Automatic Notification and Execution of Security Updates in the Django Web Framework

Magnus Nermark

Master of Science in Computer Science
Submission date: July 2018
Supervisor: Jingyue Li, IDI
Co-supervisor: Tosin Daniel Oyetoyan, Sintef

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Frameworks are actively used today as a tool to simplify development processes and to create secure and robust tailor made solutions. Using frameworks as the foundation when developing web solutions reduce the time it takes to go from an idea to a finished product, meanwhile allowing the framework to handle potential log-in processes. Problems occur if a security breach is identified in such a framework. If the flawed framework is utilized by multiple websites, these users will be vulnerable to malware or malicious actions by third parties. If the update process for the framework is simplified, it would mean an increase in the update rate by any admin.

In this thesis, research by interviews and observations have been made to identify possible improvements in the update process of the Python-based framework Django. Since 2010, more than 50 holes in the security of this framework have been discovered. Due to a complicated update process, there is reason to assume that there are multiple users on the web today with vulnerable versions of the framework. Therefore, in the work on this thesis, a tool that can be installed on existing Django-applications has been developed and tested. This tool will alert an admin if the current version of the framework is outdated. The tool includes a user interface to help the administrator installing any updates and uncover potential risks by installing the newest version of the framework.

Sammendrag

Rammeverk brukes i dag aktivt som verktøy for å forenkle utviklingsprosesser, og til å lage sikre og robuste løsninger fra bunnen av. Ved å bruke et rammeverk som grunnstein når en skal utvikle en webløsning, vil en kunne redusere tiden det tar å gå fra en idé til et produkt, samtidig som man lar rammeverket håndtere eventuelle innloggingsprosesser. Problemer oppstår dersom det blir oppdaget et sikkerhetshull i et slikt rammeverk. Dersom rammeverket brukes av mange ulike nettsider, betyr dette i praksis at alle som bruker den sårbare versjonen av rammeverket, ansees som potensielle mål for en ondsinnet bruker. Dersom prosessen for å oppdatere rammeverket er gjort enkel, vil dette medføre hyppigere oppdatering av rammeverket fra en administrator sin side.

I denne oppgaven er det blitt gjort undersøkelser i form av intervjuer og observasjoner for å avdekke mulige forbedringer i oppdateringsprosessen til det Python-baserte rammeverket Django. Siden 2010 har det blitt avduket over 50 sikkerhetshull i rammeverket, og det antas, på bakgrunn av en komplisert oppdateringsprosess, at det finnes mange sårbare versjoner av Django ute på nettet i dag. Undertegnede har gjennom arbeid med denne oppgaven utviklet og testet et verktøy som kan installeres i en allerede eksisterende Django-applikasjon, og som vil gi administrator beskjed dersom den gjeldende installasjonen av rammeverket er utdatert. Det er også laget et brukergrensesnitt i denne applikasjonen for å hjelpe administrator til å installere en ny oppdatering, samt for å avdekke eventuelle farer ved å installere nyeste versjon av rammeverket.

Preface

This thesis is submitted as the final of a five-year M.Sc. in Computer Science at the Department of Computer Science (IDI) to the Norwegian University of Science and Technology (NTNU). The work has been carried out in consultation with Jingyue Li as my supervisor and Tosin Daniel Oyetoyan as the external supervisor. I would like to thank my supervisors for good discussions through the last semester and for pushing me to make my work possible. I would also like to thank my roommates for motivating me through the final weeks of the project, and my family and friends for all the support through the last five years. Finally, I would like to thank the board of Spire Consulting for the distribution of work over the last six months, making it possible for me to combine a startup company together with my masters thesis.

Contents

List of Figures x

List of Tables xii

Glossary xiv

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Goal . . . 3
- 1.3 Research Question . . . 3
- 1.4 State of the Art . . . 4
  - 1.4.1 Notify users when a new update is available . . . 4
  - 1.4.2 Verify impact of an update . . . 5
  - 1.4.3 Installing a new update in Django . . . 6
- 1.5 Outline . . . 6

2 Background theory 7
- 2.1 Web Application Framework . . . 7
  - 2.1.1 Framework vs. web framework . . . 7
  - 2.1.2 Open Source Software . . . 9
  - 2.1.3 MVC pattern . . . 9
- 2.2 Python . . . 11
- 2.3 The Django Framework . . . 12
  - 2.3.1 Overview . . . 12
  - 2.3.2 Features . . . 13
  - 2.3.3 Why Django? . . . 14
  - 2.3.4 The architecture of Django . . . 14
  - 2.3.5 Security policy and vulnerabilities . . . 16
- 2.4 Virtual Environment . . . 19
- 2.5 PIP . . . 20
- 2.6 SSH Access . . . 21

2.7 Security Issues Related to the Use of a Framework 22
2.8 CVE 22
2.8.1 Classification of CVE data available 23
2.8.2 Brief description of different vulnerabilities listed in CVE 25
2.9 Crawling websites 26
2.10 Notifications 28
2.11 Mailgun 29
2.12 Cron 30

3 Research Approach and Similar Solutions 33
3.1 Research Question and Research Methodology 33
3.2 Approach and Process 36
3.3 WordPress 37
3.3.1 Security in WordPress 38
3.3.2 Updating WordPress 38
3.4 PyUp-Django 39

4 Results of the Observations and Interviews 43
4.1 Pre-phase Interview 43
4.2 Design 44
4.2.1 First iteration and interviews 44
4.2.2 Second iteration and interviews 47

5 Implementation 51
5.1 Assumptions 51
5.1.1 Python version 51
5.1.2 Dependencies 53
5.1.3 Django version base 53
5.2 Example application 54
5.3 Updating service application 56
5.3.1 update-script.py 56
5.3.2 Models 58
5.3.3 Views 61
5.3.4 Templates 68
5.4 Requirements 69
5.5 Development Environment 70
5.5.1 Installing Python 71
5.5.2 Setting up virtual environment 72
5.5.3 Installing the requirements 72
5.5.4 Setting up a Django Project 73
5.5.5 Installing the Updating Service application 74

5.6 WebFaction 74
5.6.1 VirtualEnv 75
5.6.2 Let's Encrypt 77
5.6.3 Cron 78

6 Evaluation and further development **81**
6.1 Methodology 81
6.2 Evaluation of solution 82
6.3 Further development 83

7 Conclusion and further work **85**
7.1 Conclusion 85
7.2 Further Work 85

Bibliography **87**

A Code-Classes **93**
A.1 update-script.py 93
A.2 views.py 95
A.2.1 imports 95
A.2.2 update_page(request) 96
A.2.3 get_current_django_version() 97
A.2.4 get_all_cve_on_given_version(VERSION_NUMBER) 97
A.2.5 add_base_and_supported_to_database() 98
A.2.6 get_django_supported_and_lts_versions() 99
A.3 Crontab 101
A.4 requirements.txt 101

List of Figures

2.1 MVC pattern overview 10
2.2 Trending programming language on Stack Overflow 12
2.3 Architecture overview [16] 15
2.4 Distribution of security vulnerabilities in Django 17
2.5 Visualization of a virtual environment 20
2.6 Illustration of SSH tunneling 21
2.7 Web scraping illustrated by MyDataCareer [33] 27

3.1 Distribution of the vulnerabilities in WordPress from 2010 to 2017 gathered from CVEdetails.com 38
3.2 Notification given to the administrator in WordPress 39
3.3 One-click update in the WordPress admin panel 39
3.4 PyUP-Django - Example of the administrator panel 40

4.1 Old design of the application 45
4.2 New design of the application 48

5.1 PyCharm survey from 2016 52
5.2 PyCharm survey from 2017 53
5.3 Models in the example application 54
5.4 Main page of the example application 55
5.5 A typical blog post page of the example application 56
5.6 Example of an E-mail sent to the administrator 58
5.7 Model overview of the updating application 59
5.8 Overview of views.py 62

List of Tables

2.1 Advantages and disadvantages of the MVC pattern 11
2.2 Attributes allowed in crontab 31

5.1 Variables and values returned to the template 63
5.2 Brief description of requirements in the project 70
5.3 Useful terminal commands 71

Glossary

API Application Programming Interface

Base version The base version of the Django installation, e.g. 5.4

Current version The current subversion of the base version, e.g. 5.4.3

CVE Common Vulnerabilities and Exposures (CVE)

DRY Don't repeat yourself

HTTP Hyper Text Transfer Protocol

HTTPS Hyper Text Transfer Protocol Secure

JSON JavasScript Object Notation. Language independent lightweight data-interchange format.

