU)	**Ok?**
RU1. Edit personal information. a) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when logged in. b) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when processing an order.	
RU2. Change password. a) It should be possible to change the password when logged in.	
RU3. Order ticket(s). a) The user should be able to order tickets. b) The order should be limited by ten tickets. c) Event headline and number of tickets ordered should be confirmed by user. d) Address, postal code and post office should be confirmed by user. *e) Payment information should be entered. f) User repudiation should be avoided.	
RU4. Logout. a) The user session should be invalidated. b) A message confirming logout should be sent to the user. c) Pressing the “logout” button should be equivalent to using the browser to end the session. *d) The user session should be invalidated after a period of 15 minutes without user action.	

Table 4.2: Requirement table - Regular users.

Regular users functionality requirements are identified in Table 4.2, “Regular users”. The first requirement RU1 states that the user should be able to edit his or her personal information, either when logged in, or when processing an order. RU2 opens the opportunity for the user to change the password. RU3 is the main user action, that is, order ticket(s). This action consists of several sub requirements. The system will require that an order is limited by ten tickets (RU3b)), and that the user confirms the number of tickets ordered and which event the order concerns (RU3c)). The user should also confirm that his or her registered address is correct (RU3d)). RU3e) requires payment information to be entered to the system each time an order is processed. This is due to strict requirements as to storage of this kind of information and it is a security critical requirement. At last, it is important that repudiation is avoided (RU3f)). RU4 requires that logout is available. This functionality is equal for all user roles, and will be repeated in the administrator users table and the super users table. The logout functionality results in session invalidation (RU4a)) and a confirmation message to the user of the application (RU4b)). If a user choose to end his or her session by closing the browser, then it should be equivalent to pressing the logout button (RU4c)), only a confirmation message is not necessary. If a user session is inactive for a period of 15 minutes, logout should happen automatically (RU4d)), and a message to the user stating that he or she was logged out due to timeout should be provided. This is also a security critical requirement.

4.2.3 Administrator users

Administrator users (AU)	**Ok?**
AU1. Edit events. a) Should be able to add event(s). b) Should be able to change event properties. c) Should be able to remove event(s) by hiding them.	
AU2. Edit personal information. a) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when logged in.	
AU3. Change password. a) It should be possible to change the password when logged in.	
AU4. Logout. a) The user session should be invalidated. b) A message confirming logout should be sent to the user. c) Pressing the “logout” button should be equivalent to using the browser to end the session. *d) The user session should be invalidated after a period of 15 minutes without user action.	

Table 4.3: Requirement table - Administrator users.

These users will have extended system rights compared to regular users. The requirements are available in Table 4.3. An administrator user should be able to edit events (AU1)). This implies adding events (AU1a)), changing event properties like for instance dates (AU1b)) and removing events from the page available to regular users (AU1c)). The administrator user should not be able to delete events, as this will be a task left for super users. AU2 states that administrator users should be able to edit their personal information when logged in to the system. AU3 gives the administrator user the opportunity to change his or her password. AU4, logout, is similar to regular users logout, RU4.

4.2.4 Super users

Super users have full system rights. Requirements are enlisted in Table 4.4. Requirement SU1, on editing events, is similar to the same functionality for administrator users AU1, the only difference being that the super users are able to completely delete an event from the system (SU1c)). Requirements SU2 and SU3 are similar to corresponding administrator user functionality, and is thus not repeated. SU4 states that a super user should be able to create an administrator user account, while SU5 states that he or she should be able to edit an administrator account. In addition,

Super users (SU)	**Ok?**
SU1. Edit events. a) Should be able to add event(s). b) Should be able to change event properties. c) Should be able to delete event(s).	
SU2. Edit personal information. a) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when logged in.	
SU3. Change password. a) It should be possible to change the password when logged in.	
SU4. Should be able to register administrator user account.	
SU5. Should be able to edit administrator user account.	
SU6. Should be able to delete administrator user account.	
**SU7. Should be able to check the log.*	
SU8. Logout. a) The user session should be invalidated. b) A message confirming logout should be sent to the user. c) Pressing the "logout" button should be equivalent to using the browser to end the session. *d) The user session should be invalidated after a period of 15 minutes without user action.	

Table 4.4: Requirement table - Super users.

a super user should be able to delete an administrator account (SU6)). SU7 regards the log, which can be a tool to help detect misuse of the system. The super user should be able to check the log. This requirement is marked with an asterisk, since it is security critical. SU8, logout, is similar to RU4, and will not be treated.

4.2.5 The database

Database requirements (DBR)	**Ok?**
**DBR1. Passwords should be stored encrypted.*	
DBR2. The database should be available 90% of the time.	
DBR3. The database should be able to cope with simultaneous access.	

Table 4.5: Requirement table - Database requirements.

The database will need to be consistent at all times. It will have to work according to ACID[1] properties. Database requirements are expressed in Table 4.5, "Database requirements". The first requirement regards password storage. According to security guidelines 2.1.5. "Defense in depth", and 2.1.6, "Secure the weakest link", passwords should not be stored in clear text, but rather encrypted (DBR1)). This is a security critical requirement. Requirement DBR2) concerns availability. Keeping the database available at all times is neither realistic nor necessary. It will have to

[1]These four properties are considered to be key transaction processing features of database management systems. Without them, the integrity of the database cannot be guaranteed. The four properties are atomicity, consistency, isolation and durability [44].

be unavailable during for instance repairs, and this is not critical as this application is not a critical one. DBR2 states that the database should be available 90% of the time. DBR3) states that simultaneous access not should cause trouble in the system. Simultaneous access needs to be synchronized so that for instance database updates are under control.

Administrators should make sure that there is always a backup of the database.

4.2.6 System requirements

System requirements (SR)	**Ok?**
SR1. Availability. a) The system needs to be available 90% of the time.	
SR2. Scalability. a) The system should be able to handle 100 events simultaneously.	
**SR3. Encryption.* a) As of the login session, all succeeding communication should be encrypted.	
**SR4. Caching.* a) Sensitive data should not be cached.	
**SR5. Input validation.* a) Headers, cookies, query strings, form fields and hidden fields needs to be validated on the server side.	
SR6. Log. a) The system should keep track of actions and transactions with a log.	

Table 4.6: Requirement table - System requirements.

In addition to functional requirements, system requirements are presented in Table 4.6 “System requirements”. The same availability requirement as for the database goes for the system in total, it should be available 90% of the time (SR1a)). The system should also be able to handle 100 events simultaneously (SR2a)). Encryption should provide confidentiality of data by providing a way that the server and client can exchange encrypted messages that cannot be understood by anybody else. As we will be using the Secure Socket Layer (SSL), the login session will also be encrypted. As of the login session, all communication should, according to security guideline 5.3,“Encrypt communication lines” be encrypted (SR3a)). This is a security critical requirement marked with an asterisk . Caching is an important topic. Caching of sensitive data, for instance credit card information, might be dangerous as attackers can view for instance the Temporary Internet Files folder in Internet Explorer. Sensitive data should not be cached (SR4a)), according to 2.2.3, “Limit cache usage”. This requirement is also marked with an asterisk, and is relevant to security. As have been mentioned before, input validation is extremely important. All kinds of input should be validated. This is expressed in requirement SR5a), according to 2.1.1, “Validate input”, which is marked with an asterisk, since it is a security requirement. The system is required to keep a log (SR6)).

4.2.7 Error handling

Error handling requirements (EHR)	**Ok?**
EHR1. System error should be reported to super user.	
**EHR2. System error should present information page to regular user.*	
EHR3. User error should result in a constructed error page being return to the user.	

Table 4.7: Requirement table - Error handling requirements.

It is of great importance that errors and exceptions in an application are treated correct. This is also according to security guideline 2.1.2, “Fail securely”. The requirements are presented in Table 4.7, “Error handling requirements”. For security it is important that errors are classified and reported to a super user so that they can be fixed. This is expressed in requirement EHR1. The user should not be provided a too detailed error message as this might be something an attacker can take advantage of. This is expressed in requirement EHR2 and is marked with an asterisk to indicate the connection to security. EHR3 expresses that a users error should return an error message to the client side.

4.3 Misuse cases

To model our requirements, we will make use of misuse cases[2] to point out where security might be threatened. The fascinating thing about misuse cases is the way that they relate functionality and threats to each other in a visual and easily surveyable manner.

Briefly, misuse cases are use cases that model both functionality in a system and the system areas where security is exposed to threats. It also implies what effort one should make to reduce, or even better, avoid, any risks present.

Following are the modeled misuse cases. They are organized in the same fashion as the requirements presented earlier in this chapter, and the focus is on both functionality and security. The textual description that succeeds each of them, focuses on how to prevent the security risks the system faces. The various threats are identified in chapter 3, “Vulnerabilities, threats and countermeasures”. This chapter also presents countermeasures. Some details about threats and attacks are repeated here, but for an in depth description of attacks and countermeasures we once again refer the readers to chapter 3.

Some misuse cases are more complex than others, and those who consist of compound actions of interest are decomposed to get a closer look at the threat we are facing. One misuse case is particularly complex, but it is left that way because it provides a survey of functionality available before login, and also illustrates that there are a number of threats facing our application.

4.3.1 Before login

The first misuse case, shown in Figure 4.2, is an overview of functionality that faces the user when he or she first browses the page. The first misuse case focuses solely on *Denial of Service* (DoS) attacks. The second misuse case shows pretty much the

[2]Appendix B gives an introduction to both use cases and misuse cases.

same functionality, but related to different threats. (Browse the page is not repeated in the second misuse case, as it is only threatened by DoS attacks.) Both of these first misuse cases are surveys of available functionality for all users browsing the page, independent of user role, or even if the user is registered in the system. The misuse cases ties together a lot of functionality, and will be decomposed to examine the login procedure and the register user account functionality in more details in separate misuse cases. The first misuse cases correspond to requirements in Table 4.1, "Before login".

Misuse case - Before login, DoS

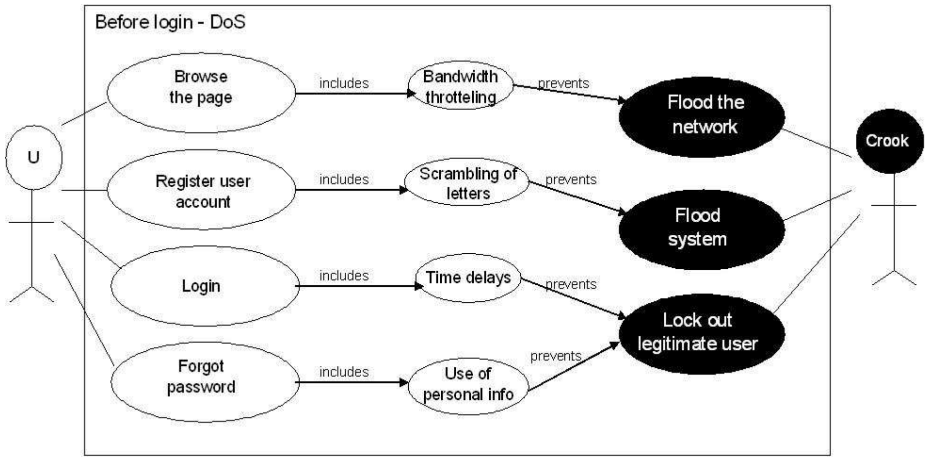


Figure 4.2: Before login - DoS.

First of all, every user that accesses the web site will be able to browse information presented on the page. The information available is for instance event-, contact- and site information (according to requirement BP, browse the page.). Should too many users browse the page at once, or an attacker flood the network with useless traffic, it is a possibility that the network could be compromised due to bandwidth shortage. This is illustrated in the top of the figure, and countermeasures to *flood the network* attacks are identified as for instance bandwidth throttling techniques in chapter 3, "Vulnerabilities, threats and countermeasures".

The page that welcomes the users is also the one that provides the opportunity to register a new user account (requirement RUA, register user account). This functionality might be offer to a *flood system* attack. The difference from a flood the network attack is presented in chapter 3. Both are defined as DoS attacks, and flood the system is a reference to an attack that compromises the system due to an overload of system capacity, as opposed to flood the network, which is an attack on network capacity. The main intent of using scrambling of letters as a countermeasure of a flood system attack is to avoid automated attacks. The idea is that the user have to enter a set of numbers and letters from a small window that is not readable by a computer script; only by the human eye. Thus an automated attack is not feasible.

The login procedure and forgot password feature might be victim to another type of DoS attack, *lock out legitimate user*. This attack locks out a legitimate user by providing invalid credentials until the system locks the account. To prevent it our login functionality will use a time delay (required in SLd), standard login). The intent of this time delay is that, instead of shutting down a user account after a user has provided wrong username and password combination a number of times, the system will slow down. That way, an attacker is not in a position to shut down user accounts by repeatedly providing fake input.

To prevent a *lock out legitimate user* attack on the forgot password feature, one may use personal information (in accordance with requirement FPa), forgot Password feature). This variant of DoS, as presented in chapter 3 sends a forgot password requests on a specific account. This may result in the user being locked out of the system because the password has been reset. It is circumvented by asking the user to provide some personal details in order to go through with the forgot password procedure. Our system will, as mentioned during requirements specification, ask for post office and username.

When we move on to the second misuse case concerning functionality available before login, Figure 4.3, we see that several functionalities are potential victims of more threats.

Before login - miscellaneous

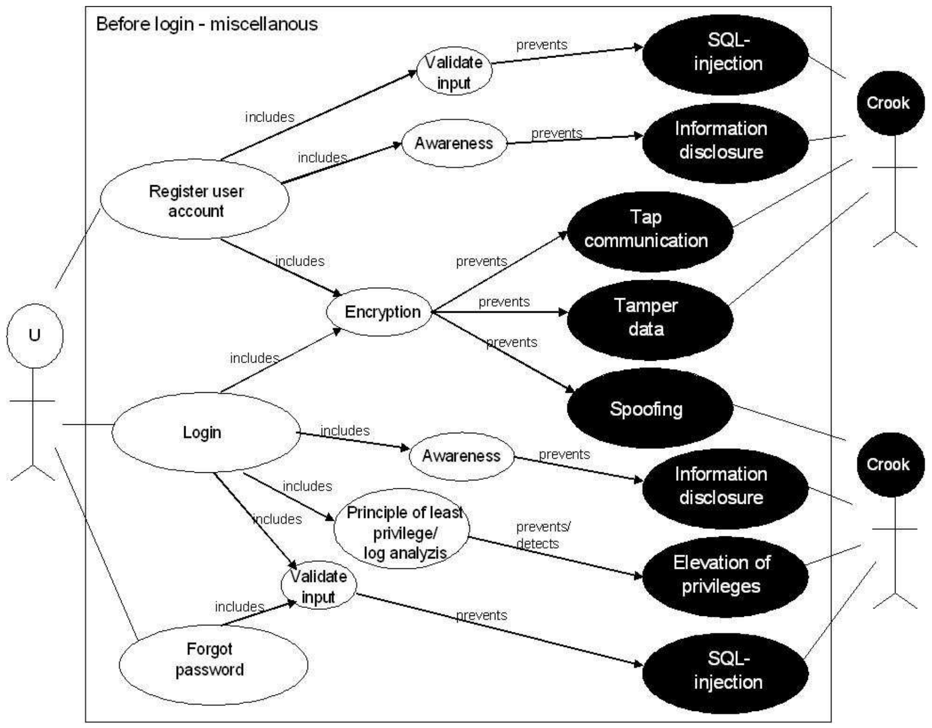


Figure 4.3: Before login- miscellaneous.

The second misuse case is fairly complex, and it might seem overwhelming, but it certainly illustrates that even a rather limited application is exposed to a lot of different threats. Some of these threats are collective terms, and these are the ones that will be decomposed to show which part of the threat is relevant to our application. Register user account, login functionality and forgot password might be victim of a *SQL-injection* attack (presented in chapter 3, "Vulnerabilities, threats and countermeasures") unless input is validated. Input is required to be validated on the server side according to requirements RUAb), register user account, SLb), standard login and FPb), forgot password feature. These functionalities are also possible targets for launching of *information disclosure* attacks. This kind of attacks are avoided by various measures identified in chapter 3 for example to avoid storing secrets, that is passwords, in plain text (requirement DBR1, database requirements).

Finally, register user account is connected to both *tampering data* and *tap communication* attacks. These attacks are compound, and the relevant parts of this threat are treated in the decomposition misuse case, presented later in this section. The same goes for login, which is, in addition to tampering data and tap communication attack, also threatened by *spoofing*. A closer look at the login procedure is provided in the next misuse case.

The threat *elevation of privilege*, in this setting, concerns a user being able to log on as a more privileged user, the countermeasures being to follow the principle of least privilege and review the log for abnormal activity (the system is required to keep track of actions and transactions by use of a log in requirement SR5, system requirements.).

4.3.2 Login

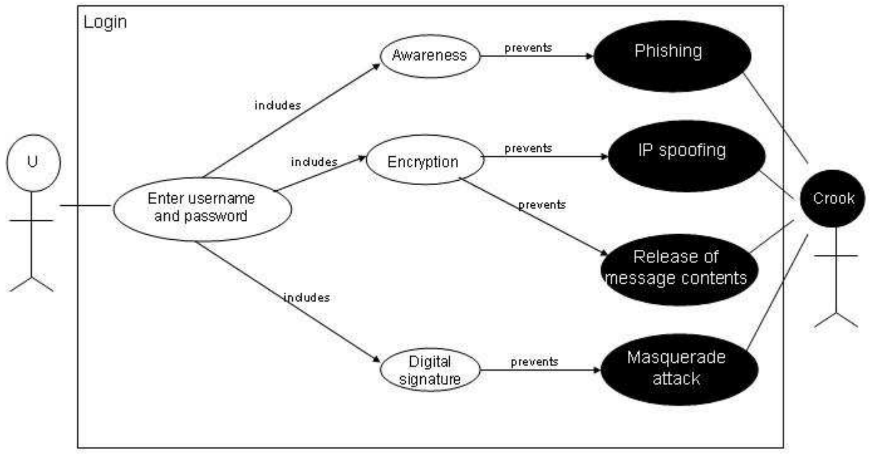


Figure 4.4: Misuse case, Log in.

The login misuse case, in Figure 4.4, shows a more detailed view of the relation between login functionality and possible threats. It shows that *phishing* and *IP Spoofing* are possible forms of spoofing attacks. Phishing is the kind of attack where an attacker sends e-mails asking the user to provide personal information because of some problem in the application. The only way to prevent it is awareness among users. IP spoofing is prevented by encryption (requirement SR3, system requirements). Encryption is also the means to prevent from *release of message contents*, which is classified as a tap communication attack. This is the tap communication attack that is relevant for our application, and later on tap communication will not be decomposed. Release of message contents is implied. The other tap communication attack, *traffic analysis*, will not be discussed. The last modeled threat to the login procedure is a tampering data attack, namely *masquerade* attack. It is defined as a tampering attack as it creates a false stream when a user logging on as a different user than himself. To conquer a masquerade attack, chapter 3, "Vulnerabilities, threats and countermeasures" identifies several encryption techniques.

4.3.3 Register user account

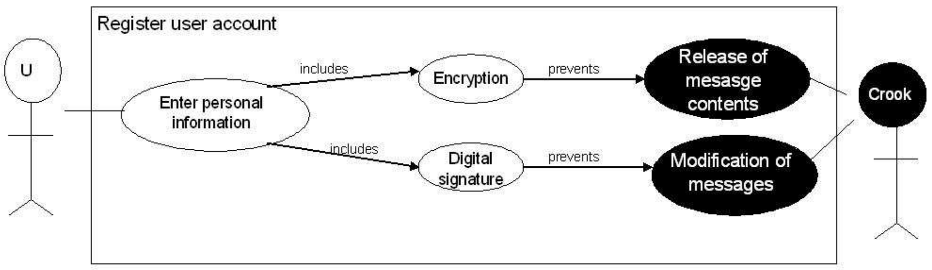


Figure 4.5: Register user account.

This misuse case, Figure 4.5, shows a decomposition of the register user account procedure. *Release of message contents* is, as in the login procedure, prevented by encryption. This is the kind of tap communication attack that is interesting in this matter. The *modification of message* threat is a subgroup of tampering data attacks, and it is prevented by the use of digital signatures. Digital signatures was presented in chapter 1, "Web applications and security issues".

Regular users

The misuse case in Figure 4.6 shows what a regular user is capable of doing when logged into the system. The actions he or she is capable of, are in accordance with Table 4.2, "Regular users". Of course, a user have to be able to order ticket(s) (requirement RU3). This is the most complicated task and the tamper data threat will be decomposed in the succeeding misuse case. Order ticket(s) is also connected to a tap communication threat, but this threat will not be decomposed, as mentioned before, because whenever this attack is described, *release of message content* is implied. *Repudiation* occurs when a user denies performing actions or transactions. This is critical, especially during ordering of tickets, and must thus be avoided (requirement RU3f)). Countermeasures are identified in chapter 3, "Vulnerabilities, threats and

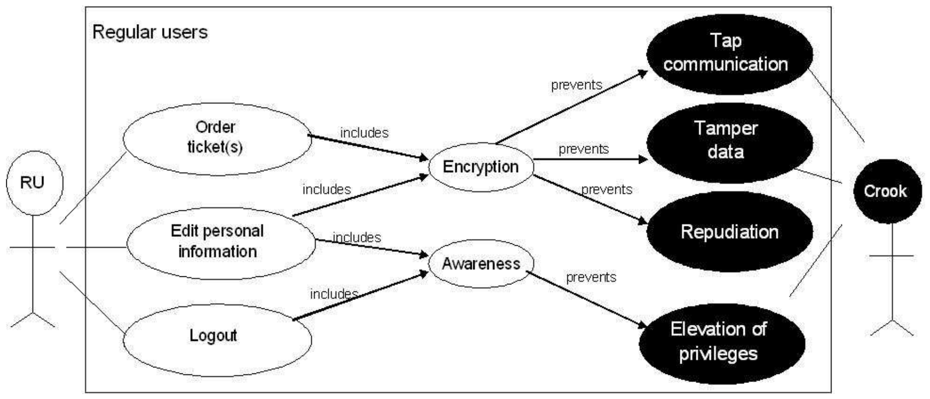


Figure 4.6: Misuse case, Regular Users.

countermeasures". In addition to order ticket(s), a regular user might edit his or her personal information (requirement RU1). This functionality is similar to register user account, and is therefore not decomposed. Instead, we refer to the previous misuse case shown in Figure 4.5. The last thing that a user is capable of doing, is logout (requirement RU4). This is threatened by *elevation of privilege* as a privileged user might leave his or her workstation without logging out, and thus open an opportunity for a less privileged user to use his or her account. In chapter 3 this is defined as a elevation of privilege on account of exploiting physical premises. The misuse case

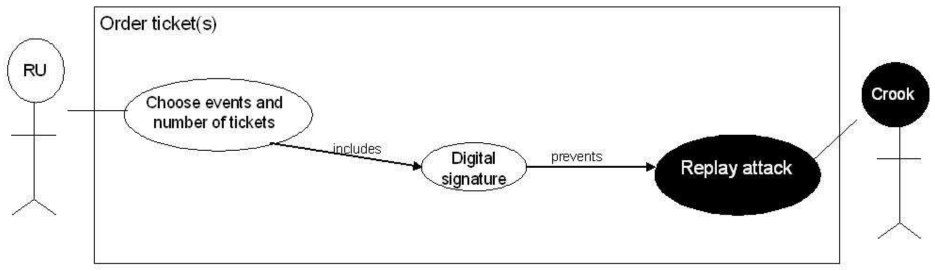


Figure 4.7: Misuse case, Order ticket(s).

in Figure 4.7 is a more detailed view of threats connected to ordering ticket(s). A digital signature will be used to avoid *replay* attack.

4.3.4 Administrator users

The next misuse case, shown in Figure 4.8 illustrates administrator users possible actions related to possible threats. The functionality is in accordance with Table 4.3, "Administrator users". Edit personal information (requirement AU2) and logout (requirement AU4) is equivalent to the same functionality in the regular users misuse case. Edit events (requirement AU1) will be decomposed to have a closer look at *tamper data* threats. In this misuse case, in Figure 4.9, edit event(s) is decomposed. Possible actions are add event(s), edit event(s) and remove event(s). All are possible victims of the same threats, namely the tampering data threats *modification of*

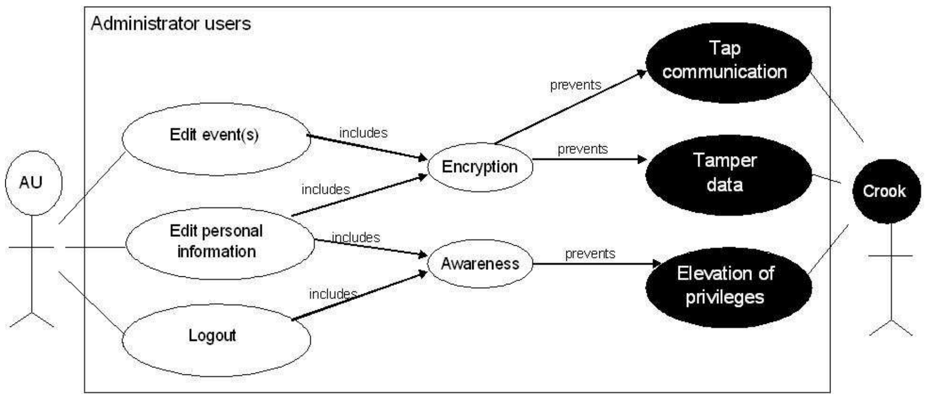


Figure 4.8: Misuse case, Administrator Users.

messages and *masquerade*. Masquerade would imply that an attacker is capable of logging on as another user, and then add, edit or remove events from the system. Remove is not the same as deleting events, the administrator user is restricted to only remove events from the system, in the sense of making them unavailable to users. This was also mentioned in the requirements part of this chapter. There is only one countermeasure identified, and this is encryption, which is a collective term. This is treated in more detail in chapter 3, “Vulnerabilities, threats and countermeasures”.

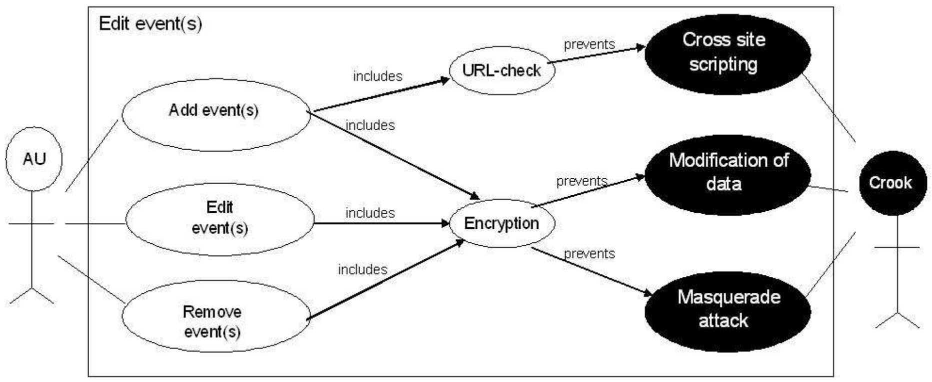


Figure 4.9: Misuse case, Edit events.

4.3.5 Super users

This misuse case, 4.10, is in accordance with Table 4.4, “Super users”, and edit personal information (requirement SU2) and logout (requirement SU8) is in accordance with previous misuse cases and it will not be treated further here. Edit event(s) (requirement SU1) is also pretty much the same, only a decomposition would show that a super user is in fact capable of actually deleting an event from the system, not only hide it as a administrator user is capable of. The super user is capable of checking the log (requirement SU7). This is threatened by a *tap communication* *attack*. As usual, this attack implies *release of message content*, not traffic analysis.

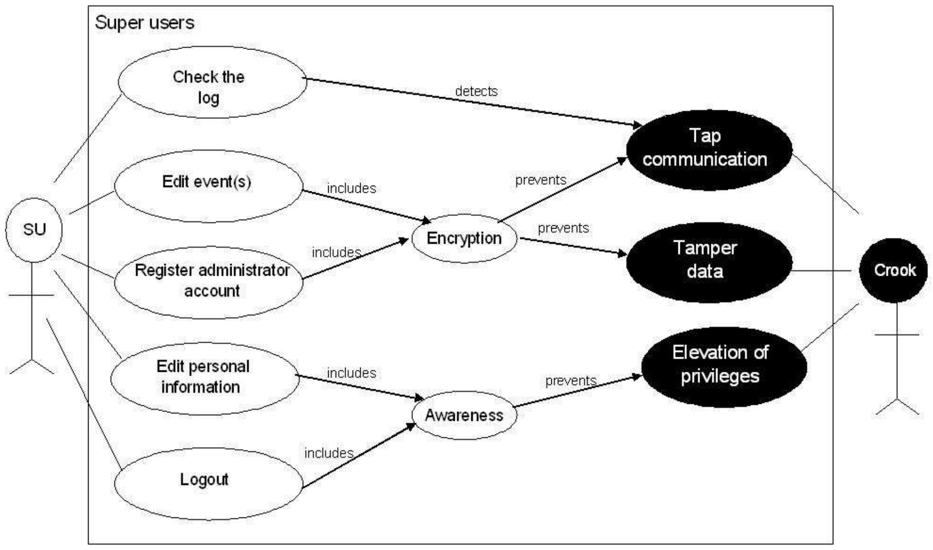


Figure 4.10: Misuse case, Super users.

4.4 Risk analysis

Figure 4.11 is an illustration of the relationship between threats and risk. We will be treating all the threats we have encountered during misuse case modeling in this chapter, and range them to find which ones are most urgent in our application. We have chosen to do this at this stage of the process to open the opportunity to design with respect to relevant threats.

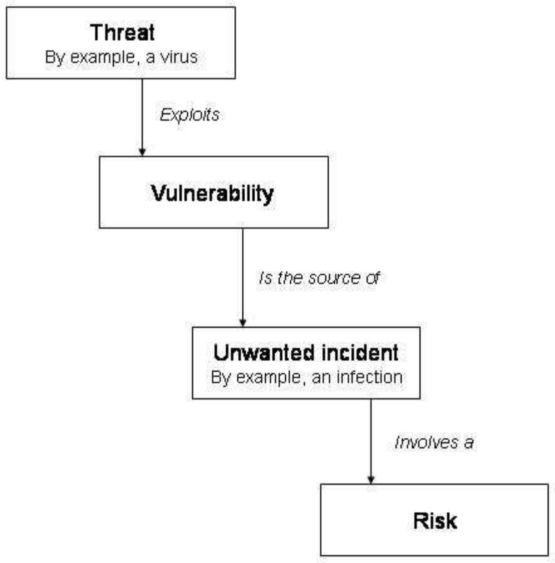


Figure 4.11: The relation between threat and risk.

At the end of this modeling process we will be left with a list of the relevant threats, ordered by the risk that they pose to our application.

A simple way to rank the threats would be to use this formula: *Risk = Probability * Consequence*. To achieve a more differentiated ranking we have chosen to apply a more fine grained method that takes more elements into account. We will be using Microsofts DREAD model [35], that contains five aspects to consider. Each threat identified relevant to the application will be evaluated and given a score on each of these five. DREAD is an abbreviation, and the content of the elements we will be evaluating is expressed below. Each element have been related to the more traditional aspects of risk analysis, that is, either the consequences of a risk, or the probability of it occurring.

Damage potential: How great is the damage if the vulnerability is exploited? This element concerns the *consequence* of an attack. A threat that would result in the attacker subverting the system, that is for instance get full trust authorization, run as administrator or be able to upload content is considered a high ranking threat. A threat that could result in leakage of trivial information is a low ranked threat.

Reproducibility: How easy is it to reproduce the attack? A high ranked threat would be possible to reproduce within a short time interval, while a low ranked threat is difficult to reproduce even with knowledge of the security hole. This element considers the *probability* of an attack.

Exploitability: How easy is it to launch an attack? If a novice programmer would be able to make the attack quickly it needs to be considered a high ranked threat, whereas if the attack requires a highly skilled person with an in-depth knowledge every time to be exploited, it would be considered a low ranged threat. This element concerns the *probability* of an attack.

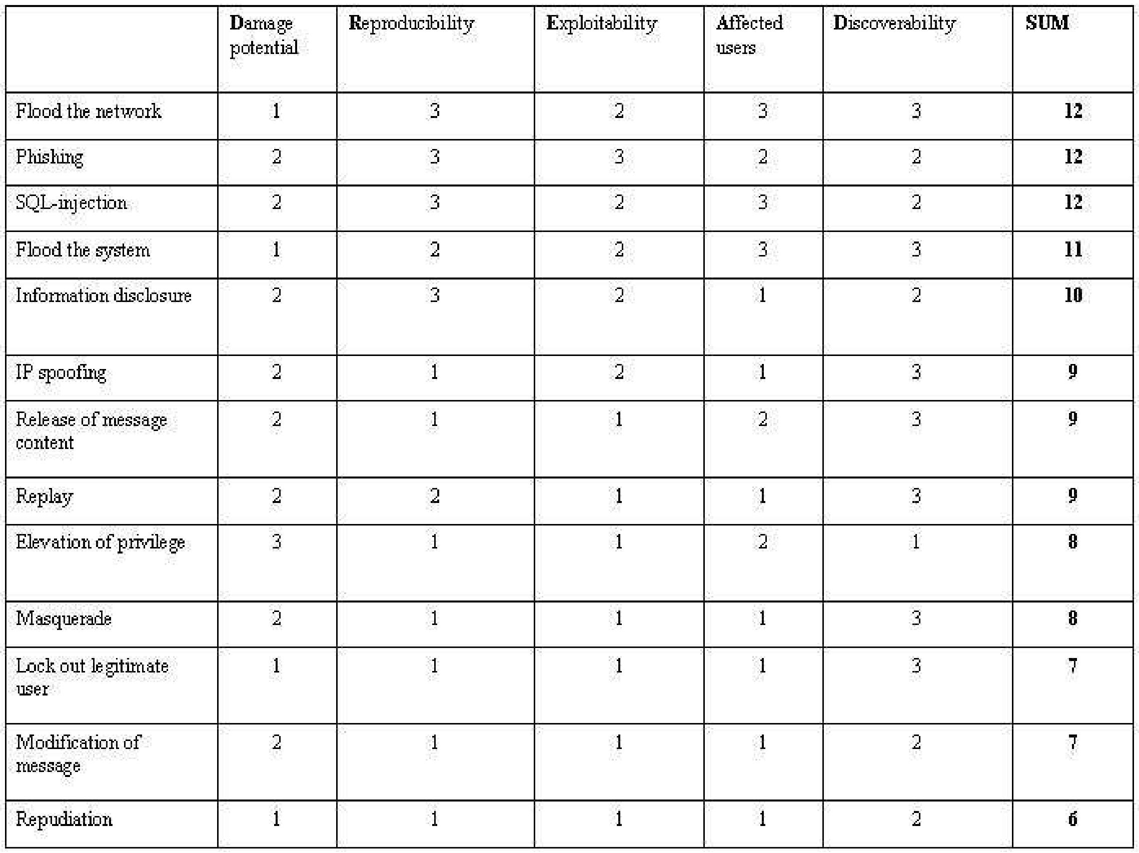
Affected users: As a rough percentage, how many users are affected? If all users are affected it is a high ranked threat. If only a very small percentage of users is affected it would be ranked as low. This element concerns the *consequence* of an attack.

Discoverability: How easy is it to find the vulnerability? If it is highly available, that is, published information explains the attack and the vulnerability is found in a commonly used feature it would be considered a high ranked threat, whereas if the threat exploits a vulnerability that is not publicly known, or it is unlikely that users will work out the damage potential, it is considered a low ranked threat. This element concerns the *probability* of an attack.

Each of these elements have been rated with a high (3)-, medium (2) or low (1) score when evaluating each of the threats identified in this chapter. This indicates that each threat will end up with a total score in the range between 5 and 15. An overall rating will classify a total score 12-15 as a *high risk*, a total of 8-11 as a *medium risk* and a total of 5-7 as *low risk*. Our threats, ranked in decreasing order on account of their score on the various elements presented is showed in Figure 4.12. Each one has been given a unique number, for simpler reference.

High risk threats

As one can see, there are three attacks that are all rated to a score 12 out of 15 possible. They are therefore considered as high risk threats. *R1, Flood the system* was evaluated to have a low damage potential, high reproducibility and it is high on affected users. It is also easily available, but requires a skilled programmer. As it is a DoS attack it is hard to completely avoid it, but some countermeasures are identified in chapter 3, "Vulnerabilities, threats and countermeasures". *R2, Phishing* also has a high ranking, it is given a medium score on damage potential, as users of the application might be fooled into disclose of their personal information, it is an attack that is easily reproduced, and a novice programmer could carry it out, affected users is a medium as probably only a group of users will fall short to this attack. Discoverability is rated a medium. *R3, SQL-injection* has a medium score on damage potential as a successful attack will probably result in leakage of sensitive information. This attack might be reproduced easily, but it requires a skilled programmer to accomplish. Affected users is rated high, as if the database is compromised all users personal information is compromised as well. Discoverability is a medium.



	Damage potential	**R**eproducibility	**E**xploitability	**A**ffected users	**D**iscoverability	**SUM**
Flood the network	1	3	2	3	3	**12**
Phishing	2	3	3	2	2	**12**
SQL-injection	2	3	2	3	2	**12**
Flood the system	1	2	2	3	3	**11**
Information disclosure	2	3	2	1	2	**10**
IP spoofing	2	1	2	1	3	**9**
Release of message content	2	1	1	2	3	**9**
Replay	2	2	1	1	3	**9**
Elevation of privilege	3	1	1	2	1	**8**
Masquerade	2	1	1	1	3	**8**
Lock out legitimate user	1	1	1	1	3	**7**
Modification of message	2	1	1	1	2	**7**
Repudiation	1	1	1	1	2	**6**

Figure 4.12: DREAD diagram.

Medium risk threats

R4, Flood the system has a total score of 11 points, which inflicts an overall rating as a medium risk. Damage potential is low, reproducibility and exploitability is medium, affected users is high. Discoverability is high. *R5, Information disclosure* has a total score of 10 points, which ranges the attack as a medium risk to the application. It is set to a medium on damage potential as it could possibly reveal sensitive system information. The attack may easily be reproduced, but requires a skilled programmer to launch. That is, high on reproducibility, medium exploitability. Affected users is set to low, as it is system information that is compromised. Discoverablilty is medium. *R6, IP spoofing* is in a group of three threats that has all been given a total of nine points, and they are therefore considered medium risks. IP spoofing has a medium damage potential, it is hard to reproduce (low score) and it requires a skilled programmer to conduct (medium score on exploitability). It will probably affect a small percentage of users (low score on affected users), but is is high on discoverability. *R7, Release of message content* also has an overall score of nine points. It has a medium damage potential, but it is hard to reproduce and requires a skilled programmer. Affected users is set to medium, and discoverability is set to high. *R8, Replay* has an overall score of nine points. It is medium ranged on damage potential and reproducibility, it is hard to accomplish and is thus ranked as a high on exploitability. Affected users is also low, but discoverability might be high. *R9, Elevation of privilege* has a total score of eight points. For this threat, the damage potential is great, but it is rather hard to conduct (Low score on reproducibility and exploitability). Affected users is a medium, and discoverability is low. *R10, Masquerade* has a medium damage potential, but it is low on reproducibility, exploitability and affected users as it is an attack that is hard to conduct and it does not affect the majority of users. Discoverability is high. *R11, Lock out legitimate user* has a total score of seven points, which leaves the threat ranged as a low risk. It has a low score on all elements, except from discoverability. *R12, Modification of message* also has an overall score of seven points. It is medium on damage potential and discoverability, on other elements it is ranked low. The last threat, *R13, Repudiation* has a overall score of six points. It is an attack that is complicated to conduct, and it will probably not affect a lot of users, but it is medium on discoverability.

All in all, although all these threats are indeed relevant to the design of our application, we will be giving the most attention to the threats that have been ranked most urgent. That is, threats that have a total score that ranges them as high risks will be taken more into account in the design phase than less urgent threats.

4.5 Summary

This chapter concludes the requirements engineering phase of the development process. Various requirements, that is, functional requirements, system requirements and database requirements have been identified, and misuse cases have been employed to relate system functionality to potential threats. At the end of this chapter a risk analysis has been conducted to range the threats relevant, so that when moving on to the design phase, one may take into consideration the most urgent ones.

Chapter 5

Design

As our requirements were identified in the previous chapter, and a risk analysis showed which threats are most likely to inflict our application, we will now design with respect to this. Vulnerable parts of the application will be highlighted. As we have pointed out earlier, in chapter 2, "Security guidelines", it is important to keep ones mind focused on security at all times in the development process. As we are entering the design phase, we will identify specific processes in the application that will be critical.

This chapter will contain a description of our chosen architecture, a brief introduction to the technologies that is to be used in the implementation, and a design of the application.

5.1 Architecture

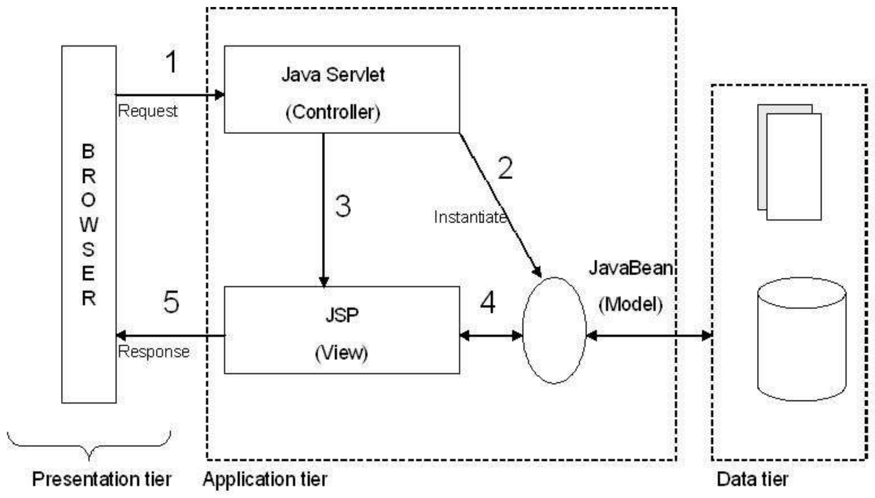


Figure 5.1: The JSP Model 2 Architecture.