MVC Model-View-Controller. A software architectural pattern.

NTNU Norwegian University of Science and Technology

PIP The PyPA recommended tool for installing Python packages.

PyPA The Python Packaging Authority.

Chapter 1

Introduction

The purpose of this chapter is to provide a short overview of the thesis, as well as present the motivation, goal, and research area of this thesis.

Web pages today are made available to everyone. One who desires to create a blog, online store, or an information page does not need to have coding knowledge to get a fully working web page platform to serve its purpose. Today, people could, and are most likely to use a framework, regardless of whether they know coding or not. A framework is a pre-made set of sites which, in many cases serves content from a database to the front page. Depending on the framework used, there are differences in how the framework is set up, what the framework provides, and what kind of programming skills are required by the user.

Common for all frameworks is the handling of user authentication, exchange of information between front-end, e.g. a web form, and back-end, e.g. a database. Some frameworks are template based, either developed by the creator of the framework, or by an external source. Some frameworks make a skeleton site in which the user can place the content into a predefined template, and some only provide functionality regarding secure communication and exchange of information inside the application.

Today there exist many different frameworks to choose from, and the selection is primarily dependent on the programming language that the developer is familiar with. The framework also depends on the site's purpose; however, in recent years, there have been good initiatives to close the gap between frameworks as much as possible. Choosing the appropriate framework mainly depends on the need of the user. Here are a couple of popular frameworks that is in use on the internet today:

- WordPress - PHP-based framework suited for Blogs and Online Stores.
- Struts - Action-based framework in Java with an MVC mindset for creating Java web applications.
- Django - A Python-based framework with an MVC-like architecture for creating a scalable website that can handle heavy traffic.

1.1 Motivation

A framework is a good way to get started when developing a web platform. However, if a framework is vulnerable, e.g., either information can get stolen or an authentication fails, this generally means that every web page that uses that particular framework is vulnerable to the given weakness. Providers or owners of frameworks take security issues seriously. Fixing weakness in the framework as soon as they are discovered is highly prioritized among them. However, industries using the framework are not always quick to update to the latest version. The updating process primarily depends on the framework and how easy it is to patch the system whenever a security fix is released. Some frameworks do this automatically, such as WordPress; however, doing this manually can be a cumbersome and time-consuming process.

After using Django on a personal basis for the last couple of years, my personal experience is that the updating process tends to be forgotten. As the administrator of a Django-powered site, one goes through the setup process when the site is created, and tend to push code to the application whenever a change is made. In Django, if one want to update to the latest version, the process involved is to log on to the server either by SSH or physically accessing the server through a terminal window. Then you would have to check whether there is any update available for your application, pull the whole application to a local repository, and apply the update locally to make sure that the website is working as expected when the update is complete. If the update is successful, one would have to do the same process on the server and hope that the application does not break. This process is complicated and time-consuming. As stated by the Django Software Foundation, by doing an update regularly, the complexity of the updating process decreases and it is easier to do small changes as they come rather than doing a major update after a longer period of time. [1]

The motivation for this thesis is to make the web application patching process easier for the administrator. If the process of installing a new security

patch is made simpler, the threshold for updating the web application to the latest version should be significantly reduced. By doing so, both the users of the web application and the administrator of the site can rely on the service to be up-to-date and safe to use.

1.2 Goal

Security is a common issue for all web-based systems. This study aims to gather information about the most common vulnerabilities in the Django framework and focuses on how the end users can cope with the vulnerabilities by making the updating process for their existing software as easy as possible.

The overall goal for this thesis is to raise awareness about the importance of security patching, and to make the patching process easier for Django users. By providing detailed research in the field of security patching and the updating process, this thesis will hopefully help administrator users of Django in the process of keeping their website up-to-date with the latest security and performance patches. With an automated updating process lowering the threshold for the administrator, the users of the site can trust the service they are using and not to be concerned over the site been outdated or vulnerable.

1.3 Research Question

The pre-study for this research was to gather information from the last eight years and create an overview of the most common vulnerabilities in the Django framework. Now, the concerned area is shifted towards the updating process and how to make it easier for the stakeholders to ensure that the web page is secure and up-to-date. The stakeholders for this study are primarily the system administrators; however, users of the website might also be interested to know that the site itself is as secure and updated as possible. The focus of this research is centered around the Django Web Framework.

RQ1: How to automatically inform web framework users when a new security update is available?

The first area of this work is to notify the stakeholders that their site is

outdated. As of today, Django takes security vulnerabilities seriously and releases security patches for the supported version of their framework whenever a vulnerability or a weakness is discovered. By default, there is no notification mechanism built into Django that tells users that their version of the framework has some security vulnerabilities. The goal is to notify the user whenever a new security patch is released for that given version of the framework.

RQ2: How to analyze the impact of web framework's update on the application using the framework

The second area of this work is to analyze and warn the system administrator of the critical part of an update. The goal is to give the user an overview of what is necessary to complete the update. After giving this overview, the administrator can justify whether the update is necessary, and decide whether he or she wants to invest time to make the update possible.

RQ3: How to perform automatic update of the web framework and the application using the framework

The updating process in Django is a cumbersome process. To lower the threshold for the administrator to keep the site up-to-date, this process should be made easier. The final part of this work is related to automating the updating process as much as possible. Many web frameworks on the market have an easy updating process; however, this is lacking in the Django web framework. By performing security patching regularly, the administrator is one step closer to securing their applications and to maintaining trustworthy applications for the end users.

1.4 State of the Art

The usage of Django has increased significantly in recent years. The setup process has been simplified and the framework has become more reliable than ever. This section describes the state of the art of the Django Web Framework with a focus on the research questions listed in the previous section.

1.4.1 Notify users when a new update is available

Django places security on the top of their priority list. If a security vulnerability is detected in the framework, Django is quick to release a patch

for that security vulnerability if the version is still in the supported range. However, if the user is not reading the development blog on the djangoproject.com website every week, he or she might miss out on a newly released patch for that vulnerability. One can request a subscription to a low traffic notification list that is used whenever Django releases a new security patch [2]. The process of attending such a list is quite extensive, and placement is not guaranteed.

The request is done by sending a detailed email to the Django organization describing your organization and how it meets the requirements for placement on the notification list. By being on this list, the administrator and the organization are the first to know about upcoming security releases and receive detailed information about a security breach.

Another way to get the information is through a more trafficked list called Django-announce. Everyone who is subscribed to this list, will be receiving information directly from Django. However, one does not know whether the information received is relevant to one's application. The messages sent through this mailing list might also be information about new alpha releases, which in many cases is not relevant when concerned about security.

There exists an additional tool that one can use to gather security information about a Django project. The tool is called *pyup-django* and is described in Section 3.4.

1.4.2 Verify impact of an update

As of today, there is no default impact analysis tool for Django to detect the impact of a change. If a system administrator wants to evaluate the impact of a security update, he or she must have a complete overview of the application and know which part of the Django core is been used, either directly or partially, by his or her own developed application.

If the system administrator detects that one component of Django is critical for the application to be operative, he or she must check whether the new changes made by Django can cause the already existing application to not functioning properly. This can be done by searching through information in either the Django documentation, the release notes or the source code. Seeking this information can be a time-consuming process, and it is not displayed in one place. Searching many different sources takes time, and the system administrator must have a good overview of where in the documentation or

code base he or she should search for the relevant information.

1.4.3 Installing a new update in Django

The Django team takes security issues seriously, and is quick to patch their framework when a new security vulnerability is detected. However, the process of installing the update is cumbersome, involving many different steps. If a web application, e.g. a web blog, running in a live environment has a known vulnerability, the steps involved in performing the update are many.

First, the system administrator must log on to the server either through an SSH-connection, or by accessing a terminal on the physical server in another way. Then, as suggested by the documentation of Django, the system administrator must clone the installed version on the server and install it in a local environment. After the local environment is up and running, the system administrator must apply the update locally, and check whether the application is still working after the update. If everything works as expected, the administrator must go back to the server and perform the update in the live environment. This can be done by pulling down and installing the latest version of Django through PIP and specifying the preferred version to be installed. Ideally nothing goes wrong in the updating process, and the server should be patched after the new installation.

1.5 Outline

The outline of the thesis is as follows; Chapter 1 is the introduction chapter describing the motivation behind the research as the research question. Chapter 2 describes the background theory needed for understanding the research better, and to have a basic knowledge of the topic. Chapter 3 describes the method used in the research as well as the questions been targeted followed by the results of the different interviews in Chapter 4. Chapter 5 gives a detailed explanation of the implemented solution as well as how to use it. Chapter 6 shows the evaluation of the solution, method and recommendations for further development. The final chapter, Chapter 7, gives the conclusion of the thesis as well as a recommendation for further research.

Chapter 2

Background theory

The purpose of this chapter is to give a brief introduction to the necessary background theory needed to understand how the challenges related to the research questions are solved.

2.1 Web Application Framework

As mentioned in the introduction, using a framework might help the developers to write components or systems with fewer lines of code, instead of having to write the same thing repeatedly. But the advantage of using a framework is not only to speed up the developing process - there is more to it.

2.1.1 Framework vs. web framework

It takes time to develop a dynamic web application. Fortunately, today there exists software in different forms that already know how to organize a web application in a given format. The most popular architecture pattern used in web frameworks is the MVC-pattern. The goal is to encapsulate data together with its processing and computation (the model) and isolate it from the user interaction (the controller) and different data presentations (the views).

A framework is an integrated set of components that collaborate to produce a reusable architecture for a family of applications. Areas that are specific must be refined by application developers by means of extending existing framework objects to provide application-specific features.

A web application framework is a software framework specifically designed to support the creation of web-based applications. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes the framework to a specific application by sub-classing and composing instances of framework classes. The goal of the web application framework should be the ability for rabid and quality development of a dynamic web application. [3]

Design

Frameworks codify expertise in the form of reusable algorithms, component implementations, and extensible architectures. A good framework can reduce the cost of developing an application by an order of magnitude because it allows for reuse of both design and code. [3]

Benefit

There are some benefits involved when using a framework. According to Mohamed Fayad and Douglas C. Schmid, the primary benefits is *Modularity*, *Reusability*, *Extensibility*, and the *Inversion of Control* to the developers [4]:

- Modularity - Frameworks enhance modularity by encapsulating volatile implementation details behind stable interfaces. Framework modularity helps improve software quality by localizing the impact of design and implementation changes.
- Reusability - The stable interfaces provided by frameworks enhance reusability by defining generic components that can be reapplied to create new applications. Framework reusability leverages the domain knowledge and prior effort of experienced developers to avoid re-creating and re-validating common solutions to recurring application requirements and software design challenges. Reuse of framework components can yield substantial improvements in programmer productivity, as well as enhance the quality, performance, reliability, and interoperability of software.
- Extensibility - A framework enhances extensibility by providing explicit hook methods that allow applications to extend its stable interfaces.
- Inversion of control - The run-time architecture of a framework is characterized by an inversion of control. When events occur, the framework's dispatcher reacts by invoking a hook method, which performs application-specific processing on the event.

2.1.2 Open Source Software

Open-source software (OSS) is a software for which the programming code is available on the Web so that developers can modify and redistribute it [5]. This contrasts with most commercial software, for which the source code is a closely guarded trade secret for a company or organization. The basic idea behind OSS is that, when developers on the Web can read and modify the source code as freely as they wish, the software itself evolves.

For students who are studying computer science, for example, OSS is important for multiple reasons. Most of the OSS out there is free, and students can create an exciting development environment on their personal computers. It is a beneficial way for students to see how the different components of the system communicate with each other and to see how programmers have organized the code. Open-source software is among the best ways for young software developers recieve international recognition for their work by participating in the OSS community. For companies, organizations, governments, and students, OSS provides a way to prevent the widespread illegal copying of software, and an opportunity to raise the level of software development to international standards. [3]

2.1.3 MVC pattern

The model-view-controller (MVC) pattern, was addressed by Glenn E. Krasner and Stephen T. Pope in 1988. They presented the equine of MVC programming in the application as a three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the view), and the user's interaction with the model and the view (the controller). This application structuring paradigm of thinking about interactive components was developed where they saw the possibility of updating one of the components separately to get the interactive aspect. [6]

The *model* in an application is the implementation of the main structure of the application data. The model represents the data of the application. The model holds the information of the necessary fields with the technical specification, as well as the relation between other models.

The *view* deals with everything graphical. The view requests data from the model and display this data to the user. A view is likely to be dynamic, meaning that if an entrance or data in the model is changed, the added or

modified data can be fetched to the view and the view is updated without needing to refresh the page. For example, a button click is registered through the view and the view sends the user action to the controller.

The final part of the MVC pattern is the *controller*. It is used to communicate between the model and the view. The controller handles input from the user through the view, places this in the model and database, and can give feedback to the view regarding whether the communication was successful or not. The controller handles any computation or gathering of data from, for example, a sensor or an API endpoint. The MVC pattern is illustrated in Figure 2.1.

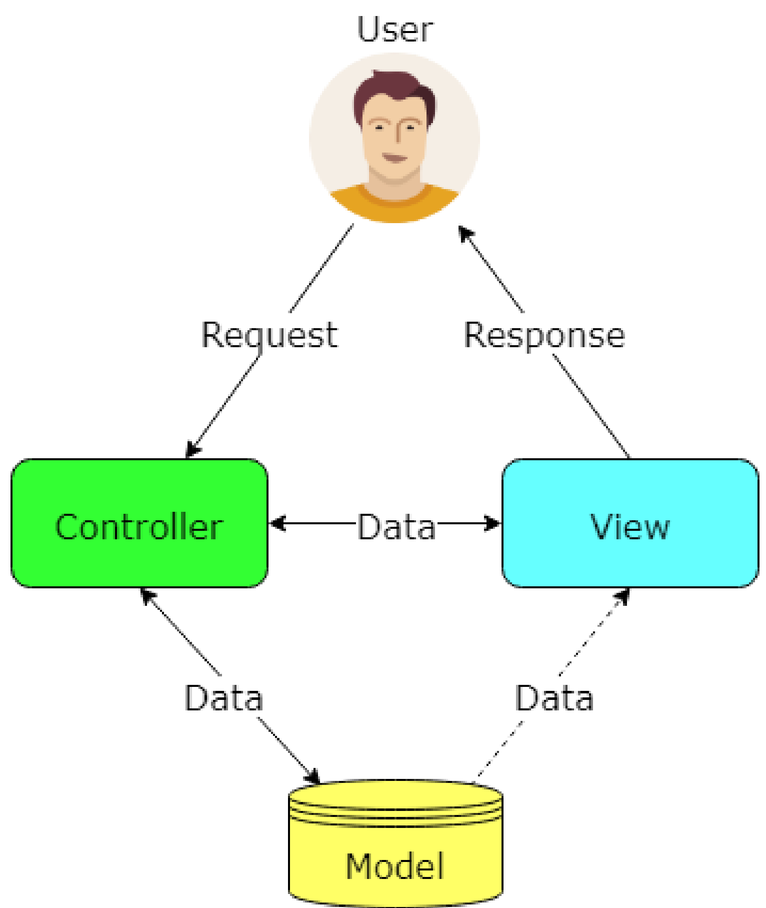


Figure 2.1: MVC pattern overview

As an example of using the MVC-pattern, consider a user who wants to create a Blog application that everyone can access unsing an URL. As soon as the user sends a request to the URL, the server returns the view to the user. This

is what the user will interact with while reading the articles presented by the Blog application. If the user wants to read the next article, the user clicks inside the view. This triggers the controller, which retrieves the data from the database and returns it to the view. Some benefits and disadvantages of using such a pattern are listed in Table 2.1.