Figure 5.1 shows a high level view of the architecture we have chosen for the application. It is called a Model-View-Controller (MVC) architecture, and its intent is to separate what the user sees from the logic of the application. It defines a separation of concerns in an application where the *model* defines the internal data structures of the system, the *view* defines how the model is rendered to the user, and the *controller* performs the actual actions in the program that affect the model [17]. In our design, the view is the graphical user interface (GUI) that the user interacts with. The controller is where the business logic is placed, and the database will be referred to as the model. This architecture is related to our security guidelines in chapter 2, as it realize the guideline 2.1.8, "Compartmentalization". Breaking the system into units, and isolating code that has security privileges on the server, will minimize the amount of damage if someone attacks a part of the application. This architecture also refers to the guideline 2.1.3, "Keep it simple", as the controller will serve as a choke point to handle requests.

From the figure, one can see that the view is called *JSP*, the controller *Java Servlet*, and the model *JavaBean*. Before presenting our design in more detail, we would like to introduce the technologies that will be used later on in the implementation phase.

5.2 Technologies

One of the goals in this report is to design a web application on the Java platform. When using the MVC architecture, we need to interweave several techniques, for instance an application server and a database. There are several different technologies to choose from, but we have chosen those that are widespread and free of charge. This is according to the security guideline 2.1.4, "Use and reuse trusted components". Doing this, it is important to emphasize that components must be developed by trusted third parties, tested out by multiple users, and have no documentation indicating weaknesses. In addition, choosing popular technologies, makes it easier to find answers to problems along the way.

5.2.1 Hyper Text Markup Language (HTML)

HTML is a *static* authoring language used to create documents on the web. It defines the structure and layout of a web document by using a variety of tags and attributes. HTML looks a lot like old-fashioned typesetting code, where you surround a block of text with code that indicate how it should appear. Additionally, in HTML you can specify that a block of text, or a word, is linked to another file on the Internet [44].

When creating a web application on the Java platform, we use the JavaServer Pages (JSP) technology. JSP is Java code embedded in a Hyper Text Markup Language (HTML) page using special tags.

5.2.2 The Java Platform

The Java platform is a software-only platform that runs on top of other hardware-based platforms. Because hardware-based platforms vary in their storage, memory, network connection, and computing power capabilities, specialized Java platforms are available to address applications development for those different environments [11].

Using the Java platform makes the software developed less vulnerable to the *buffer overflow* attack described in chapter 3, "Vulnerabilities, threats and countermeasures". Scripting languages such as Java are less exposed to this threat, as it use dynamic memory management to allocate space for variables. This does not mean that input lengths should be ignored when creating a web application on the Java platform. It is still possible that these variables could be passed to other programs that are susceptible to buffer overflows [12].

5.2.3 Java programming language

The Java programming language lets you write programs that run in the browser, from the desktop, on a server, or on a consumer device. Java programs are run on, and interpreted by, another program called the Java Virtual Machine (JVM). Rather than running directly on the native operating system, the program is interpreted by the JVM for the native operating system. This means that any computer system with the JVM installed can run a Java program regardless of the computer system on which the application was originally developed [11].

5.2.4 JavaServer Pages (JSP)

The JavaServer Pages (JSP) technology is an open, freely available specification developed by Sun Microsystems. JSP pages are written on the Java platform, and is therefore platform independent.

The fact that Java is platform independent, means that JSP pages does not tie you to any specific web server or operating system [14], which is a great advantage when it comes to security. All security functionality implemented in JSP-files, will function in any web browser. If you would use for instance JavaScript[1], there are several web browsers that won't understand it. It is only supported by recent browsers from Netscape and Microsoft Internet Explorer (MSIE), though MSIE supports only a subset. If security functionality is to be implemented using JavaScript, there is no guarantee that the security efforts will work properly.

A JSP page contains both regular HTML, and also special JSP elements that allows the server to insert dynamic content into a web page. JSP elements can be used for a wide variety of purposes, such as retrieving information from a database or registering user preferences. JSP uses tags to allow for instance embedded code in an HTML page, session tracking, and database connection [2].

When JSP pages are requested by the user, the server executes the JSP elements, merges the result with the static parts of the page, and sends the dynamically composed page back to the browser [23]. What happens is that the pages are compiled into *Java Servlets*[2] and loaded into memory the first time they are called, and executed for all subsequent calls.

There are several advantages about JSP. Speed and scalability are two of them [14]. In addition, JSP has its own embedded mechanism for handling runtime exceptions. You can provide your own exception handling using JSP, but it is hard to

[1]JavaScript is a scripting language developed by Netscape to enable web authors to design interactive sites. Although it shares many of the features and structures of the full Java language, it was developed independently. JavaScript can interact with HTML source code, enabling web authors to spice up their sites with dynamic content [44].

[2]Servlets are programs written in Java that run on a web server and can produce dynamic pages [44].

anticipate all situations. By making use of the embedded “page” directives “error” attribute, it is possible to forward an uncaught exception to an error handling JSP page for processing [27]. This embedded error handling functionality, makes JSP supported by the security guideline 2.1.2, “Fail securely”.

5.2.5 JavaBeans

As a component of the JSP architecture, one may use JavaBeans (often referred to as beans), technology. A JavaBean is a Java servlet that follows certain coding conventions so that it can be used as a component in a larger application [2]. A bean is often used in JSP as the container for the dynamic content to be displayed on a web page. A bean is always created by a server process and given to the JSP page, that is, it handles business logic as a part of the controller.

A JavaBean can act as a choke point where all critical security operations are lead through it. This is according to security guideline 2.1.3, “Keep it simple”.

5.2.6 Tomcat

An application server is a server program in a computer within a distributed network, that executes applications when requested by other clients [44]. It also stores business logics and the business models classes of applications.

We have installed the Tomcat-server from the Apache Jakarta Project in the application tier. This is an application server developed by a trusted third party, and tested out by multiple users. This is according to the security guideline 2.1.4, “Use and reuse trusted components”.

5.2.7 eXtensible Markup Language (XML)

XML (eXtensible Markup Language) is a standard for creating markup languages which describe the structure of data. It is not a fixed set of elements like HTML, but rather it is a metalanguage, or a language for describing languages [47].

As for security, there exist a project on XML security by the Apache Jakarta project, that presents solutions for how to handle security issues like encryption and digital signatures for XML in interoperability with Java [47].

5.2.8 MySQL

In the a three-tired application, as presented in chapter 1, “Web applications and security issues”, we need a database in the data tier. MySQL is a multithreaded, multi-user, SQL (Structured Query Language) relational database server. MySQL is most commonly used for web applications and for embedded applications, and has become a popular alternative to proprietary database systems because of its speed and reliability [44].

As of security, MySQL is ACID[3] compliant, it can establish a secure connection with a remote server using OpenSSL[4], which means that it is possible to establish a secure communication line between the database and the applications server [15]. It is also possible to encrypt data values in MySQL using built-in encryption functions [18].

[3]For definition of ACID, see “Glossary” in Appendix A

[4]For definition of OpenSSL, see “Glossary” in Appendix A.

Java Database Connectivity (JDBC)

When making a web application like ours, we need a tool that will handle the communication between the JSP pages and the database.

The Java Database Connectivity (JDBC) is a Java API[5] that enables Java programs to execute SQL statements [44]. This allows Java programs to interact with, for instance, the chosen MySQL database.

Some of the technologies introduced in this section, will be referred to later on in this chapter. Others will we not be getting into before the implementation in the following chapter. But now, we will move on to presenting our design.

The design is divided into parts in accordance with our architecture. This means that we will be presenting the design in three parts, the model, the view and the controller. Each of the three parts will have contain a subsection describing solutions to important security issues. We will be drawing sequence diagrams to show the data flow between the three application tiers, namely the presentation tier (web browser), application tier (view and controller) and the data tier (model).

The application we are developing will be designed with regard to security. The previous chapter, "Requirement engineering", provided a basis from which the work in this chapter will proceed.

Some of the requirements, especially the security critical ones, marked with an asterisk in the requirement tables, will be referred during the design. The misuse cases showed where the system is vulnerable to attacks. Countermeasures identified, will be refined according to our choices of technologies. Throughout the design, we will keep on stressing where security is threatened, and design the system in such way, that the threats identified will be prevented. Those threats that has the highest rank in our risk analysis in Figure 4.12, is given high priority.

5.3 The Model

We start off introducing the design of our model, namely our MySQL database, which defines the internal data structures of the system.

Figure 5.2 shows the tables in the applications database, and the relations between them. The database has six tables, namely "Person", "Order", "Event", "Users", "User_roles", and "Roles".

The "Person" table contains personal information on all users of the system, for instance full name, username (userID) and password. To fulfill the security guideline 2.1.5, "Defense in depth", and according to requirement DBR1, the password must be stored encrypted in the database.

The "Order" table contains information about every order made by users in the system. It holds information on who order the tickets, to what event, how many tickets was ordered, when did the order take place and did the payment go through.

The "Event" table contains information on events, like type of events, descriptions, where it is, and how many tickets there are left. The "TypeOfEvent" element should have the value 1, 2, or 3, according to which type of event it is. 1 is concert, 2 is theater and 3 is musicals. In addition, there is a row with a binary value called "Visible". In this table the value 1 states that an event should be visible to users,

[5]Short for Application Program Interface. A set of routines provided in libraries that extends a language's functionality.

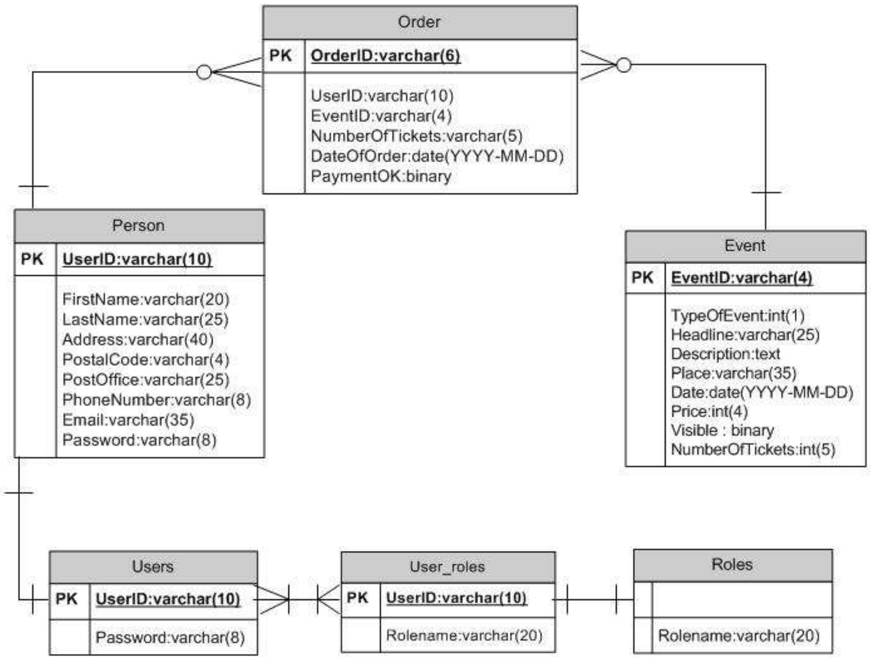


Figure 5.2: The Model is the database.

and 0 that it should not. This row is necessary to satisfy requirement AU1c), that administrators and super users should be able to hide events.

The "Users", "'User_roles", and "Roles", are tables that should be used when *authorizing* users to the system. If the users role is set to *preLogin*, the user have no privileges but to browse the page. If the role is set to *user*, the user have regular user privileges,. If the user role is set to *administrator*, the user gets privileges as a system administrator. If the user role is *superuser*, the user gets super user privileges in the system.

Security

The database should have a secure connection to the application server to avoid *information disclosure*, described in chapter 3, "Vulnerabilities, threats and countermeasures". If the application server and the database both lies within the same closed, secure environment, the communication between the two tiers will not be a problem. But if the database lies on an other server, a potential attacker may monitor data passed in plain text, between the database and the application server, over the network. Then the security guideline , "Encrypt communication lines", should be followed, and all communication flowing between the server and the database should be encrypted.

The risk of *information disclosure* (R5) has a high value (10) on the ranking of risk in the table shown in Figure 4.12, in chapter 4, "Requirements engineering". Therefore, this risk should be avoided.

In chapter 3, it was suggested to use encryption over the communication channel to prevent information disclosure. Now, knowing which technologies to use, we can refine this solution. If the application server and database is in the same, secure environment, the problem is already solved. But if they need to communicate over a network, there should either be a Secure Socket Layer (SSL) connection, using OpenSSL[6], or a secure connection using the IP Security Protocol (IPSec)[7] should be established.

Encryption of sensitive data stored in the database, for instance the password, is important, and is a part of the security guideline 2.1.5, "Defence in depth". In the "Person" table, we should therefore encrypt the password, in accordance with requirement DBR1..

5.4 The View

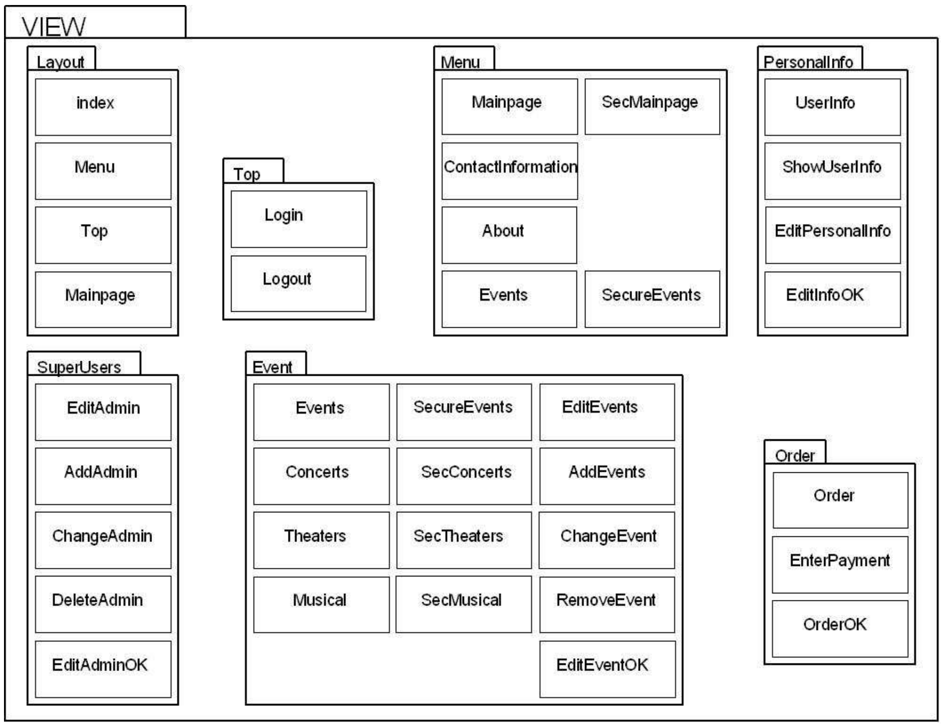


Figure 5.3: Files in the view.

The view defines how the model is rendered to the user, and is the graphical user interface (GUI). The web pages shown to the user, can be looked upon as classes. In our design, these classes will be JSP files, that is, static HTML files containing dynamic content using JSP. Figure 5.3 shows all the JSP class files in the view, bundled into small packages, which then again is put in one big package. Together, these files presents the applications content to the users through the web browser

[6]For definition of OpenSSL, see "Glossary" in Appendix A.
[7]For definition of IPSec, see "Glossary" in Appendix A.

(presentation tier). During the design of the view, we refer to the application as the (web) *site*.

This section will explain what functionality the different classes in the view have. How the JSP classes in the view interacts with the business logic, is left to the following section, “The Controller”.

We have sorted the classes into smaller packages according to what actions the classes perform. Similar and subsequent actions, have been put in the same package. These small packages are, as the figure shows, called “Layout”, “Top”, “Menu”, “PersonalInfo”, “Administrator”, “Event” and “Order”. Notice, that some of the classes has the prefix “Secure” or “Sec”. These secure versions are to be used when a user is logged on to the system, and should not be cached. This is according to both security guideline 2.2.3, “Limit cache usage” and the system requirement SR4 in chapter 4, “Requirements engineering”.

Layout

The *“Layout”* package contains classes that forms the web page’s layout. The web page should have a vertical menu on the left side (“Menu”), a top field lying horizontally across the entire upper part of the page (“Top”), and a main page in the middle that shows information about the site (“Mainpage”).

Top

The *“Top”* package holds the links “Login” before a user logs on to the site, and “Logout” when the user is logged on. The “Login” page shown to the user, contains one field for entering username, and one field for entering password. In addition, it contains a link that says “Register new user account”, that the user can click on if he or she is a new user, and a forgot password function.

Menu

The *“Menu”* package holds the links a user may click on. The links leads to pages containing information the user might find useful, like event-, contact information, and information about the site. It is the classes “Events”, “ContactInformation” and “About” that provides the user this information through the web browser.

PersonalInfo

The *“PersonalInfo”* package holds the classes “UserInfo”, “ShowUserInfo”, “EditPersonalInfo” and “EditInfoOK”. “UserInfo” presents the user a form through the web browser where the user can enter personal information. The form should contain the fields enlisted in requirement RUA1 in chapter 4, “Requirements Engineering”, and a hidden field called “role” with default value “user”. “ShowUserInfo” shows the personal information that the user has stored in the database. The class “Edit-Information” presents the user a form through the web browser where he or she can edit his or her personal information. The information the user may edit, is according to requirement RU1 in chapter 4, “Requirements Engineering”. Both regular-, administrator- and super users may be presented with this view.

SuperUsers

The *"SuperUsers"* package holds the classes "EditAdmin", "AddAdmin", "ChangeAdmin", "RemoveAdmin" and "EditAdminOK". "EditAdmin"shows the super user which administrator users he or she can choose from, and a button "Add administrator". "AddAdmin" shows a form where the super user can add information on a user (according to requirement SU4), using a form equal as in the "UserInfo" class in the "PersonalInfo" package, except that the hidden field now should have the value "administrator". The class "ChangeAdmin" equals the "EditPersonalInfo" from the "PersonalInfo" package. "DeleteAdmin" is a class that presents a list to the super user through the browser where all enlisted elements have a box next to them. The super user can tick off all the administrators he or she wants to delete, and press a "Delete" button.

Event

The *"Event"* package contains several classes. The "Events" class shows the users through the web browser which event the user can choose from. The classes"Concerts", "Theaters" and "Musicals" presents information about the events. Administrator- and super users are presented the content from the classes "EditEvents", "AddEvents", "ChangeEvents", "RemoveEvents" and "EditEventsOK". "EditEvents" shows a list with all of the registered events, and a button "Add event". "AddEvents" shows a form the administrator- or super user can fill out in order to add an event. "ChangeEvents" shows a form where the administrator- and super users can enter updated information on events. "RemoveEvents" shows a list of all events, with a box behind each element, and a button that says "Remove event". Then the administrator- and super user can tick off the elements they want to remove, and push the button. The Super users is also presented with a "Delete" button (requirement SU6). The "EditEventOK" class presents through the web browser a message saying that the editing of an event went well.

Order

The *"Order"* package contains three classes, namely "Order", "EnterPayment" and "OrderOK". The "Order" class presents the user with a form with event information, and a question about how many tickets he or she wants to order. The "EnterPayment" shows a form where the user can enter payment information. The "OrderOK" class presents through the web browser a message saying that the ordering tickets for an event went well.

Security

Since we are using the Model-View-Controller approach, there is no business logic in the view. Therefore, when it comes to security, we have to make the end users aware of the security breaches that may occur, like telling them not to leave their work stations when logged on to systems or applications and never to give away their user credentials. This is according to the security guideline 2.1.9, "Awareness". If users doesn't learn about the threats that are out there, and aimed at them, they can easily be exposed to social engineering attacks like *phishing*. The most regular form of a phishing attack, is that the attacker sends the user an email, claiming to be system administrator, saying he or she needs the users credentials, and asks the user to fill out a form accessible through an appended link.

The risk of a phishing attack (R2) occurring, has a very high estimated risk value (12) on the ranking of risks in the table shown in Figure 4.12 in chapter 4, "Requirements engineering". It is therefore important to make the users aware of this type of attack.

In chapter 3, "Vulnerabilities, threats and countermeasures", it is suggested to solve the problem of phishing by making the users of the application aware of the problem. This could be done by asking the user to be selective to what information that is good, and what is bad. Following are a list with suggestions to the user of how to deal with situations where he or she gets insecure. The list is from The Center for Information Security in Norway [30]:

- If something seems to good to be true, it might be the case.
- If you are insecure about if the content in an email is malicious or not, check with others.
- Be sceptic towards contact information or links that arrives in an email.
- Check the addresses on the web. Fake web pages can look identical as the legitimate ones.

5.5 The Controller

The pages presented to the user is only a GUI. If the application should be able to perform actions, there is need for some business logic, which is placed in the controller of the MVC architecture.

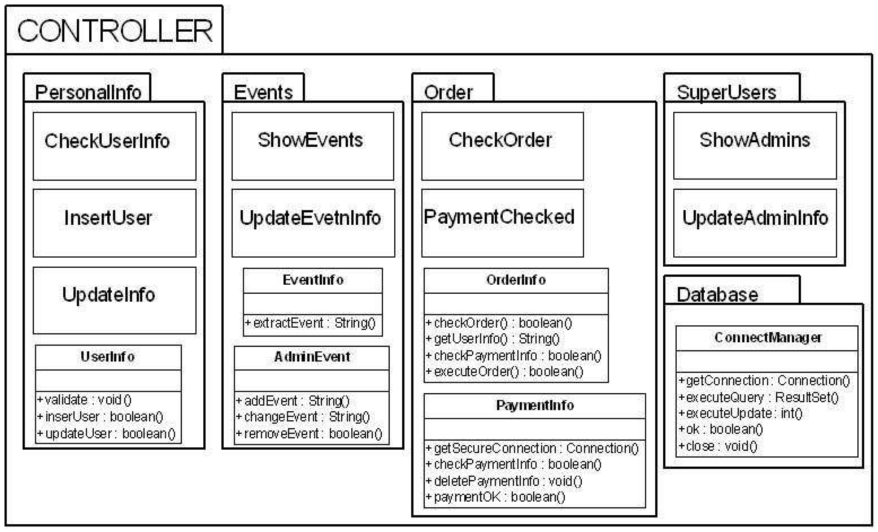


Figure 5.4: Files in the controller.

Figure 5.4 shows a package called "Controller" which contain smaller packages called "PersonalInfo", "Events", "Order", "SuperUsers" and "Database". As in the previous section, the classes are categorized with regard to what actions they perform. This time, it is both JSP- and Java classes in the smaller packages. This is because some of the business logic should be implemented in Java classes.

5.5.1 JSP classes

To explain the functionality we will divide the JSP classes from the Java classes, and start off with the JSP classes. To get a better overview of what action they should perform, we have drawn a component diagram where the files from the view and the JSP files from the controller are interweaved, showing how they are connected to each other. The diagrams are inspired from regular class diagrams, showing relations between entities [40].

We would like to emphasize that these diagrams does not show the flow between the classes, only the connection between them. The diagrams are divided into the subsections *Before login*, *Regular users*, *Administrator users* and *Super users* like the requirements in chapter 4, "Requirements engineering". This gives a better picture of what different actions users are able to perform, and it makes it easier to explain what the JSP classes in the "Controller" package does. The control elements are put in yellow boxes, so that they are easy to spot.

Before login

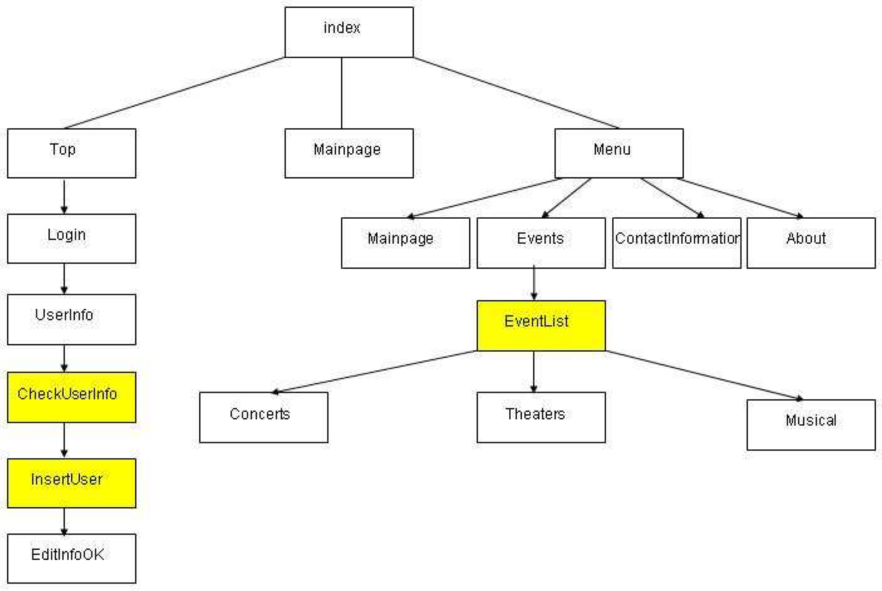


Figure 5.5: Component diagram - Before login.

Figure 5.5 shows how the web page is built up before the user logs on to the system. The classes in the view was explained in previous section. Following is a description of the controller JSP classes.

“EventList” should get information about the event the user chose through the “Event” class in the view. Then the class checks if it is a user before log in, a regular user or an administrator user that asks for this information. Here, it is the user before log in, and therefore, “EventList” returns the information to either the “Concerts”, “Theaters” or “Musicals” class, depending on the type of event the user chose.

“CheckUserInfo” should check if the input the user entered in the form in the “UserInfo” class, is valid according to requirement RUA1b in chapter 4, “Requirement engineering”.

“InsertUser” should receive the validated information from the “CheckUserInfo” class, and insert it into the database.

Regular users

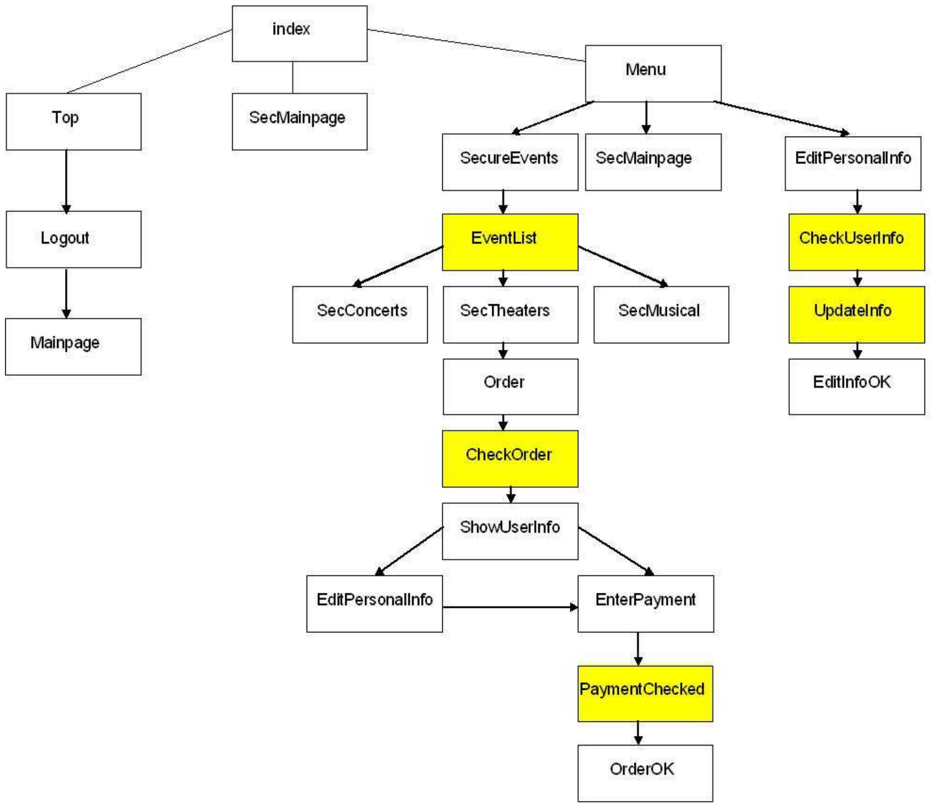


Figure 5.6: Component diagram - Regular users.

Figure 5.6, shows how the web page is built up when a regular user is logged on to the system. The layout is the same as for users before login. The difference is that the user now may log out (instead of log in), order tickets, and edit personal information.

The “EventList” should get information about the event the user chose through the “Event” class in the view. Then the class checks if it is a user before log in, a regular user or an administrator user that asks for this information. Here, it is the regular user , and therefore, “EventList” returns the information to the “Sec-Concerts”, “SecTheaters” or “SecMusicals” class, depending on the type of event the user chose.

The “CheckOrder” class, should get input from the “Order” class, that says how many tickets the user wants to order. The “CheckOrder” class checks if there are more tickets left, and if so, it returns the users personal information to the web browser through the “ShowUserInfo” class in the view.

“PaymentChecked” should receive payment information that the user have entered into a form in the “EnterPayment” class from the view. If the user has enough money on his or her account, the “PaymentChecked” will let the order go through.

“CheckUserInfo” has the same functionality as before login. The “UpdateInfo” is similar to the “InsertUser” class, but instead of inserting each field in the form coming from the “EditPersonalInfo” class in the view, it updates only those fields where changes have been made by the user.

Administrator users

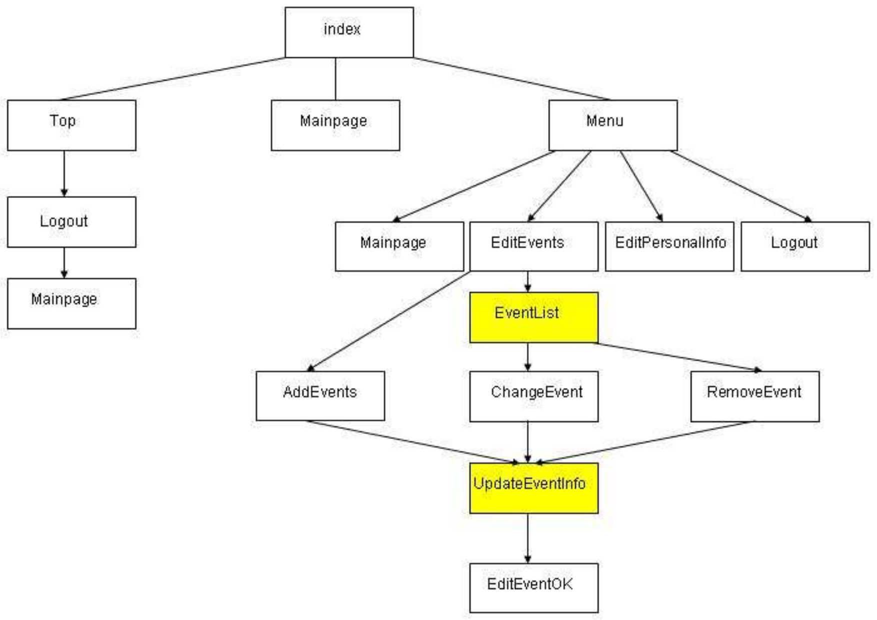


Figure 5.7: Component diagram - Administrator users.

The administrators gets a specific URL to where they can find their login in page. Figure 5.7 has the same layout as the regular user, but with some variations in the functionality available. The functionality is according to requirements in Table 4.3 in chapter 4, “System requirements”.

"EditPersonalInfo" is similar to the same functionality for regular users, and are not drawn into the figure to keep from getting too complex.

"EventList" should get information about the event the user chose through the "Event" class in the view. Then the class checks if it is a user before log in, a regular user or an administrator user that asks for this information. Here, it is the administrator, and therefore, "EventList" returns the information to either the "ChangeEvent" or "RemoveEvent" class, depending on the type of event the user chose. (Note, that if the administrator chooses the "Add event" button in the "Event" class, he or she is presented the view of the "AddEvent" class.)

"UpdateEventInfo" either inserts a new event in the database, or updates the information that the administrator has entered in the "ChangeEvent" or "RemoveEvent" classes.

Super users

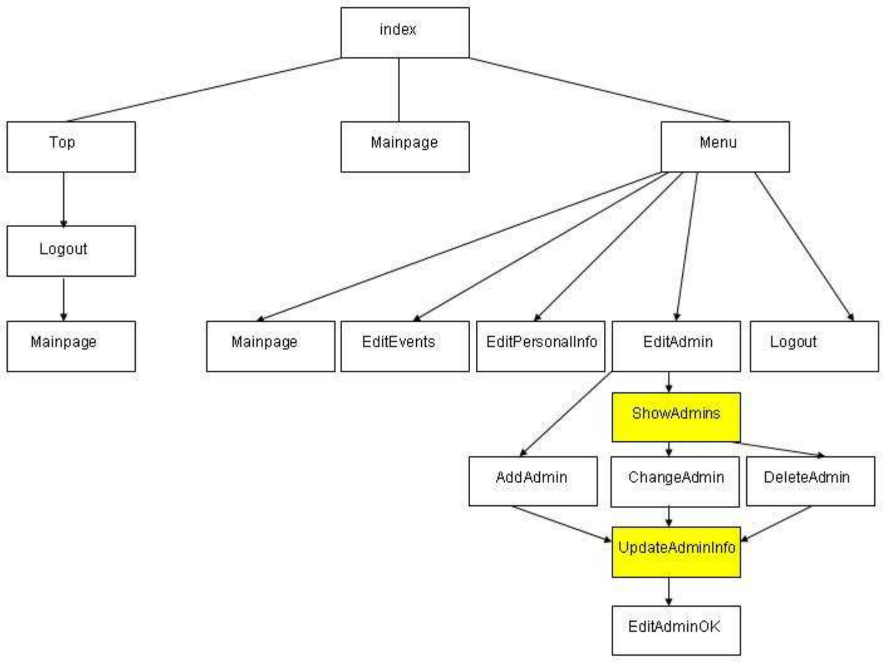


Figure 5.8: Component diagram - Super users.

Figure 5.8 shows what the page looks like after a super user logs in. The super users finds the login page as the administrator, by entering a known address in the URL.

According to requirements in Table 4.4 in the previous chapter, a super user is capable of editing a administrator account.

The "EditEvent" procedure is equal to the one for the administrator, except that the super user also have the privilege to delete an event. This is according to requirement SU1c in chapter 4, "Requirement engineering".

The "EditAdmin" class enlisted all administrator users to the super user through the web browser. The "ShowAdmins" class should get information about the chosen administrator, and send it to the "ChangeAdmin" or the "DeleteAdmin" class.

The "UpdateAdminInfo" class either inserts a new administrator user in the database, or update the changed information the super user has entered on the chosen administrator.

5.5.2 Java classes

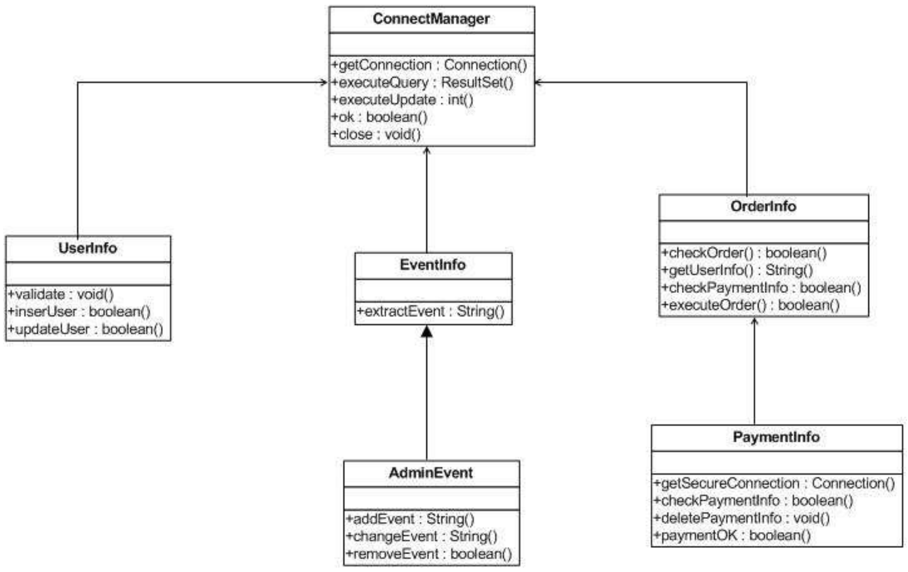


Figure 5.9: Class diagram of the business logics Java files.

As the "Controller" package in Figure 5.4 showed, the control part of our MVC architecture, also contain Java files. Using Java files helps us centralizing the business logic, and makes it easier to expand functionality. This is according to security guideline 2.1.3, "Keep it simple", where the Java files can be choke points where all security critical operations are lead through.

Figure 5.9 shows the Java classes and the relation between them. The model is from [40]. The classes contains some methods that is necessary to perform wanted operations, and these will be used in sequence diagrams in section 5.6, to explain the flow of the operations.

But first we will enlist the classes shown in Figure 5.9, and give a brief description of how what functions they should contain.

ConnectManager. The "ConnectManager" should be the only class to establish a connection with the database using JDBC. All other classes should connect to the database through thhe “ConnectManager”. The “ConnectManager” should execute queries, execute updates, and close the connection.

UserInfo. The “UserInfo” should be a JavaBean that validates all personal information coming from regular users, administrator users and super users, and insert and update information through the “ConnectManager” using SQL queries.

EventInfo. The “EventInfo’ class should extract information from the database through the “ConnectManager” using SQL queries.

AdminEvent. The “AdminEvent” contains methods for adding events, changing events and removing events using SQL queries. If the super user chooses to delete an event, this should also be handled in this class.

OrderInfo. The “OrderInfo” class should send an SQL query to the database to check if there are enough tickets left according to the users order. If so, the class must check if the user has enough money on his or her account. If there is enough money, the “OrderInfo” class executes the order by sending an SQL query through the “ConnectManager” to the database.

PaymentInfo. The “PaymentInfo” class establishes a secure connection to the VISA terminal, checks if the user has enough cash on his or her account and returns true or false to the “OrderInfo” class.

Security

Using a MVC architecture keeps a simple and an easy-to-follow design of the applications business logic. It makes it easy to create choke points where we lead critical security operations through. This is according to security guideline 2.1.3, “Keep it simple”. Using the “ConnectManager” to handle all connections with the database, is one choke point. If the database and the application server is connected over a network, it is only the “ConnectMangagar” file that has to make sure that a secure connection between the two of them is established.

The business logic handles all input from the user, that is, from the client side. One of the biggest risks that comes with user inputs, is the risk of *SQL injections* (R3), that are ranked with a high risk value (12) in our risk analysis in Figure 4.12. There is also a risk that some users might try to *elevate their privileges* (R9), by logging in as a higher privileged user. In addition, there is a risk of *release of message content* (R7) when payment information is sent from the application server to the VISA terminal. All of these attacks can be circumvented.

Using the “UserInfo” class to take care of all input validation, is a good countermeasure against *SQL injections*. If there is discovered a security flaw in the validation procedure, that lets an SQL injection attack take place, this can be repaired in the “UserInfo” class. All forms that takes input from the user and uses the “UserInfo” class to validate input, will be fixed, as soon as the “UserInfo” is fixed. This makes the “UserInfo” class a choke point where critical security operations are lead through, which is according the security guideline 2.1.3. In addition it follows the guideline 2.1.1, “Validate input”.

As to *elevation of privileges*, this may be an attack performed by an attacker that has gotten his or her hands on an administrator- or super users user credentials. By giving the administrator and super users an URL where they can find their log on page, instead of putting a link on the front page of the application, available to all users, we make it a little bit harder for the attacker to find a way in to the system.

When it comes to the *release of message content* attack, there must be a secure, encrypted connection established between the “PaymentInfo” and the VISA terminal. This can be done using IPSec.

5.6 Sequence diagrams

Objects communicate by sending messages. To carry out a certain task, a particular sequence of messages may have to be exchanged between two or more entities [40]. The time ordering in which this sequence of messages has to occur may be depicted in a sequence diagram. Such a diagram depicts the interactions between class instances, in the form of method calls and call returns [44]. We have chosen to model security critical operations in our application in sequence diagrams to shed light on processes that it is important to security.

We will focus on modeling the actions that takes place between the JSP files, the Java files, and the database. The view is presented as the web browser, the controllers with the names of files, and the model as the database (DB).

5.6.1 Login

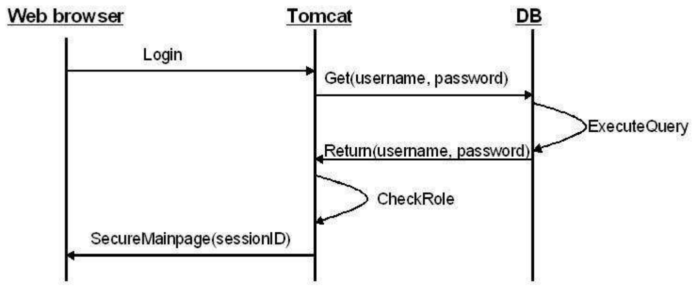


Figure 5.10: Sequence diagram - Login.

We’ll start off presenting the log on routine. Figure 5.10 shows what happens in the application tier when a user chooses to log on to the system.

First, the user must enter his or her user credentials into a form on the “Login” page in the view (web browser) to be authenticated. Then this information is sent to the application server, which is the Tomcat server introduced in section 5.2. The Tomcat asks the database for user credentials. The database executes a query, and returns the values to the Tomcat server. The Tomcat checks if the users input matches the stored values in the database. If so, it returns a session identifier to the user, and the secure mainpage, and the user is logged in.

Security

When it comes to security, it is important that the database either is in a safe, closed environment with the Tomcat server, or establishes a secure connection to the database, using for instance OpenSSL with MySQL [15]. If this is not done, both the application servers log on information to the database, and the user's credentials might be revealed to a potential attacker, who might perform the passive attack, release of message content (R7).

5.6.2 Logout

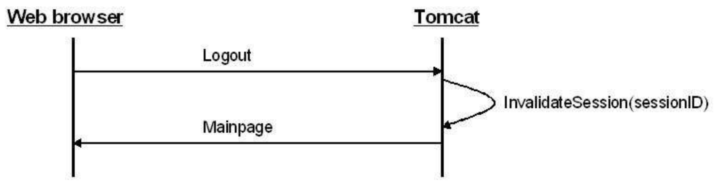


Figure 5.11: Sequence diagram - Logout.

Figure 5.11 shows how the logout procedure takes place in the system. The user presses the logout button in the "Top" window, and a the "Logout" JSP class in the view, sends the message to the Tomcat server. The tomcat server invalidates the message.