Table 2.1: Advantages and disadvantages of the MVC pattern

Advantages	Disadvantages
Makes it easy to have multiple views of the same model, which can be connected and disconnected at run-time.	Increases complexity. May lead to many unnecessary updates for user actions.

2.2 Python

Python is an interpreted, interactive, object-oriented programming language. It provides high-level data structures, such as list and associative arrays (called dictionaries), dynamic typing and dynamic binding, modules, classes, exceptions, automatic memory management, etc. It has a remarkably simple and elegant syntax and yet is a powerful and general purpose-programming language.

Python was designed in 1990 by Guido van Rossum. Like many other scripting languages, Python is free, even for usage in commercial purposes, and it can be run on virtually any modern computer. A Python program is compiled automatically by the interpreter into platform-independent byte code, which is then interpreted. We are running unmodified components written in Python under Linux, Windows NT, 98, 95, IRIX, SunOS, OSF. [7]

An article published in 2017 by data scientist David Robinson illustrates the incredible growth of Python over the last 5 years. [8] The article is based on data gathered from the Stack Overflow database and shows how technologies have trended over time, based on their tags, since 2008, when Stack Overflow was founded. Stack Overflow serves as a platform where users can ask and answer technical computer science questions. The article shows rapid growth in high-income countries such as United States, United Kingdom, and Germany. As shown in Figure 2.2, Python has been growing rapidly in the last few years, and the popularity is still increasing [].

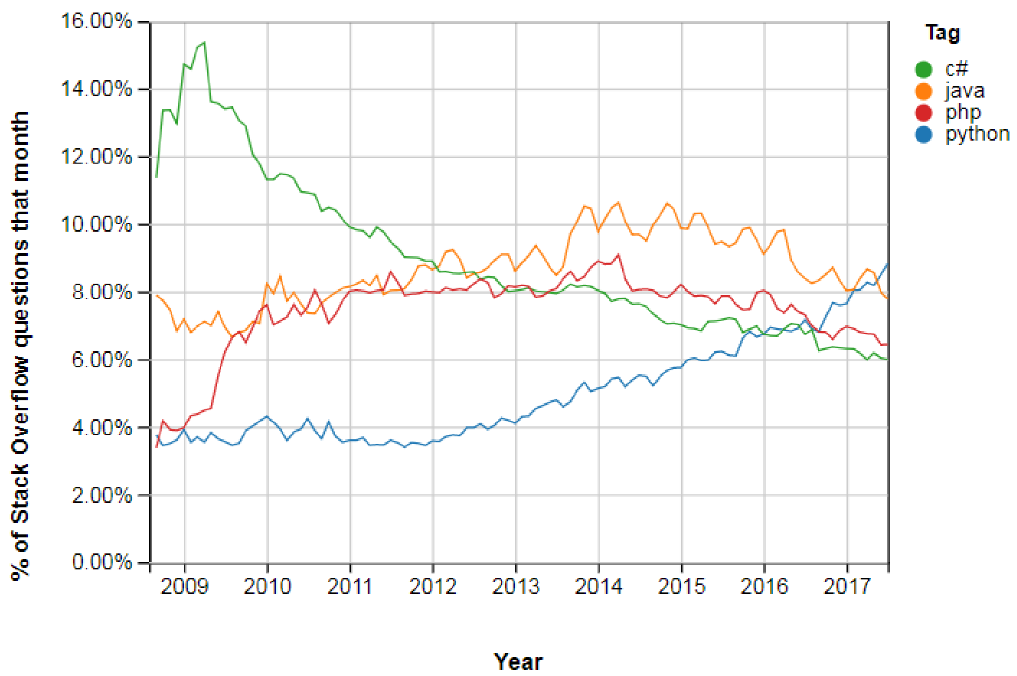


Figure 2.2: Trending programming language on Stack Overflow

The article points out that Java has a popularity spike in the Spring and in the Fall, and has a small drop in the Summer. Robinson suggests that this is due to students that are learning Java at their schools and universities. Python, on the other hand, is stable throughout the year. The trend only shows the frequency of questions asked about that particular programming language; however it is a good indicator of which programming language the users are using.

2.3 The Django Framework

This section provides a brief introduction to the Django framework, its architecture, and its worldwide usage.

2.3.1 Overview

Django is described as follows by the creators:

"The Web framework for perfectionists (with deadlines). Django makes it easier to build better Web apps more quickly and with less code. Django is a high-level Python Web framework that encourages rapid development

and clean, pragmatic design. It lets you build high-performing, elegant Web applications quickly. Django focuses on automating as much as possible and adhering to the DRY (Don't Repeat Yourself) principle" [9].

It is one of the most popular external packages in Python, and has been downloaded over 34 million times [10]. The framework is known globally and is used by many major companies, including Instagram, Pinterest, and National Geographic [11], for running their service. Django was released in 2005 and has since been available to everyone as an open-source project [12].

2.3.2 Features

Django has many features, including templating and automatic generation of database, database access layer, and admin interface generation from a model description given in straight Python code. At the Python Wiki page, they have listed and summarized some functionalities and requirements for using Django [13]. They are shortened and listed as follows:

Deployment Platforms:

mod_python. Has full WSGI support. Comes with a standalone Web service for development purposes.

Suitability:

Django serves many different sites such as Instagram, Pinterest, and National Geographic. Due to its many features, Django can be used, e.g., as a content-management system.

URL dispatching:

URLs are mapped to request handler functions using simple regular expressions.

Environment Access:

Accessed through an HTTP Request object that contains metadata about the request.

Session, Identification, and Authentication:

Sessions are created and managed using cookies. The cookies and the request object are stored in a dictionary.

Persistence Support:

The automatic creation of database tables and database abstraction layer from Pythonic model definition is quite elegant and likely Django's most distinctive feature.

Presentation Support:

Django is using a template language such as: {% block jumbatron %}.

Documentation:

Documentation of Django is phenomenal, and the team behind it updates the documentation continuously.

2.3.3 Why Django?

The team behind *The Django Project* gives an overview of why a developer should consider Django when developing a web application [14]. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Django has been developed and used, since its beginning in 2005, by a Web newspaper operation, The Washington Post, and is well-suited for developing content-management systems [15]. It was designed from scratch to handle the intensive deadlines of a newsroom and the stringent requirements of experienced Web developers. It focuses on automating as much as possible and adhering to the DRY principle. It includes a template system, object-relational mapper, and a framework for dynamically creating admin interfaces. [13]

Ruby on Rails is similar to it, but Django is written in Python and has a few more advanced conveniences for a rapid Web development.

2.3.4 The architecture of Django

Django uses a "shared-nothing" architecture, which means one can add hardware at any level – database servers, caching servers, or Web/application servers. The framework cleanly separates components such as its database layer and application layer. It addition, it ships with a simple-yet-powerful cache framework [12].

Django is an MVC framework (MTV- model, template, view). The model is the database structure, the view is the template (how content is shown), and the controller is the view (how and which information flows with an URL dispatcher). An overview is shown in Figure 2.3.

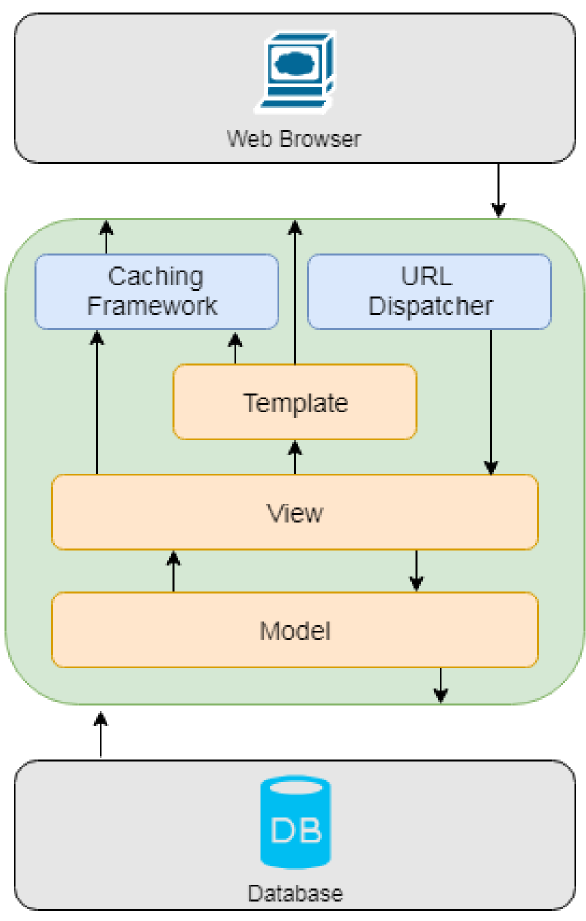


Figure 2.3: Architecture overview [16]

Caching Framework

If caching is enabled, the caching framework is used. When a request is made, the view can check the cache to see whether a version is already stored there. If so, it can skip all other steps involved and return the cached version back to the web browser.

URL Dispatcher

The URL Dispatcher is used to map the requested URL to the associated view function and calls it.

Template

A template returns an HTML page. It contains a *Simple-to-Learn* syntax and provides programmable HTML for the users, such as *if* and *for* statements.

View

The view performs the action requested by the URL Dispatcher. This typically involves operations such as reading or writing to the database.

Model

The model defines the data in Python and interacts with it. In Django, the model-instances are created and can be accessed as objects by the application.

Web Browser

The web browser is what the user is using when interacting with a web page. The web browser initiates the handshake with the server and makes a request to the URL Dispatcher.

Database

The database is where all user data is stored and accessed when needed. This is an external part, and is not implemented directly by the Django team.

2.3.5 Security policy and vulnerabilities

Django has a solid security protocol. The protocol states that the security issues should be reported through email, and not through the public Trace instance. This is due to the sensitive nature of security issues [17]. The Django Project has its own priority of security levels, which are as follows:

- **High:** *Remote code execution* and *SQL injection*
- **Moderate:** *Cross site scripting (XSS)*, *Cross site request forgery (CSRF)*, and *Broken authentication*

- **Low:** *Sensitive data exposure*, *Broken session management*, *Invalidated redirects/forwards*, and *Issues requiring an uncommon configuration option*

The Django team provides official security support for several versions of Django. They are supporting the two most recent version released, meaning that if Version 5.5 is the latest release, Versions 5.3 and 5.4 will also get security updates. Django also releases an LTS (long-term support), which will receive security updates for 3 years from its first release. As mentioned in a blog post in 2015, the team at Django have a feature release scheduled every 8 months, meaning that they will evolve from, e.g., *5.3* to *5.4*, and will have a new long-term support release every 2 years [18]. Security updates are released to the supported versions as soon as they are patched and evolve from, e.g., *5.3.4* to *5.3.5*.

Despite the fact that Django takes security seriously, they have had numbers of security vulnerabilities, like many other frameworks. Data gathered from the CVEDetails website [19] from 2010 through October of 2017 show that Django has suffered through over 50 different vulnerabilities and weaknesses over recent years. The main vulnerabilities and weaknesses have been related to DoS – Denial of Service - and XSS – Cross-site Scripting. This is shown in Figure 2.4.

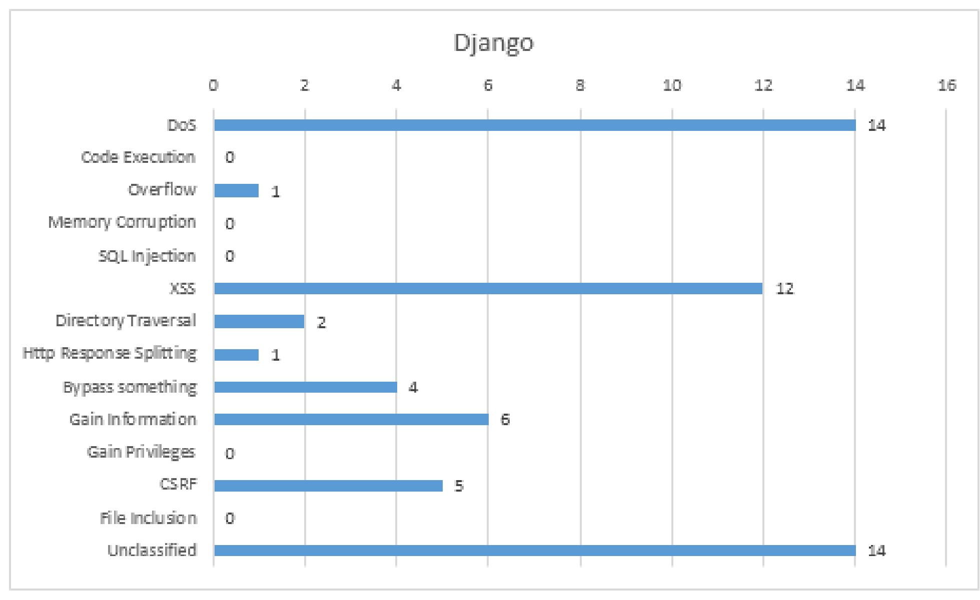


Figure 2.4: Distribution of security vulnerabilities in Django

This means that potentially thousands of websites that are running Django have most likely been vulnerable to these attacks. The only way to ensure that the sites are not suffering from these weaknesses is to always patch the application, either to the latest version, or at least to the latest LTS version.

Structure in a project

If a project is developed from scratch, or reused from another developer, the project should have the following structure:

```
1  myproject/
2  ...app1/
3  ......templates/
4  .........app1/
5  .........base.html
6  .........content.html
7  ......static/
8  .........css/
9  .........img/
10 .........js/
11 ......admin.py
12 ......apps.py
13 ......forms.py
14 ......models.py
15 ......tests.py
16 ......urls.py
17 ......views.py
18 ...app2/
19 ...myproject/
20 ......settings.py
21 ......urls.py
22 ...manage.py
```

myproject is the name of the root folder. *manage.py* is the file used for running the test server, make the migrations between models and the database and can be used to run tests on. This file uses all the settings that are listed inside *settings.py*. If an external app, in this case, *app1*, is going to be installed, all one have to do is to add 'app1' into the settings file variable named *INSTALLED_APPS*. This is a dictionary containing all the installed apps connected to the application. The *urls.py* file located inside *myproject* keeps track of all the URLs allowed in the application. If *app1* is installed, the URLs from that application should be included into the main *urls.py* file

of the project. An example of this is demonstrated in subsection 5.5.5.

2.4 Virtual Environment

When developing a Python application, the need for external packages and modules that are not part of the standard library is often the case. The developer can use external libraries to, e.g., pull data from an API, send an email through an external provider, or transform data into the preferred format. Sometimes the application will need a specific version of that external package or module to work properly with, for example, a specific bug fix in that given version.