Security

As the user logs out of the system, it should not be possible to press the "back" button in the web browser, and then reenter the logged in session. This can be avoided, by setting the header information to "no-cache" in the classes used by the regular user with the prefix "Secure" and "Sec". All of the classes to the administrator and super user should have this restriction as well. This is according to the system requirement SR4a, "Sensitive data should not be cached", and security guideline 2.2.3, "Limit cache usage"..

If the user then presses the "back" button after he or she has logged out, there will be displayed an error message stating that the session has expired.

5.6.3 ConnectManager

Figure 5.12 pictures the interaction between the Java file "ConnectManager" and the database.

What happens, is that the "ConnectManager" establish a connection between the application server and the database, using the method *getConnection()*. Further, the "ConnectManager" should execute all queries that comes from other classes in the business logic, and execute all updates. When the "ConnectManager" is finished with its work, it closes the connection with the database.

Security

All correspondance the controller has with the database, should go through the "ConnectManager" file. Then this file will work as a choke point for data going from the controller to the database, according to security guideline 2.1.3, "Keep it simple".

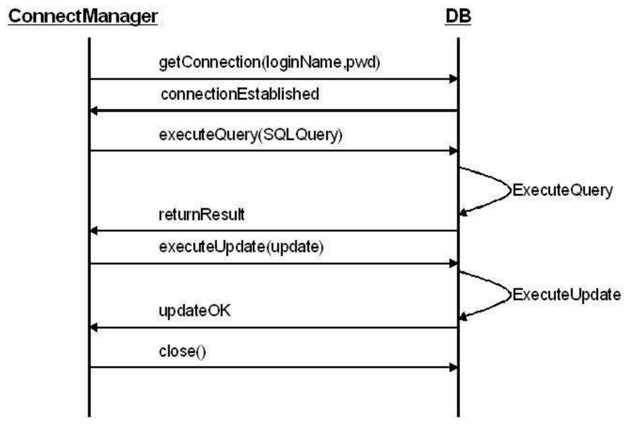


Figure 5.12: Sequence diagram - ConnectManager.

Again it is important that the database either is in a safe, closed environment with the Tomcat server, or that it is established a secure connection to the MySQL database using either IPSec, or OpenSSL[8]

5.6.4 Register and edit user information

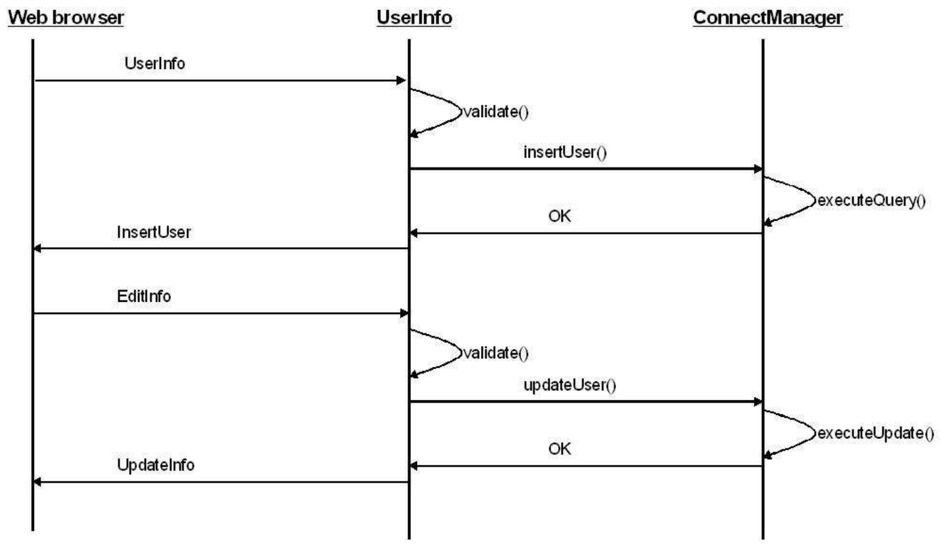


Figure 5.13: Sequence diagram - Register and edit user information.

Figure 5.13 describes what happens between the view and the controller when a user registers to the system for the first time, and when a user updates personal information.

[8].

First, the user chooses to register as a new user, and fills out a form with his or her personal information. The web browser posts the form containing user information to the application server. There, a JavaBean called “UserInfo” checks if the input the user has entered, is on a correct form. If the information is correct, the “UserInfo” bean uses an *insertUser()*-method to insert the users information into the database. The connection between the JavaBean and the database, goes through the “ConnectManager”. The change is sent to the view through the “InsertUser” JSP file, which is shown in the “Controller package” in Figure 5.4.

When the user wants to edit his or her personal information, the web browser sends the users updates, to the application server. There, the “UserInfo” bean again validates the users input, and sends them off to the “ConnectManager” if they are correct. Then the “ConnectManager” connects to the database and executes an update. The update is sent to the view through the “UpdateInfo” JSP file, which is shown in the “Controller” package in Figure 5.4.

Security

One of the biggest threats this application faces, according to our risk analysis in Figure 4.12, is the SQL-injection (R3) that has a high risk value (12). By validating all user input, the risk of SQL injection will be reduced.

According to security guideline 2.2.1, “Validate input on the server side”, all input coming from the client side should be validated. Keeping all the validation logic in a JavaBean, makes it easier to maintain the validation process. For instance, if there is found a vulnerability in one of the input fields, this is repaired in the JavaBean. Everywhere in the application where this input field is present, will be updated.

When using centralized security mechanisms, like putting the validation process in a JavaBean in the controller, there is no need to identify every page that handles this type of input, and update every single page. Not only is it easy to overlook a page containing this input field, but it is hard to test if the improvement has been implemented correctly at each one of them. Leading all critical security operations through a choke point, like all validation through a JavaBean, is according to security guideline 2.1.3, “Keep it simple”.

5.6.5 Events

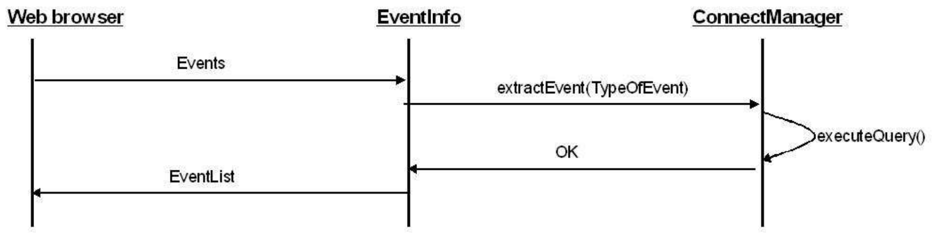


Figure 5.14: Sequence diagram - Event.

Figure 5.14 explains the flow of actions when the user wants to watch which events that the web application has information on.

The user clicks on the event he or she wants more information about, in the web browser. The Java file "EventInfo" on the controller side, contacts the "ConnectManager" and asks that file to extract the information on the chosen event from the database. The "ConnectManager" executes its actions, and returns the result to the "EventInfo" file. The "EventInfo" uses the "EventList" JSP file to return the result to the view.

Security

As for security, all the connections with the database goes through the "ConnectManager". This will help to keep an easy-to-follow architecture, and is according to the security guideline 2.1.3, "Keep it simple".

5.6.6 Administrator editing events

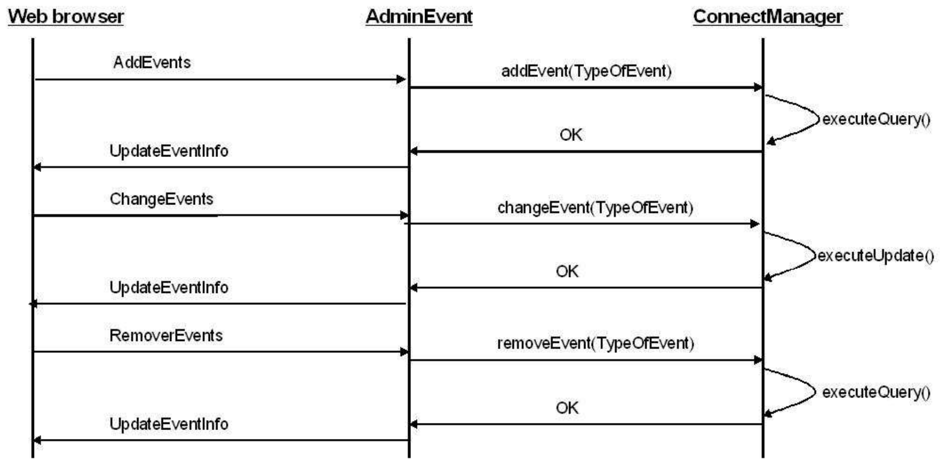


Figure 5.15: Sequence diagram - Administrators editing events.

Figure 5.15 describes how an administrator can add, change and remove events from the database. The super user can edit events the same way as the administrator. There will not be drawn a sequence diagram for that operation, since it is exactly the same.

The administrator chooses through the web browser, for instance, to change an event. He chooses which event he wants to change, the same way as shown in the sequence diagram in Figure 5.14. When the event is shown in the administrators web browser, he makes the necessary changes through a form, and sends the form to the controller.

The request is sent to the Java file "AdminEvent" on the controller side. This Java file takes the information from the form, creates an SQL query to the database to insert the information. The SQL query is sent to the "ConnectManager", and this file handles all the database communication. When the database returns the result to the "ConnectManager", it is forwarded to the "AdminEvent', which then again uses the JSP controller file "UpdateEventInfo" to return the result to the view.

The administrator receives a response through the web browser that the update was successful, through the JSP view file, "EditEventOK".

Security

It is assumed that the administrator is a good guy. But if the administrator is a crook who wants to tamper with information stored in the database, he or she would have the possibility to try to send an SQL-injection through the form used to edit event information.

But when using Java to handle all the business logic, this will filter out the error messages coming directly from the database, and not reveal core database information to the administrator. Since having Java files in the business logic, detailed error messages won't arrive on the client side. This is according to the security guideline 2.1.2 "Fail securely".

5.6.7 OrderInfo

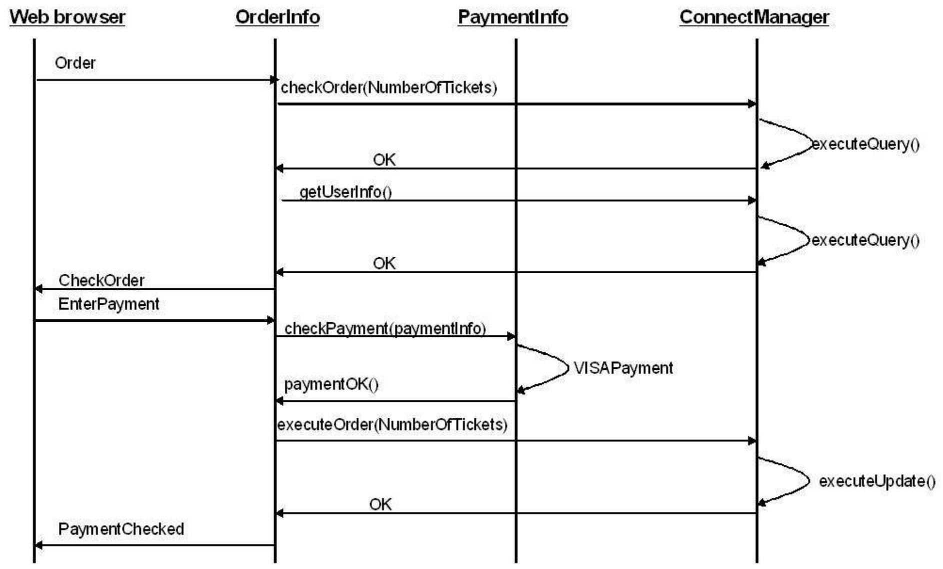


Figure 5.16: Sequence diagram - Order.

Figure 5.16 illustrates the coherence between the web browser, the "OrderInfo", the "PaymentInfo" and the "ConnectManager".

When the user is logged on to the system, he or she can order tickets to desired events through the web browser. Information about the order is posted in a form to the controller.

The controller has a Java file called "OrderInfo", that handles the business logic behind the order. It checks in the database, via the 'ConnectManager", if there are more tickets left to the requested event. If so, it sends a request to get the users personal information from the database, again via the "ConnectManager". When the information arrives, the JSP file "CheckOrder", that hands over the information to a JSP page in the view called “ShowUserInfo”. This page tells the user to either edit the personal information end then enter payment information, or confirm that personal information is correct, and then enter payment information.

The user enters payment information in a form within the JSP view file, “EnterPayment”. The “OrderInfo” checks if the user has enough money on his or her account, through the “PaymentInfo” Java file. (This file handles all connections to the VISA terminal, and the procedure will be described in the following subsection.) If so, the “OrderInfo” execute the order, updates the “Order” table in the database with order information, and sends the JSP file “PaymentChecked” back to the web browser, which shows the page “OrderOK” to the user.

Security

With regard to security, it is important that the communication line between the web browser and the business logic is encrypted. Encrypting this connection, is according to the security guideline 5.3, “Encrypt communication lines”. If the communication line is not encrypted, there will be a risk of *replay* attacks occurring. That is, an attacker may resend a users order repeatedly, and thereby orders an unwanted amount of tickets on the users behalf.

In addition, it is very important that the payment information data is not stored in the users cache when a web page is posted. Then sensitive payment information will be stored locally on the users computer, and could therefore be exploited by others using the same machine. Not storing the payment information in cache, is according to security guideline 2.2.3, “Limit cache usage”.

5.6.8 VISA payment

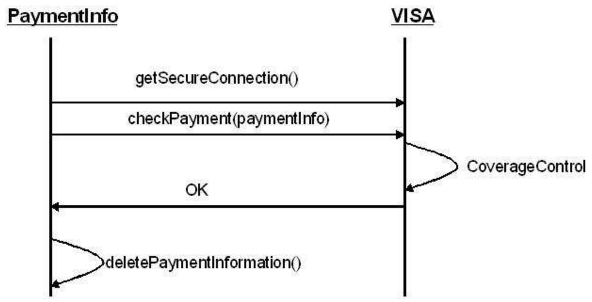


Figure 5.17: Sequence diagram - VISA payment.

Figure 5.17 shows how the “PaymentInfo” Java file interacts with a VISA terminal. First, the “PaymentInfo” establish a secure connection with the VISA terminal. Then it sends a request to check if the user has enough money on his or her account. Then the VISA terminal does a coverage control. If the customer has enough money on his or her account, the buy can go through.

Security

As for the application, all the payment information from the user must be sent over an encrypted communication channel from the web browser to the application server.

This can be done using the Secure Socket Layer (SSL) offered by the Tomcat.

The communication between the "PaymentInfo" and the VISA terminal must also be secured. This could for instance be done using IPSec.

5.7 Summary

This chapter started off presenting the architecture we have chosen for our application, and why we chose exactly this one. There have also been a short introduction to technologies that is advantageous to have knowledge about for better understanding our design.

The design of the web application have, in addition to modeling the functionality, emphasized where the system is vulnerable and might be exposed to threats. Designing an application with constant focus on where the security is threatened, requires that you have thought through different threats and vulnerabilities that is likely to occur in the front edge of the design. Otherwise it might be hard to spot eventual threats.

The upcoming chapter will describe an implementation we did of the application that we have designed. The goal is to see how we can solve some of the security issues pointed out in the requirements and design, on the Java platform.

Chapter 6

Implementation

Throughout the development process, and working on this report, the focus has been set on security guidelines. The process of identifying needed functionality and relating it to potential threats, has been a main goal of this project. We have implemented a online ticket ordering system according to our requirements and design, but also according to our security guidelines. The requirements tables, which were identified in chapter 4, "Requirements engineering" have been repeated in Appendix D, this time with marks to indicate implemented functionality. In this chapter we will be presenting interesting parts of the implementation, that is, parts of it relevant to security. From the risk analysis at the end of chapter 4, "Requirements engineering" we have chosen one of the highest ranking threats to our application, SQL injection, which will be further treated in this chapter. In addition, we have chosen to present how prevention of cross site scripting attack is implemented, as this is an attack that requires that an authenticated administrator is a crook. This is interesting, and also a reminder that one of the threats to applications of any kind is attacks from the inside.

Security guidelines one through six is shown implemented in various sections of this chapter. In addition, all of our specific web application guidelines have been illustrated, in addition to a presentation of two widespread technologies, Secure Socket Layer (SSL) and the use of J2EE containers. Concurrency is also treated briefly at the end of the chapter.

As the implementation is presented, screen shots from our application is included along the way. Appendix F contains all of our code[1].

6.1 Assumptions and simplifications

Developing a web application with limited resources requires that some assumptions be taken. They are present in the list below.

- All ordered tickets will be sent, with a bill, by post to customers. The reason why we have chosen this approach is that to connect a web application to a VISA terminal cost money, and is not necessary as we have already identified the important concepts concerning this procedure.

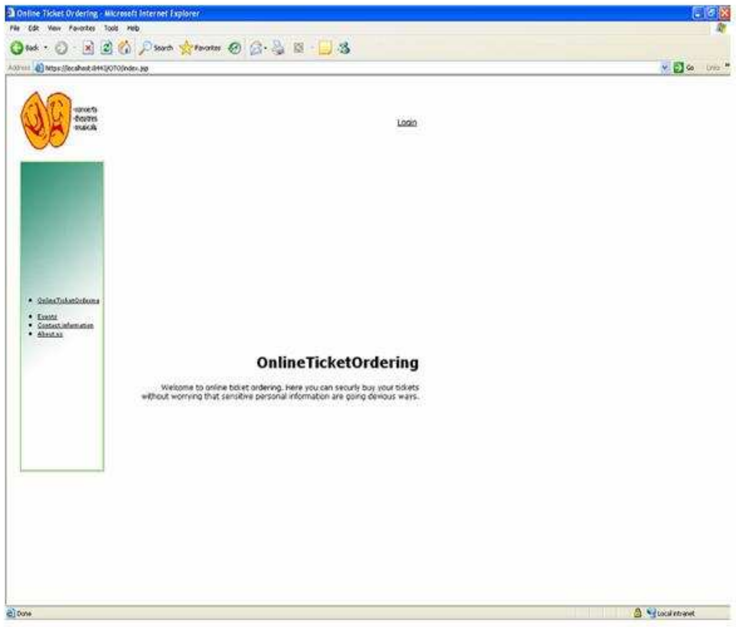
[1]The installation guide is provided on a CD which contains all of the applications code.

- The procedure of editing personal information when logged in is left out, because this functionality would not shed light on any aspects different from the register user procedure.
- The administrator and super users will not be implemented.

6.2 The application and security considerations in Java

The application we have implemented, picks up on some of the security issues according to our design. We have focused on making the application as illustrative as possible.

The web site we have created is a ticket ordering site, called "OnlineTicketOrdering" (OTO), and Figure 6.1 shows a screen dump of the front page. More screen


Online Ticket Ordering - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://localhost:8443/OTO/index.jsp
Go Links
-concerts -theatres -musicals
Login
OnlineTicketOrdering
Events
Contact information
About us
Done
Local intranet

OnlineTicketOrdering

Welcome to online ticket ordering. Here you can securly buy your tickets without worrying that sensitive personal information are going devious ways.

Figure 6.1: This is what the front page in our application looks like.

dumps are found in Appendix D. Those that describes pages where security considerations has been made, are also to be found in this chapter where it is necessary, along with code examples and a textual description.

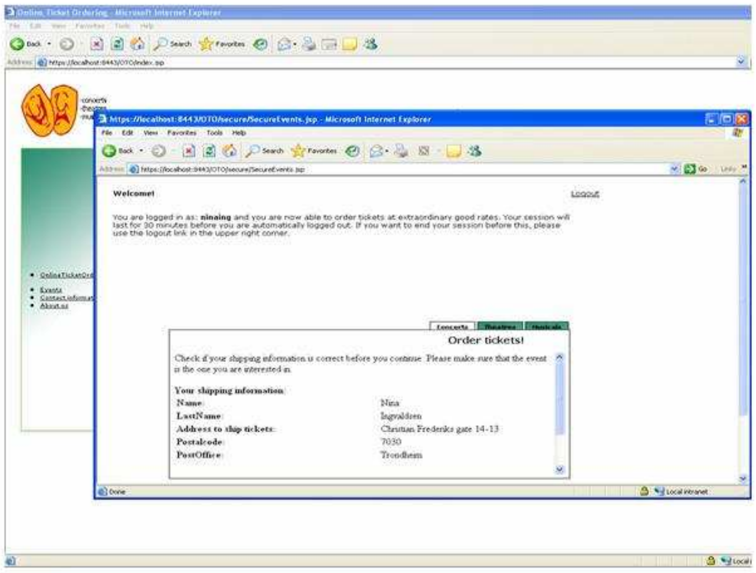
6.2.1 Using the Secure Socket Layer

To safeguard communication between the browser and the server, we make use of the Secure Socket Layer (SSL), which is supported by Tomcat. SSL is a technology which allows web browsers and web servers to communicate over a secured connection. This means that the data that is sent is first encrypted by the sender, transmitted, and then decrypted by the receiver before processing continues. This is a two-way process, in the sense that both the server *and* the browser encrypt all traffic before sending data [38].

Another important aspect of the SSL protocol is *authentication.* During an initial attempt to communicate with a web server over a secure connection, the server will present your web browser with a set of credentials, in the form of a digital certificate[2], as proof of who the site is and what it claims to be. In certain cases, the server may also request a certificate from your web browser, asking for proof that you are who you claim to be. This is known as *client authentication* [37].

Since we have been using SSL, we also maintain *confidentiality and integrity* of data, since the encrypted communication makes it hard for an attacker to tap- and/or tamper the data.

For instance, when a user orders a ticket, this is done under the protection of SSL. Figure 6.2 shows the ordering ticket part, where the user has ordered a ticket, and have to confirm shipping information. Since SSL is used to protect these transactions, the risk of a replay attack is reduced.


Online Ticket Ordering - Microsoft Internet Explorer
https://localhost:8443/OTO/index.jsp
https://localhost:8443/OTO/secure/SecureEvents.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
https://localhost:8443/OTO/secure/SecureEvents.jsp
Go
Logout
OnlineTicketOrd
Events
Contact informat
About us
Concerts
Theatres
Musicals
Done
Local intranet

Welcome!

You are logged in as: **ninaing** and you are now able to order tickets at extraordinary good rates. Your session will last for 30 minutes before you are automatically logged out. If you want to end your session before this, please use the logout link in the upper right corner.

Order tickets!

Check if your shipping information is correct before you continue. Please make sure that the event is the one you are interested in.

Your shipping information:

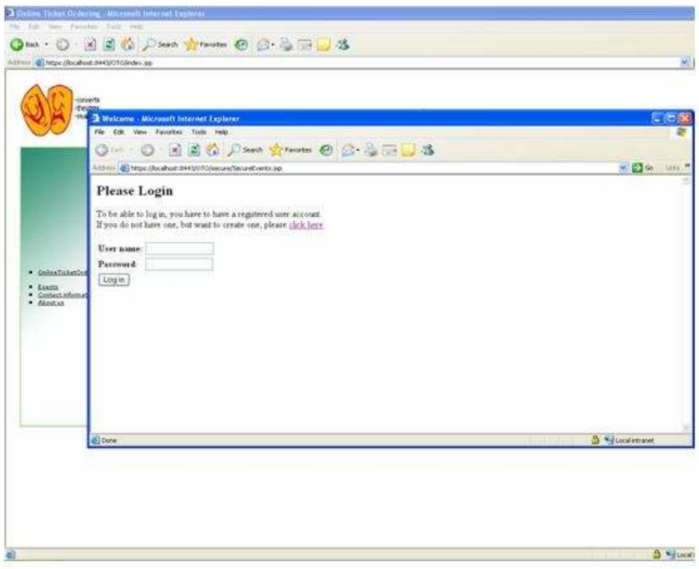
Name:	Nina
LastName:	Ingvaldsen
Address to ship tickets:	Christian Frederiks gate 14-13
Postalcode:	7030
PostOffice:	Trondheim

Figure 6.2: This is what the application looks like when a user has decided to order a ticket.

[2]A digital certificate is a digital representation of information which at least identifies the certification authority issuing it, names or identifies its Subscriber, contains the subscribers public key, identifies its operational period, and is digitally signed by the certification authority issuing it [44].

6.2.2 Using containers

When a user has registered him- or herself in the system, he or she can log on to it through the login window shown in Figure 6.3. The log on procedure is put in


Online Ticket Ordering - Microsoft Internet Explorer
Welcome - Microsoft Internet Explorer
https://localhost:8443/OTO/index.jsp
https://localhost:8443/OTO/secure/SecureEvents.jsp
Events
Contact information
About us
Done
Local intranet

Please Login

To be able to log in, you have to have a registered user account.
If you do not have one, but want to create one, please click here

User name:
Password:
Log in

Figure 6.3: This screen dump shows how the login window looks like.

a *container*. A servlet container is the connection between a web server and the servlets. It provides the runtime environment for all the servlets on the server as defined by the servlet specification, and is responsible for loading and invoking those servlets when the time is right [2].

The servlet container is responsible for mapping incoming requests to a servlet registered to handle the resource identified by the URL, and passing the request message to that servlet. After the request is processed, it is the containers responsibility to convert the response object created by the servlet into a response message, and send it back to the client [2].

In our log on procedure, the container is used in the *authentication* process [3]. This process involves associating the current servlet request object, the current execute thread, and the internal session, with the user's identity. By associating a session with the identity, the container can guarantee that the current request and all subsequent requests by the same user can be associated with the same session until that user's session is invalidated [2].

J2EE-based web containers offer three types of authentication mechanisms; basic, form-based and mutual authentication [10]. Most web applications use the form-based authentication mechanism, because it allows the application to customize he authentication user interface. Our application does the same. The **¡login-config¿** section of `web.xml` defines the type of authentication mechanism, and the URLs to login and error pages. The following is an excerption of our `web.xml` file:

```
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/OTO/Login.jsp</form-login-page>
        <form-error-page>/OTO/LoginError.jsp</form-error-page>
    </form-login-config>
</login-config>
```

In this excerption, the authentication method is set to form-based, and whenever a user requests a protected page, if he or she has not yet been authenticated, they will be redirected to the `login.jsp` so that they may provide their credentials. Should an error occur, or the authentication fail, the user will be taken to the `LoginError.jsp` page.

The pages that are to be protected are identified in the next excerption, encapsulated by the `<url-pattern>` tags. In our application, all files placed in the `secure` sub directory of the OTO directory are protected. It is also possible to specify protection of HTTP-methods, if you choose to specify any in the list of `<http-method>`s, only those specified will be protected. We have protected all HTTP-methods.

```
 <security-constraint>
    <web-resource-collection>
        <web-resource-name>Protected area</web-resource-name>
        <url-pattern>/OTO/secure/*</url-pattern>
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
        <http-method>PUT</http-method>
    </web-resource-collection>

    <auth-constraint>
        <role-name>user</role-name>
    </auth-constraint>

    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
<security-role>
    <role-name>user</role-name>
</security-role>
```

`<security-role>`-elements specify the configured roles that the application may use in the `<security-constraint>`. The `<auth-constraint>` tag, that lies inside the `<security-constraint>`, defines what roles in the system should have access to the area in question. In other words, containers provides both authentication and authorization. Authorization is enforced at page level. In our application, users, which translates to "regular users" have access to the secure directory after successful login. To be able to accommodate separate areas for the administrators and super users, corresponding areas would have to be defined.

Whenever we send a user ID and password over the network, the password is sent in base64-encoding[3]. This is considered to be clear text, so the transport connection should be encrypted. The `<security-constraint>` requests an SSL connection by specifying CONFIDENTIAL in the `<transport-guarantee>` tag.

Database realm

A realm is a "database" of usernames and passwords that identify valid users of a web application, plus an enumeration of the list of roles associated with each user. A particular user may have any number of roles associated with their username [26]. Tomcat 5 defines a Java interface (org.apache.catalina.Realm) that provides support for connection to existing authentication databases. We have stored authentication information in our relational database, and thus the security realm is set to our database. It will be accessed through JDBC, and it is configured in `server.xml`. An excerpt is shown below:

```
<Realm className="org.apache.catalina.realm.JDBCRealm" digest="SHA" debug="99"
        driverName="org.gjt.mm.mysql.Driver"
        connectionURL="jdbc:mysql://mysql.stud.ntnu.no/ninai_db"
        connectionName="ninai_admin" connectionPassword="1LEfant"
        userTable="Users" userNameCol="UserID" userCredCol="Password"
        userRoleTable="User_roles" roleNameCol="Rolename" />
```

Some requirements needs to be fulfilled when using the database as the security realm. There must be a table that contains one row for each valid user this `Realm` should recognize. The table is "Users", as seen in the excerption of the database related to the authentication process in Figure 6.4. The "Users" table is required to contain at least two columns, one for username and one for password. This expressed in the `server.xml` file, in the last but one line, and can be seen in the database figure. There must also be a table that contains one row for every valid role the user is assigned to. This table must contain at least two columns, username and rolename. We have called this table "User_roles". The last table seen in Figure 6.4 is the "Role" table. This table maintains the list of available roles in the application.

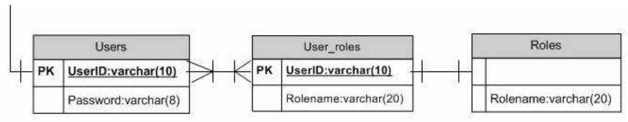


Figure 6.4: Database realm.

Alternative realms is memory realm, which is the Tomcat default realm (and which can be configured in `tomcat-users`) and LDAP[4].

In sum, web containers perform the the following steps to implement security of a web application.

[3]In computing, base64 is a data encoding scheme whereby binary-encoded data is converted to printable ASCII characters [44].

[4]Lightweight Directory Access Protocol.

- Determines whether the user has been authenticated when the protected web resources are accessed.
- If the user has not been authenticated yet, requests that the user provide security credentials by redirecting to the login page.
- Validates the users credentials against the security realm configured for the container.
- Determines whether the authenticated user is authorized to access web resources.

In our application, the session is invalidated (requirement RU4a)) when the user logs out from the system using the logout button (requirement RU4c)), or if the inlogged-user has not performed any action in the application in 15 minutes (requirement RU4d)). When the users session is invalidated, the user is taken back to the mainpage.

We could have created a JavaScript that would have ended the session when the user closes the web browser. But this requires that the popup-blocker is not turned on in the browser. Today, it is practically default that people have popup-blockers turned on in their web browsers. In addition, only a few web browsers support JavaScript. Since we then cannot guarantee that closing the web browser ends a users session, we don't implement this functionality either.

6.2.3 Password encryption

When the user enters his or her password, the password will be sent over the encrypted SSL communication lines to the application server. Here, the password is encrypted with the SHA1-algorithm using the Tomcat and MySQL API. The first code excerption shows how, when the realm is set to the database in `server.xml`, SHA is set to be the encryption algorithm.

```
Realm className="org.apache.catalina.realm.JDBCRealm" digest="SHA"
debug="99"
```

Next, in the JavaBean, the password is encrypted in the `insertUser()` method:

```
String sqlUpdateUsers = "insert into Users values
('"+login+"',SHA1('"+pwd+"'));"
```

If a user enters “password” as their password, the SHA1-algorithm will transform it into “5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8”, which is useless outside the scope of the application.

Since all passwords are stored encrypted in the database, they will not be readable if someone manages to hack into the database. This maintains confidentiality, and it satisfy three of our security guidelines, namely 2.1.4 “Use and reuse trusted components”, 2.1.5 “Defence in depth” and 2.1.6 “Secure the weakest link”.

When encrypting the password, we made use of a library (API) that has been tested by multiple users over a certain time period. We have not found any documentation indicating that the SHA1-algorithm should not be considered safe. To use the SHA1-algorithm is thus in accordance with the security guideline “Use and reuse trusted components”.

By encrypting of the password, we have employed the security guideline “Defence in depth”. Passwords are first transferred safely by the use of SSL from the client to the server, they are then encrypted, and stored in the database. This results in, that if anyone by any chance gets access to the password file in the database, the passwords there are cryptic and not useful to the hacker.

By encrypting the password, we have encrypted a weak link in our application. Therefore, we have also followed the security guideline “Secure the weakest link”.

6.2.4 No cache

In most web browsers “back” and “forward” buttons exist and allow users to go back or forward on a page. If the “back” button causes the web browser to display stale pages from their caches after the logout process, the application has a security problem.

When the “back” button is clicked, the browser on default does not request a page from the web server. Instead, the browser reloads the page from its cache. This problem is common across all technologies, and not only Java-based ones.

The caches are advantageous to static HTML pages or pages that are graphic- or image intensive. Web applications, on the other hand, are more data-oriented. As data in web applications is likely to change more frequently, it is also more important to display fresh data to end users than saving some response time by going to the cache and displaying stale or of out-of-date information.

The HTTP protocol has headers that will be useful in this setting [16]. “Expires” and “Cache-Control” headers offers the application server a mechanism for controlling the browsers and proxies[5] caches. The HTTP “Expires” header dictates the proxies' caches when the pages “freshness” will expire. The HTTP “Cache-Control”, which was new under the HTTP 1.1 Specification, contains attributes that instruct the browsers to prevent caching on any desired page in the web application. When the “back” button encounters such a page, the browser sends the HTTP request to the application server for a new copy of that page. The descriptions for necessary Cache-Control header's directives follow:

`no-cache`: forces the caches to obtain a new copy of the page from the origin server.

`no-store`: directs the caches not to store the page under any circumstances.

The directives to caching somewhat overlap, but combined they give good protection against unwanted caching [9]. For backward compatibility to HTTP 1.0, the `Pragma:no-cache` directive, which is equivalent to HTTP 1.1 `Cache-Control: no-cache` in HTTP 1.1, may be included in the headers response.

The following scriptlet is executed at the beginning of our JSP pages to prevent them from being cached at the browser.

```
<%
 response.setHeader("Cache-control","no-cache");
 response.setHeader("Cache-control","no-store");
 response.setDateHeader("Expires", 0);
 response.setHeader("Pragma","no-cache");
 %>
```

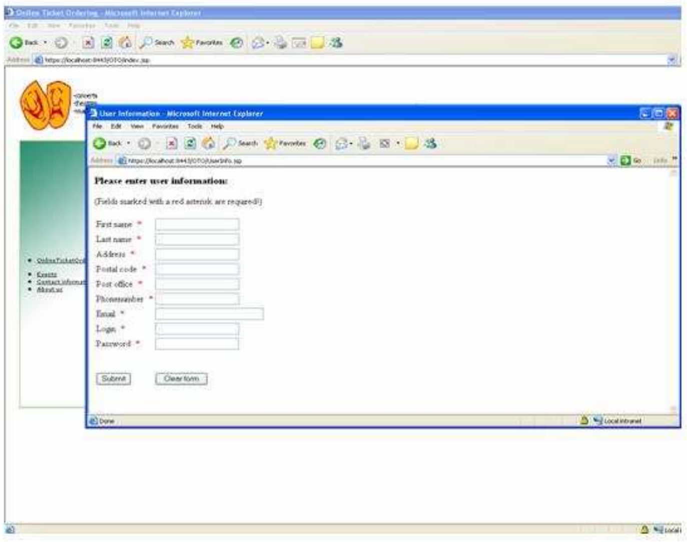
[5]Server placed between a user's machine and the Internet [44].

The `response.setDateHeader("Expires", 0);` prevents caching at the proxy server. The reason why we have employed these HTTP header directives is to fulfill requirement SR4, "Sensitive data should not be cached" and the security guideline 2.2.3 , "Limit cache usage". If the user is logged out of the system, and presses the "back" button in the web browser, he or she will be sent directly back to the login-page.

6.2.5 Preventing SQL injections

The SQL injection attack was presented in chapter 3, "Vulnerabilities, threats and countermeasures" with an example and countermeasures. As was mentioned, almost all platforms are vulnerable to this attack and any input field in the application that make up clauses of a database query is candidate for an SQL injection. As this threat was ranked among the top three threats, with a score of 12 out of 15 possible points in our risk analysis, we will be treating it quite thoroughly, and give details as to how we avoided it in our application.

In our application, if the user wants to order tickets, he or she must be logged in. To be able to log in, the user needs to be registered. Figure 6.5 shows the registration form.


Online Ticket Ordering - Microsoft Internet Explorer
User Information - Microsoft Internet Explorer
https://localhost:8443/OTO/index.jsp
https://localhost:8443/OTO/userInfo.jsp
Submit
Clear form
Done
Local intranet

Please enter user information:

(Fields marked with a red asterisk are required!)

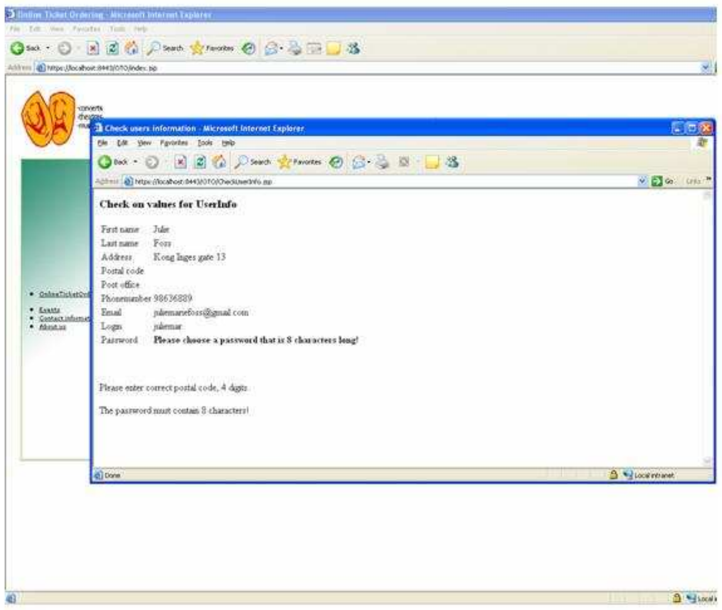
First name *
Last name *
Address *
Postal code *
Post office *
Phonenumber *
Email *
Login *
Password *

Figure 6.5: Screnn dump of how the registration form looks like in the web application.

When the user fills out this form the input is sent from the client to the server. At the server side, this input is validated by a JavaBean. All the input fields are tested to see if the input has appropriate form and size in accordance with security guideline 2.1.1, “Validate input”. The input is in addition validated on the server side according to the security guideline 2.2.1, “Validate input on the server side”.

After validation, it is shipped off to the database. The validation assures that the input will not cause a SQL injection or buffer overflow[6] attack.

As we have checked all input coming from the client before building the SQL query, we have thus used the JavaBean, or a part of the *controller*, as a choke point. Being able to direct all input validation through the JavaBean adds to the MVC-architecture, and it is in accordance with security guideline 2.1.3, “Keep it simple” witch states that security critical operations should go through a choke point.


Check users information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
https://localhost:8443/OTO/CheckUserInfo.jsp
Done
Local intranet

Check on values for UserInfo

```
First name   Julie
Last name    Foss
Address      Kong Inges gate 13
Postal code
Post office
Phonenumber 98636889
Email        juliemariefoss@gmail.com
Login        juliemar
Password     Please choose a password that is 8 characters long!


Please enter correct postal code, 4 digits

The password must contain 8 characters!
```

Figure 6.6: Screen dump that shows how the page looks like when a user has entered wrong input in the registration form.

In our application, if the information in the registration form is incorrect, the user gets feedback from the application on which fields that are filled in incorrectly. This is shown in Figure 6.6.

Stored procedures

Server side input validation is the most effective method of preventing SQL injection [21]. Another way to work around the SQL injection problem is by avoiding dynamic SQL queries. This can be achieved by using stored procedures, in Java “CallableStatement” and “PreparedStatement” [1] can be used. A stored procedure

[6]Buffer overflow attacks are not likely to occur on the Java-platform, since arrays and fields have strict regulations on what kind of input that is allowed. Although, one should be aware of, that some compilers does not check the code good enough, and therefore makes buffer overflow attacks possible to bring about.

is a group of precompiled SQL statements. The procedure accepts input as parameters so that a dynamic query is avoided. Although input is put into the precompiled query as it is, since the query itself is in a different format, it does not have the effect of changing the query as expected. By using stored procedures one may let the database handle the execution of the query instead of asking it to execute a query we have built.

Client side scripts

It is definitely possible to check user input on the client side, by using JavaScript. This approach has nothing to do with security though, because the input will need to be revalidated when reaching the server side. Although client side checking disallows the attacker to enter malicious data directly into the input fields, that alone is not enough to prevent an SQL injection. Client side scripts only check for input in the browser, and this does not guarantee that the information will remain the same till it reaches the server. There are tools that can capture the request going from the client and change it before sending it to the server. The attacker can also inject commands into the querystring variables which are not checked by the client side script [21]. According to security guideline 2.2.1, "Validate input on the server side" all data that passes the invisible security barrier needs to be validated when reaching the server side. Client side scripts may be used to enhance functionality, or help the user. Most of the time, invalid input in forms is not a malicious attacker, but rather a mistyping from an end user. By checking input on the client side, one may short down response time, so that a user don't have to wait for the validation procedure on the server side. This is the only advantage of client side checking, because all validation needs to be duplicated on the server side.

6.2.6 Preventing cross site scripting

Our system is a ticket ordering system. It would be an advantage if, when the user is looking at the number of tickets left, he or she would be able to refresh the page to see if the number of tickets is decreasing. In that case, the user might want to hurry to order.

To be able to use the refresh-button in the browser, we had to use include[7] instead of frames[8] in our applications index-file. When using "include", the risk of Cross Site Scripting (XSS) exist. Therefore, we have entered following code into our index-file to show how this can be avoided if using include.

```
<%
  String pagestr = request.getParameter("page");
  Vector pagelist = new Vector();
  pagelist.add("index.jsp");
  pagelist.add("Mainpage.jsp");
  pagelist.add("Top.jsp");
  pagelist.add("Login.jsp");
  pagelist.add("Logout.jsp");
  pagelist.add("About.jsp");
  pagelist.add("ContactInformation.jsp");
```

[7]For a definition of Server Side Include, see "Glossary" in Appendix A.
[8]For a definition of frames, see "Glossary" in Appendix A.

```
  pagelist.add("secure/SecureEvents.jsp");
  pagelist.add("Events.jsp");
  pagelist.add("SecEvents.jsp");

  if(pagelist.contains(pagestr)){

  }else{
       pagestr="Mainpage.jsp";
  }
  %>
```

By doing this, the browser can only open URLs that are allowed by the user, because the URL is validated by the code. After the user has logged in, we use frames.

What is a problem here, though, is that the URL is validated on the client-side. Throughout this report we have stressed that all validation should occur on the server side. But users can't enter any URL's into the system, since all of the input is validated. Only administrators and super users can do that. Therefore, this validation of the URL is a way to protect the user on the client-side from malicious actions conducted by administrators and/or super users with cruel intentions.

6.2.7 Error messages

If a user tries to log onto the system with false user credentials, we have implemented an error message that is returned to the client. This is done instead of revealing system information that might be exploited by malicious hackers.

Figure 6.7 shows the error message presented the user when the login functionality fails. Using this type of error messages is not only according to system requirement EHR2, in chapter 4, "Requirements engineering", but also according to the security guideline 2.1.2 "Fail securely" in chapter 2, "Security guidelines".

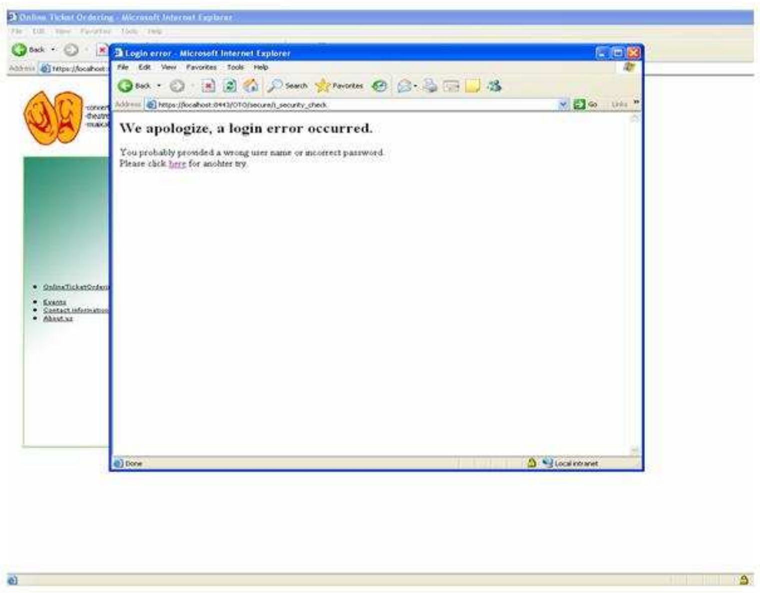
Multithreading

The JavaBean we created uses the synchronized-block in all methods that handles requests from the client side. The setEmail()-method shown underneath is an example of a method using the synchronized block. The synchronized block prevents problems with multithreading.

```
public synchronized void setEmail(String newEmail)
   {
      if(newEmail != null)
         email = newEmail.trim();
      else
         email = " ";
   }
```

When using JavaBeans, the business logic becomes more structured and maintainable in the application. But it is important to be aware of that beans shared between multiple pages, must be thread-safe. This is why we make use of the synchronized block.

Multithreading is an issue only for beans in the session and the application scopes. Beans in the page and request scope can be executed by more than one thread,


Online Ticket Ordering - Microsoft Internet Explorer
Login error - Microsoft Internet Explorer
https://localhost:8443/OTO/secure/j_security_check
OnlineTicketOrdering
Events
Contact information
About us
Done
Local intranet

We apologize, a login error occurred.

You probably provided a wrong user name or incorrect password.
Please click here for anohter try.

Figure 6.7: Screen dump of the error message that occurs when a user enters wrong login credentials to the system.

initiated by requests from the same client. This may happen if the user opens multiple browsers, repeatedly clicks a “Submit button” in a form, or if the application uses frames to request multiple JSP pages at the same time. Application scope beans are shared by all application users, hence it is very likely that more than one thread is using an application scope bean.

Java provides mechanisms for dealing with concurrent access to resources, such as synchronized block and thread notification methods [2].

By using the principle of multithreading, *integrity* will be maintained. In addition, the system requirement system requirement SR2, on simultaneous users is be taken into account.

6.3 Summary

This chapter has introduced an implementation of the application designed earlier in this report. It has also shown how some of the security issues pointed out in previous chapters has been solved, both according to security guidelines and system specification.

In this report there has been described the development of a web application, where the focus was aimed at security throughout the development process. The following chapter will contain a discussion of the results we came up with, and a conclusion that presents what we have achieved.

Chapter 7

Discussion

There are two important trends to note concerning web applications. First of all, they represent an increasing share of the total number of applications developed. Second, they get more available every day. Over the last few years, more security critical operations have been included in them as well. For instance, online shops that allows you to pay by credit card are widespread.

Security is an important topic, not only to web applications, but to all kinds of software. The reason why software needs to focus on security is that it tend to contain sensitive information that should not be compromized. The amount of threats relevant to web applications puts them in a special position. The fact that they are open to attacks potentially launched by anyone connected to the Internet, implies that a heavy load of threat countermeasures needs to be employed in order to protect vulnerable assets. Web applications do require new security measures, as traditional external ones, like firewalls and intrusion detection systems (IDS), are no longer sufficient.

It is important to understand that security is not something that may be added to a system after a complete development process. This is practically infeasible, as is adding new functionality at the end of a software development project.

Some people argue that being security conscious all the way through the development process is costly as well, and it does require some extra time and effort, but it saves you the surprise at the end of things. You will need security, and the best way to make sure that you are doing a good job protecting assets, is making sure you pay attention to potential pitfalls along the way.

We started off this report by providing security guidelines for software development. Guidelines are useful to avoid well-known threats, but they also encourage a clean design that will result in a surveyable system, so they may in fact help prevent future threats as well. Our guidlines included a separate part concerning web application security specifically. We also produced a security guideline focusing on how to avoid attacks in a web application. This guideline was developed for The Centre for Information Security (SIS), and will be published on their web page. The development of guidelines represented the first sub goal of our project, as they formed a basis on which the rest of the report was built, and an overall goal was to be security concious throughout the process.

Another goal was to design a web application with regards to various requirements, identify what vulnerabilities and threats the application would be exposed to, and perform a risk analysis ranking threats identified. Through the entire development process we were to *take our security guidelines into consideration*, and present specific technology solutions at the end of the implementation phase.

In this discussion, we would like to shed light on how it has been developing a system using our proposed security guidelines, and focusing on making a system secure through the entire development process. We will be looking back at our security guidelines, and present some examples of how they have translated to specific design and implementation details. We will also go through the development phases, and identify what was done with regard to security in each of them.

Developing with regard to security

Software systems, and especially those accessible via the Internet, are continuously exposed to threats, both from the inside as well as the outside. We have made the assumption that once a user has been authenticated, his or her intentions are good, and we have thus set the focus on external attacks. Where it was relevant we did present measures on how to avoid internal attacks as well. Most of the time, internal attacks are not preventable by measures one may implement. Rather they must be prevented by user awareness. Logging might be used detecting inside attacks. They may be hard to prevent during a development phase, because an inside attack might invole that a crook is part of the development team. At the end of a development process, checking the code for hidden backdoors[1], is a possible countermeasure to circumvent the work of a crook.

Security guidelines, general

We have come to use the majority of our security guidelines throughout the development process, and their relation to various design decisions and implementation details have been stated throughout previous chapters. Here, some examples showing how the security guidelines have mapped to various parts of the development process will be repeated. The *validate input* guideline is important, validating properly helps avoiding SQL injection attacks. It is taken care of in our JavaBean. The second guideline concerns *proper handling of failure*. One needs to design a system that has fault tolerance, as there is a 100% guarantee that at some point, the system will fail. The question is how you deal with it. The system should not reveal critical information under any circumstances as this might be exploited to launch an attack on the basis of information disclosure. Our system has a default error message that is sent to the user in case of failure during the login process. The guideline that says the system should be kept *as simple as possible* is realized with the Model-View-Controller (MVC) architecture. The choke point handling requests is part of the controller in our Model 2 version of the MVC architecture. Trusted components have also been used, for instance, our password encryption algorithm is the approved SHA-1, embedded in the MySQL database implementation. We have strived to practice *defense in depth*, which is illustrated by the password encryption. Should the database be compromised, the passwords will be useless as they are encrypted. The password encryption is also an illustration of the guideline instructing

[1]Also called a trapdoor. An undocumented way of gaining access to a program, online service or an entire computer system. The backdoor is written by the programmer who creates the code for the program. It is often only known by the programmer. A backdoor is a potential security risk [44].

that one should *protect the weakest link*. The next guideline concerns *privileges* of users, and states that they should be kept at a minimum. We have designed three different system roles to fulfill this requirement, and they are each provided the least privileges possible. *Compartmentalization* is also realized by the use of the MVC architecture. *Awareness* is not something that one may implement to a system, but rather to potential users.

Security guidelines, web application specific

One of the main lessons one should remember when developing web applications, is not to trust the client side. This is expressed in our security guideline *validate input on the server side*. This is necessary as it is only the server side that we are in control of. One may do validation on the client side as well, but only with the intent to speed up the system. There is nothing to gain on doing this in the sense of security. *Encrypt communication lines* is another important guideline, it is realized with the use of Secure Socket Layer (SSL). This will avoid for instance relese of message content, and traffic analysis. The last guideline, *limit cache usage*, is treated in the implementation chapter, where it is explicitly shown how to avoid caching of sensitive information.

Requirements engineering phase

We started focusing on security already in the requirements engineering phase. All requirements related to security were identified and emphasized. Using misuse cases, we modelled where a potential crook illegitimately might try to compromise the system, for instance by elevation of privileges, phishing or SQL injections. In addition, misuse cases introduced a link from each threat to possible countermeasures. We believe that using misuse cases is effective, because it awakens developers, it is a visual tool relating functionality to threats, and it identifies countermeasures. We were in fact surprised that so many threats were relevant to our application. Looking back, we are pleased we knew this early in the process. In addition to the advantages mentioned, the threats identified by our misuse cases served as a basis for our risk analysis. The fact that we chose to adapt the waterfall model to include a risk analyis at the end of the requirements engineering phase, proved to be a good choice. It enabled us to proceed in the development process with knowledge of what threats were most urgent. It was also important that the threats were provided scores on different elements. This was important to understand the possible impact of a threat, as some might be easily reproduced and relaunched, wheras others might have a great damage potential. Making sure that various elements were included in the risk analysis gave a good basis to rank the threats on. Conducting the risk analysis as early on in the process, made it possible to design to avoid trouble.

Design phase

Identifying problem areas in the first phase, became a stepping-stone into the next, the design phase. It gave us a head start since we already had thought about where security might be threatened, and what countermeasures we could use to prevent potential attacks.

In the design phase, the choice of technologies has been in accordance with the guideline “Use and reuse trusted components”. Making use of components developed by trusted third parties, tested by multiple users, and with no documented indication as to why it should not be considered safe, turned out to be advantageous.

The architecture we chose was the MVC. This architecture is well tested, and has proven to be appropriate when developing applications on the Java platform. This model also applies to several of our security guidelines.

Implementation phase

In the implementation phase we realized security functionality planned for in previous development phases. Technology specific solutions were provided, and high ranked threats from the requirements engineering phase were prioritized. Reaching this phase, the value of our previous work on planning for a secure system, became evident. With a clean cut design, and fuctionality that was well planned for, the implementation was straighforward. We focused on illustrating security critical parts of the system, such as login functionality, cache, and password protection in the database.

Summary

Such a thing as a 100 % secure computer system does not, and probably will never exist, but by being security conscious throughout the development process, one may reduce system vulnerabilities. Focusing on a security oriented development, many pitfalls may be avoided.

We have performed one iteration of the waterfall model, reaching the implementation phase. Throughout the development process, we have worked by our own guidelines. We believe that this has helped us to stay focused and security conscious along the way. It also contributed to making the application that we developed less vulnerable to threats. The security guidelines have been used to support our design and implementation decisions. It has been a valuable reference. We have learned that there are plenty of threats and vulnerabilities that needs to be handled, even when developing a small system. The value of ranking these threats, so that one may take care of the most urgent threats, carefully has become evident, and also the fact that one have to decide on an acceptable level of risk. Zero risk is neither necessary nor practical, and it comes down to the value of the assets in the system. The risk rating has been referenced throughout the report, and it has been of great help. It is possible that if the system were to be scaled, and more iterations were to be conducted, one would have to reevaluate the threat ranking on account of an incrase in system complexity.

There are many small web applications today, by example in-house applications. They may represent great security problems, especially as they are often part of a bigger environment, and when compromized they may for instance provide illegitimate acces to connected units. The system developed in this project is a small one. We believe that the principals that one should keep in mind when developing systems are independent of the size of the system, and that bigger applications are exposed to exactly the same threats as smaller ones. This report might thus be a valuable resource even if you are developing a much bigger system. The only difference would be the complexity. The techniques are still the same. It would not have been feasible within the limits of this project, to design for, and develop a gigantic web application. Nor would it have illustrated different security principals.

Web applications exist in a security critical environment. There is a difference between systems that are security critical, like Internet banks, in which faults are unacceptable, and other web applications, where security might be important but not crucial. Either way, these systems can not rely on traditional external measures. This report has underlined the positive effect of being security concious when developing a system. Security is something that should be planned for, and kept in focus throughout the development process.

Chapter 8

Conclusion and further work

The application described in this report has been described with the focus directed at security.

Focusing on security throughout the development process has been a challenge. But having the security guidelines as reference points through the development process, gave us a clear image of how to build the application with regard to identified vulnerabilities. After we had identified relevant threats, and suggested countermeasures, we could begin designing solutions. This made it relatively easy to implement countermeasures to threats that had proven themselves dominant through the development process. We believe the security measures we implemented on the Java platform, can be used as a suggestion to how others can solve the same problems. We found the security guidelines we suggested very useful throughout the development of our application, and we believe these guidelines can be of good help to others that are to develop web applications.

When working on this report, we experienced the difficulty of finding information on how to create secure web applications, in addition to descriptions of vulnerabilities, threats, and countermeasures. We therefore believe that our report will provide information that will come in handy for those who are to develop a web application, both smaller ones, like ours, and large ones. The fundamental principles for security are the same. As our guidelines are platform independent, they could be applied even if an application were to be developed using a different programming language.

8.1 Further work

Further work that can be done after this report, is to complete the implementation of the web application, and scale it up according to our design. This might imply that new iterations of the waterfall model have to be conducted, and this may for instance result in a update of the risk analysis. Then it could be tested, and maintained according to the waterfall model.

Appendix A

Glossary

Here are explanation to words that might come in handy reading the report. The definitions are mainly from Webopedia [44], while some are from Wikipedia [46].

ACID. ACID stands for atomicity, consistency, isolation and durability. These four properties are considered to be key transaction processing features of database management systems.

API. Short for Application Program Interface. A set of routines provided in libraries that extends a language's functionality.

Asset. An asset is a resource of value such as the data in a database or on the file system, or a system resource.

Atomicity. Atomicity refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them are.

Attack. An attack is an action taken to harm an asset.

Auditing. Effective auditing and logging is the key to non-repudiation. Non-repudiation guarantees that a user cannot deny performing an operation or initiating a transaction.

Authentication. The process of uniquely identifying the clients of your applications and services. In security parlance authenticated clients are referred to as principals.

Authorization. This is the process that governs the resources and operations that authenticated clients are permitted to access.

Audit trails. An audit trail is a series of records of computer events, about an operating system, an application, or user activities.

Availability. Means that the system remains available to legitimate users.

Cache. From an Internet browsers point of view, when something is cached, the files and graphics is saved locally from web sites you have previously visited.

Confidentiality. Also referred to as privacy, the process of making sure that date remains private and confidential, and that it cannot be viewed by unauthorized users or eavesdroppers. Encryption is frequently used to enforce confidentiality.

Consistency. Consistency refers to the database being in a legal state when the transaction begins and when it ends. They are the key transaction processing features of a database management system, or DBMS.

Cookies. A cookie is a small bit of information that a web server can store temporarily within your browser. A common use of cookies is to store information entered into a form so it does not need to be reentered on subsequent visits.

Countermeasures. A safeguard that addresses a threat and mitigates a risk. In this report it is important to notice that the main goal is to prevent attacks on a security level.

Data hashing. A mathematical summary that can be used to provide message integrity.

Digital signature. An authentication mechanism that enables the creator of a message to attach a code that acts as a signature.

DNS. Domain Name Server. Is an Internet service that translates domain names into IP addresses. Because domain names are alphabetic, they're easier to remember. The Internet however, is really based on IP addresses. Every time you use a domain name, therefore, a DNS service must translate the name into the corresponding IP address.

Durability. Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone.

Encryption. Application of a specific algorithm to plaintext data so as to alter the appearance of the data making it incomprehensible to those who are not authorized to see the information.

Form fields. Form fields are sections in an outline where to enter input data.

Frames. A feature supported by most modern web browsers that enables the web author to divide the browser display area into two or more sections (frames). The contents of each frame are taken from a different web page. Frames provide great flexibility in designing web pages, but many designers avoid them because they are supported unevenly by current browsers.

Handshake protocol. The handshake protocol allows the server and client to authenticate each other and to negotiate an encryption and MAC[1] algorithm and encryption keys to be used to protect data sent in an SSL record.

Header. Header refers to supplemental data placed at the beginning of a block of data being stored or transmitted, which contain information for the handling of the data block.

Hidden fields. Hidden fields are form fields that not are shown to the user through the web browser, but contains information from the server.

HTML. The Hyper Text Markup Language (HTML) is the authoring language used to create documents on the WWW. It defines the structure and layout of a web document by using a variety of tags and attributes.

[1]Message Authentication Code is a short piece of information used to authenticate a message.

HTTP. The Hyper Text Transfer Protocol (HTTP) defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands.

Integrity. It is the guarantee that data is protected from accidental or deliberate modification. Like privacy, integrity is a key concern for data passed over networks. Integrity for data in transit is typically provided by hashing techniques and message authentication codes (MAC).

Internet. The Internet is a massive network of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the Internet. Information that travels over the Internet does so via a variety of languages known as protocols.

Invisible security barrier. The invisible security barrier is a conceptual barrier between the server and the client. Everything that has crossed this barrier from the client to the server, may be unsafe. Never trust data that have passed the security barrier, no matter how deep in the HTML it is hidden.

IPSec. IPSec is an extended IP protocol which enables secure data transfer. It provides services similar to SSL/TLS, however. IPSec can be used for creation of encrypted tunnels between networks (Virtual Private Network (VPN)) so called tunnel mode, or for encryption of traffic between two hosts so called transport mode.

Isolation. Isolation refers to the ability of the application to make operations in a transaction appear isolated from all other operations.

MAC. A Message Authentication Code (MAC) is a one-way hash computed from a message and some secret data. Its purpose is to detect if the message has been altered.

OpenSSL. OpenSSL is an open source implementation of the SSL and TLS protocols. The core library (written in the C programming language) implements the basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

Principle of least privilege. The principle of least privilege requires that a user be given no more privilege than necessary to perform a job. Ensuring least privilege requires identifying what the user's job is, determining the minimum set of privileges required to perform that job, and restricting the user to a domain with those privileges and nothing more. By denying to subjects transactions that are not necessary for the performance of their duties, those denied privileges cannot be used to circumvent the organizational security policy.

Query strings. Query strings are text strings containing for instance SQL queries.

Server Side Include. Abbreviated SSI. A type of HTML comment that directs the web server to dynamically generate data for the web page whenever it is requested. The simplest command is #include, which inserts the contents of another file. This is especially useful for ensuring that boilerplate components, such as headers and footers, are the same on all pages throughout a web site. To change a boilerplate element, you need only modify the include file, instead of updating every individual web page.

Sessions. The period of time a user interfaces with an application. The user session begins when the user accesses the application and ends when the user quits the application.

SSL. The Secure Socket Layer (SSL) is a commonly-used protocol for managing the security of a message transmission on the Internet. SSL uses a public-and-private key encryption system, which also includes the use of a digital certificate.

Strong authentication. Strong authentication is authentication in which the identities of networked users, clients and servers are verified without transmitting passwords over the network.

Strong encryption. Strong encryption are plaintext encrypted with ciphers that are essentially unbreakable without the decryption keys.

SQL. Standard Query Language is a standardized query language for requesting information from a database.

TCP/IP. The Transmission Control Protocol/Internet Protocol (TCP/IP) are protocols that are the basis for transmitting and routing data packets on the Internet. The Internet Protocol is the one thing that all current Internet sites have in common.

Threat. A threat is a potential occurrence, malicious or otherwise, that may harm an asset.

URL. Uniform Resource Locator. Specifies the location of resources. It is the global address of documents and other resources on the WWW. The first part of the address indicates what protocol to use, and the second part specifies the IP address or the domain name where the resource is located.

Vulnerability. A weakness that make a threat possible.

World Wide Web. The World Wide Web, or simply web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet. The Web uses the HTTP protocol, only one of the languages spoken over the Internet, to transmit data.

XML. Extensible Markup Language is a meta language used to define data formats. It was originally designed to meet the challenges of large-scale electronic publishing, but XML is also playing an increasingly important role in the exchange of a wide variety of data on the web and elsewhere. Modern systems makes often use of XML when storing data, when this is advantageous for inter-operability with other systems and sources.

Appendix B

Use cases

This is an introduction to misuse cases. To understand the concept and functionality of misuse cases, we start off presenting use cases. Information about use cases is from [7] and [39]. Facts on misuse cases comes from [28] and [29].

B.1 Use cases

Use cases are helpful tools for the elicitation of, communication about and documentation of requirements [28]. This section contains a short introduction to use cases. The use cases are described to familiarize the reader with this tool and hopefully provide a understanding of misuse cases as well. They will be introduced later on in this chapter.

The standard employed in the use cases is the Unified Modeling Language (UML) [7]. Notations is explained in the succeeding sections.

We split the use case templates into two groups. These are:

1. **Use case diagrams** - Use case diagrams are sketches of a scenario between an actor[1] and a system.

2. **Textual use cases** - Textual use cases are a detailed description of every step in the scenario which an actor must go through to achieve a wanted goal. We will not go into these here.

It is important to notice that use cases only describe what a system is doing - not *how* the system is doing it.

B.1.1 Use case diagrams

Figure B.1 depicts a use case diagram.

- A *scenario* is a sequence of steps describing an interaction between a user and a system.

- The person is an *actor*. An actor is a role that a user plays with respect to the system.

[1]An actor is a role that a user plays with respect to the system.

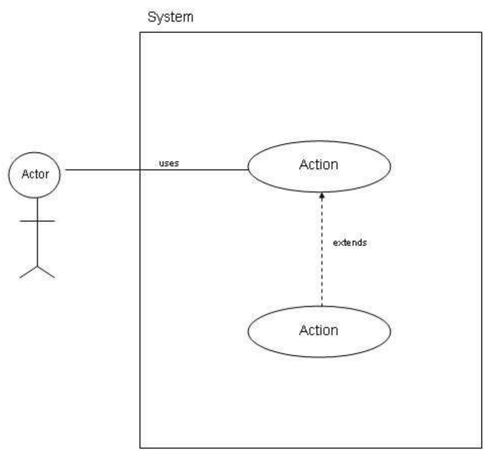


Figure B.1: A use case is a set of scenarios tied together by a common user goal.

- The *square* illustrates the system where all of the action takes place.
- The *action* explains the specific plot the actor is doing to the system.

There are four types of relationships in use cases. These are:

uses This is used when describing a link between an actor and an action. Default notation.

include This is used when repeating oneself in two or more separate use cases. One does this to avoid repetition.

generalization This is used when describing a variation on normal behavior. One does this to describe the incident casually.

extends This is used when describing a variation on normal behavior. One does this when a more controlled form is wanted, declaring extension points in the base use case.

B.2 Misuse cases

A misuse case is the inverse of a use case. It describes a function that the system should not allow. While use cases represents *positive* actions in a system, misuse cases represents the *negative*. Misuse cases describe functions in a system that results in losses for an organization or some specific stakeholder, and they identify potential risks and misuse.

B.2.1 Misuse case diagrams

Figure B.2 depicts a misuse case. The diagram shows the positive use case *and* the negative misuse case.

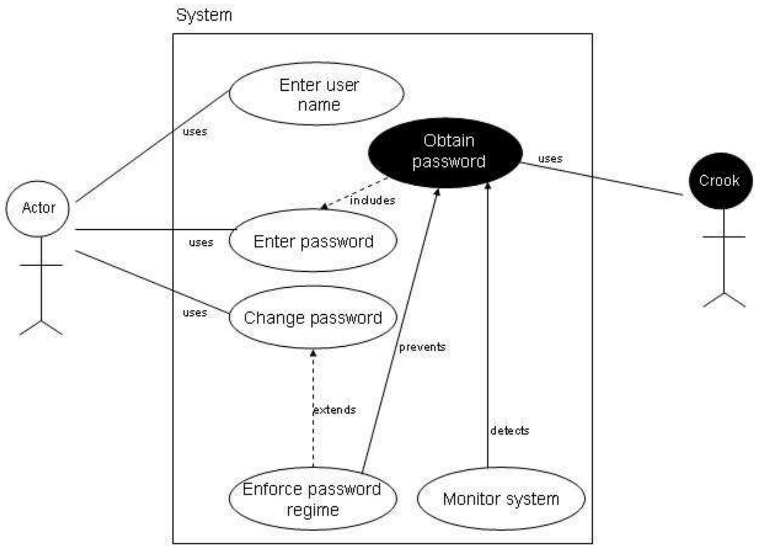


Figure B.2: A misuse case is the inverse of a use case. In this example, a crook get his hands on an actors password.

- A new actor is introduced in the misuse cases, namely the *mis-actor*. The mis-actor is the inverse of an actor, that is, an actor that one does not want the system to support, an actor who initiates misuse cases.
- A new action is also introduced, the *mis-action*. The mis-action describes the negative plot the mis-actor is doing to the system.

The relations in misuse cases are:

uses This is used when describing a link between an actor and an action. Default notation.

include This is used when repeating oneself in two or more separate use cases. One does this to avoid repetition.

generalization This is used when describing a variation on normal behavior. One does this to describe the incident casually.

extends This is used when describing a variation on normal behavior. One does this when a more controlled form is wanted, declaring extension points in the base use case.

detects This is used when one makes a link between an action that may detect misuse in the system and a mis-action.

prevents This is used when one makes a link between an action that may prevent misuse in the system and the mis-action it is preventing.

B.3 Summary

This chapter has given a brief introduction on use cases and misuse cases. The misuse cases will be used in the chapter on requirements engineering 4.

Appendix C

J2EE

Java 2 Enterprise Edition is a platform that is suitable for designing large, complex web applications that interacts with other programs and applications using Java technologies. The J2EE technology is a product from Sun Microsystems, and a technology widely adapted by the software industry today. JSP is a key part of the J2EE platform, and can take advantages of the many Java Enterprise libraries.

To get some understanding of the benefits one can achieve using J2EE, some of the technologies included in J2EE are briefly introduced in this section.

C.1 Struts

Struts is a project where the goal is to provide an open source framework for building Java web applications, created by The Apache Software Foundation.

The core of the Struts framework is a flexible control layer based on standard technologies like Java Servlets, JavaBeans, and XML. Struts encourages application architectures based on the Model 2 approach[1], a variation of the classic Model-View-Controller design paradigm.

Struts provides its own Controller component and integrates with other technologies to provide the Model and the View. For the Model, Struts can interact with standard data access technologies, like the Java Database Connectivity driver (JDBC)[2] and Enterprise Java Beans (EJB). For the View, Struts works well with JavaServer Pages (JSP).

The Struts framework provides the invisible underpinnings every professional web application needs to survive. Struts helps you create an extensible development environment for your application, based on published standards and proven design patterns. For small applications, Struts is not recommended due to a lot of overhead [37].

C.2 Enterprise Java Beans (EJB)

Enterprise JavaBeans (EJB) is a Java API developed by Sun Microsystems that defines a component architecture for multi-tier client/server systems.

[1]Described in chapter5, "Design".
[2]Described in chapter 5 "Design".

EJB systems allow developers to focus on the actual business architecture of the model, rather than worry about endless amounts of programming and coding needed to connect all the working parts. This task is left to EJB server vendors. Developers just design (or purchase) the needed EJB components and arrange them on the server.

Because EJB systems are written in Java, they are platform independent. Being object oriented, they can be implemented into existed systems with little or no recompiling and configuring [44].

The Struts framework contains readily implemented EJBs that developers can make use of.

C.3 Java Authentication and Authorization Service (JAAS)

The Java Authentication and Authorization Service (JAAS) is a set of APIs that can be used for two purposes. First of all, for authentication of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet. Second, for authorization of users, to determine reliably and securely who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet.

JAAS authentication is performed in a pluggable fashion. This permits Java applications to remain independent from underlying authentication technologies. New or updated technologies can be plugged in without requiring modifications to the application itself. JAAS authorization extends the existing Java security architecture that uses a security policy to specify what access rights are granted to executing code. That is, permissions are granted based on code characteristics, where the code comes from and whether it is digitally signed, and if so, by whom [13].

JAAS APIs can be used for both authentication and authorization, as follows [13]:

- For authenticating users to determine securely who is executing Java code, regardless of whether the code is a stand-alone Java technology-based application, an applet, an Enterprise JavaBean (EJB) component, or a servlet.
- For authorization of users to make sure they have the right permissions required to perform their actions.
- Pure Java technology implementation
- Pluggable Authentication Module (PAM) framework implementation for authenticating users
- Single sign-on support
- Flexible access control policy for user-based, group-based, and role-based authorization
- Sample authentication modules using Java Naming and Directory Interface (JNDI), UNIX Operating Environment, Windows NT, Kerberos or Keystore.

C.4 Common Object Request Broker Architecture (CORBA)

CORBA is a short for Common Object Request Broker Architecture, an architecture that enables pieces of programs to communicate with one another regardless of what programming language they were written in or what operating system they're running on. CORBA is developed by an industry consortium known as the Object Management Group (OMG) [44].

C.5 Remote Method Invocation (RMI)

RMI is a short for Remote Method Invocation, a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects. RMI is a relatively simple protocol, but unlike the more complex protocol CORBA, it works only with Java objects [44].

C.6 Java Naming and Directory Interface (JNDI)

Java Naming and Directory Interface (JNDI) enables Java platform-based applications to access multiple naming and directory services. Part of the Java Enterprise application programming interface (API) set, JNDI makes it possible for developers to create portable applications that are enabled for a number of different naming and directory services, including, file systems, directory services such as Lightweight Directory Access Protocol (LDAP), and distributed object systems such as CORBA, RMI, and EJB [44].

JNDI organizes and locates components within a distributed computing environment similarly to the way that card catalogs (and increasingly computer applications) organize and represent the locations of books within a library. A distributed application needs a mean of locating components in the same way that the library patron needs a mean of locating the book. Just rummaging around inside a library - or an application - is not an efficient way to find a particular object. JNDI makes it possible for application components to find each other. Because different naming and directory service providers can be seamlessly connected through the API, Java applications using it can be easily integrated into various environments and coexist with legacy applications.

C.7 eXtensible Markup Language (XML)

eXtensible Markup Language (XML) is a set of guidelines for constructing and ordering data in a document. XML contains three cooperating components. The XML parsers, the XML processors and the XML applications. The parsers parse the XML document into memory, the processors handle he parsed data, perhaps making transformation using a stylesheet. The parser and processor functionality can be bundled together in an inseparable component. The applications act in the data structure generated by the XML structure [2].

One can use XML to store data on a web server, and use JSP files to display the data. Using XML for data storage is getting more and more popular. To learn more about advantages of using XML, we refer to the article [45] from JavaWorld.

Java Cryptography Extension (JCE)

The Java Cryptography Extension (JCE) is a set of packages that provides a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects.

C.8 Java Secure Socket Extension (JSSE)

The Java Secure Socket Extension (JSSE) is a set of packages that enable secure Internet communications. It implements a Java technology version of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. It includes functionality for data encryption, server authentication, message integrity, and optional client authentication.

The SunJSSE implementation now supports a number of additional ciphersuites. They include ciphersuites using AES as symmetric cipher and ephemeral Diffie-Hellman with RSA authentication.

In addition to the simple X.509 based trustmanager previously available in the SunJSSE provider, it now supports a second, PKIX-compliant trust manager. It is implemented using the default CertPath PKIX implementation.

C.9 Summary

J2EE is more and more adapted within the software industry today. It is a well developed tool to use when creating an application from scratch, and when linking several programs together as an application.

Appendix D

Ticked requirements

Following are the requirements tables

- before login,
- regular users,
- administrator users,
- super users,
- the database,
- system requirements, and
- error handling.

The requirements implemented in our application, are ticked off.

Browse the page (BP)	**Ok?**
a) Site information, contact information and a list of upcoming events should be available.	V
Register a user account (RUA)	
RUA1. User states personal information.	
a) First name, last name, address, postal code, post office, phone number and email should be provided by the user.	V
*b) Input should be validated on the server side.	V
RUA2. User selects a username.	
a) The username should be unique.	V
RUA3. User selects a password.	
a) Password should contain eight characters.	V
b) Password should contain small and big characters.	
RUA4. User should read and agree to a personal information protection statement.	
**RUA5. User should read and state scrambled letters to the application.*	
Standard login (SL)	
*a) A new session should be created.	V
*b) Input should be validated on the server side.	V
*c) The user should be taken to another, secure page.	V
*d) Time delays should be used when a user provides wrong	
*e) Administrators and super users should log in from a hidden URL. credentials.	
SL1. User states username.	
SL2. User states password.	
*a) Password should never be transferred in clear text.	V
Forgot password feature (FP)	
FP1. User clicks the "Forgot password?"-link.	
*a) Ask the user to supply username and postal code.	
*b) Input should be validated on the server side.	
*c) Send mail to the users registered email address with link to a page for resetting the password and a one-time password.	

Table D.1: Ticked off requirements - Before login.

Regular users (RU)	**Ok?**
RU1. Edit personal information. a) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when logged in. b) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when processing an order.	
RU2. Change password. a) It should be possible to change the password when logged in.	
RU3. Order ticket(s). a) The user should be able to order tickets. b) The order should be limited by ten tickets. c) Event headline and number of tickets ordered should be confirmed by user. d) Address, postal code and post office should be confirmed by user. *e) Payment information should be entered. f) User repudiation should be avoided.	
RU4. Logout. a) The user session should be invalidated. b) A message confirming logout should be sent to the user. c) Pressing the "logout" button should be equivalent to using the browser to end the session. *d) The user session should be invalidated after a period of 15 minutes without user action.	 V V

Table D.2: Ticked off requirements - Regular users.

Administrator users (AU)	**Ok?**
AU1. Edit events. a) Should be able to add event(s). b) Should be able to change event properties. c) Should be able to remove event(s) by hiding them.	
AU2. Edit personal information. a) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when logged in.	
AU3. Change password. a) It should be possible to change the password when logged in.	
AU4. Logout. a) The user session should be invalidated. b) A message confirming logout should be sent to the user. c) Pressing the "logout" button should be equivalent to using the browser to end the session. *d) The user session should be invalidated after a period of 15 minutes without user action.	

Table D.3: Ticked off requirements - Administrator users.

Super users (SU)	**Ok?**
SU1. Edit events. a) Should be able to add event(s). b) Should be able to change event properties. c) Should be able to delete event(s).	
SU2. Edit personal information. a) It should be possible to edit first name, last name. address, postal code, post office, phone number and email when logged in.	
SU3. Change password. a) It should be possible to change the password when logged in.	
SU4. Should be able to register administrator user account.	
SU5. Should be able to edit administrator user account.	
SU6. Should be able to delete administrator user account.	
**SU7. Should be able to check the log.*	
SU8. Logout. a) The user session should be invalidated. b) A message confirming logout should be sent to the user. c) Pressing the “logout” button should be equivalent to using the browser to end the session. *d) The user session should be invalidated after a period of 15 minutes without user action.	

Table D.4: Ticked off requirements - Super users.

Database requirements (DBR)	**Ok?**
**DBR1. Passwords should be stored encrypted.*	V
DBR2. The database should be available 90% of the time.	
DBR3. The database should be able to cope with simultaneous access.	

Table D.5: Ticked off requirements - Database requirements.

System requirements (SR)	**Ok?**
SR1. Availability. a) The system needs to be available 90% of the time.	
SR2. Scalability. a) The system should be able to handle 100 events simultaneously.	
**SR3. Encryption.* a) As of the login session, all succeeding communication should be encrypted.	V
**SR4. Caching.* a) Sensitive data should not be cached.	V
**SR5. Input validation.* a) Headers, cookies, query strings, form fields and hidden fields needs to be validated on the server side.	
SR6. Log. a) The system should keep track of actions and transactions with a log.	

Table D.6: Ticked off requirements - System requirements.

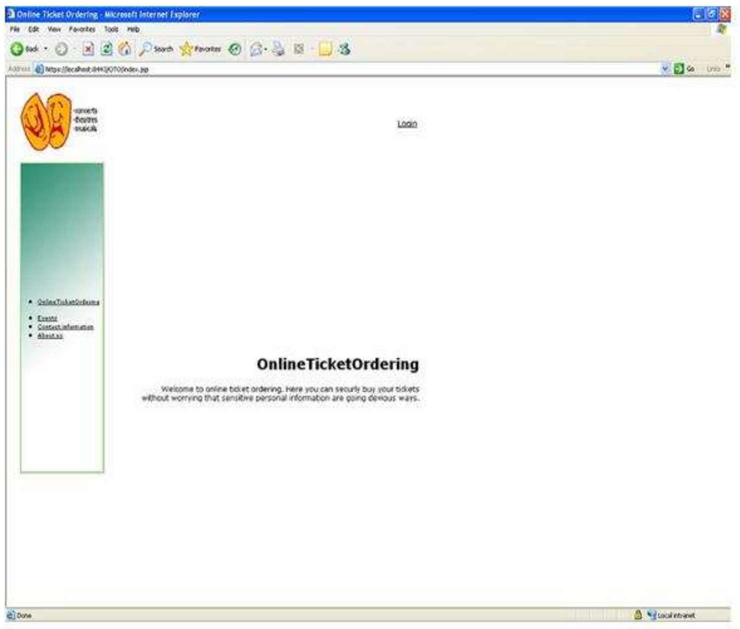
Error handling requirements (EHR)	**Ok?**
EHR1. System error should be reported to super user.	
**EHR2. System error should present information page to regular user.*	
EHR3. User error should result in a constructed error page being return to the user.	V

Table D.7: Ticked off requirements - Error handling requirements.

Appendix E

Screen dumps

This chapter contains screen dumps from our implemented application. Each figure has a short comment on what the screen dump shows. Those pages that have been relevant to our project, is presented and described in chapter 6, “Implementation”.

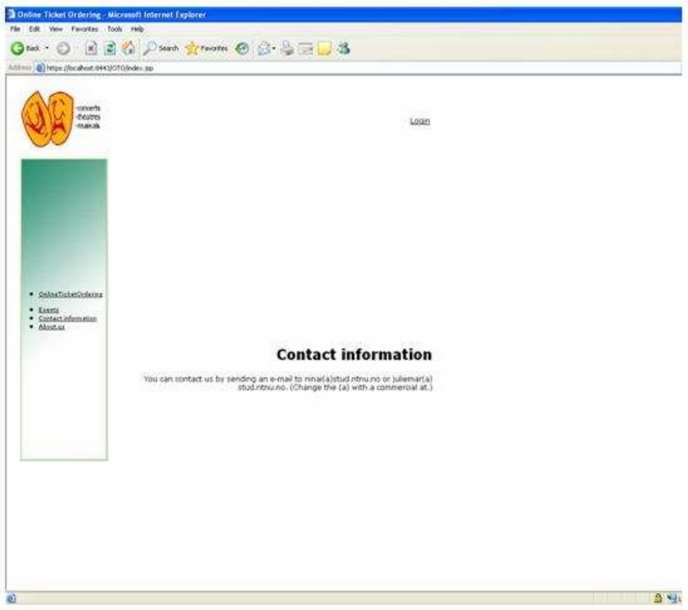


Online Ticket Ordering - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://localhost:8443/OTO/index.jsp
Go Links
concerts
theatres
musicals
Login
OnlineTicketOrdering
Events
Contact information
About us
Done
Local intranet

OnlineTicketOrdering

Welcome to online ticket ordering. Here you can securly buy your tickets without worrying that sensitive personal information are going devious ways.

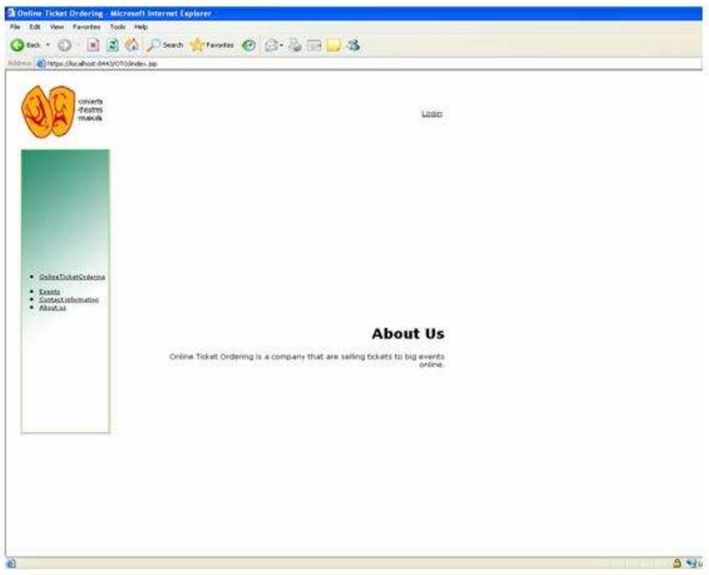
Figure E.1: OnlineTicketOrdering, mainpage.


Online Ticket Ordering - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://localhost:8443/OTO/index.jsp
-concerts
-theatres
-musicals
Login
OnlineTicketOrdering
Events
Contact information
About us

Contact information

You can contact us by sending an e-mail to nina(a)stud.ntnu.no or juliemar(a)stud.ntnu.no. (Change the (a) with a commercial at.)