If, for example, a server contains two different Python-based applications, and both are using the external library *requests* which allows the users to send organic, grass-fed HTTP/1.1 requests, without the need for manual labor [20], the scenario might be that application 1 might need Version 1.0 of the external library, and application 2 might need Version 2.0 to work properly. This will cause conflict in the requirements for running the applications, and installing either Version 1.0 or 2.0 will leave one application unable to run [21].

By creating a virtual environment for each of the applications, the conflict in the requirements is eliminated, and each of the applications will be able to have their external packages and modules installed separately inside that virtual environment. This is illustrated in Figure 2.5.

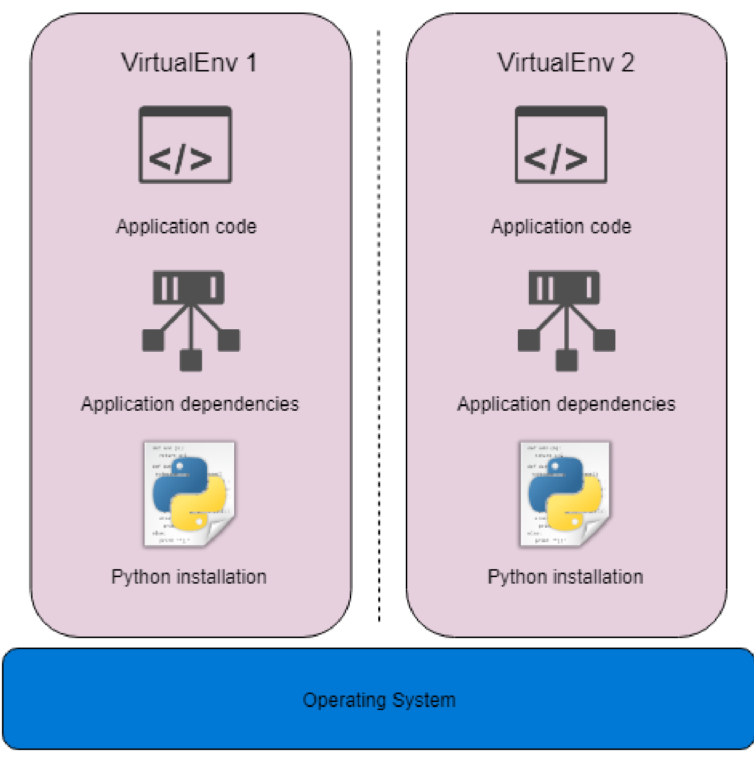


Figure 2.5: Visualization of a virtual environment

If application 2 needs to upgrade the library to Version 3.0, this could be done inside the virtual environment and will not affect application 1, as this application is running Version 1.0 inside the virtual environment.

2.5 PIP

PIP is a package management system used for installing and managing external packages in Python [22]. PIP is installed by default in the latest release of Python 2 and Python 3. Most of the packages that can be installed are found on the official third-party software repository for Python, the Python Package Index (PyPI).

PIP can be used from the command line when it is installed by the following commands:

```
1 $ pip install PACKAGENAME
```

This installs the latest version available for the package. *PACKAGENAME* can be changed to any packages listed in the repository [23], e.g., *Django*. If

a user wants to install a specific version of a given package, this can be done by specifying the version number:

```
1 $ pip install Django="1.11.13"
```

2.6 SSH Access

The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network [24]. Secure Shell is a multi-channel security protocol running over the Transmission Control Protocol (TCP), which offers channels for several services over a secured connection, such as remote shells and connection forwarding [25].

After Taty Ylönen and his university were victims of a password sniffing attack in the early 1990s, Ylönen started the development of SSH [26]. The first release came in July of 1995, and the response in the security community was positive. By the end of the year, SSH had around 20,000 users [27].

To create the secure connection using SSH, we need to have 1. a target server, typically offering services such as HTTP, VPN, etc., 2. an SSH server for the user to connect to, and 3. an SSH client which forwards the traffic through the SSH tunnel [28].

This is illustrated in Figure 2.6, provided from Berkeley University.

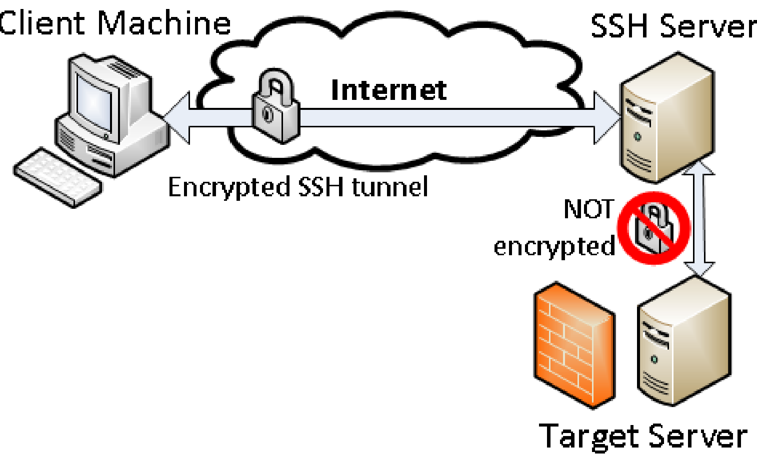


Figure 2.6: Illustration of SSH tunneling

As the figure above illustrates, the connection between the SSH server and the target server is a non-encrypted connection, meaning that the SSH server and the target server must either be on the same server, or placed in a secure internal network.

Typical usage of SSH is when a user wants to manage his or her external web server. The user can securely access the server through SSH and is able to run shell commands or to configure the server as he or she wants. The syntax for connecting to an external server is:

```
1 $ ssh username@example.com
```

2.7 Security Issues Related to the Use of a Framework

As mentioned in the introduction, the use of frameworks has grown over recent years, becoming a common approach when wanting to create a webpage or platform. This is to speed up the developing process and not have to write everything from scratch every time one wants to create a new web page or platform. The framework makes the process of creating a web page or platform easier, and handles the issue of authentication/authorization, rendering web pages, and storing data efficiently and securely in a database.

However, if a framework has a vulnerability associated with it, this vulnerability is distributed among all of the services using this framework and potentially many millions of websites across the Internet.

As part of the pre-study for this thesis, I gathered data from the CVEdetails.com website. The data was gathered from 2010 to October 2017, and focuses on the four most used frameworks for web development. To understand the data that was gathered, let us have a look at the classification of CVEs given by the CVEdetails website.

2.8 CVE

Common Vulnerabilities and Exposures (CVE) is a list of common identifiers for publicly known cybersecurity vulnerabilities [29]. The goal of CVE is to make it easier to share data across separate vulnerability capabilities such as tools, databases, and services with these definitions.

In 1999, most cybersecurity tools used their own database with their own naming of vulnerabilities and security exposures. This lead to a lack of coverage and no unified method for keeping track of different security errors. Each tool used different metrics to state the number of vulnerabilities or exposures they detected, which meant there was no standardized basis for evaluation among the tools. Common Vulnerabilities and Exposures, with its standardized identifiers, provided a solution for these problems [29]. Common Vulnerabilities and Exposures is now the industry standard for exposure and vulnerability identification.

2.8.1 Classification of CVE data available

Whenever a vulnerability is discovered and reported, that vulnerability gets a CVE tag, and the CVE is broken down and classified. The classification of a vulnerability listed on the CVEdetalis.com website [19] is as follows:

This is a counter for the total number of vulnerabilities in the framework. Depending on sort filters, this number is not unique for each CVE in the framework or software.

CVE ID

This is a unique ID that the vulnerability has been assigned. The ID is generated based on when the vulnerability is detected, e.g., CVE-2017-7233, where CVE-2017 indicates that the CVE was listed in year 2017, and the number after, 7233, is assigned as the number in the series of CVE weaknesses that year. In this case, this CVE is the number 7233 detected in 2017.