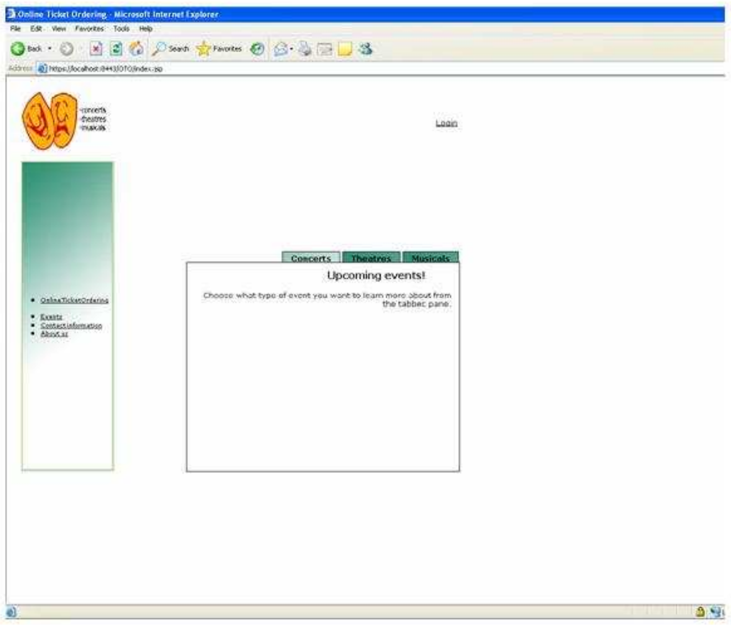
Figure E.2: OnlineTicketOrdering, contact information.


Online Ticket Ordering - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://localhost:8443/OTO/index.jsp
-concerts
-theatres
-musicals
Login
OnlineTicketOrdering
Events
Contact information
About us

About Us

Online Ticket Ordering is a company that are selling tickets to big events online.

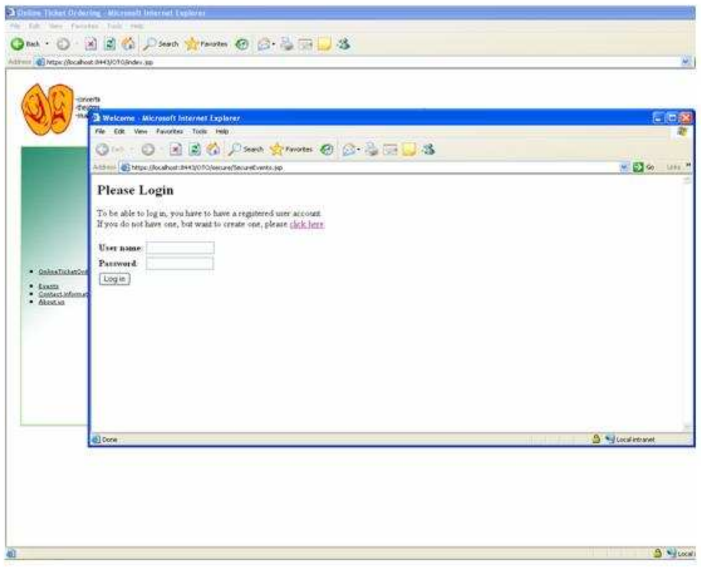
Figure E.3: OnlineTicketOrdering, about us.



Upcoming events!

Choose what type of event you want to learn more about from the tabbec pane.

Figure E.4: OnlineTicketOrdering, event information.

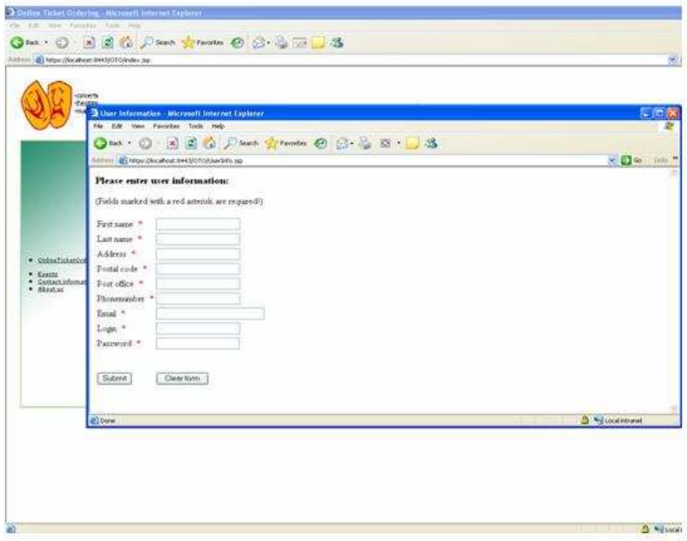

Online Ticket Ordering - Microsoft Internet Explorer
Welcome - Microsoft Internet Explorer
https://localhost:8443/OTO/secure/SecureEvents.jsp
Log in
Done
Local intranet

Please Login

To be able to log in, you have to have a registered user account.
If you do not have one, but want to create one, please click here.

User name:
Password:

Figure E.5: OnlineTicketOrdering, login.

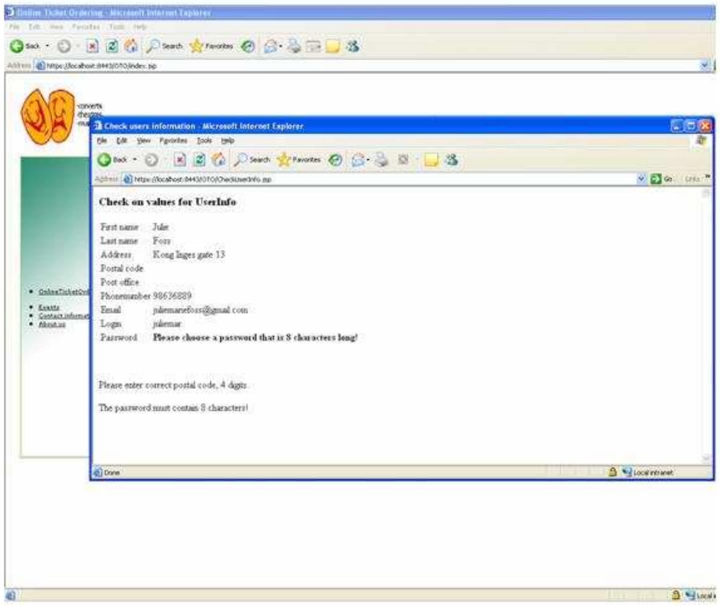

Online Ticket Ordering - Microsoft Internet Explorer
User Information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
https://localhost:8443/OTO/UserInfo.jsp
Submit
Clear form
Done

Please enter user information:

(Fields marked with a red asterisk are required!)

First name *	
Last name *	
Address *	
Postal code *	
Post office *	
Phonenumber *	
Email *	
Login *	
Password *	

Figure E.6: OnlineTicketOrdering, registration form.


Online Ticket Ordering - Microsoft Internet Explorer
Check users information - Microsoft Internet Explorer
File Edit View Favorites Tools Help
https://localhost:8443/OTO/CheckUserInfo.jsp
Done

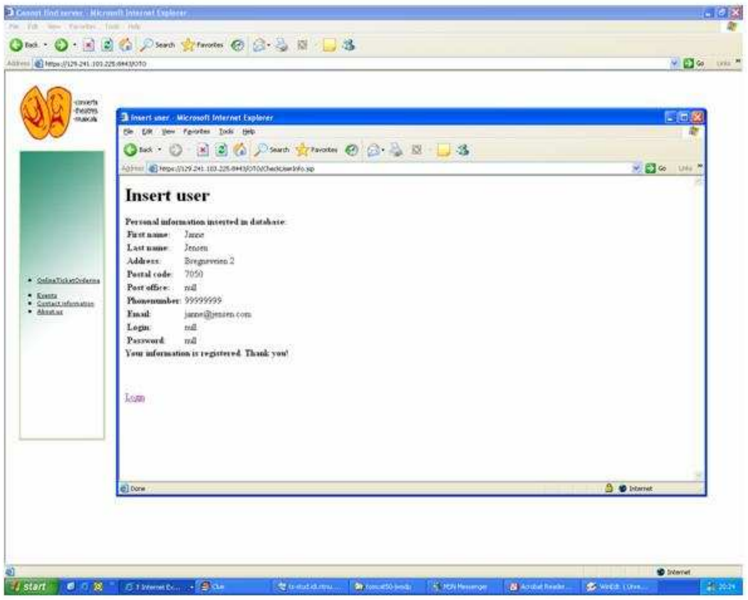
Check on values for UserInfo

First name	Julie
Last name	Foss
Address	Kong Inges gate 13
Postal code	
Post office	
Phonenumber	98636889
Email	juliemariefoss@gmail.com
Login	juliemar
Password	**Please choose a password that is 8 characters long!**

Please enter correct postal code, 4 digits.

The password must contain 8 characters!

Figure E.7: OnlineTicketOrdering, registration error page.


Insert user - Microsoft Internet Explorer
https://129.241.103.225:8443/OTO/CheckUserInfo.jsp
OnlineTicketOrdering
Events
Contact information
About us
Done
Internet

Insert user

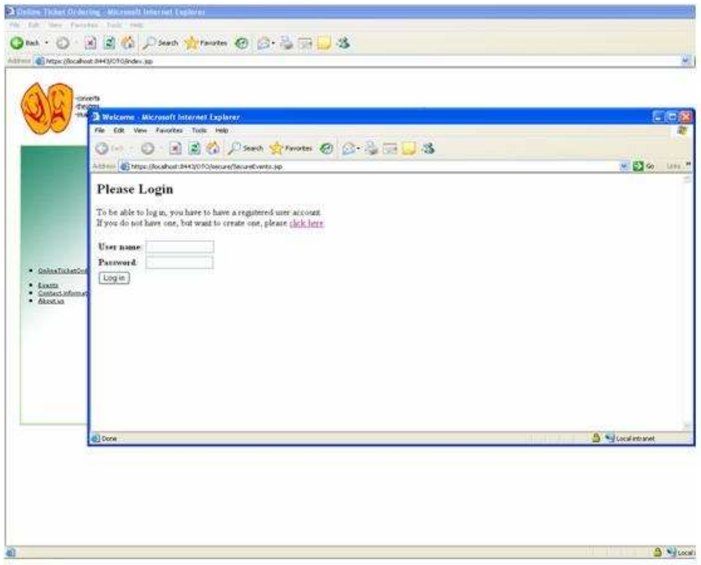
Personal information inserted in database:

First name:	Janne
Last name:	Jensen
Address:	Bregneveien 2
Postal code:	7050
Post office:	null
Phonenumber:	99999999
Email:	janne@jensen.com
Login:	null
Password:	null

Your information is registered. Thank you!

Login

Figure E.8: OnlineTicketOrdering, registration ok.

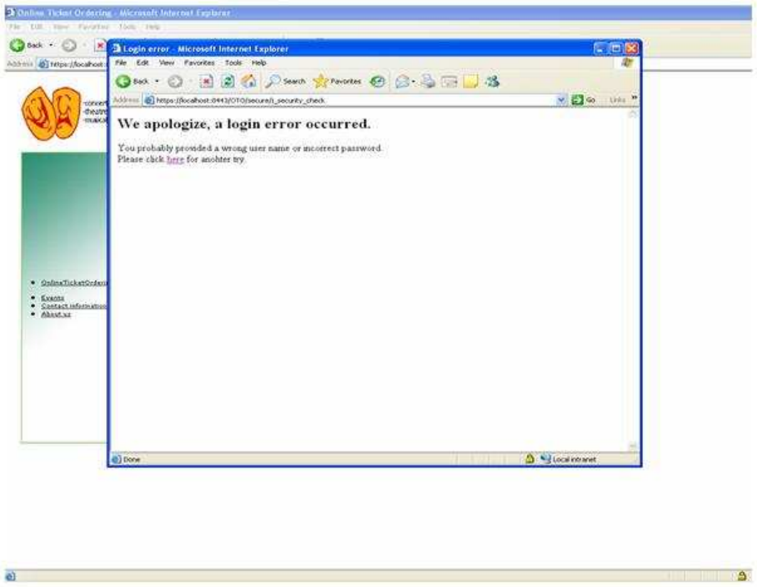

Welcome - Microsoft Internet Explorer
https://localhost:8443/OTO/secure/SecureEvents.jsp
Done
Local intranet

Please Login

To be able to log in, you have to have a registered user account.
If you do not have one, but want to create one, please click here.

User name:
Password:
Log in

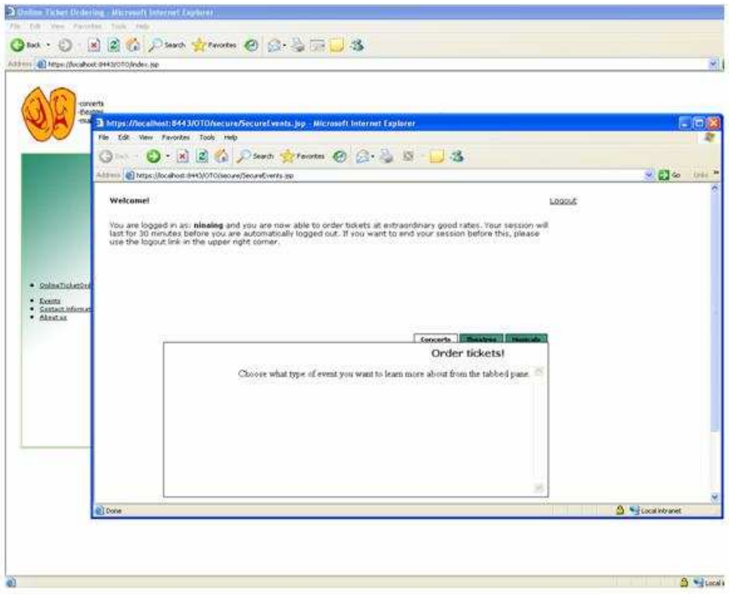
Figure E.9: OnlineTicketOrdering, login.


Online Ticket Ordering - Microsoft Internet Explorer
Login error - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://localhost:8443/OTO/secure/j_security_check
Go Links
Done
Local intranet

We apologize, a login error occurred.

You probably provided a wrong user name or incorrect password.
Please click here for anohter try.

Figure E.10: OnlineTicketOrdering, login error page.


Online Ticket Ordering - Microsoft Internet Explorer
https://localhost:8443/OTO/index.jsp
https://localhost:8443/OTO/secure/SecureEvents.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Search Favorites
Address https://localhost:8443/OTO/secure/SecureEvents.jsp
Go Links
-concerts
Concerts
Done
Local intranet

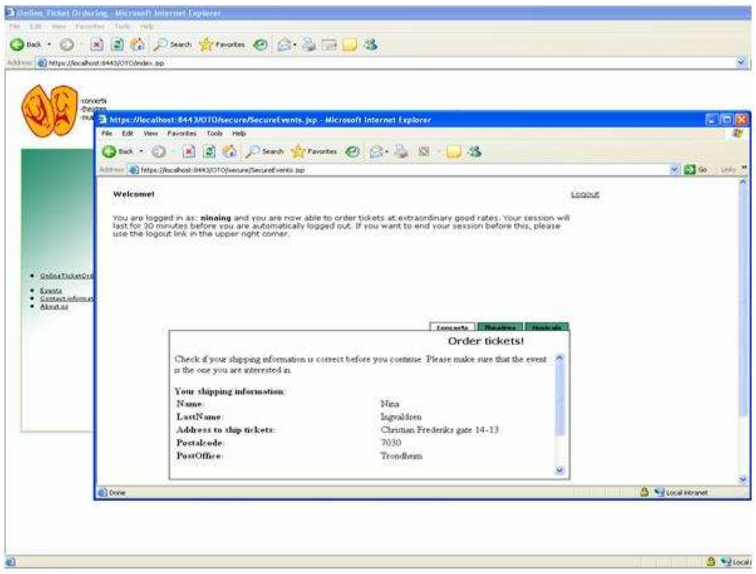
Welcome! Logout

You are logged in as: **ninaing** and you are now able to order tickets at extraordinary good rates. Your session will last for 30 minutes before you are automatically logged out. If you want to end your session before this, please use the logout link in the upper right corner.

Order tickets!

Choose what type of event you want to learn more about from the tabbed pane.

Figure E.11: OnlineTicketOrdering, logged in.


Online Ticket Ordering - Microsoft Internet Explorer
https://localhost:8443/OTO/index.jsp
https://localhost:8443/OTO/secure/SecureEvents.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
https://localhost:8443/OTO/secure/SecureEvents.jsp
Logout
Concerts
Theatres
Musicals
Done
Local intranet

Welcome!

You are logged in as: **ninaing** and you are now able to order tickets at extraordinary good rates. Your session will last for 30 minutes before you are automatically logged out. If you want to end your session before this, please use the logout link in the upper right corner.

Order tickets!

Check if your shipping information is correct before you continue. Please make sure that the event is the one you are interested in.

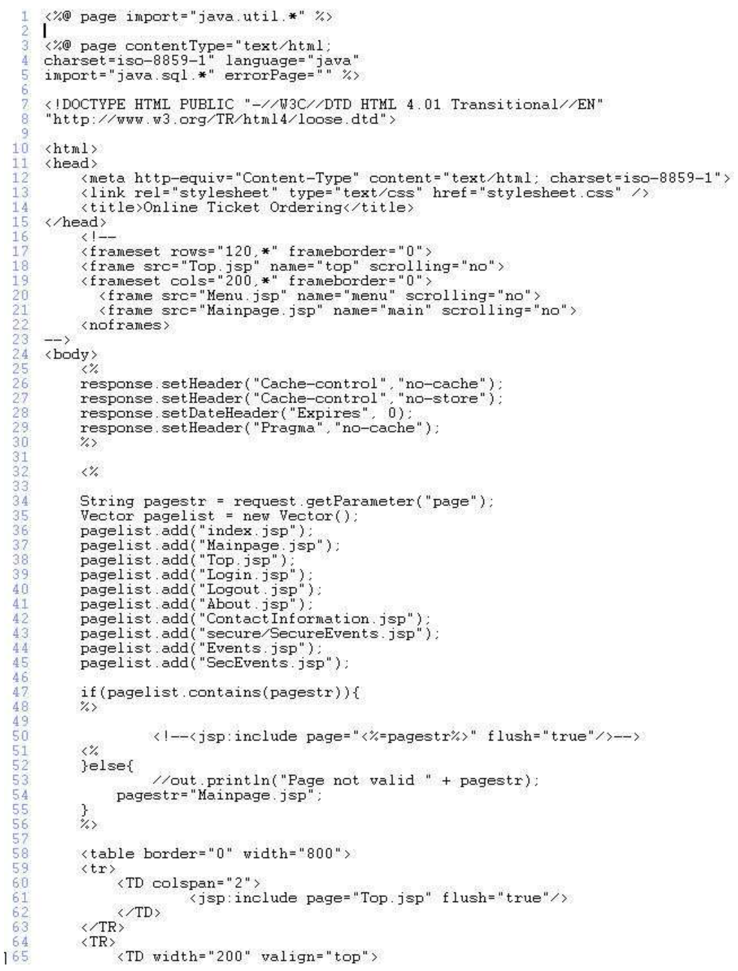
Your shipping information:

Name:	Nina
LastName:	Ingvaldsen
Address to ship tickets:	Christian Frederiks gate 14-13
Postalcode:	7030
PostOffice:	Trondheim

Figure E.12: OnlineTicketOrdering, order.

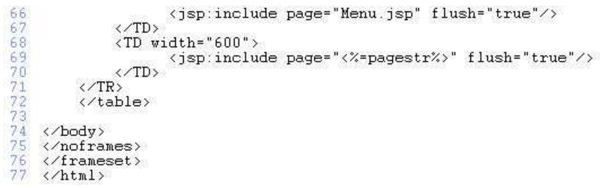
Appendix F

Code



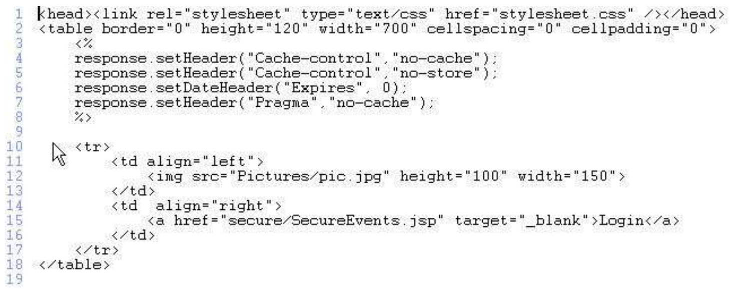
```
<%@ page import="java.util.*" %>

<%@ page contentType="text/html;
charset=iso-8859-1" language="java"
import="java.sql.*" errorPage="" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link rel="stylesheet" type="text/css" href="stylesheet.css" />
    <title>Online Ticket Ordering</title>
</head>
    <!--
    <frameset rows="120,*" frameborder="0">
    <frame src="Top.jsp" name="top" scrolling="no">
    <frameset cols="200,*" frameborder="0">
      <frame src="Menu.jsp" name="menu" scrolling="no">
      <frame src="Mainpage.jsp" name="main" scrolling="no">
    <noframes>
-->
<body>
    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    <%

    String pagestr = request.getParameter("page");
    Vector pagelist = new Vector();
    pagelist.add("index.jsp");
    pagelist.add("Mainpage.jsp");
    pagelist.add("Top.jsp");
    pagelist.add("Login.jsp");
    pagelist.add("Logout.jsp");
    pagelist.add("About.jsp");
    pagelist.add("ContactInformation.jsp");
    pagelist.add("secure/SecureEvents.jsp");
    pagelist.add("Events.jsp");
    pagelist.add("SecEvents.jsp");

    if(pagelist.contains(pagestr)){
    %>

            <!--<jsp:include page="<%=pagestr%>" flush="true"/>-->
    <%
    }else{
            //out.println("Page not valid " + pagestr);
        pagestr="Mainpage.jsp";
    }
    %>

    <table border="0" width="800">
    <tr>
        <TD colspan="2">
                <jsp:include page="Top.jsp" flush="true"/>
        </TD>
    </TR>
    <TR>
        <TD width="200" valign="top">
```

Figure F.1: Code - index.jsp, Part I.



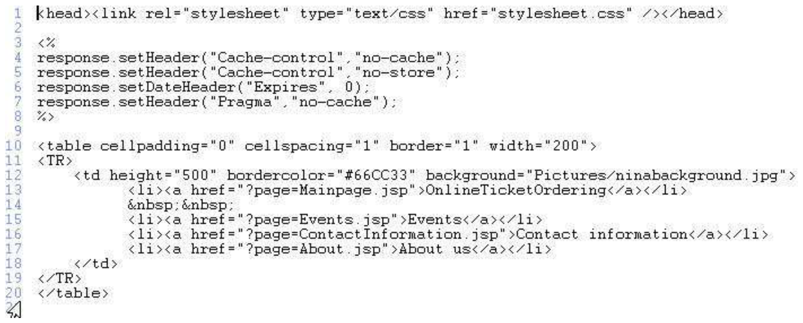
```
                <jsp:include page="Menu.jsp" flush="true"/>
            </TD>
            <TD width="600">
                <jsp:include page="<%=pagestr%>" flush="true"/>
            </TD>
        </TR>
        </table>

</body>
</noframes>
</frameset>
</html>
```

Figure F.2: Code - index.jsp, Part II.



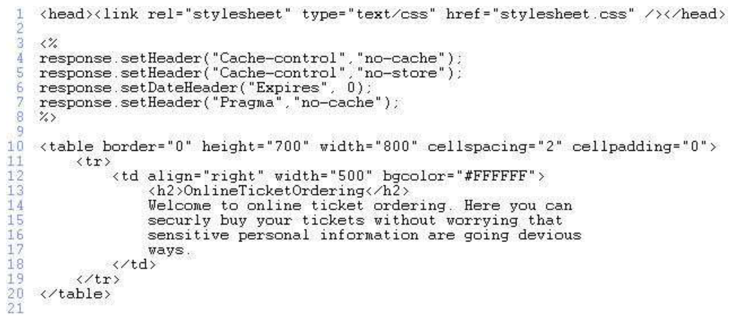
```
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<table border="0" height="120" width="700" cellspacing="0" cellpadding="0">
    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    <tr>
        <td align="left">
            <img src="Pictures/pic.jpg" height="100" width="150">
        </td>
        <td  align="right">
            <a href="secure/SecureEvents.jsp" target="_blank">Login</a>
        </td>
    </tr>
</table>
```

Figure F.3: Code - Top.jsp.



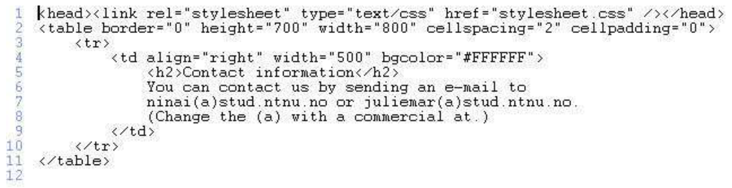
```
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>

<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>

<table cellpadding="0" cellspacing="1" border="1" width="200">
<TR>
    <td height="500" bordercolor="#66CC33" background="Pictures/ninabackground.jpg">
          <li><a href="?page=Mainpage.jsp">OnlineTicketOrdering</a></li>
          &nbsp;&nbsp;
          <li><a href="?page=Events.jsp">Events</a></li>
          <li><a href="?page=ContactInformation.jsp">Contact information</a></li>
          <li><a href="?page=About.jsp">About us</a></li>
    </td>
</TR>
</table>
```

Figure F.4: Code - Menu.jsp.



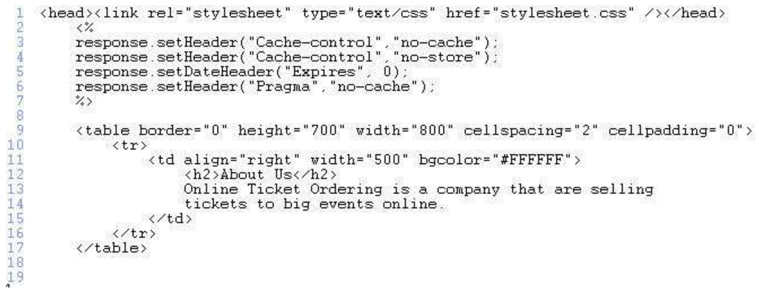
```
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>

<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>

<table border="0" height="700" width="800" cellspacing="2" cellpadding="0">
    <tr>
        <td align="right" width="500" bgcolor="#FFFFFF">
            <h2>OnlineTicketOrdering</h2>
            Welcome to online ticket ordering. Here you can
            securly buy your tickets without worrying that
            sensitive personal information are going devious
            ways.
        </td>
    </tr>
</table>
```

Figure F.5: Code - Mainpage.jsp.



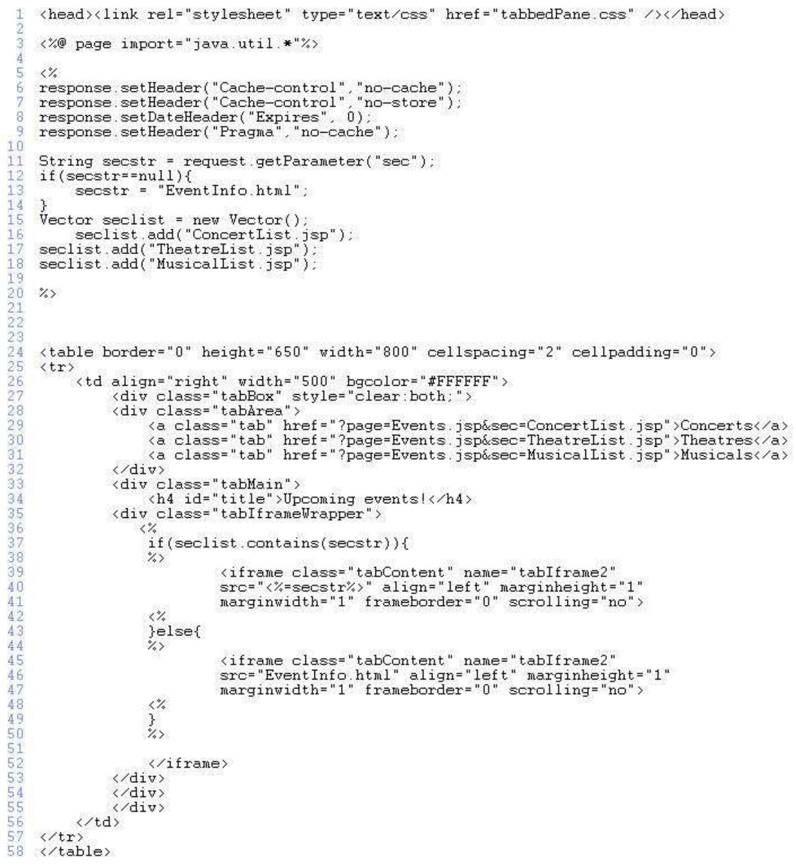
```
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<table border="0" height="700" width="800" cellspacing="2" cellpadding="0">
    <tr>
        <td align="right" width="500" bgcolor="#FFFFFF">
            <h2>Contact information</h2>
            You can contact us by sending an e-mail to
            ninai(a)stud.ntnu.no or juliemar(a)stud.ntnu.no.
            (Change the (a) with a commercial at.)
        </td>
    </tr>
</table>
```

Figure F.6: Code - ContactInformation.jsp.



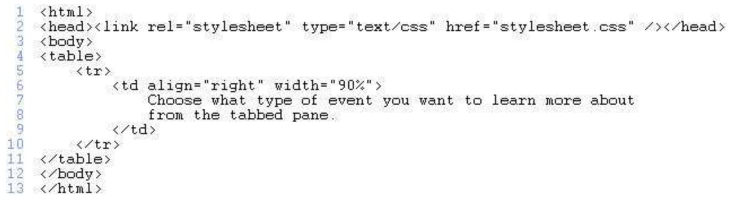
```
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    <table border="0" height="700" width="800" cellspacing="2" cellpadding="0">
        <tr>
            <td align="right" width="500" bgcolor="#FFFFFF">
                <h2>About Us</h2>
                Online Ticket Ordering is a company that are selling
                tickets to big events online.
            </td>
        </tr>
    </table>
```

Figure F.7: Code - About.jsp.



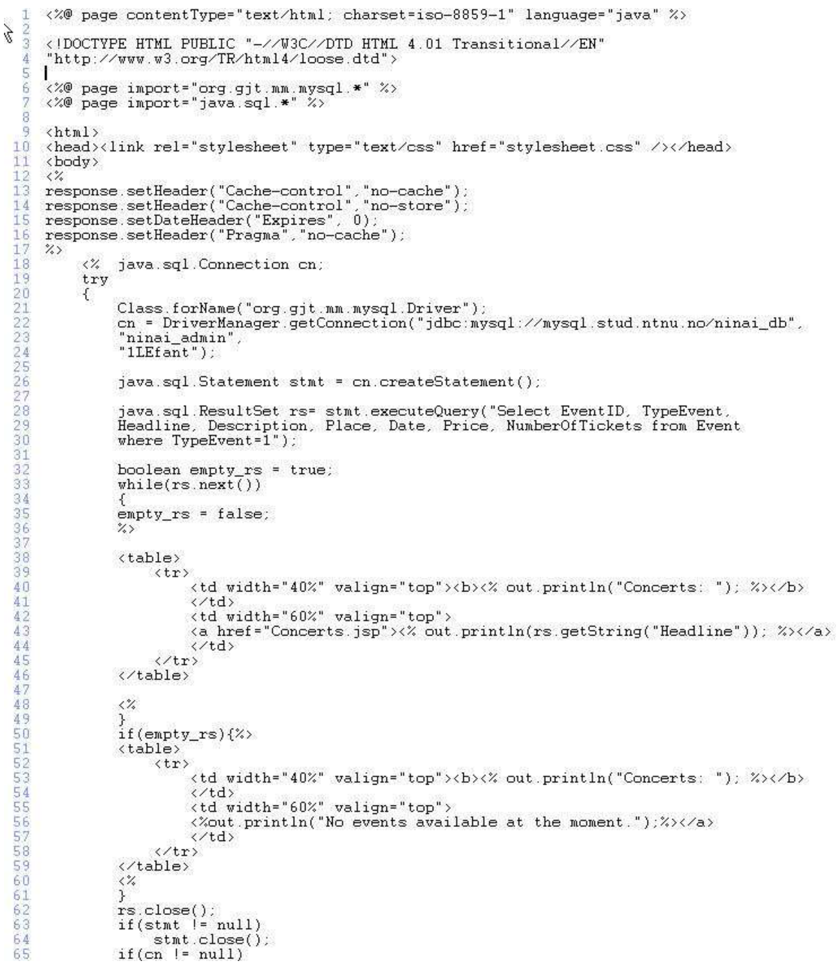
```
<head><link rel="stylesheet" type="text/css" href="tabbedPane.css" /></head>

<%@ page import="java.util.*"%>

<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");

String secstr = request.getParameter("sec");
if(secstr==null){
    secstr = "EventInfo.html";
}
Vector seclist = new Vector();
    seclist.add("ConcertList.jsp");
seclist.add("TheatreList.jsp");
seclist.add("MusicalList.jsp");

%>



<table border="0" height="650" width="800" cellspacing="2" cellpadding="0">
<tr>
    <td align="right" width="500" bgcolor="#FFFFFF">
        <div class="tabBox" style="clear:both;">
        <div class="tabArea">
            <a class="tab" href="?page=Events.jsp&sec=ConcertList.jsp">Concerts</a>
            <a class="tab" href="?page=Events.jsp&sec=TheatreList.jsp">Theatres</a>
            <a class="tab" href="?page=Events.jsp&sec=MusicalList.jsp">Musicals</a>
        </div>
        <div class="tabMain">
            <h4 id="title">Upcoming events!</h4>
        <div class="tabIframeWrapper">
           <%
            if(seclist.contains(secstr)){
            %>
                    <iframe class="tabContent" name="tabIframe2"
                    src="<%=secstr%>" align="left" marginheight="1"
                    marginwidth="1" frameborder="0" scrolling="no">
            <%
            }else{
            %>
                    <iframe class="tabContent" name="tabIframe2"
                    src="EventInfo.html" align="left" marginheight="1"
                    marginwidth="1" frameborder="0" scrolling="no">
            <%
            }
            %>

            </iframe>
        </div>
        </div>
        </div>
    </td>
</tr>
</table>
```

Figure F.8: Code - Events.jsp.



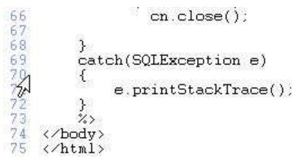
```
<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<table>
    <tr>
        <td align="right" width="90%">
            Choose what type of event you want to learn more about
            from the tabbed pane.
        </td>
    </tr>
</table>
</body>
</html>
```

Figure F.9: Code - EventInfo.html.



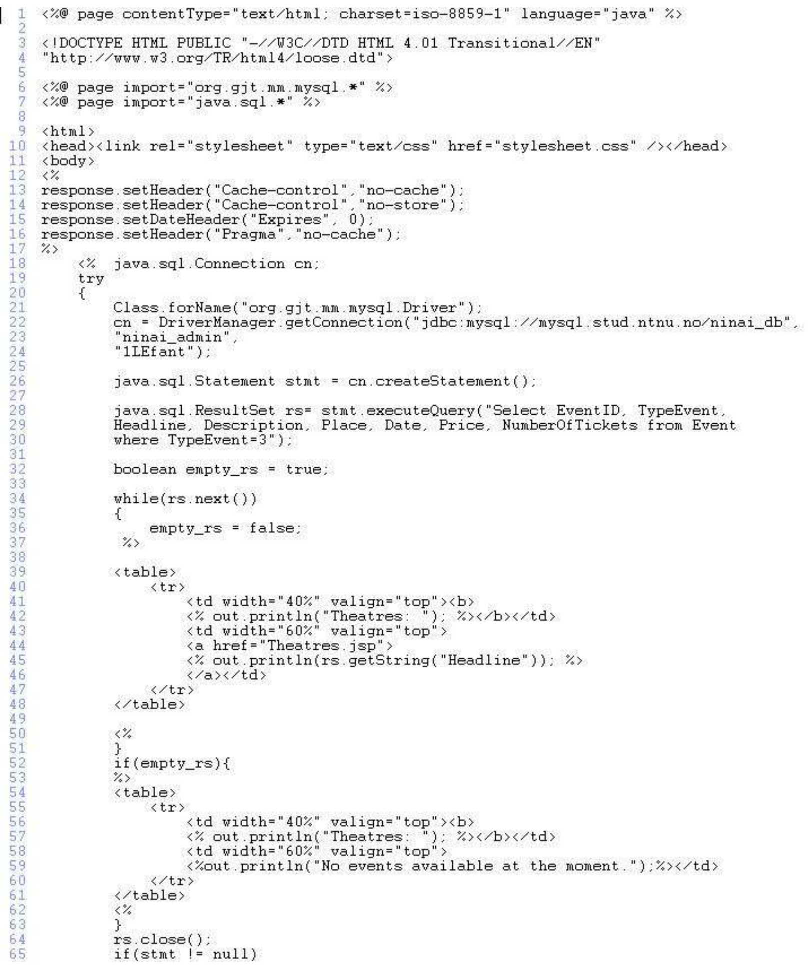
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>
    <%  java.sql.Connection cn;
    try
    {
        Class.forName("org.gjt.mm.mysql.Driver");
        cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
        "ninai_admin",
        "1LEfant");

        java.sql.Statement stmt = cn.createStatement();

        java.sql.ResultSet rs= stmt.executeQuery("Select EventID, TypeEvent,
        Headline, Description, Place, Date, Price, NumberOfTickets from Event
        where TypeEvent=1");

        boolean empty_rs = true;
        while(rs.next())
        {
        empty_rs = false;
        %>

        <table>
            <tr>
                <td width="40%" valign="top"><b><% out.println("Concerts: "); %></b>
                </td>
                <td width="60%" valign="top">
                <a href="Concerts.jsp"><% out.println(rs.getString("Headline")); %></a>
                </td>
            </tr>
        </table>

        <%
        }
        if(empty_rs){%>
        <table>
            <tr>
                <td width="40%" valign="top"><b><% out.println("Concerts: "); %></b>
                </td>
                <td width="60%" valign="top">
                <%out.println("No events available at the moment.");%></a>
                </td>
            </tr>
        </table>
        <%
        }
        rs.close();
        if(stmt != null)
            stmt.close();
        if(cn != null)
```

Figure F.10: Code - ConcertList.jsp, part I.



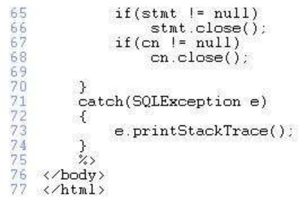
```
            cn.close();

    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }
    %>
</body>
</html>
```

Figure F.11: Code - ConcertList.jsp, part II.



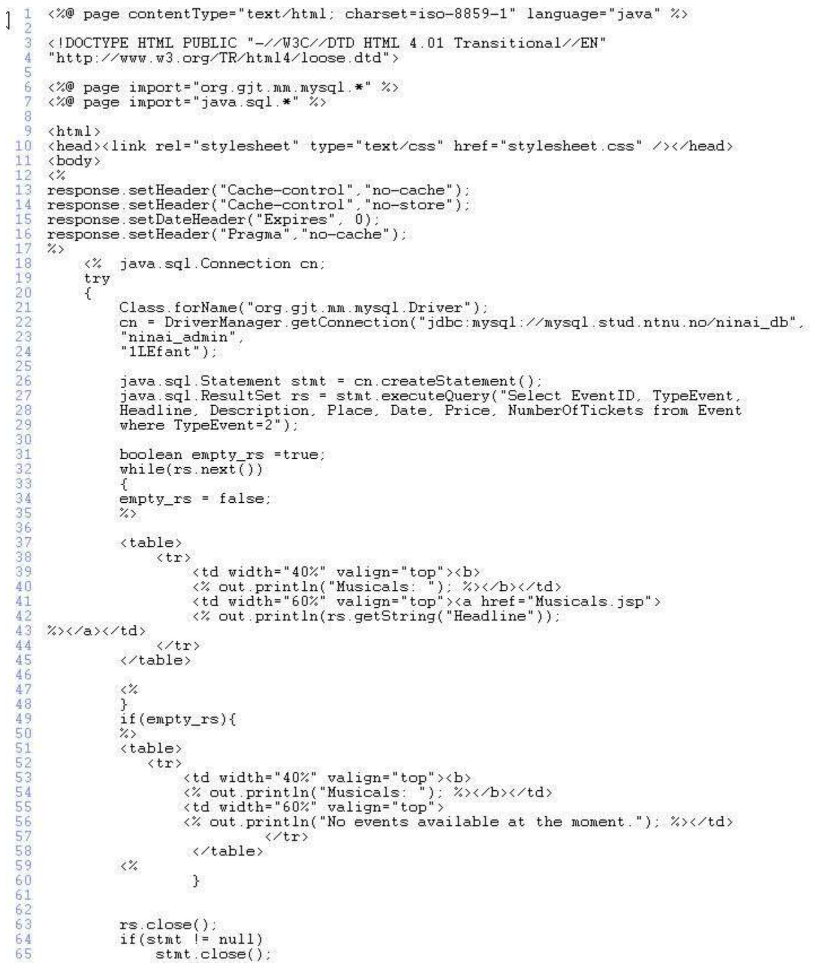
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>
    <%  java.sql.Connection cn;
    try
    {
        Class.forName("org.gjt.mm.mysql.Driver");
        cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
        "ninai_admin",
        "1LEfant");

        java.sql.Statement stmt = cn.createStatement();

        java.sql.ResultSet rs= stmt.executeQuery("Select EventID, TypeEvent,
        Headline, Description, Place, Date, Price, NumberOfTickets from Event
        where TypeEvent=3");

        boolean empty_rs = true;

        while(rs.next())
        {
            empty_rs = false;
         %>

        <table>
            <tr>
                <td width="40%" valign="top"><b>
                <% out.println("Theatres: "); %></b></td>
                <td width="60%" valign="top">
                <a href="Theatres.jsp">
                <% out.println(rs.getString("Headline")); %>
                </a></td>
            </tr>
        </table>

        <%
        }
        if(empty_rs){
        %>
        <table>
            <tr>
                <td width="40%" valign="top"><b>
                <% out.println("Theatres: "); %></b></td>
                <td width="60%" valign="top">
                <%out.println("No events available at the moment.");%></td>
            </tr>
        </table>
        <%
        }
        rs.close();
        if(stmt != null)
```

Figure F.12: Code - TheaterList.jsp, part I.



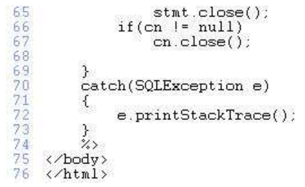
```
        if(stmt != null)
            stmt.close();
        if(cn != null)
            cn.close();

    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }
    %>
</body>
</html>
```

Figure F.13: Code - TheaterList.jsp, part II.



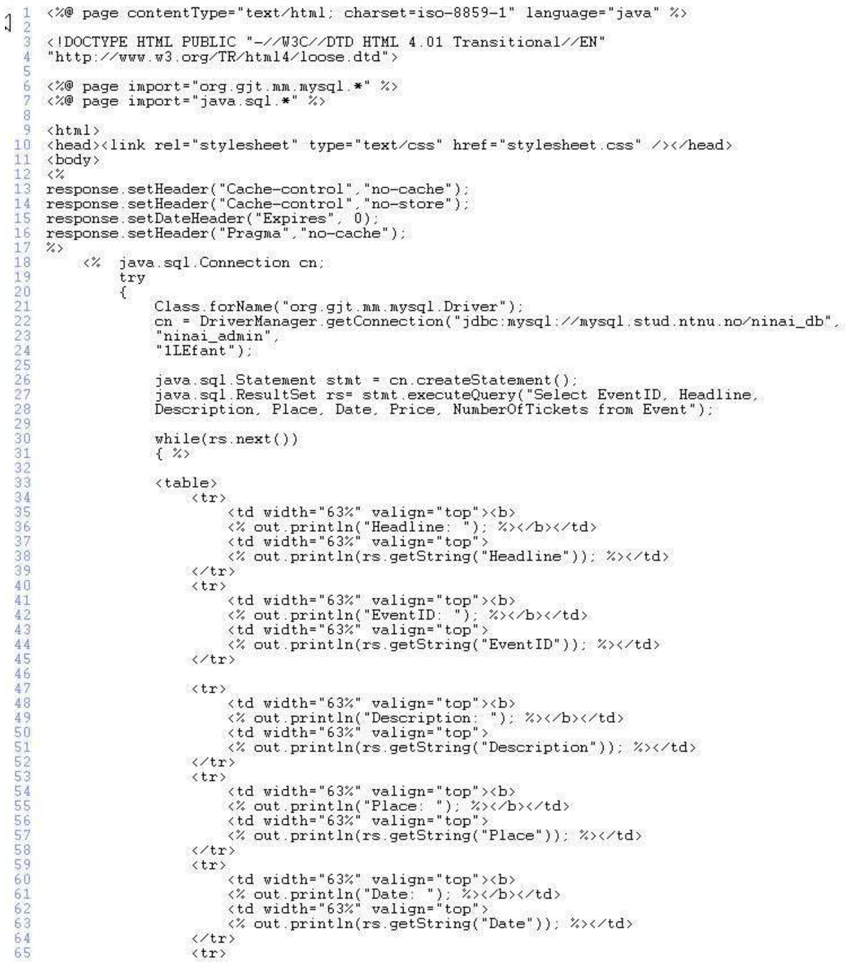
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>
    <%  java.sql.Connection cn;
    try
    {
        Class.forName("org.gjt.mm.mysql.Driver");
        cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
        "ninai_admin",
        "1LEfant");

        java.sql.Statement stmt = cn.createStatement();
        java.sql.ResultSet rs = stmt.executeQuery("Select EventID, TypeEvent,
        Headline, Description, Place, Date, Price, NumberOfTickets from Event
        where TypeEvent=2");

        boolean empty_rs =true;
        while(rs.next())
        {
        empty_rs = false;
        %>

        <table>
            <tr>
                <td width="40%" valign="top"><b>
                <% out.println("Musicals: "); %></b></td>
                <td width="60%" valign="top"><a href="Musicals.jsp">
                <% out.println(rs.getString("Headline"));
%></a></td>
            </tr>
        </table>

        <%
        }
        if(empty_rs){
        %>
        <table>
           <tr>
               <td width="40%" valign="top"><b>
               <% out.println("Musicals: "); %></b></td>
               <td width="60%" valign="top">
               <% out.println("No events available at the moment."); %></td>
                        </tr>
                </table>
        <%
                }


        rs.close();
        if(stmt != null)
            stmt.close();
```

Figure F.14: Code - MusicalList.jsp, part I.



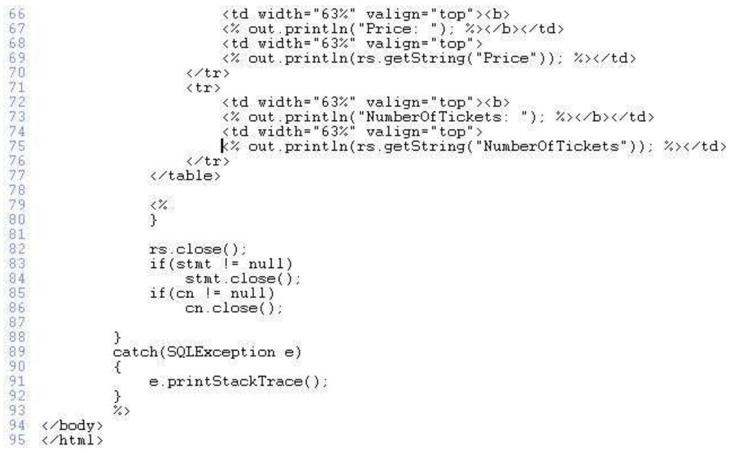
```
            stmt.close();
        if(cn != null)
            cn.close();

    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }
    %>
</body>
</html>
```

Figure F.15: Code - MusicalList.jsp, part II.



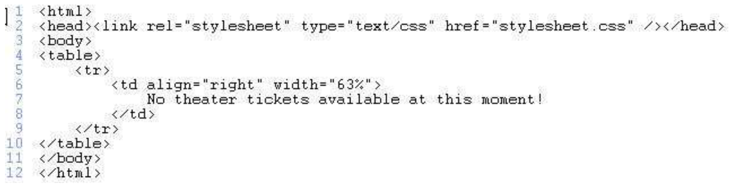
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>
    <%  java.sql.Connection cn;
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver");
            cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
            "ninai_admin",
            "1LEfant");

            java.sql.Statement stmt = cn.createStatement();
            java.sql.ResultSet rs= stmt.executeQuery("Select EventID, Headline,
            Description, Place, Date, Price, NumberOfTickets from Event");

            while(rs.next())
            { %>

            <table>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Headline: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Headline")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("EventID: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("EventID")); %></td>
                </tr>

                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Description: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Description")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Place: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Place")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Date: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Date")); %></td>
                </tr>
                <tr>
```

Figure F.16: Code - Concerts.jsp, part I.



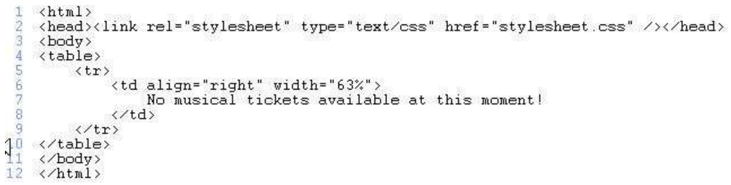
```
                    <td width="63%" valign="top"><b>
                    <% out.println("Price: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Price")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("NumberOfTickets: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("NumberOfTickets")); %></td>
                </tr>
            </table>

            <%
            }

            rs.close();
            if(stmt != null)
                stmt.close();
            if(cn != null)
                cn.close();

        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        %>
</body>
</html>
```

Figure F.17: Code - Concerts.jsp, part II.



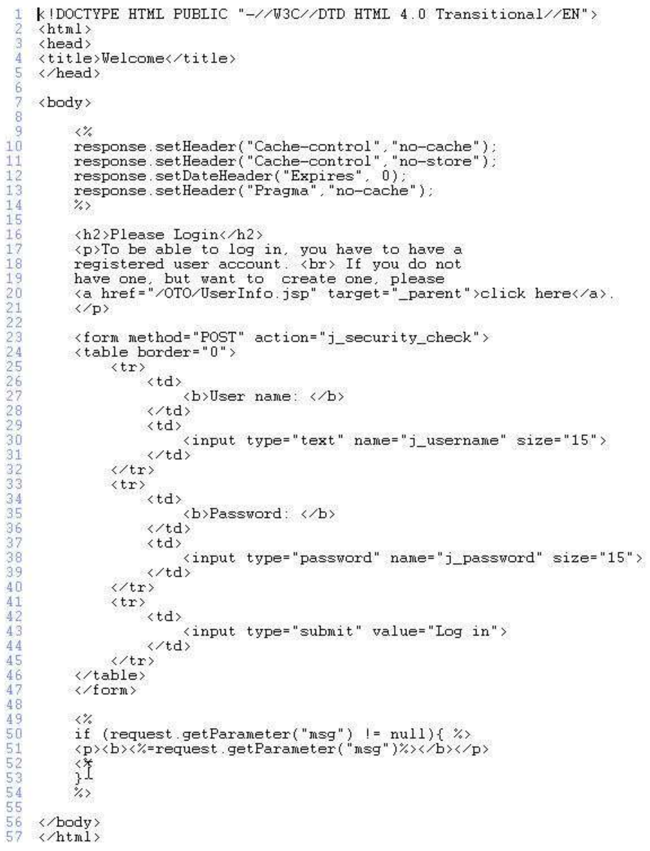
```
<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<table>
    <tr>
        <td align="right" width="63%">
            No theater tickets available at this moment!
        </td>
    </tr>
</table>
</body>
</html>
```

Figure F.18: Code - Theaters.html.



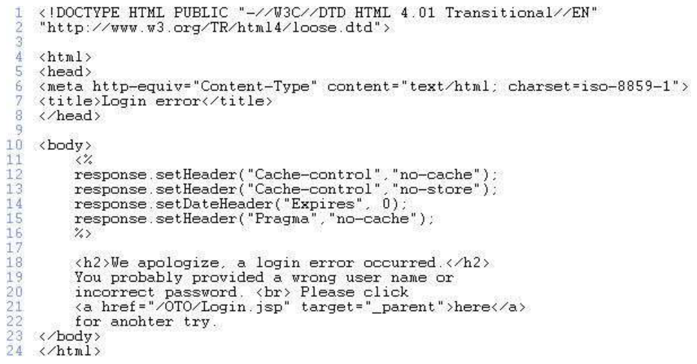
```
<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<table>
    <tr>
        <td align="right" width="63%">
            No musical tickets available at this moment!
        </td>
    </tr>
</table>
</body>
</html>
```

Figure F.19: Code - Musicals.html.



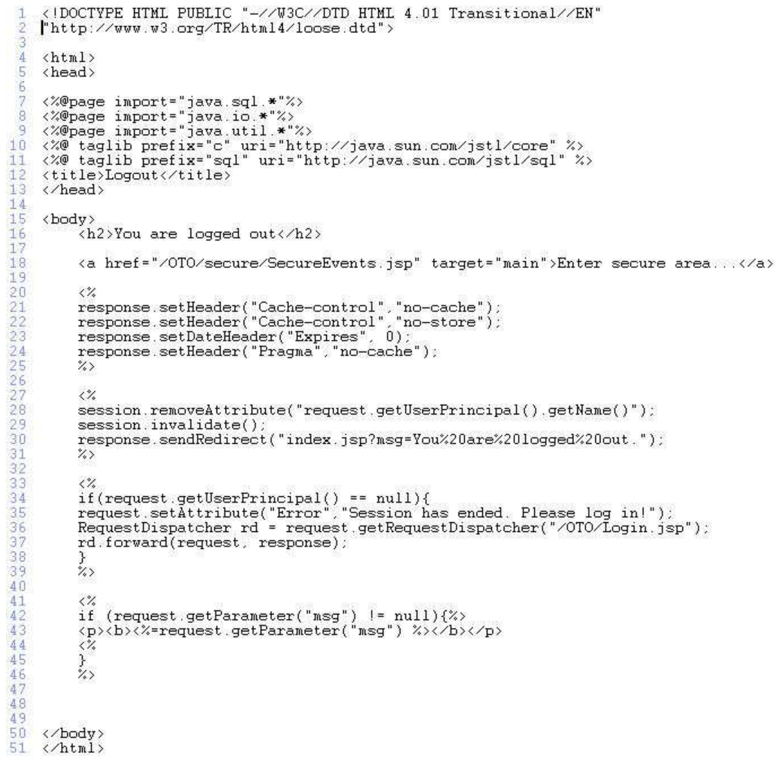
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Welcome</title>
</head>

<body>

    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    <h2>Please Login</h2>
    <p>To be able to log in, you have to have a
    registered user account. <br> If you do not
    have one, but want to  create one, please
    <a href="/OTO/UserInfo.jsp" target="_parent">click here</a>.
    </p>

    <form method="POST" action="j_security_check">
    <table border="0">
        <tr>
            <td>
                <b>User name: </b>
            </td>
            <td>
                <input type="text" name="j_username" size="15">
            </td>
        </tr>
        <tr>
            <td>
                <b>Password: </b>
            </td>
            <td>
                <input type="password" name="j_password" size="15">
            </td>
        </tr>
        <tr>
            <td>
                <input type="submit" value="Log in">
            </td>
        </tr>
    </table>
    </form>

    <%
    if (request.getParameter("msg") != null){ %>
    <p><b><%=request.getParameter("msg")%></b></p>
    <%
    }
    %>

</body>
</html>
```

Figure F.20: Code - Login.jsp.



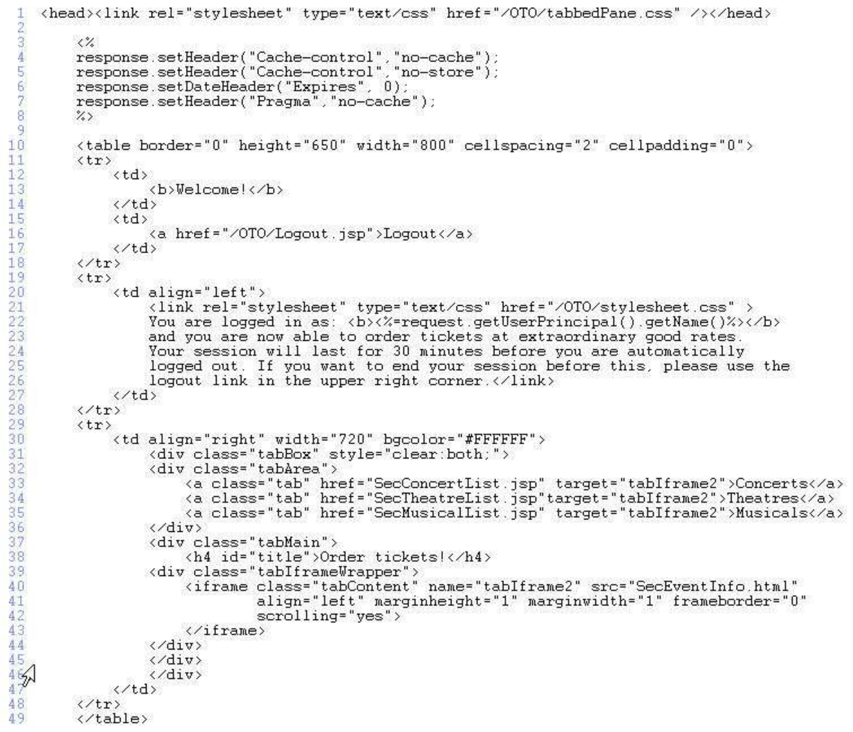
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Login error</title>
</head>

<body>
    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    <h2>We apologize, a login error occurred.</h2>
    You probably provided a wrong user name or
    incorrect password. <br> Please click
    <a href="/OTO/Login.jsp" target="_parent">here</a>
    for anohter try.
</body>
</html>
```

Figure F.21: Code - LoginError.jsp.



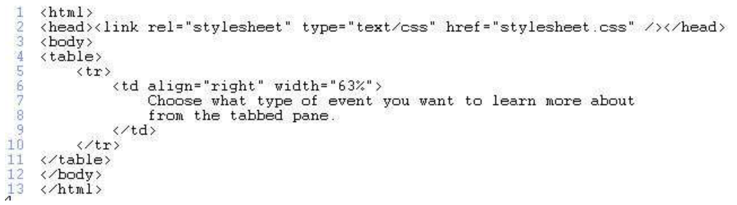
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>

<%@page import="java.sql.*"%>
<%@page import="java.io.*"%>
<%@page import="java.util.*"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<title>Logout</title>
</head>

<body>
    <h2>You are logged out</h2>

    <a href="/OTO/secure/SecureEvents.jsp" target="main">Enter secure area...</a>

    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    <%
    session.removeAttribute("request.getUserPrincipal().getName()");
    session.invalidate();
    response.sendRedirect("index.jsp?msg=You%20are%20logged%20out.");
    %>

    <%
    if(request.getUserPrincipal() == null){
    request.setAttribute("Error","Session has ended. Please log in!");
    RequestDispatcher rd = request.getRequestDispatcher("/OTO/Login.jsp");
    rd.forward(request, response);
    }
    %>

    <%
    if (request.getParameter("msg") != null){%>
    <p><b><%=request.getParameter("msg") %></b></p>
    <%
    }
    %>



</body>
</html>
```

Figure F.22: Code - Logout.jsp.



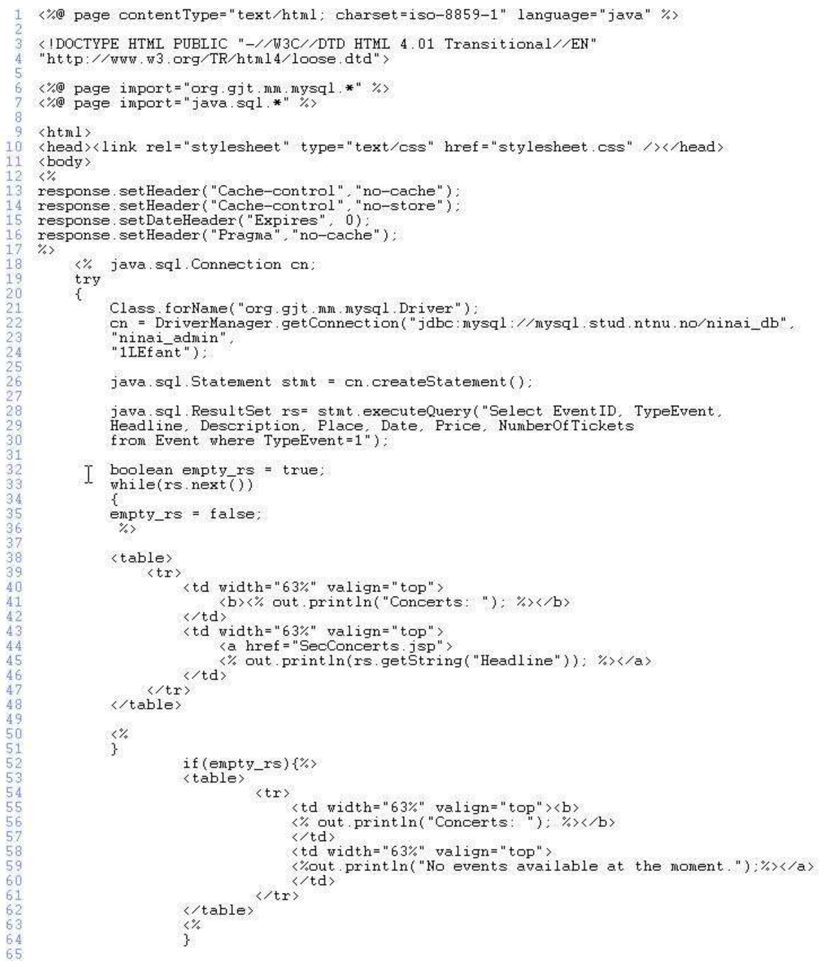
```
<head><link rel="stylesheet" type="text/css" href="/OTO/tabbedPane.css" /></head>

    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    <table border="0" height="650" width="800" cellspacing="2" cellpadding="0">
    <tr>
        <td>
            <b>Welcome!</b>
        </td>
        <td>
            <a href="/OTO/Logout.jsp">Logout</a>
        </td>
    </tr>
    <tr>
        <td align="left">
            <link rel="stylesheet" type="text/css" href="/OTO/stylesheet.css" >
            You are logged in as: <b><%=request.getUserPrincipal().getName()%></b>
            and you are now able to order tickets at extraordinary good rates.
            Your session will last for 30 minutes before you are automatically
            logged out. If you want to end your session before this, please use the
            logout link in the upper right corner.</link>
        </td>
    </tr>
    <tr>
        <td align="right" width="720" bgcolor="#FFFFFF">
            <div class="tabBox" style="clear:both;">
            <div class="tabArea">
                <a class="tab" href="SecConcertList.jsp" target="tabIframe2">Concerts</a>
                <a class="tab" href="SecTheatreList.jsp"target="tabIframe2">Theatres</a>
                <a class="tab" href="SecMusicalList.jsp" target="tabIframe2">Musicals</a>
            </div>
            <div class="tabMain">
                <h4 id="title">Order tickets!</h4>
            <div class="tabIframeWrapper">
                <iframe class="tabContent" name="tabIframe2" src="SecEventInfo.html"
                        align="left" marginheight="1" marginwidth="1" frameborder="0"
                        scrolling="yes">
                </iframe>
            </div>
            </div>
            </div>
        </td>
    </tr>
    </table>
```

Figure F.23: Code - SecureEvents.jsp.



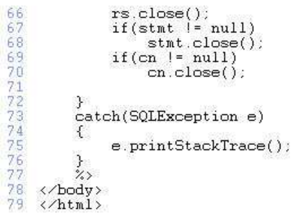
```
<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<table>
    <tr>
        <td align="right" width="63%">
            Choose what type of event you want to learn more about
            from the tabbed pane.
        </td>
    </tr>
</table>
</body>
</html>
```

Figure F.24: Code - SecEventInfo.html.



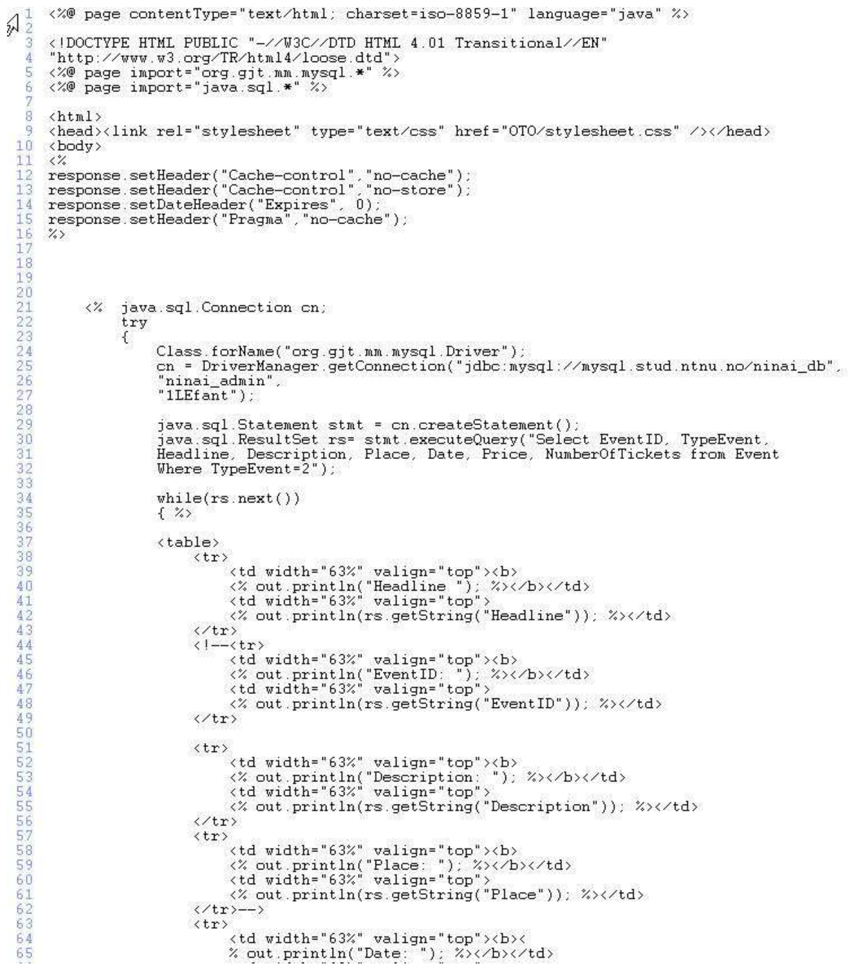
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>
    <%  java.sql.Connection cn;
    try
    {
        Class.forName("org.gjt.mm.mysql.Driver");
        cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
        "ninai_admin",
        "1LEfant");

        java.sql.Statement stmt = cn.createStatement();

        java.sql.ResultSet rs= stmt.executeQuery("Select EventID, TypeEvent,
        Headline, Description, Place, Date, Price, NumberOfTickets
        from Event where TypeEvent=1");

        boolean empty_rs = true;
        while(rs.next())
        {
        empty_rs = false;
         %>

        <table>
            <tr>
                <td width="63%" valign="top">
                    <b><% out.println("Concerts: "); %></b>
                </td>
                <td width="63%" valign="top">
                    <a href="SecConcerts.jsp">
                    <% out.println(rs.getString("Headline")); %></a>
                </td>
            </tr>
        </table>

        <%
        }
                if(empty_rs){%>
                <table>
                        <tr>
                            <td width="63%" valign="top"><b>
                            <% out.println("Concerts: "); %></b>
                            </td>
                            <td width="63%" valign="top">
                            <%out.println("No events available at the moment.");%></a>
                            </td>
                        </tr>
                </table>
                <%
                }

```

Figure F.25: Code - SecConcertList.jsp, part I.



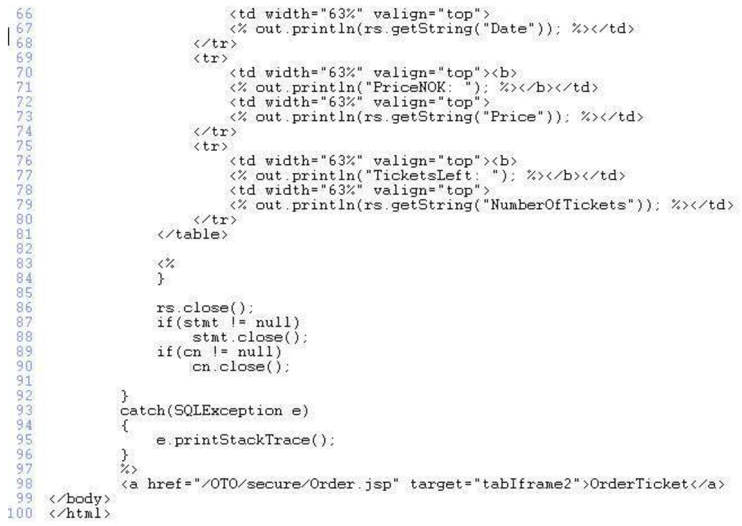
```
        rs.close();
        if(stmt != null)
            stmt.close();
        if(cn != null)
            cn.close();

    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }
    %>
</body>
</html>
```

Figure F.26: Code - SecConcertList.jsp, part II.



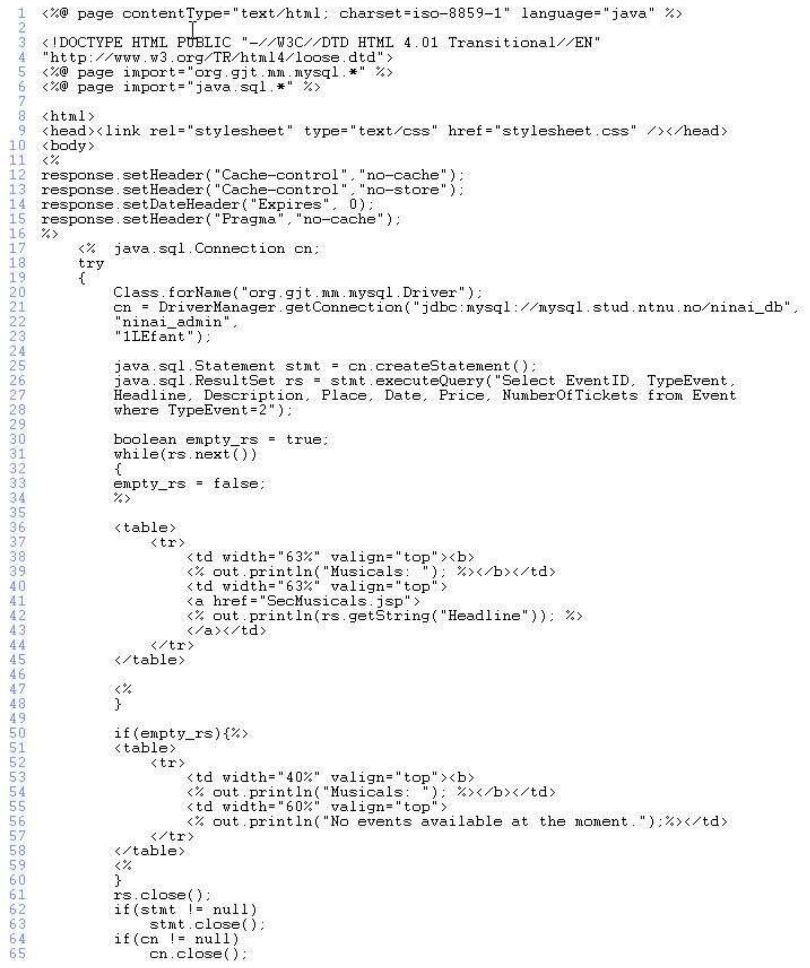
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="OTO/stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>




    <%  java.sql.Connection cn;
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver");
            cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
            "ninai_admin",
            "1LEfant");

            java.sql.Statement stmt = cn.createStatement();
            java.sql.ResultSet rs= stmt.executeQuery("Select EventID, TypeEvent,
            Headline, Description, Place, Date, Price, NumberOfTickets from Event
            Where TypeEvent=2");

            while(rs.next())
            { %>

            <table>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Headline "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Headline")); %></td>
                </tr>
                <!--<tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("EventID: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("EventID")); %></td>
                </tr>

                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Description: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Description")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Place: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Place")); %></td>
                </tr>-->
                <tr>
                    <td width="63%" valign="top"><b><
                    % out.println("Date: "); %></b></td>
```

Figure F.27: Code - SecTheaterList.jsp, part I.



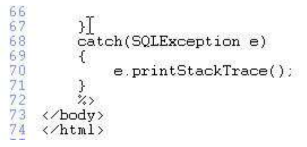
```
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Date")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("PriceNOK: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Price")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("TicketsLeft: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("NumberOfTickets")); %></td>
                </tr>
            </table>

            <%
            }

            rs.close();
            if(stmt != null)
                stmt.close();
            if(cn != null)
                cn.close();

        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        %>
        <a href="/OTO/secure/Order.jsp" target="tabIframe2">OrderTicket</a>
</body>
</html>
```

Figure F.28: Code - SecTheaterList.jsp, part II.



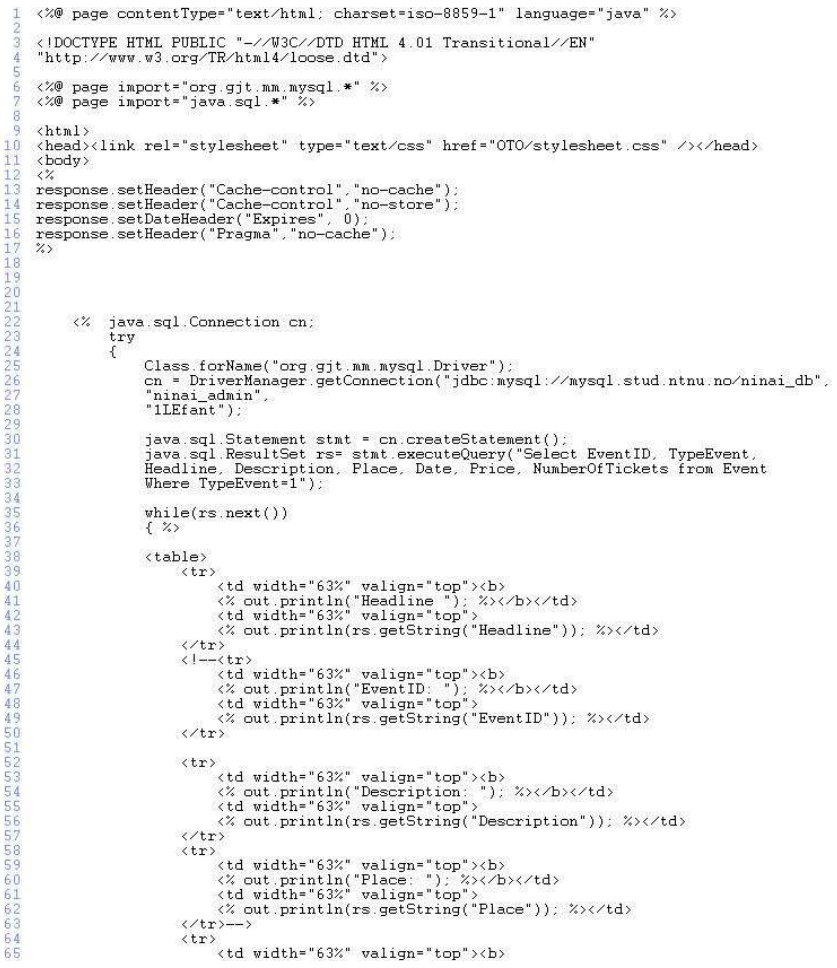
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>
    <%  java.sql.Connection cn;
    try
    {
        Class.forName("org.gjt.mm.mysql.Driver");
        cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
        "ninai_admin",
        "1LEfant");

        java.sql.Statement stmt = cn.createStatement();
        java.sql.ResultSet rs = stmt.executeQuery("Select EventID, TypeEvent,
        Headline, Description, Place, Date, Price, NumberOfTickets from Event
        where TypeEvent=2");

        boolean empty_rs = true;
        while(rs.next())
        {
        empty_rs = false;
        %>

        <table>
            <tr>
                <td width="63%" valign="top"><b>
                <% out.println("Musicals: "); %></b></td>
                <td width="63%" valign="top">
                <a href="SecMusicals.jsp">
                <% out.println(rs.getString("Headline")); %>
                </a></td>
            </tr>
        </table>

        <%
        }

        if(empty_rs){%>
        <table>
            <tr>
                <td width="40%" valign="top"><b>
                <% out.println("Musicals: "); %></b></td>
                <td width="60%" valign="top">
                <% out.println("No events available at the moment.");%></td>
            </tr>
        </table>
        <%
        }
        rs.close();
        if(stmt != null)
            stmt.close();
        if(cn != null)
            cn.close();
```

Figure F.29: Code - SecMusicalList.jsp, part I.



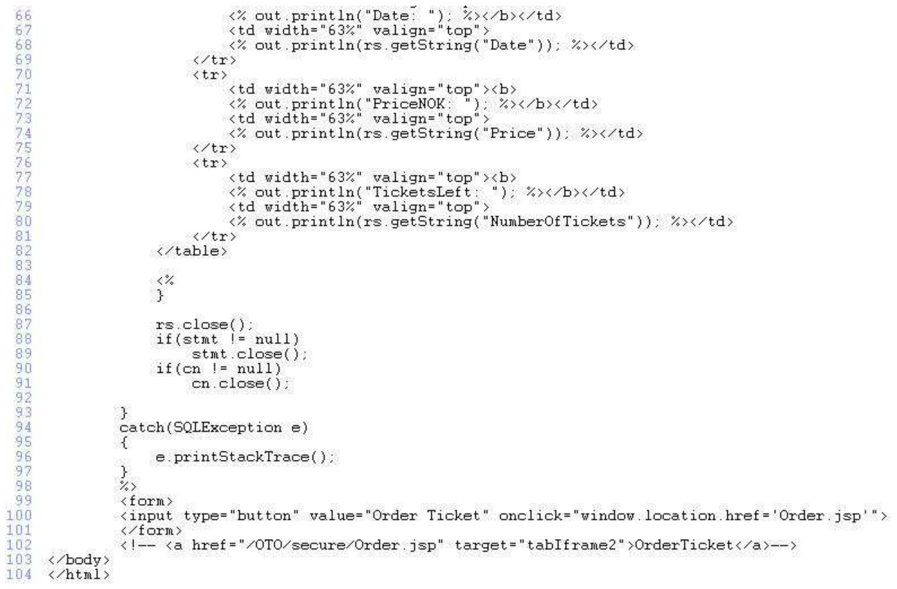
```

    }
    catch(SQLException e)
    {
        e.printStackTrace();
    }
    %>
</body>
</html>
```

Figure F.30: Code - SecMusicalList.jsp, part II.



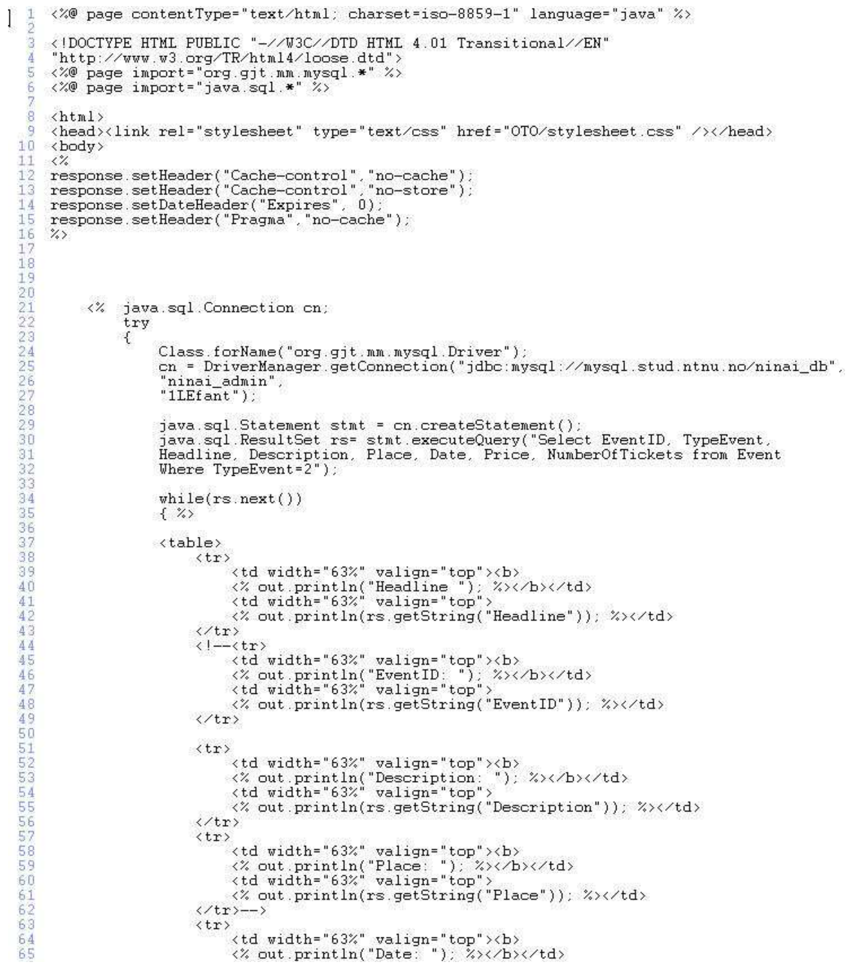
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="OTO/stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>




    <%  java.sql.Connection cn;
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver");
            cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
            "ninai_admin",
            "1LEfant");

            java.sql.Statement stmt = cn.createStatement();
            java.sql.ResultSet rs= stmt.executeQuery("Select EventID, TypeEvent,
            Headline, Description, Place, Date, Price, NumberOfTickets from Event
            Where TypeEvent=1");

            while(rs.next())
            { %>

            <table>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Headline "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Headline")); %></td>
                </tr>
                <!--<tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("EventID: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("EventID")); %></td>
                </tr>

                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Description: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Description")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Place: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Place")); %></td>
                </tr>-->
                <tr>
                    <td width="63%" valign="top"><b>
```

Figure F.31: Code - SecConcerts.jsp, part I.



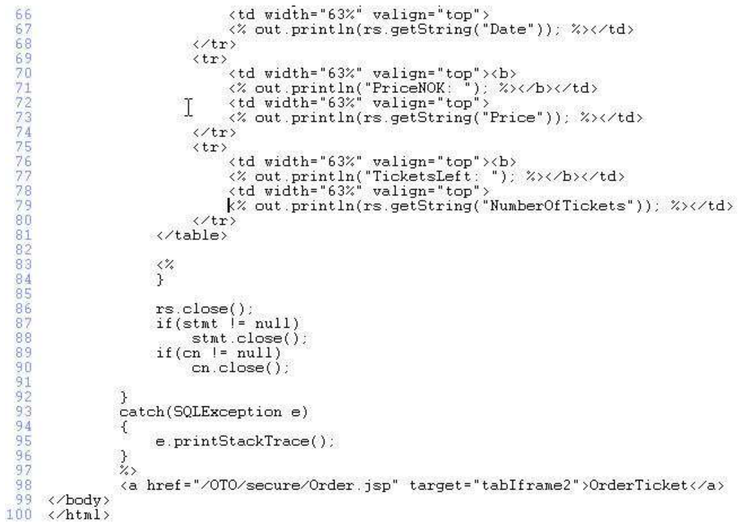
```
                    <% out.println("Date: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Date")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("PriceNOK: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Price")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("TicketsLeft: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("NumberOfTickets")); %></td>
                </tr>
            </table>

            <%
            }

            rs.close();
            if(stmt != null)
                stmt.close();
            if(cn != null)
                cn.close();

        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        %>
        <form>
        <input type="button" value="Order Ticket" onclick="window.location.href='Order.jsp'">
        </form>
        <!-- <a href="/OTO/secure/Order.jsp" target="tabIframe2">OrderTicket</a>-->
</body>
</html>
```

Figure F.32: Code - SecConcerts.jsp, part II.



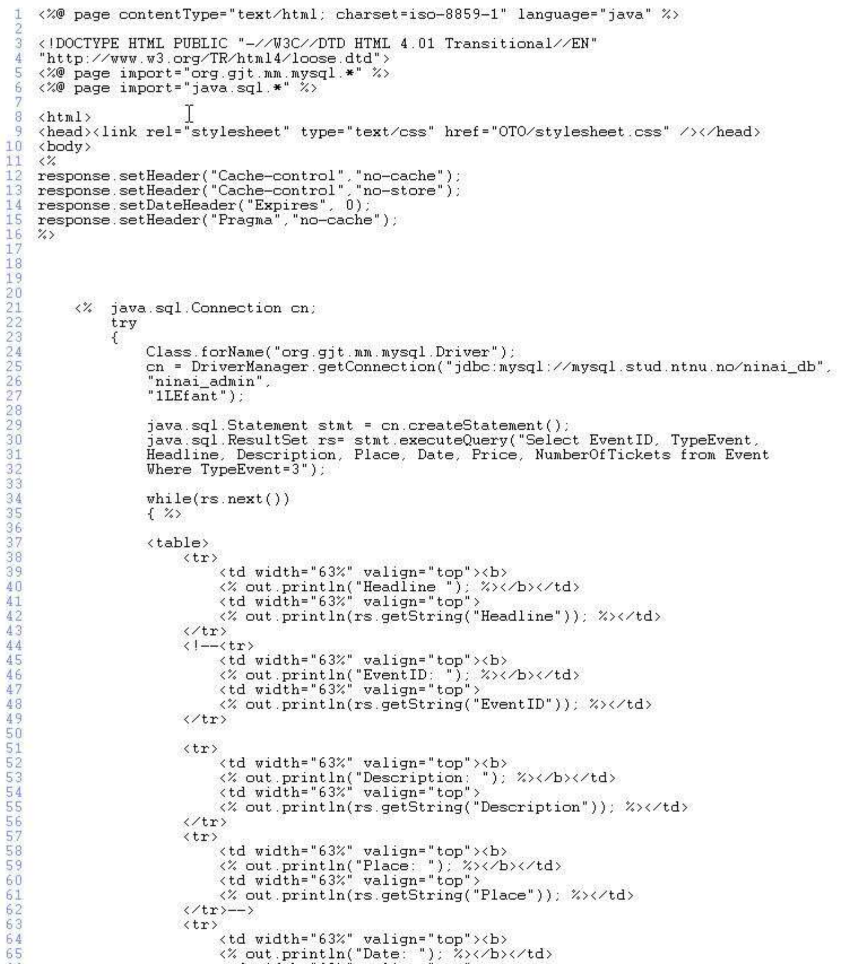
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="OTO/stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>




    <%  java.sql.Connection cn;
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver");
            cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
            "ninai_admin",
            "1LEfant");

            java.sql.Statement stmt = cn.createStatement();
            java.sql.ResultSet rs= stmt.executeQuery("Select EventID, TypeEvent,
            Headline, Description, Place, Date, Price, NumberOfTickets from Event
            Where TypeEvent=2");

            while(rs.next())
            { %>

            <table>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Headline "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Headline")); %></td>
                </tr>
                <!--<tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("EventID: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("EventID")); %></td>
                </tr>

                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Description: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Description")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Place: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Place")); %></td>
                </tr>-->
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Date: "); %></b></td>
```

Figure F.33: Code - SecTheaters.jsp, part I.