Vulnerability Type(s)

This is the classification done by the CVE organization. The classification does not distinguish between weakness, vulnerability, and impact. See subsection 2.8.2 for a complete description of the different vulnerabilities.

Publish Date

This is the date when the CVE was first announced and listed on their website.

Update Date

The date the CVE was last updated; in most cases, it gives an idea of when the vulnerability or exploit was fixed.

Score

After a CVE is issued, it is computed a score based on the vulnerability that the weakness or attack provides. There are many factors that go into the process of computing the score, such as access level, complexity of the exploit, and the level of confidentiality, integrity, and/or availability the vulnerability provides. The score ranges from *1* to *10*.

Gained Access Level

This gives a description of the gained access level an attacker gets if the given vulnerability is exploited. In most cases the Gained Access Level is *None*, but it might also be *User* or, in some rare cases, even *Admin*.

Access

What kind of access is necessary to perform or exploit the given CVE. Typically, the Access needed is *Remote*.

Complexity

Complexity is how complex the vulnerability is to exploit, ranging from *low* to *high*. If there is low complexity involved, it means that, potentially, people with less technical skills can exploit and take advantage of this vulnerability.

Authentication

What kind of authentication is needed to exploit this vulnerability. Typically distinguished between *Not required* and *Single system*.

Confidentiality

This describes whether the confidentiality of the application is compromised, and to what extent. Confidentiality has one of the following attributes: *None*, *Partial*, and *Complete*.

Integrity

Is the integrity of the application compromised, and to what extent? Integrity has one of the following attributes: *None*, *Partial*, and *Complete*.

Availability

Is the availability of the application compromised, and to what extent? Availability has one of the following attributes: *None*, *Partial*, and *Complete*.

Comment

Comment is the official description provided to classify the vulnerability, and to give a short summary of the problem.

2.8.2 Brief description of different vulnerabilities listed in CVE

The vulnerability is also assigned to a *Vulnerability type* as part of the classification. Definitions of the different vulnerabilities are gathered from the OWASP 2013 project [30].

- *DoS:* Attack that focused on making a resource unavailable for the purpose it was designed.
- *Code Execution:* Run code on the remote server/platform through, e.g., filename, arguments, etc.
- *Overflow:* Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code.
- *Memory Corruption:* Occurs in a computer program when the contents of a memory location are unintentionally modified; this is termed violating memory safety.
- *SQL Injection:* Insertion of SQL query via the input data from the client to the application.
- *XSS:* Cross-Site Scripting is a type of injection in which malicious scripts are injected into otherwise benign and trusted websites.
- *Directory Traversal:* Aims to access files and directories that are stored outside the web root folder.

- *Http Response Splitting:* Might happen if some data is included in an HTTP response header sent to a web user without being validated for malicious characters.
- *Bypass something:* This is when an attacker can bypass a check of some kind, e.g., bypass using https in browser.
- *Gain Privileges:* Allows a malicious user to modify his or her privileges or roles inside the application to gain unauthorized access.
- *CSRF:* An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.
- *File Inclusion:* Allows an attacker to include a file, usually exploiting "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

2.9 Crawling websites

Web scraping is the set of techniques used to automatically obtain information from a website instead of manually copying it. The goal of a Web scraper is to look for certain kinds of information, extract it, and aggregate it into new web pages [31]. Web scraping automatically extracts data and presents it in a format one can easily make sense of.

In the application that has been developed in this thesis, a Python library is used for scraping web pages. The library is named BeautifulSoup4 and is used for pulling data out of HTML and XML files [32].

By scraping a website, we should be careful not to break the Terms and Conditions of the site we are scraping. Generally, the data we are scraping should not be used for commercial purposes. We should "act like a human" when completing the requests, and be sure not to request data from the website too aggressively with our program. Finally, the layout of a website may change in the future, and it is important to keep this in mind and revisit the site to check, and possible rewrite, our code if necessary. This is illustrated in Figure 2.7.

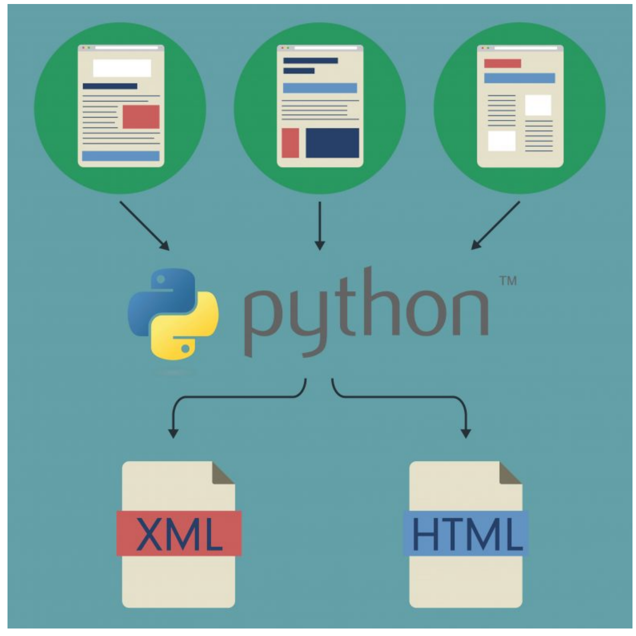


Figure 2.7: Web scraping illustrated by MyDataCareer [33]

How to use it

BeautifulSoup4 (bs4) can be installed through PIP. After the installation, we must import bs4 along with urllib into the file we want to process our crawled data. The process involved when crawling a webpage is as follows:

```
1 from bs4 import BeautifulSoup
2 from urllib.request import Request, urlopen
3
4 req = Request("https://www.example.com/", headers={'
    User-Agent': 'Mozilla/5.0'})
5 webpage = urlopen(req)
6 soup = BeautifulSoup(webpage, 'html.parser')
7 value = soup.find('div', attrs={'class': 'wanted-
    informaiton'})
```

First a request is opened towards the preferred website. Then the opened request is parsed into BeautifulSoup and stored in the soup variable as a bs4 object. Finally, one can search through the soup variable using .find() method which will search through the HTML, looks for a 'div' with the preferred class name, in this case 'wanted-information'.

2.10 Notifications

In modern information systems, notifications or notification systems is a software that has the task of delivering vital information or messages to a set of recipients [34]. Notifications, or alerts, can be delivered to the recipients in many ways such as text message, push notifications, email, letter, a phone call, and so on. The methods for reaching the correct person are numerous, and it is valuable to understand when to use the different methods.

Text message

The first text message that was sent between two devices occurred in December in 1992. The message was sent between two computers, and one year later the first SMS was sent from a phone. Despite the fact that text messages were discovered in early 1990, it was not until early-mid 2000 that text messaging really took off. In 2002, more than 250 billion SMS messages were sent throughout the year, and the service peaked in 2010 when 6.1 trillion were sent over the entire year [35].

Today, the number of text messages has decreased signigicantly, and other instant message (IM) services have taken over. However, text messaging is still present, and advertisers and service provides use direct text marketing to send messages to their users about information, promotions, and reminders.

Push notification

Push notifications are a way for an app publisher to speak directly to the user. In 2009, Apple launched the first push service to their users' smartphones. The response was positive and the support later extended to support other devices in their ecosystem. Google released their version of push notification a year after Apple.

Today, push notification is used in smartphones, smart watches, on computers, and even in cars. The notification is triggered and pushed to the users, typically whenever an event triggers the notification. An example of such an event could be when someone follows another person on Facebook - the Facebook app on one's phone sends out a push notification on the phone or smart watch if the user has allowed this feature.

Email

Email is an electronic postal service which can send and receive documents and messages from one computer device to another over a computer network. Email is sent between parties through an email service, typically SMTP, and it is widely used today on the Internet [36].