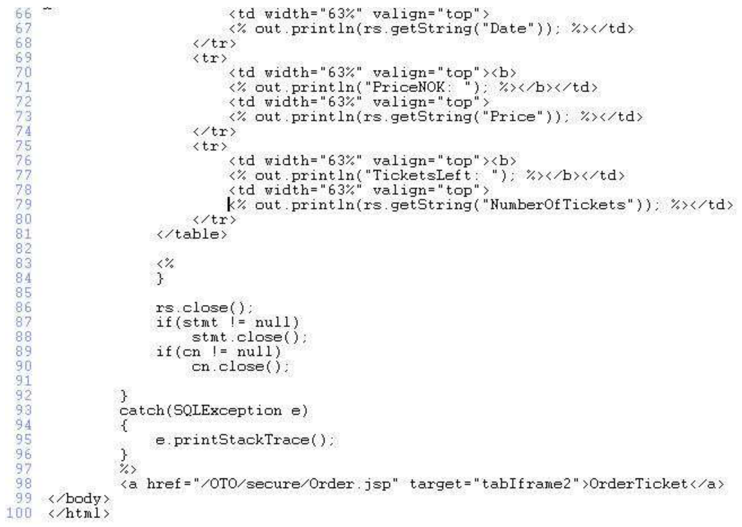
```
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Date")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("PriceNOK: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Price")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("TicketsLeft: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("NumberOfTickets")); %></td>
                </tr>
            </table>

            <%
            }

            rs.close();
            if(stmt != null)
                stmt.close();
            if(cn != null)
                cn.close();

        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        %>
        <a href="/OTO/secure/Order.jsp" target="tabIframe2">OrderTicket</a>
</body>
</html>
```

Figure F.34: Code - SecTheaters.jsp, part II.



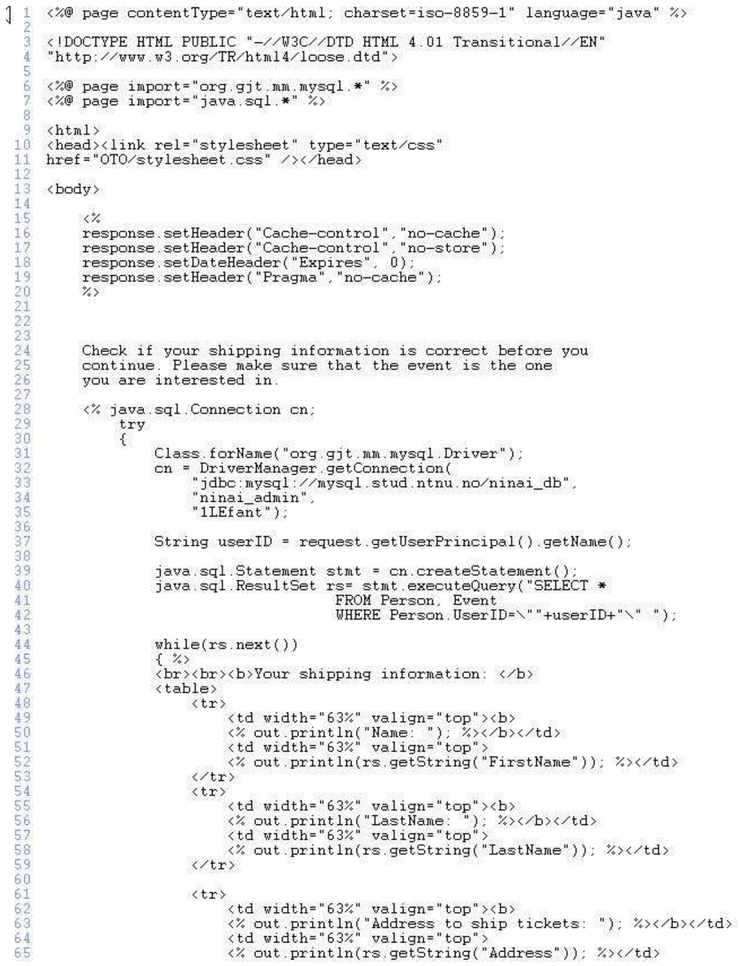
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="OTO/stylesheet.css" /></head>
<body>
<%
response.setHeader("Cache-control","no-cache");
response.setHeader("Cache-control","no-store");
response.setDateHeader("Expires", 0);
response.setHeader("Pragma","no-cache");
%>




    <%  java.sql.Connection cn;
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver");
            cn = DriverManager.getConnection("jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
            "ninai_admin",
            "1LEfant");

            java.sql.Statement stmt = cn.createStatement();
            java.sql.ResultSet rs= stmt.executeQuery("Select EventID, TypeEvent,
            Headline, Description, Place, Date, Price, NumberOfTickets from Event
            Where TypeEvent=3");

            while(rs.next())
            { %>

            <table>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Headline "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Headline")); %></td>
                </tr>
                <!--<tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("EventID: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("EventID")); %></td>
                </tr>

                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Description: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Description")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Place: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Place")); %></td>
                </tr>-->
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Date: "); %></b></td>
```

Figure F.35: Code - SecMusicals.jsp, part I.



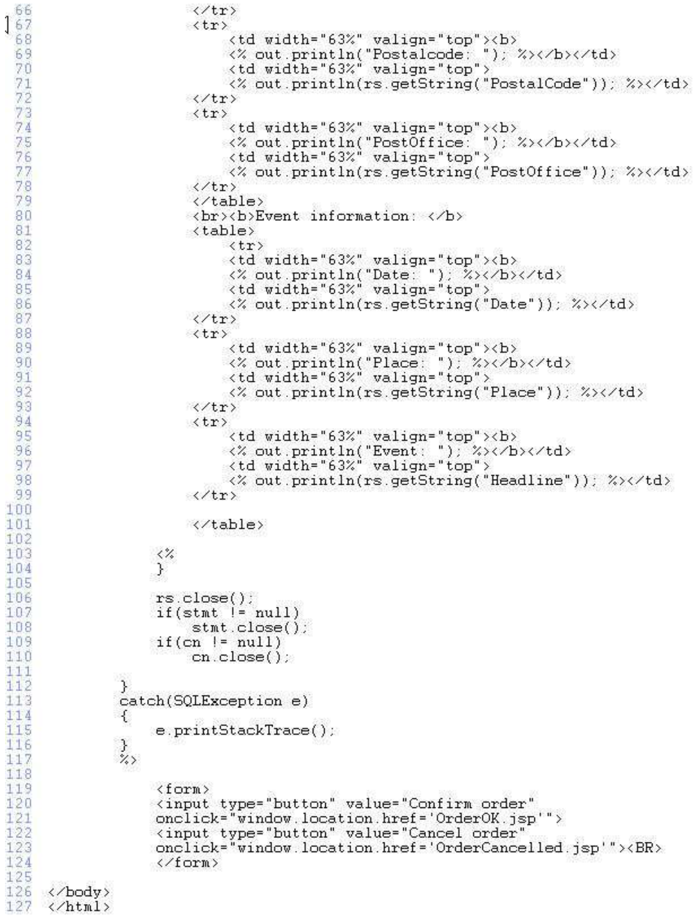
```
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Date")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("PriceNOK: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Price")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("TicketsLeft: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("NumberOfTickets")); %></td>
                </tr>
            </table>

            <%
            }

            rs.close();
            if(stmt != null)
                stmt.close();
            if(cn != null)
                cn.close();

        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        %>
        <a href="/OTO/secure/Order.jsp" target="tabIframe2">OrderTicket</a>
</body>
</html>
```

Figure F.36: Code - SecMusicals.jsp, part II.



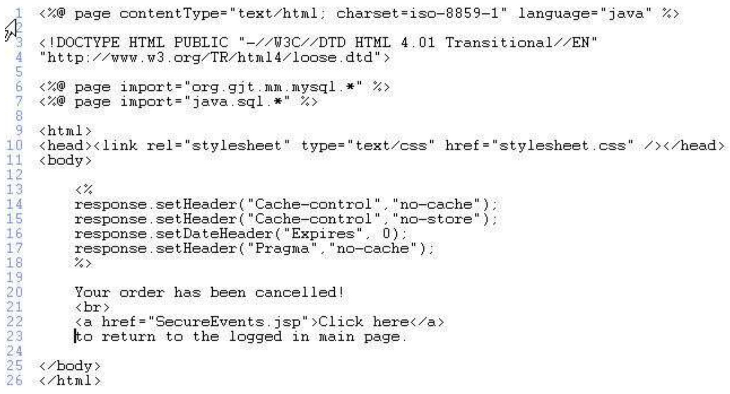
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css"
href="OTO/stylesheet.css" /></head>

<body>

    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>



    Check if your shipping information is correct before you
    continue. Please make sure that the event is the one
    you are interested in.

    <% java.sql.Connection cn;
        try
        {
            Class.forName("org.gjt.mm.mysql.Driver");
            cn = DriverManager.getConnection(
                "jdbc:mysql://mysql.stud.ntnu.no/ninai_db",
                "ninai_admin",
                "1LEfant");

            String userID = request.getUserPrincipal().getName();

            java.sql.Statement stmt = cn.createStatement();
            java.sql.ResultSet rs= stmt.executeQuery("SELECT *
                                    FROM Person, Event
                                    WHERE Person.UserID=\""+userID+"\" ");

            while(rs.next())
            { %>
            <br><br><b>Your shipping information: </b>
            <table>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Name: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("FirstName")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("LastName: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("LastName")); %></td>
                </tr>

                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Address to ship tickets: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Address")); %></td>
```

Figure F.37: Code - Order.jsp, part I.



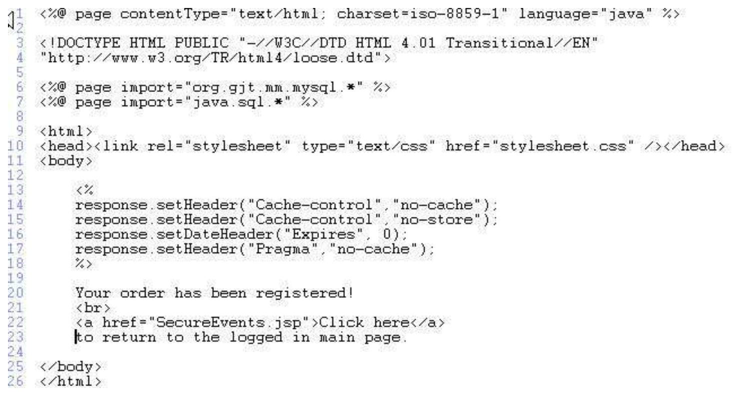
```
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Postalcode: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("PostalCode")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("PostOffice: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("PostOffice")); %></td>
                </tr>
                </table>
                <br><b>Event information: </b>
                <table>
                    <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Date: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Date")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Place: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Place")); %></td>
                </tr>
                <tr>
                    <td width="63%" valign="top"><b>
                    <% out.println("Event: "); %></b></td>
                    <td width="63%" valign="top">
                    <% out.println(rs.getString("Headline")); %></td>
                </tr>

                </table>

            <%
            }

            rs.close();
            if(stmt != null)
                stmt.close();
            if(cn != null)
                cn.close();

        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        %>

            <form>
            <input type="button" value="Confirm order"
            onclick="window.location.href='OrderOK.jsp'">
            <input type="button" value="Cancel order"
            onclick="window.location.href='OrderCancelled.jsp'"><BR>
            </form>

</body>
</html>
```

Figure F.38: Code - Order.jsp, part II.



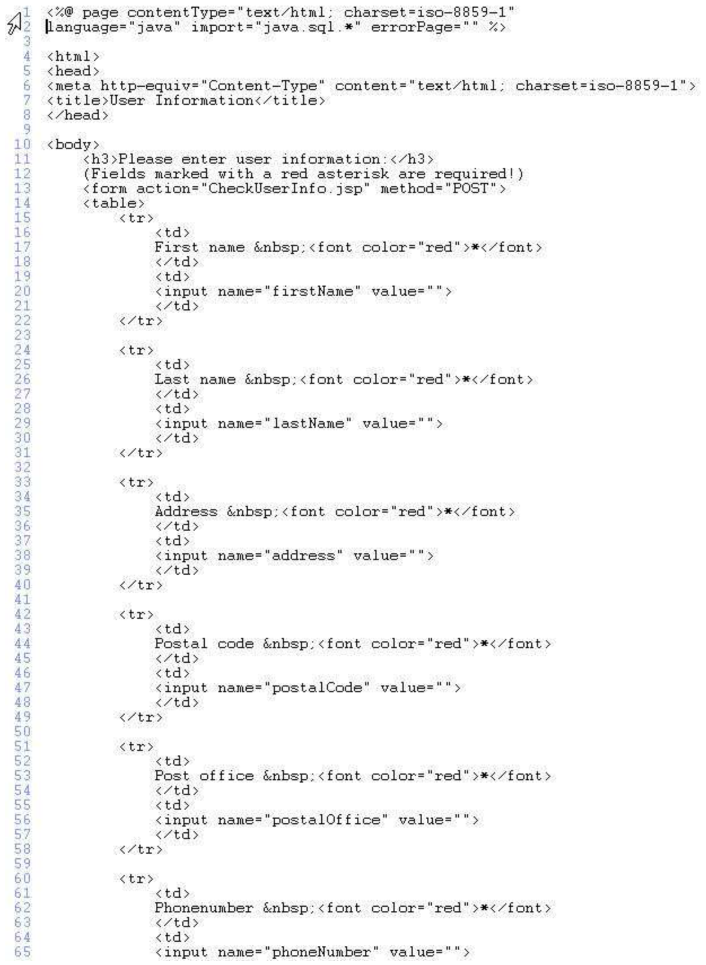
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>

    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    Your order has been cancelled!
    <br>
    <a href="SecureEvents.jsp">Click here</a>
    to return to the logged in main page.

</body>
</html>
```

Figure F.39: Code - OrderOK.jsp.



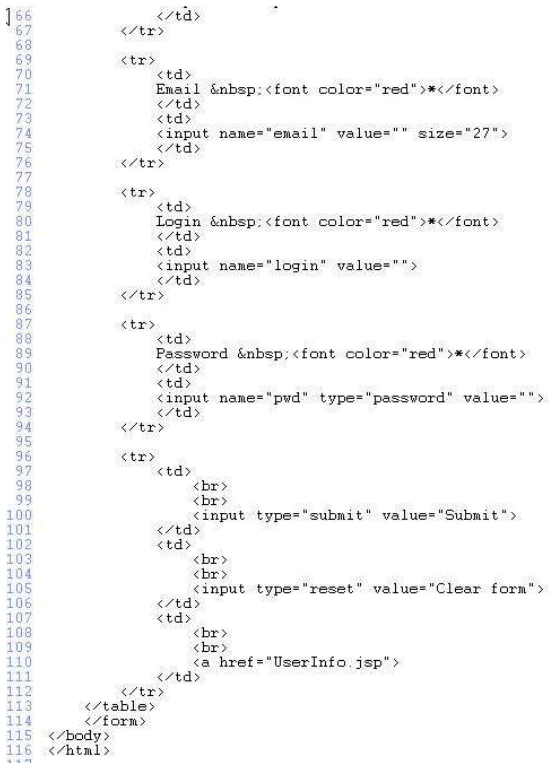
```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<%@ page import="org.gjt.mm.mysql.*" %>
<%@ page import="java.sql.*" %>

<html>
<head><link rel="stylesheet" type="text/css" href="stylesheet.css" /></head>
<body>

    <%
    response.setHeader("Cache-control","no-cache");
    response.setHeader("Cache-control","no-store");
    response.setDateHeader("Expires", 0);
    response.setHeader("Pragma","no-cache");
    %>

    Your order has been registered!
    <br>
    <a href="SecureEvents.jsp">Click here</a>
    to return to the logged in main page.

</body>
</html>
```

Figure F.40: Code - OrderCancel.jsp.



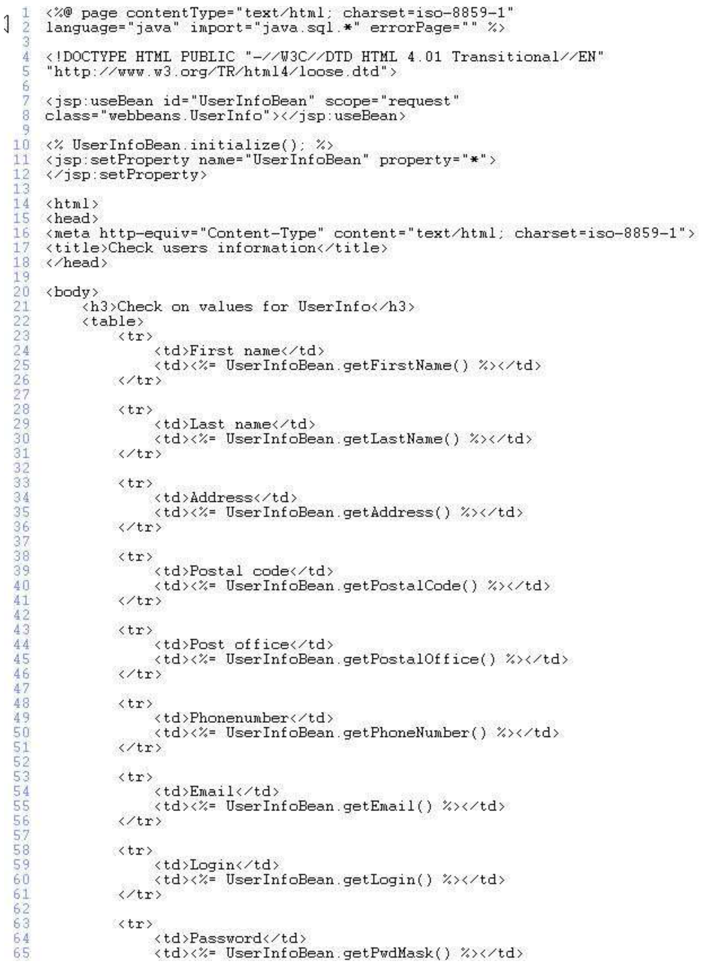
```
<%@ page contentType="text/html; charset=iso-8859-1"
language="java" import="java.sql.*" errorPage="" %>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>User Information</title>
</head>

<body>
    <h3>Please enter user information:</h3>
    (Fields marked with a red asterisk are required!)
    <form action="CheckUserInfo.jsp" method="POST">
    <table>
        <tr>
            <td>
            First name &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="firstName" value="">
            </td>
        </tr>

        <tr>
            <td>
            Last name &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="lastName" value="">
            </td>
        </tr>

        <tr>
            <td>
            Address &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="address" value="">
            </td>
        </tr>

        <tr>
            <td>
            Postal code &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="postalCode" value="">
            </td>
        </tr>

        <tr>
            <td>
            Post office &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="postalOffice" value="">
            </td>
        </tr>

        <tr>
            <td>
            Phonenumber &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="phoneNumber" value="">
```

Figure F.41: Code - UserInfo.jsp, part I.



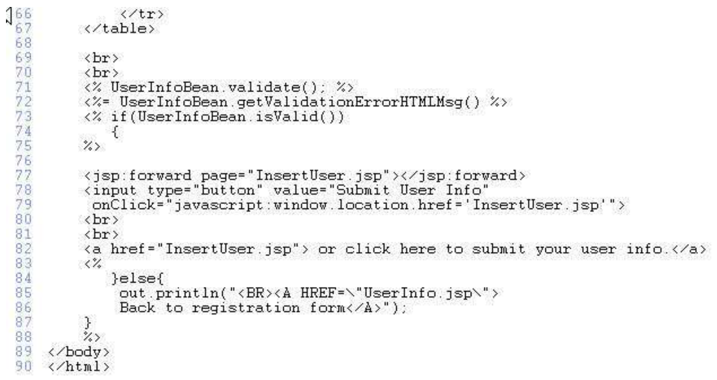
```
            </td>
        </tr>

        <tr>
            <td>
            Email &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="email" value="" size="27">
            </td>
        </tr>

        <tr>
            <td>
            Login &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="login" value="">
            </td>
        </tr>

        <tr>
            <td>
            Password &nbsp;<font color="red">*</font>
            </td>
            <td>
            <input name="pwd" type="password" value="">
            </td>
        </tr>

        <tr>
            <td>
                <br>
                <br>
                <input type="submit" value="Submit">
            </td>
            <td>
                <br>
                <br>
                <input type="reset" value="Clear form">
            </td>
            <td>
                <br>
                <br>
                <a href="UserInfo.jsp">
            </td>
        </tr>
    </table>
    </form>
</body>
</html>
```

Figure F.42: Code - UserInfo.jsp, part II.



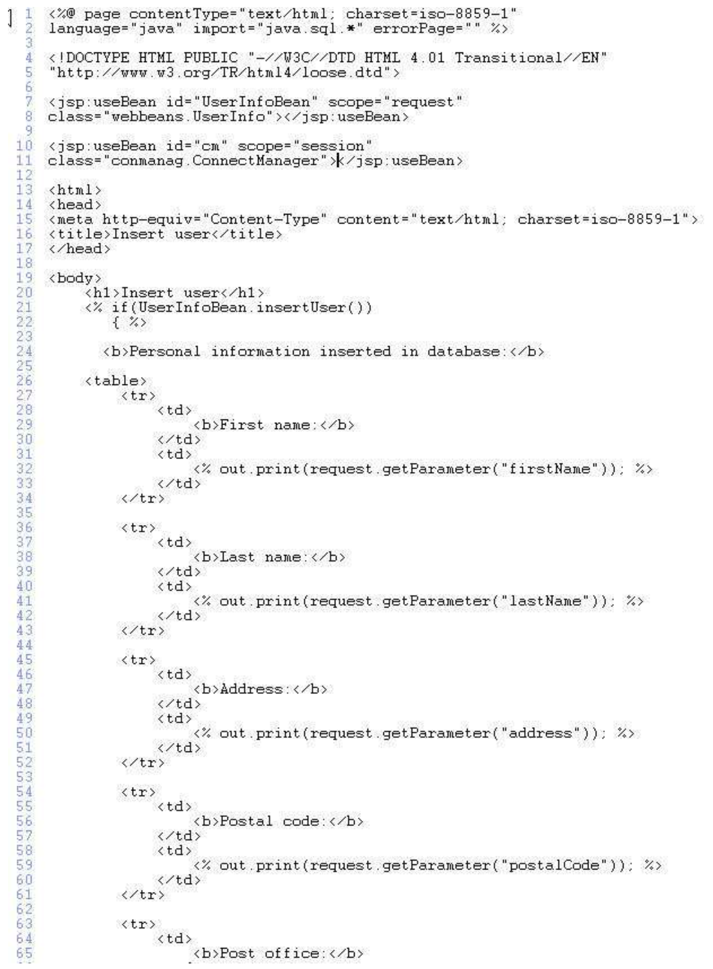
```
<%@ page contentType="text/html; charset=iso-8859-1"
language="java" import="java.sql.*" errorPage="" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<jsp:useBean id="UserInfoBean" scope="request"
class="webbeans.UserInfo"></jsp:useBean>

<% UserInfoBean.initialize(); %>
<jsp:setProperty name="UserInfoBean" property="*">
</jsp:setProperty>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Check users information</title>
</head>

<body>
    <h3>Check on values for UserInfo</h3>
    <table>
        <tr>
            <td>First name</td>
            <td><%= UserInfoBean.getFirstName() %></td>
        </tr>

        <tr>
            <td>Last name</td>
            <td><%= UserInfoBean.getLastName() %></td>
        </tr>

        <tr>
            <td>Address</td>
            <td><%= UserInfoBean.getAddress() %></td>
        </tr>

        <tr>
            <td>Postal code</td>
            <td><%= UserInfoBean.getPostalCode() %></td>
        </tr>

        <tr>
            <td>Post office</td>
            <td><%= UserInfoBean.getPostalOffice() %></td>
        </tr>

        <tr>
            <td>Phonenumber</td>
            <td><%= UserInfoBean.getPhoneNumber() %></td>
        </tr>

        <tr>
            <td>Email</td>
            <td><%= UserInfoBean.getEmail() %></td>
        </tr>

        <tr>
            <td>Login</td>
            <td><%= UserInfoBean.getLogin() %></td>
        </tr>

        <tr>
            <td>Password</td>
            <td><%= UserInfoBean.getPwdMask() %></td>
```

Figure F.43: Code - CheckUserInfo.jsp, part I.



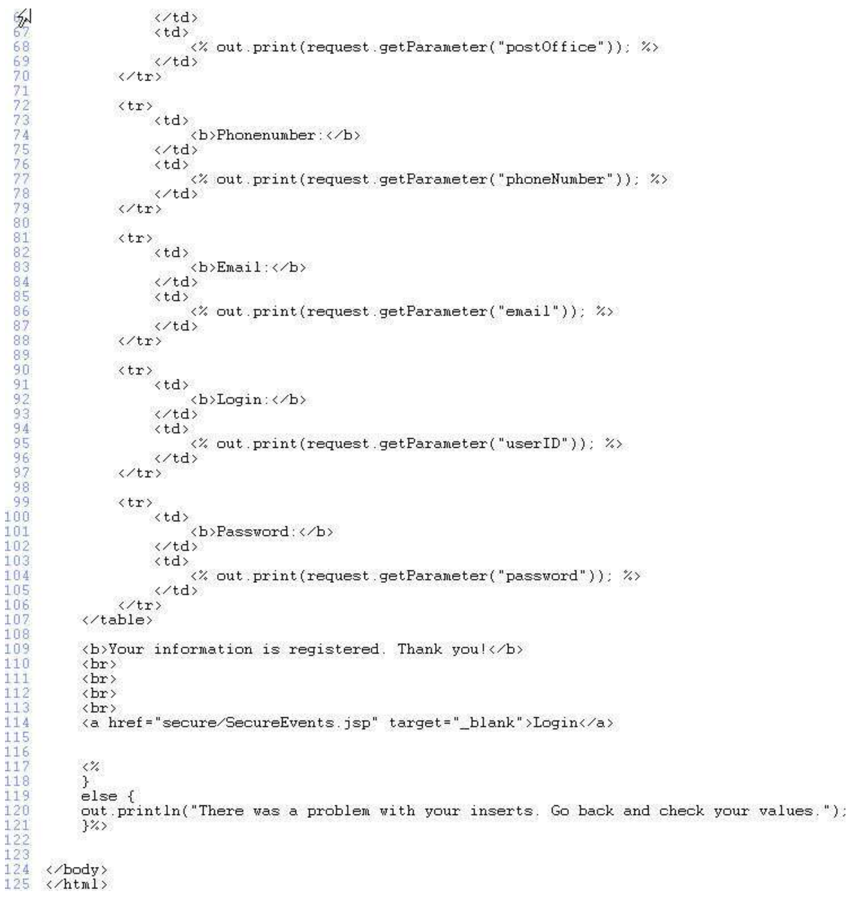
```
        </tr>
    </table>

    <br>
    <br>
    <% UserInfoBean.validate(); %>
    <%= UserInfoBean.getValidationErrorHTMLMsg() %>
    <% if(UserInfoBean.isValid())
       {
    %>

    <jsp:forward page="InsertUser.jsp"></jsp:forward>
    <input type="button" value="Submit User Info"
     onClick="javascript:window.location.href='InsertUser.jsp'">
    <br>
    <br>
    <a href="InsertUser.jsp"> or click here to submit your user info.</a>
    <%
       }else{
        out.println("<BR><A HREF=\"UserInfo.jsp\">
        Back to registration form</A>");
    }
    %>
</body>
</html>
```

Figure F.44: Code - CheckUserInfo.jsp, part II.



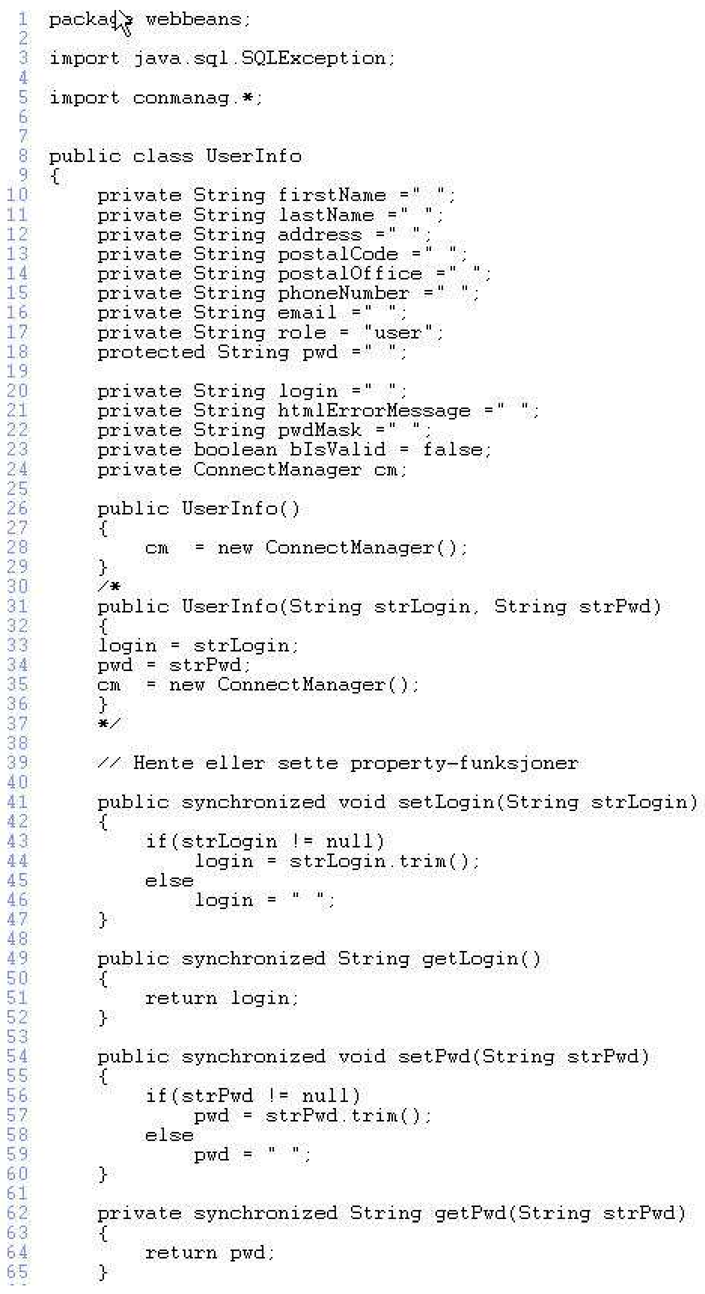
```
<%@ page contentType="text/html; charset=iso-8859-1"
language="java" import="java.sql.*" errorPage="" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<jsp:useBean id="UserInfoBean" scope="request"
class="webbeans.UserInfo"></jsp:useBean>

<jsp:useBean id="cm" scope="session"
class="conmanag.ConnectManager"></jsp:useBean>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Insert user</title>
</head>

<body>
    <h1>Insert user</h1>
    <% if(UserInfoBean.insertUser())
       { %>

      <b>Personal information inserted in database:</b>

    <table>
        <tr>
            <td>
                <b>First name:</b>
            </td>
            <td>
                <% out.print(request.getParameter("firstName")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Last name:</b>
            </td>
            <td>
                <% out.print(request.getParameter("lastName")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Address:</b>
            </td>
            <td>
                <% out.print(request.getParameter("address")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Postal code:</b>
            </td>
            <td>
                <% out.print(request.getParameter("postalCode")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Post office:</b>
```

Figure F.45: Code - InsertUser.jsp, part I.



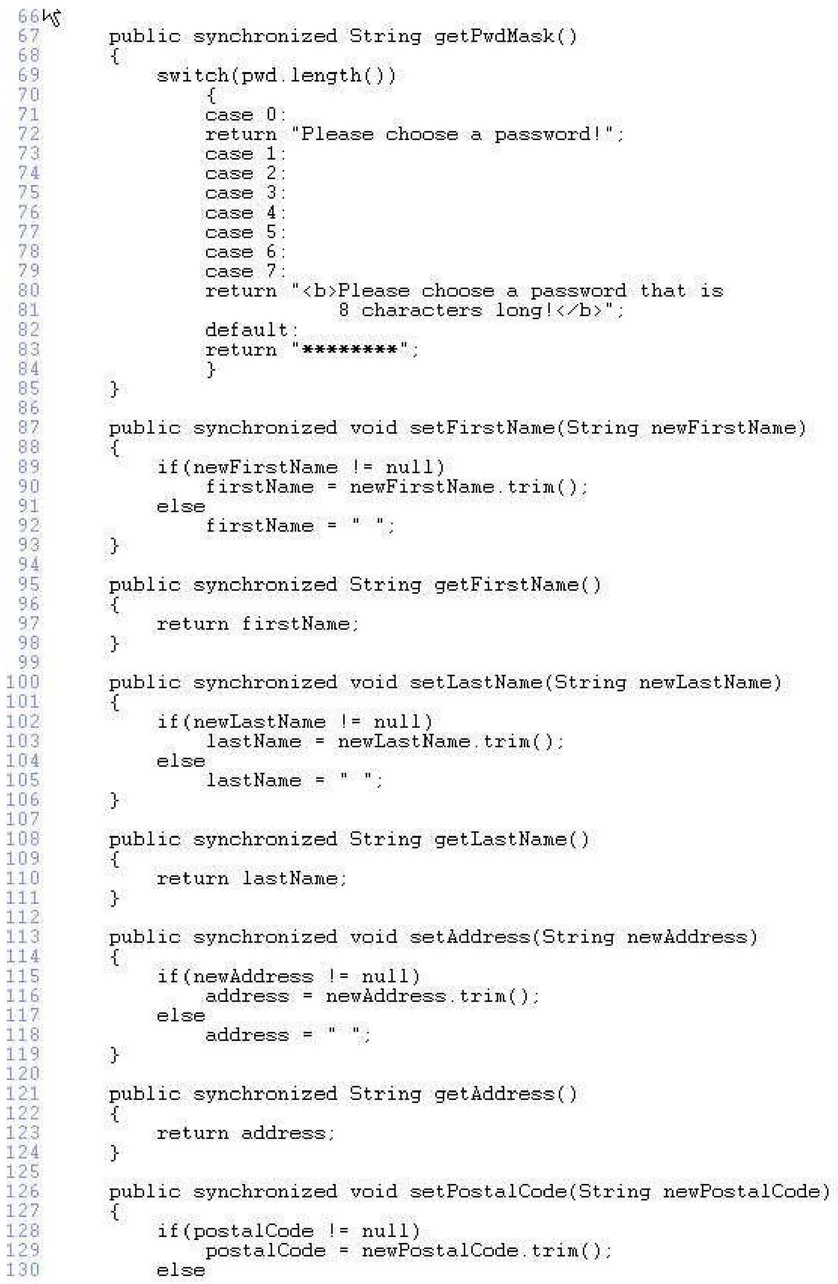
```
            </td>
            <td>
                <% out.print(request.getParameter("postOffice")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Phonenumber:</b>
            </td>
            <td>
                <% out.print(request.getParameter("phoneNumber")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Email:</b>
            </td>
            <td>
                <% out.print(request.getParameter("email")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Login:</b>
            </td>
            <td>
                <% out.print(request.getParameter("userID")); %>
            </td>
        </tr>

        <tr>
            <td>
                <b>Password:</b>
            </td>
            <td>
                <% out.print(request.getParameter("password")); %>
            </td>
        </tr>
    </table>

    <b>Your information is registered. Thank you!</b>
    <br>
    <br>
    <br>
    <br>
    <a href="secure/SecureEvents.jsp" target="_blank">Login</a>


    <%
    }
    else {
    out.println("There was a problem with your inserts. Go back and check your values.");
    }%>


</body>
</html>
```

Figure F.46: Code - InsertUser.jsp, part II.



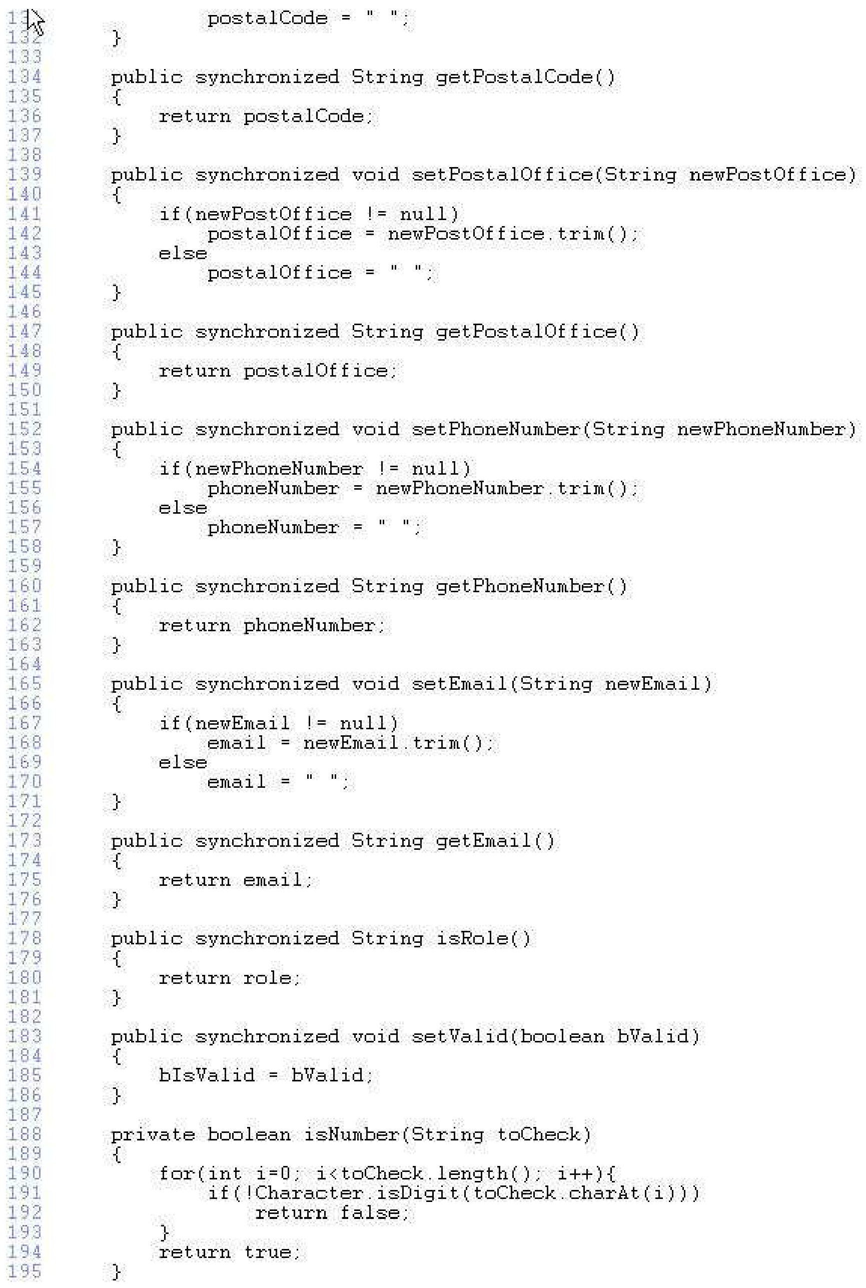
```java
package webbeans;

import java.sql.SQLException;

import conmanag.*;


public class UserInfo
{
    private String firstName =" ";
    private String lastName =" ";
    private String address =" ";
    private String postalCode =" ";
    private String postalOffice =" ";
    private String phoneNumber =" ";
    private String email =" ";
    private String role = "user";
    protected String pwd =" ";

    private String login =" ";
    private String htmlErrorMessage =" ";
    private String pwdMask =" ";
    private boolean bIsValid = false;
    private ConnectManager cm;

    public UserInfo()
    {
        cm  = new ConnectManager();
    }
    /*
    public UserInfo(String strLogin, String strPwd)
    {
    login = strLogin;
    pwd = strPwd;
    cm  = new ConnectManager();
    }
    */

    // Hente eller sette property-funksjoner

    public synchronized void setLogin(String strLogin)
    {
        if(strLogin != null)
            login = strLogin.trim();
        else
            login = " ";
    }

    public synchronized String getLogin()
    {
        return login;
    }

    public synchronized void setPwd(String strPwd)
    {
        if(strPwd != null)
            pwd = strPwd.trim();
        else
            pwd = " ";
    }

    private synchronized String getPwd(String strPwd)
    {
        return pwd;
    }
```

Figure F.47: Code - UserInfo.java, part I.



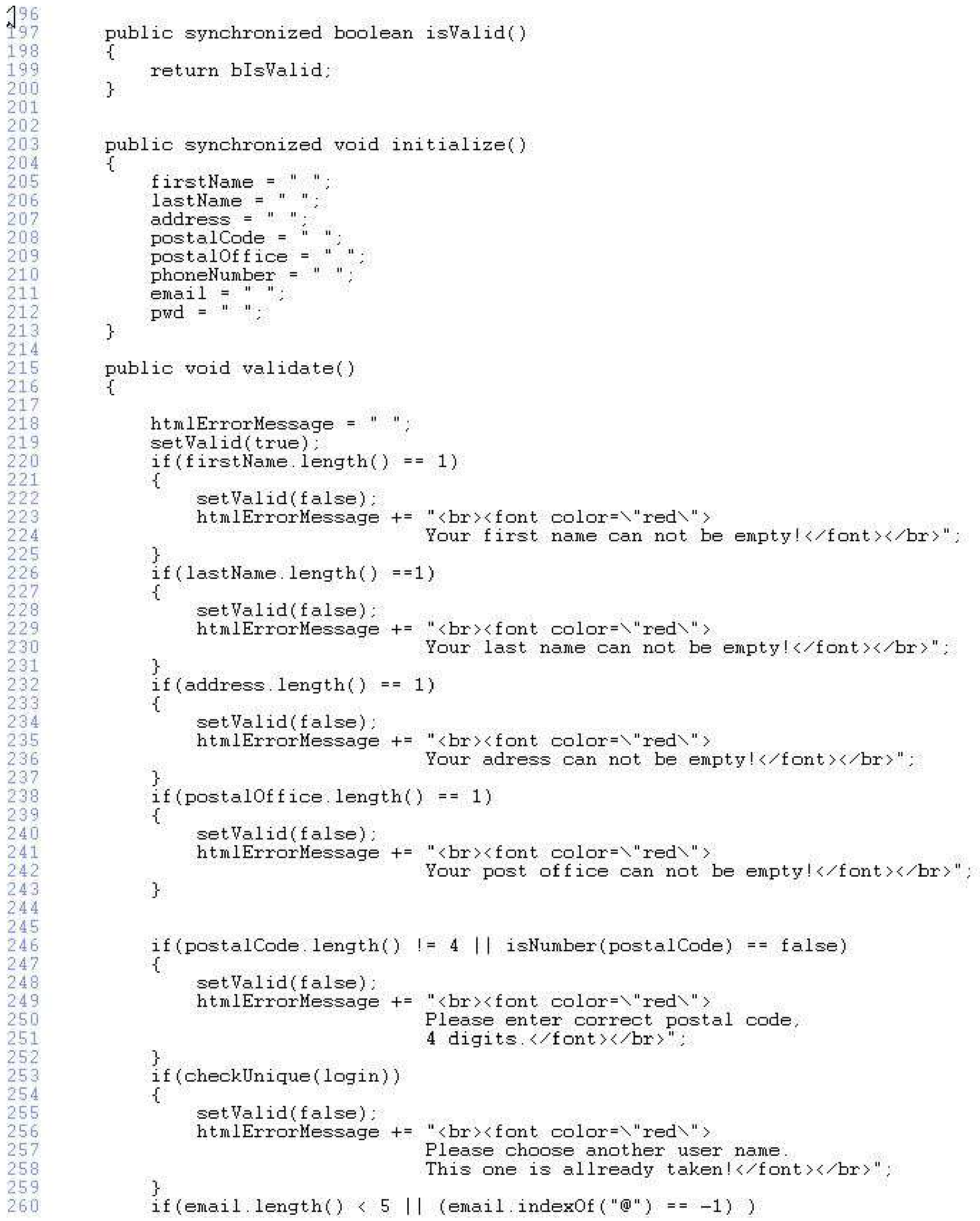
```java

    public synchronized String getPwdMask()
    {
        switch(pwd.length())
            {
            case 0:
            return "Please choose a password!";
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
            return "<b>Please choose a password that is
                       8 characters long!</b>";
            default:
            return "********";
            }
    }

    public synchronized void setFirstName(String newFirstName)
    {
        if(newFirstName != null)
            firstName = newFirstName.trim();
        else
            firstName = " ";
    }

    public synchronized String getFirstName()
    {
        return firstName;
    }

    public synchronized void setLastName(String newLastName)
    {
        if(newLastName != null)
            lastName = newLastName.trim();
        else
            lastName = " ";
    }

    public synchronized String getLastName()
    {
        return lastName;
    }

    public synchronized void setAddress(String newAddress)
    {
        if(newAddress != null)
            address = newAddress.trim();
        else
            address = " ";
    }

    public synchronized String getAddress()
    {
        return address;
    }

    public synchronized void setPostalCode(String newPostalCode)
    {
        if(postalCode != null)
            postalCode = newPostalCode.trim();
        else
```

Figure F.48: Code - UserInfo.java, part II.



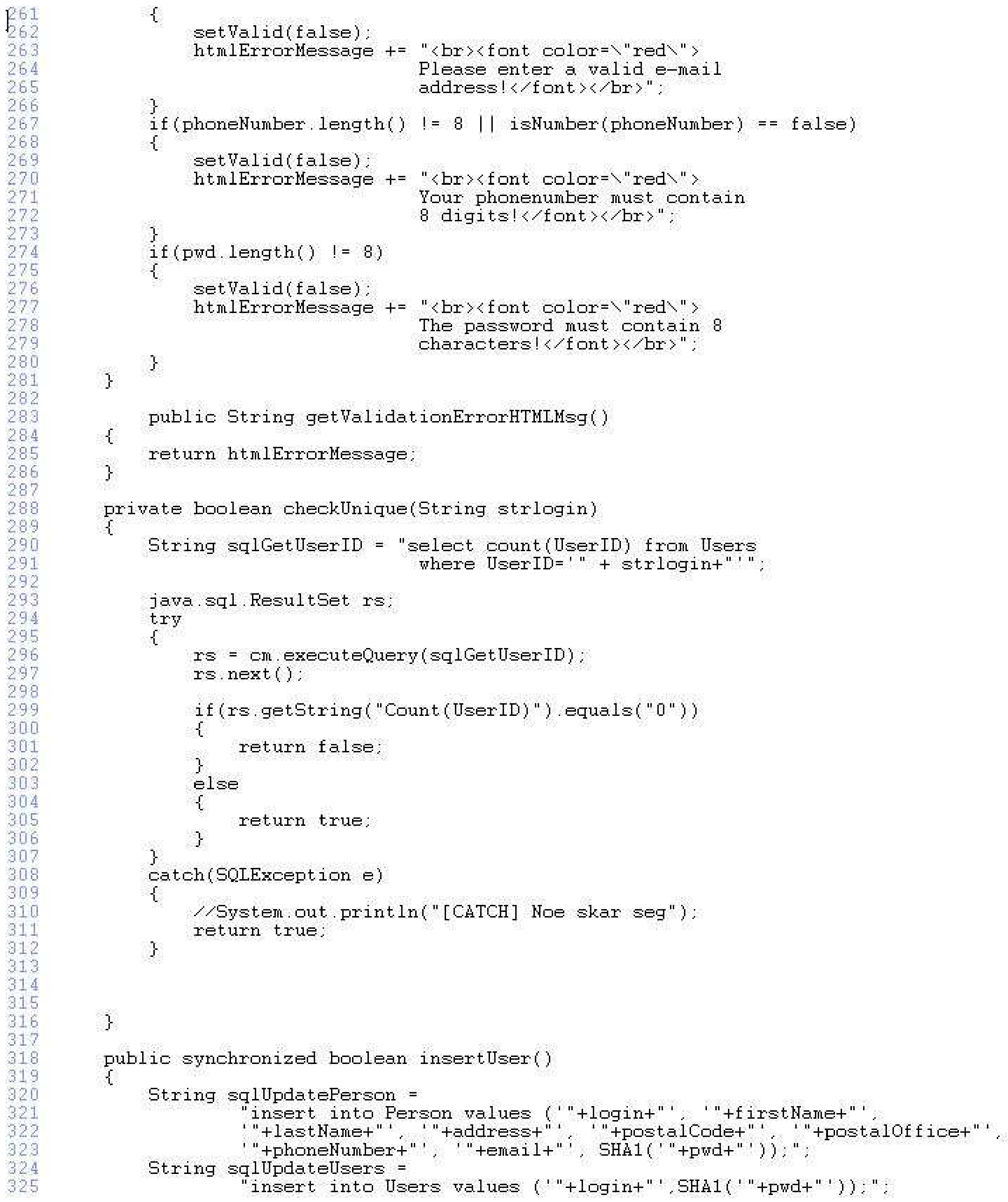
```
            postalCode = " ";
    }

    public synchronized String getPostalCode()
    {
        return postalCode;
    }

    public synchronized void setPostalOffice(String newPostOffice)
    {
        if(newPostOffice != null)
            postalOffice = newPostOffice.trim();
        else
            postalOffice = " ";
    }

    public synchronized String getPostalOffice()
    {
        return postalOffice;
    }

    public synchronized void setPhoneNumber(String newPhoneNumber)
    {
        if(newPhoneNumber != null)
            phoneNumber = newPhoneNumber.trim();
        else
            phoneNumber = " ";
    }

    public synchronized String getPhoneNumber()
    {
        return phoneNumber;
    }

    public synchronized void setEmail(String newEmail)
    {
        if(newEmail != null)
            email = newEmail.trim();
        else
            email = " ";
    }

    public synchronized String getEmail()
    {
        return email;
    }

    public synchronized String isRole()
    {
        return role;
    }

    public synchronized void setValid(boolean bValid)
    {
        bIsValid = bValid;
    }

    private boolean isNumber(String toCheck)
    {
        for(int i=0; i<toCheck.length(); i++){
            if(!Character.isDigit(toCheck.charAt(i)))
                return false;
        }
        return true;
    }
```

Figure F.49: Code - UserInfo.java, part III.



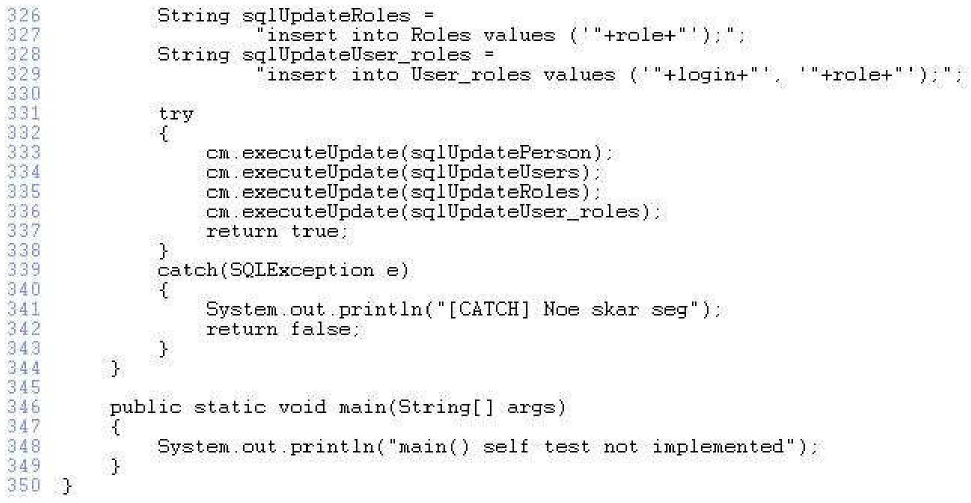
```java
    public synchronized boolean isValid()
    {
        return bIsValid;
    }


    public synchronized void initialize()
    {
        firstName = " ";
        lastName = " ";
        address = " ";
        postalCode = " ";
        postalOffice = " ";
        phoneNumber = " ";
        email = " ";
        pwd = " ";
    }

    public void validate()
    {

        htmlErrorMessage = " ";
        setValid(true);
        if(firstName.length() == 1)
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Your first name can not be empty!</font></br>";
        }
        if(lastName.length() ==1)
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Your last name can not be empty!</font></br>";
        }
        if(address.length() == 1)
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Your adress can not be empty!</font></br>";
        }
        if(postalOffice.length() == 1)
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Your post office can not be empty!</font></br>";
        }


        if(postalCode.length() != 4 || isNumber(postalCode) == false)
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Please enter correct postal code,
                                4 digits.</font></br>";
        }
        if(checkUnique(login))
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Please choose another user name.
                                This one is allready taken!</font></br>";
        }
        if(email.length() < 5 || (email.indexOf("@") == -1) )
```

Figure F.50: Code - UserInfo.java, part IV.



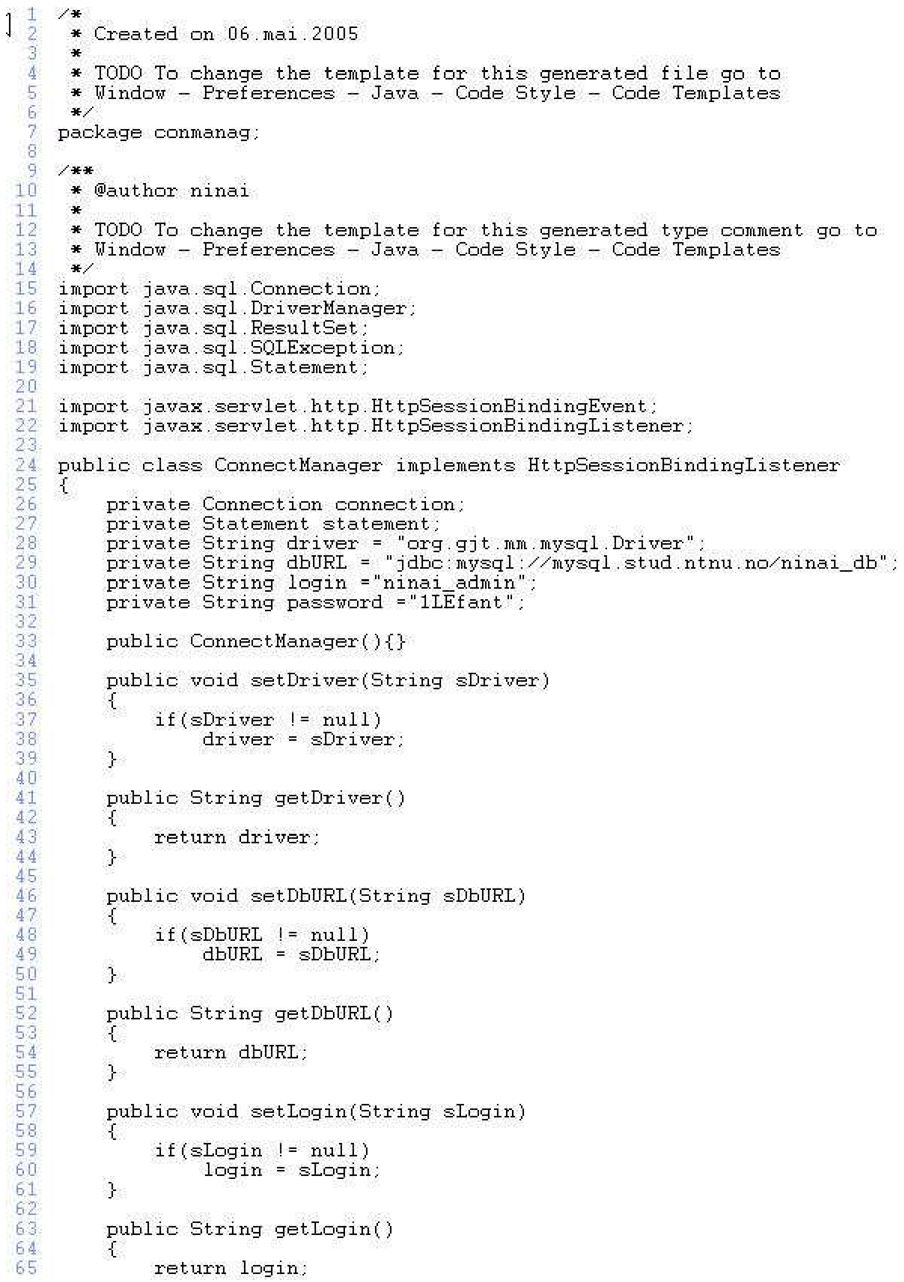
```
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Please enter a valid e-mail
                                address!</font></br>";
        }
        if(phoneNumber.length() != 8 || isNumber(phoneNumber) == false)
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                Your phonenumber must contain
                                8 digits!</font></br>";
        }
        if(pwd.length() != 8)
        {
            setValid(false);
            htmlErrorMessage += "<br><font color=\"red\">
                                The password must contain 8
                                characters!</font></br>";
        }
    }

        public String getValidationErrorHTMLMsg()
    {
        return htmlErrorMessage;
    }

    private boolean checkUnique(String strlogin)
    {
        String sqlGetUserID = "select count(UserID) from Users
                                where UserID='" + strlogin+"'";

        java.sql.ResultSet rs;
        try
        {
            rs = cm.executeQuery(sqlGetUserID);
            rs.next();

            if(rs.getString("Count(UserID)").equals("0"))
            {
                return false;
            }
            else
            {
                return true;
            }
        }
        catch(SQLException e)
        {
            //System.out.println("[CATCH] Noe skar seg");
            return true;
        }



    }

    public synchronized boolean insertUser()
    {
        String sqlUpdatePerson =
                "insert into Person values ('"+login+"', '"+firstName+"',
                '"+lastName+"', '"+address+"', '"+postalCode+"', '"+postalOffice+"',
                '"+phoneNumber+"', '"+email+"', SHA1('"+pwd+"'));";
        String sqlUpdateUsers =
                "insert into Users values ('"+login+"',SHA1('"+pwd+"'));";
```

Figure F.51: Code - UserInfo.java, part V.



```
        String sqlUpdateRoles =
                "insert into Roles values ('"+role+"');";
        String sqlUpdateUser_roles =
                "insert into User_roles values ('"+login+"', '"+role+"');";

        try
        {
            cm.executeUpdate(sqlUpdatePerson);
            cm.executeUpdate(sqlUpdateUsers);
            cm.executeUpdate(sqlUpdateRoles);
            cm.executeUpdate(sqlUpdateUser_roles);
            return true;
        }
        catch(SQLException e)
        {
            System.out.println("[CATCH] Noe skar seg");
            return false;
        }
    }

    public static void main(String[] args)
    {
        System.out.println("main() self test not implemented");
    }
}
```

Figure F.52: Code - UserInfo.java, part VI.

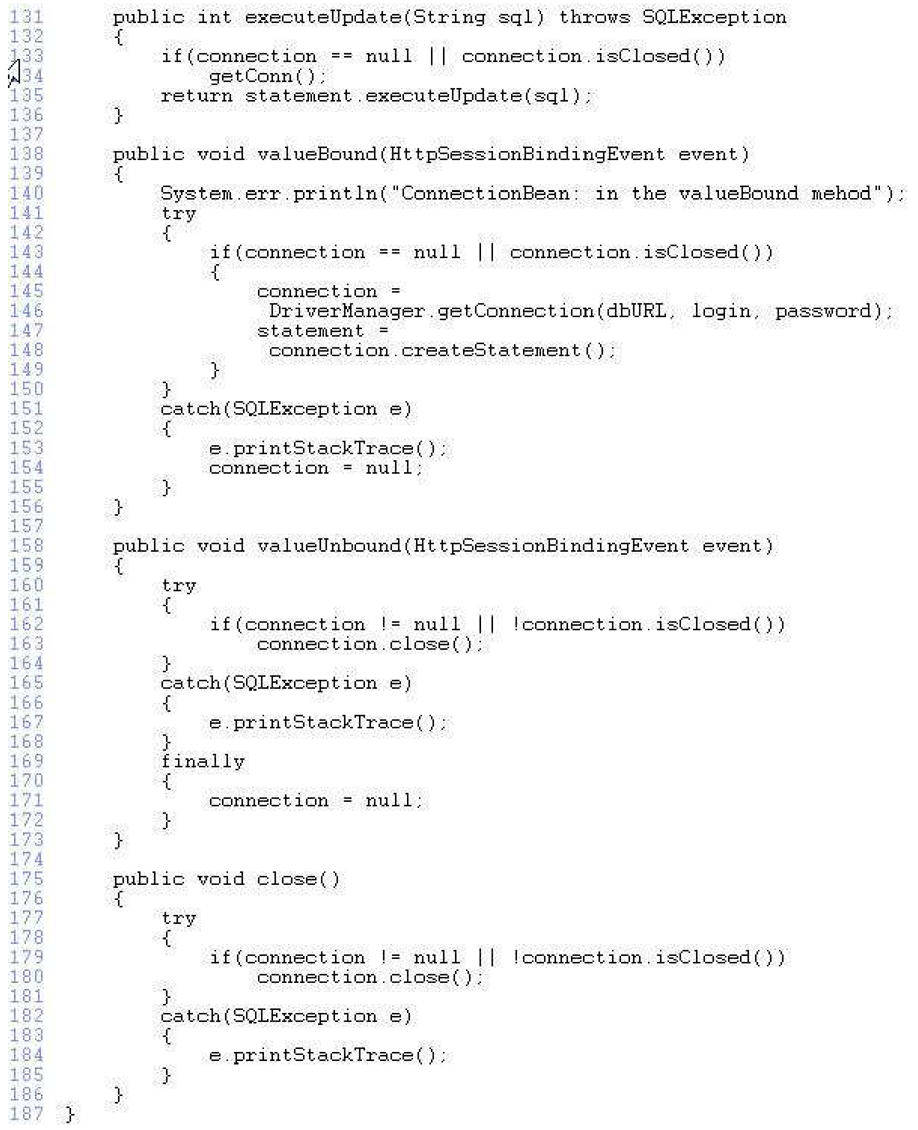


```java
/*
 * Created on 06.mai.2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package conmanag;

/**
 * @author ninai
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionBindingListener;

public class ConnectManager implements HttpSessionBindingListener
{
    private Connection connection;
    private Statement statement;
    private String driver = "org.gjt.mm.mysql.Driver";
    private String dbURL = "jdbc:mysql://mysql.stud.ntnu.no/ninai_db";
    private String login ="ninai_admin";
    private String password ="1LEfant";

    public ConnectManager(){}

    public void setDriver(String sDriver)
    {
        if(sDriver != null)
            driver = sDriver;
    }

    public String getDriver()
    {
        return driver;
    }

    public void setDbURL(String sDbURL)
    {
        if(sDbURL != null)
            dbURL = sDbURL;
    }

    public String getDbURL()
    {
        return dbURL;
    }

    public void setLogin(String sLogin)
    {
        if(sLogin != null)
            login = sLogin;
    }

    public String getLogin()
    {
        return login;
```

Figure F.53: Code - ConnectManager.java, part I.

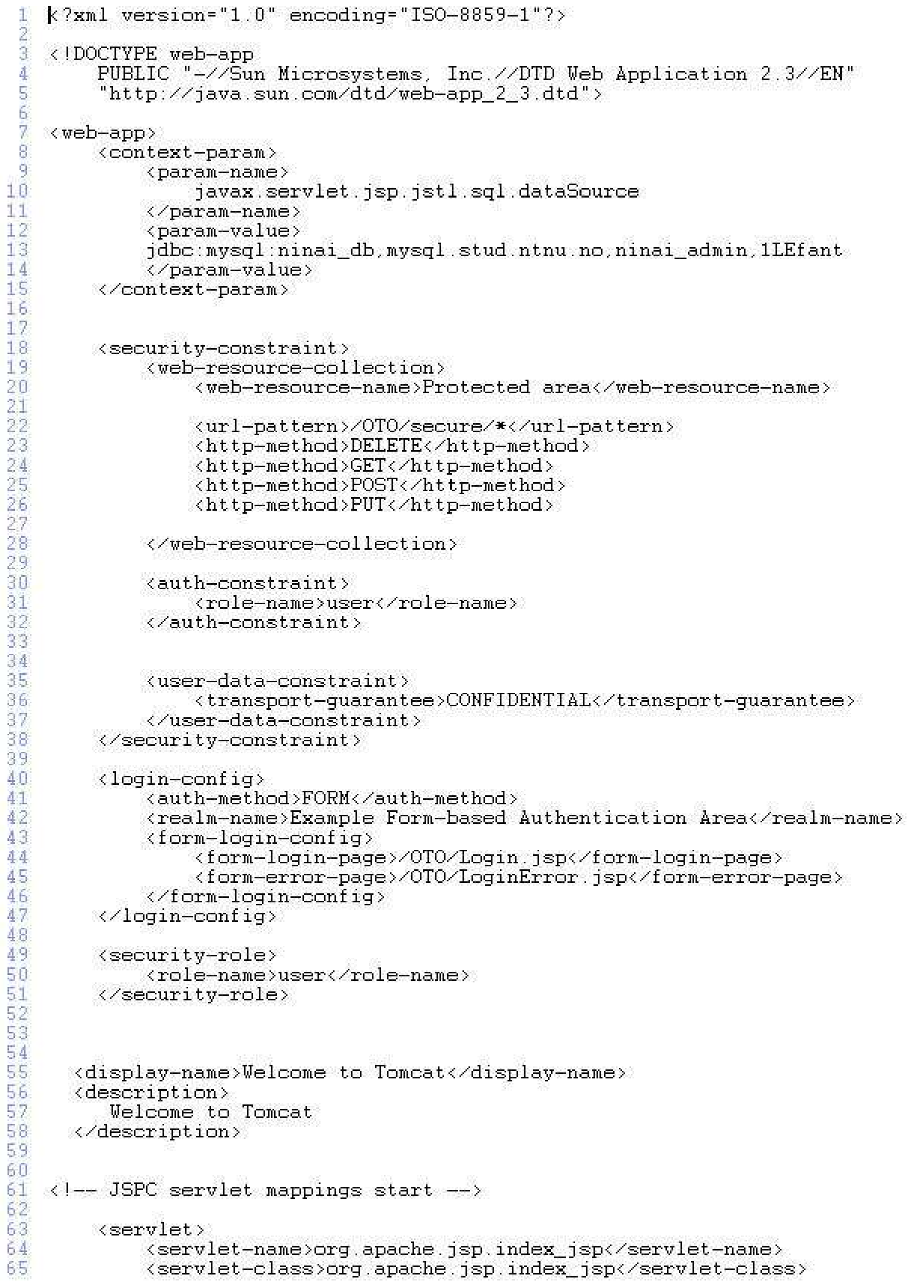
```
    }

    public void setPassword(String sPassword)
    {
        if(sPassword != null)
            password = sPassword;
    }

    public String getPassword()
    {
        return password;
    }

    private void getConn()
    {
        try
        {
            Class.forName(driver);
            connection =
             DriverManager.getConnection(dbURL, login, password);
            statement =
             connection.createStatement();
        }
        catch(ClassNotFoundException e)
        {
            System.out.println("ConnectionManager: driver unavailable");
            connection = null;
        }
        catch(SQLException e)
        {
            System.out.println("ConnectionManager: driver not loaded");
            connection = null;
        }
    }

    public Connection getConnection()
    {
        if(connection == null)
            getConn();
        return connection;
    }

    public void commit() throws SQLException
    {
        connection.commit();
    }

    public void rollback() throws SQLException
    {
        connection.rollback();
    }

    public void setAutoCommit(boolean autoCommit) throws SQLException
    {
        connection.setAutoCommit(autoCommit);
    }


    public ResultSet executeQuery(String sql) throws SQLException
    {
        if(connection == null || connection.isClosed())
            getConn();
        return statement.executeQuery(sql);
    }

```

Figure F.54: Code - ConnectManager.java, part II.



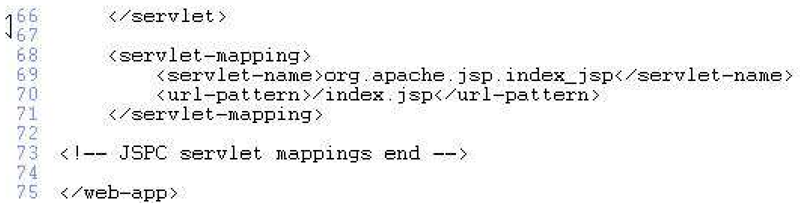
```
    public int executeUpdate(String sql) throws SQLException
    {
        if(connection == null || connection.isClosed())
            getConn();
        return statement.executeUpdate(sql);
    }

    public void valueBound(HttpSessionBindingEvent event)
    {
        System.err.println("ConnectionBean: in the valueBound mehod");
        try
        {
            if(connection == null || connection.isClosed())
            {
                connection =
                 DriverManager.getConnection(dbURL, login, password);
                statement =
                 connection.createStatement();
            }
        }
        catch(SQLException e)
        {
            e.printStackTrace();
            connection = null;
        }
    }

    public void valueUnbound(HttpSessionBindingEvent event)
    {
        try
        {
            if(connection != null || !connection.isClosed())
                connection.close();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        finally
        {
            connection = null;
        }
    }

    public void close()
    {
        try
        {
            if(connection != null || !connection.isClosed())
                connection.close();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

Figure F.55: Code - ConnectManager.java, part III.



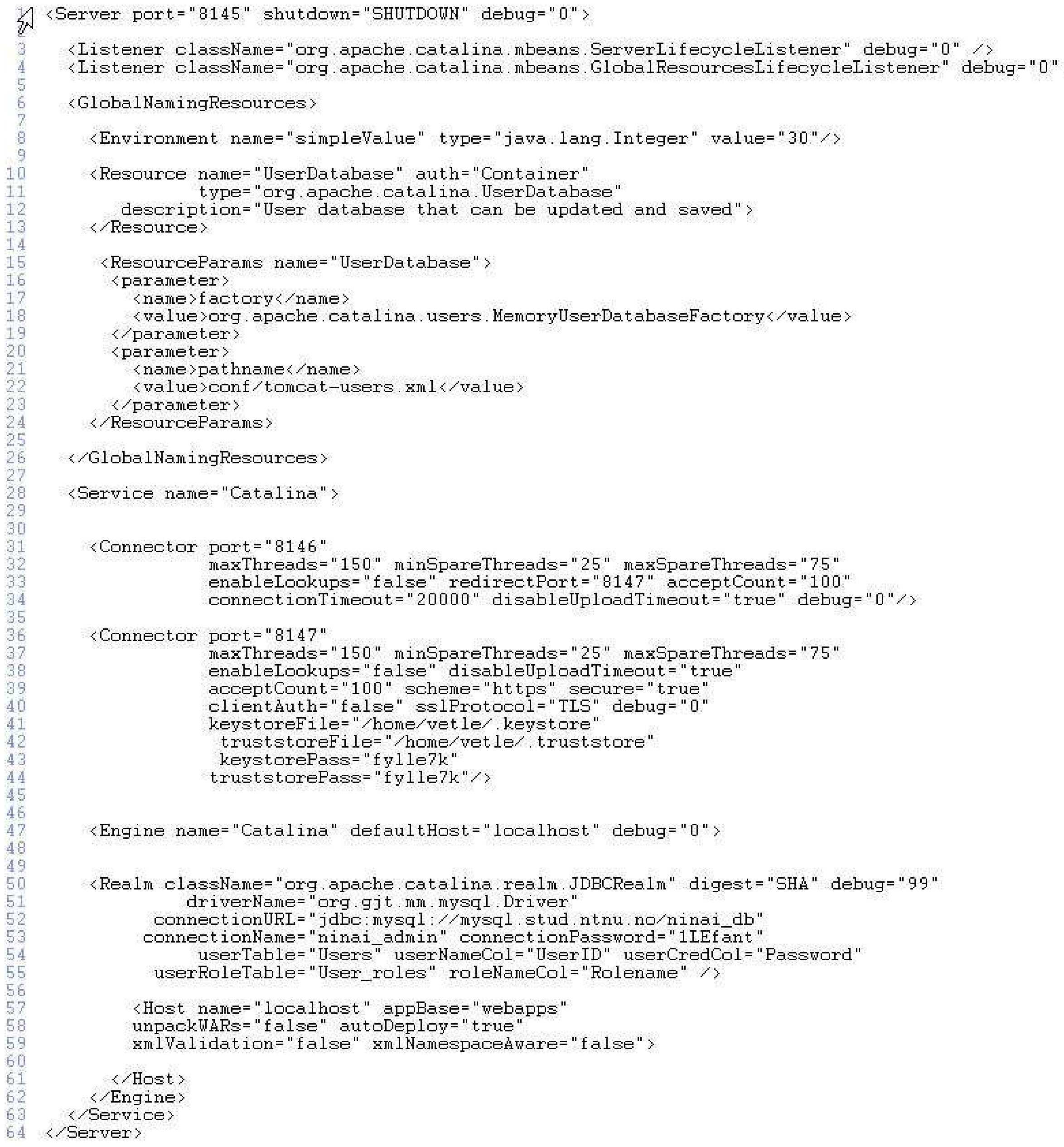
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <context-param>
        <param-name>
            javax.servlet.jsp.jstl.sql.dataSource
        </param-name>
        <param-value>
        jdbc:mysql:ninai_db,mysql.stud.ntnu.no,ninai_admin,1LEfant
        </param-value>
    </context-param>


    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Protected area</web-resource-name>

            <url-pattern>/OTO/secure/*</url-pattern>
            <http-method>DELETE</http-method>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
            <http-method>PUT</http-method>

        </web-resource-collection>

        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>


        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>FORM</auth-method>
        <realm-name>Example Form-based Authentication Area</realm-name>
        <form-login-config>
            <form-login-page>/OTO/Login.jsp</form-login-page>
            <form-error-page>/OTO/LoginError.jsp</form-error-page>
        </form-login-config>
    </login-config>

    <security-role>
        <role-name>user</role-name>
    </security-role>



  <display-name>Welcome to Tomcat</display-name>
  <description>
     Welcome to Tomcat
  </description>


<!-- JSPC servlet mappings start -->

    <servlet>
        <servlet-name>org.apache.jsp.index_jsp</servlet-name>
        <servlet-class>org.apache.jsp.index_jsp</servlet-class>
```

Figure F.56: Code - Web.xml, part I.



```
    </servlet>

    <servlet-mapping>
        <servlet-name>org.apache.jsp.index_jsp</servlet-name>
        <url-pattern>/index.jsp</url-pattern>
    </servlet-mapping>

<!-- JSPC servlet mappings end -->

</web-app>
```

Figure F.57: Code - Web.xml, part II.



```
<Server port="8145" shutdown="SHUTDOWN" debug="0">

  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" debug="0" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" debug="0"

  <GlobalNamingResources>

    <Environment name="simpleValue" type="java.lang.Integer" value="30"/>

    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
       description="User database that can be updated and saved">
    </Resource>

     <ResourceParams name="UserDatabase">
      <parameter>
        <name>factory</name>
        <value>org.apache.catalina.users.MemoryUserDatabaseFactory</value>
      </parameter>
      <parameter>
        <name>pathname</name>
        <value>conf/tomcat-users.xml</value>
      </parameter>
    </ResourceParams>

  </GlobalNamingResources>

  <Service name="Catalina">


    <Connector port="8146"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" redirectPort="8147" acceptCount="100"
               connectionTimeout="20000" disableUploadTimeout="true" debug="0"/>

    <Connector port="8147"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" disableUploadTimeout="true"
               acceptCount="100" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS" debug="0"
               keystoreFile="/home/vetle/.keystore"
                truststoreFile="/home/vetle/.truststore"
                keystorePass="fylle7k"
               truststorePass="fylle7k"/>


    <Engine name="Catalina" defaultHost="localhost" debug="0">


    <Realm className="org.apache.catalina.realm.JDBCRealm" digest="SHA" debug="99"
             driverName="org.gjt.mm.mysql.Driver"
          connectionURL="jdbc:mysql://mysql.stud.ntnu.no/ninai_db"
         connectionName="ninai_admin" connectionPassword="1LEfant"
              userTable="Users" userNameCol="UserID" userCredCol="Password"
          userRoleTable="User_roles" roleNameCol="Rolename" />

        <Host name="localhost" appBase="webapps"
        unpackWARs="false" autoDeploy="true"
        xmlValidation="false" xmlNamespaceAware="false">

      </Host>
    </Engine>
  </Service>
</Server>
```

Figure F.58: Code - Server.xml.

References

[1] JavaTM 2 Platform, Standard Edition, v 1.4.2 API Specification,
http://java.sun.com/j2se/1.4.2/docs/api/,
Date of lookup: 21th of March 2005.

[2] Hans Bergsten. *JavaServer Pages*. O'Reilly and Associates, first edition, 2001.

[3] J2EE Security, Container versus custom,
http://www.javaworld.com/,
Date of lookup: 4th of May 2005.

[4] Cryptography Dictionary,
http://www.cryptnet.net/,
Date of lookup: 8th of May 2005.

[5] Digme AS,
http://www.digme.no/tips/infosikkerhet.html,
Date of lookup: 19th of January 2005.

[6] Steve Fallin and Scott Pinzon. *What we mean by elevation of privileges*. Watchguard Technologies, 2002.

[7] Martin Fowler. *UML Distilled*. Addison Wesley, second edition, 2000.

[8] French Security Internet Response Team,
http://www.frsirt.com/english/,
Date of lookup: 4th of April 2005.

[9] Sverre H. Huseby. *Innocent Code*. John Wiley & Sons, Ltd, first edition, 2004.

[10] J2EE Form-Based Authentication,
http://www.onjava.com/lpt/a/2411,
Date of lookup: 13th of May 2005.

[11] Java Sun - Overview Java,
http://java.sun.com/overview.html,
Date of lookup: 9th of March 2005.

[12] SearchSecurity.com,
http://searchsecurity.techtarget.com,
Date of lookup: 13th of February 2005.

[13] Java Sun Developers page,
http://java.sun.com/,
Date of lookup: 15th of March 2005.

[14] Java Sun - Overview JSP,
http://java.sun.com/products/jsp,
Date of lookup: 1th of March 2005.

[15] MySQL Reference Manual, SSL,
http://dev.mysql.com/doc/mysql/en/secure-requirements.html,
Date of lookup: 9th of June 2005.

[16] Solving the logout problem properly and elegantly ,
http://www.javaworld.com/javaworld/jw-09-2004/jw-0927-logout_p.html,
Date of lookup: 4th of March 2005.

[17] Understanding JavaServer Pages Model 2 architecture,
http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html,
Date of lookup: 9th of March 2005.

[18] MySQL Reference Manual, Encryption functions,
http://dev.mysql.com/doc/mysql/en/encryption-functions.html,
Date of lookup: 9th of June 2005.

[19] The Open Web Application Security Project,
http://www.owasp.org/,
Date of lookup: 14th of February 2005.

[20] The Open Web Application Security Project - The Guide project,
http://www.owasp.org/documentation/guide/guide_downloads.html,
Date of lookup: 30th of January 2005.

[21] The Open Web Application Security Project, Web application security FAQ,
http://www.owasp.org/documentation/appsec_faq.html/,
Date of lookup: 22th of February 2005.

[22] The Centre for Information Security (SIS), Monthly report,
http://www.norsis.no/,
Date of lookup: 14th of May 2005.

[23] Frode Eika Sandnes. *Moderne applikasjonsutvikling med JAVA for web: tynne klienter og fete tjenere.* Tapir akademisk forlag, first edition, 2002.

[24] Secode - Security holes in web applications,
http://www.secode.com/,
Date of lookup: 10th of March 2005.

[25] Secode,
http://www.secode.com/,
Date of lookup: 10th of March 2005.

[26] The Apache Jakarta Tomcat 5 Servlet - JSP Container Realm Configuration,
http://jakarta.apache.org/tomcat/tomcat-5.0-doc/,
Date of lookup: 17th of March 2005.

[27] Java Sun - Introduction to JSP,
http://java.sun.com/developer/onlinetraining/jspintro/,
Date of lookup: 13th of February 2005.

[28] Guttorm Sindre and Andreas L. Opdahl. *Capturing Security Requirements through Misuse Cases*. IDI, 2002.

[29] Guttorm Sindre and Andreas L. Opdahl. *Eliciting security requirements with misuse cases*. Springer-Verlag London Limited, 2004.

[30] The Centre for Information Security (SIS),
http://www.norsis.no,
Date of lookup: 4th of April 2005.

[31] The Centre for Information Security (SIS), On bot nets,
http://www.norsis.no/,
Date of lookup: 14th of May 2005.

[32] William Stallings. *Network Security*. Prentice Hall, second edition, 2003.

[33] Symantec,
http://www.symantec.no,
Date of lookup: 10th of March 2005.

[34] Symantec Internet Threat report,
http://www.symantec.com,
Date of lookup: 10th of April 2005.

[35] Microsoft Threat modelling,
http://msdn.microsoft.com/library,
Date of lookup: 9th of May 2005.

[36] Microsoft - Threats and countermeasures,
http://msdn.microsoft.com/library/,
Date of lookup: 1th of March 2005.

[37] The Jakarta Site - Apache Jakarta Tomcat,
http://jakarta.apache.org/tomcat/index.html,
Date of lookup: 9th of March 2005.

[38] The Apache Jakarta Tomcat 5 Servlet/JSP Container SSL Configuration ,
http://jakarta.apache.org/tomcat/tomcat-5.0-doc/ssl-howto.html,
Date of lookup: 8th of March 2005.

[39] UML,
http://www.uml.org/,
Date of lookup: 11th of November 2004.

[40] Hans Van Vliet. *Software Engineering. Principles and Practice*. John Wiley & Sons, LTD, second edition, 2001.

[41] Arnstein Vestad and Magnus Alsaker. *Sikkerhet i webløsninger - Autentisering og tilgangskontroll*. KITH, 2003.

[42] John Viega and Gary McGraw. *Building Secure Software*. Addison Wesley, first edition, 2004.

[43] Web Application Security Consortium,
http://www.webappsec.org/,
Date of lookup: 14th of March 2005.

[44] Online Computer Dictionary for Computer and Internet Terms and Definitions,
http://www.webopedia.com/data,
Date of lookup: 7th of March 2005.

[45] XML,
http://www.xml.com,
Date of lookup: 20th of May 2005.

[46] Wikipedia,
http://www.wikipedia.com,
Date of lookup: 12th of February 2005.

[47] The Apache XML Project - XML Security,
http://xml.apache.org/security/index.html,
Date of lookup: 17th of April 2005.