Email is a central part of our everyday life, and it is used in many different scenarios. In advertisements, email is used to promote products and services that the recipient might find interesting and hopefully wants to buy after having been exposed to them through email. In a work environment, email is an efficient and structural way to communicate with internal or external parties. Since email is a central part of the business industry, emails tend to be used actively in cyber-attacks against companies. Digital guardian states that as much as 91% of cyber-attacks begin with a phishing email [37].

2.11 Mailgun

Mailgun is a *Transactional Email API Service For Developers* [] which helps with the process of sending out emails. The emails are sent using a POST request to a server located at Mailgun. The API allows up to 10,000 emails to be sent through the service for free every month. Below is an example of how to use it. The request is made towards the API server and authenticated with an API key. The data is then extracted from the request and put into an email format, to then be sent from their mail servers:

```
1 def send_simple_message():
2     return requests.post(
3         "https://api.mailgun.net/v3/samples.mailgun.
             org/messages",
4         auth=("api", "key-3
             ax6xnjp29jd6fds4gc373sgvjxteol0"),
5         data={"from": "Excited User <excited@samples.
             mailgun.org>",
6             "to": ["devs@mailgun.net"],
7             "subject": "Hello",
8             "text": "Testing some Mailgun
                 awesomeness!"})
```

2.12 Cron

Cron, also called cronjob or crontab, is the name of program that enables Unix users to execute commands or scripts automatically at a specified time [38]. The cron, or the job itself, is driven by the crontab file which holds a table over the cron jobs that are scheduled to be executed. When working with web servers, a cron can ensure that the Apache service is started every 20 minutes in case the service has gone down. This gives the system administrator and the people relying on the service a lower possibility for downtime.

The syntax of a cron job has not changed since the '90s and is in the following format:

```
+ ---------------- minute
| + -------------- hour
| | + ------------ day of month
| | | + ---------- month
| | | | + -------- day of week
| | | | |
* * * * * command to be executed
* * * * * command --arg1 --arg2 file1 file2 2>&1
```

Given the specification above, a typical cron can look something like this:

```
1 5 18 * * 0 /path/to/file
```

This means that the file located at */path/to/file* is executed every Monday at 18:05. A full list of the allowed commands is provided in Table 2.2.

Time-variable	**Allowed format**
minute	This controls what minute of the hour the command will run on, and is between '0' and '59'
hour	This controls what hour the command will run on, and is specified in the 24-hour clock; values must be between 0 and 23 (0 is midnight)
day of month	This is the Day of Month that one wants the command run on; e.g., to run a command on the 19th of each month, the dom would be 19
month	(1 - 12) OR jan, feb, mar, apr...
day of week	(0 - 6) (Sunday = 0 or 7) OR sun, mon, tue, wed, thu, fri, sat

Table 2.2: Attributes allowed in crontab

Chapter 3

Research Approach and Similar Solutions

The purpose of this chapter is to give an overview of the research, method as well as an introduction of similar solutions to the one that is developed in this thesis. The examples are partially or directly related to the use of Django as a Web application framework.

3.1 Research Question and Research Methodology

One of the first activities in performing this thesis is the process of defining the research questions. Three research questions were created for this thesis. The first one is "**How to automatically inform web framework users when a new security update is available?**". The second research question that was defined is "**How to analyze the impact of web framework's update on the application using the framework?**". The third and final research question is "**How to perform automatic update of the web framework and the application using the framework?**"

After defining the research questions, the process of finding a good research method began. There are numerous of ways of doing research, however, to target the questions in the right form creating the best possible answer, it is important to select the correct method. The method could be either *observing* how people are acting in the real world, *interviewing* people and asking them how they are coping with certain problems and challenges, *surveying* to measure many recipients and try to discover a trend, and finally, performing *tests* to measure the skills and knowledge of a group of people.

The four different approaches have their advantages and drawbacks [39]:

Observation:

An observation is when one blends in and observes how a task is done, or how people act in the given environment. By observing, one does not influence the person been observed, and tries to map their behavior. Observation can be used in all parts of the research process; however, it is time-consuming and does require some data to be gathered and analyzed.

Interview:

Interview is used when one wants a deeper understanding of what the person is doing or thinking. It is a deeper conversation in which the subject is questioned more in detail than, e.g., in a survey. Interviews can be conducted in three different ways: fully-structured, semi-structured, and unstructured [39]:

A *fully-structured* interview is done when a set of predefined question is asked in the same sequence for all of the interviews and with limited opportunity in the interview to improvise or ask follow-up questions.

A *semi-structured* interview typically has an interview guideline; however, the questions do not need to be asked in the same way as in a fully-structured interview. A semi-structured interview opens for discussion and follow-ups. This interview form has a more dynamic flow to it.

In an *unstructured* interview, there are no prepared questions for the interviewer to ask the subject; however, a theme for the interview is set. It is worth mentioning that in semi- or non-structured interviews, it is important for the interviewer to have good knowledge of the topic at hans. A fully-structured, interview on the other hand, can be performed by someone who barely has prior knowledge of the topic in the interview.

Surveys

Surveys or questionnaires are used when one wants to know what people feel, think, or believe. A survey consists of many questions and, in most cases, predefined answers. The survey can be handed out in person, sent by mail or email, or even performed over a phone call. [40]

Tests

Testing is a method that measures different dimensions of persons, qualities, skills, and other traits that separate people from each other. Tests typically focus on individual differences in answers, achievements, and responses. The test situations are standardized, meaning that everyone taking the test does so under the same test conditions [41]

The method that was best suited for each of the research questions is discussed below. The method that was applied consisted of first observing and researching the questions to see what is already present today, followed by face-to-face interviews.

RQ1: How to automatically inform web framework users when a new security update is available?

The wording of the question is quite accurate, and the goal is to discover a method of notifying the administrator whenever a new security update is available. This can be done either by observing what people generally prefer as their notification method, conducting a survey questioning their perspective, interviewing people for a deeper understanding of their behavior regarding the different notification methods, or performing tests on the users. The tests can be automatic tests detecting the click rate on a notification, or giving the user a task to perform with the notification. In this research, a combination of first observing what kind of notification systems work and which do not, and after the observation conducting interviews to get a deeper understanding about people's preferences, was used. The interviews gave some useful indicators of what kind of notification is preferred by the users. It is worth mentioning that the stakeholders for this question are typically system administrators; however, because of limited resources and networks, the interviews were conducted with students at IDI – Department of Computer Science, NTNU.

RQ2: How to analyze the impact of web framework's update on the application using the framework?

This question is a more advanced one of "identifying the potential consequences of a change, or estimate what needs to be modified to accomplish a change" [42]. The process can be automated, however, developing a reliable system for automatically analyzing the impact of a change is beyond the scope of this thesis. However, the research method for this question was at the beginning of the process to see what kind of software already existed and

to see whether such a tool was available or possible to adapt to the Django framework. As this research method did not give any noteworthy results, the focus shifted from automating the process to informing the user of the potential risk for changes that might cause the application to not function properly. In the first round of interviews, this topic was only partially mentioned, and was given more focus in the second and third round of interviews.

RQ3: How to perform automatic update of the web framework and the application using the framework?

Like the first research question, the final area of this research is quite accurate. To find the best way of performing the update of the application, the method used was again *observation* of what other similar solutions were doing, followed by *face-to-face interviews* to map the preferred updating method. Before the research began, the idea of always having the application perform an update was present. However, by doing the time-consuming process of conducting interviews to discuss this with the users, this idea quickly changed, and a more satisfying solution was developed.

3.2 Approach and Process

From the beginning of the process, the focus was to have an iterative process, so that by getting continuously feedback from users, the final solution would satisfy the need and expectation. By following the principle of "Testing one user early in the project is better than testing 50 near the end" [43], the final solution could adapt more to the user feedback early in the process, rather than towards the end. The work-flow and development process of this thesis was done in iterative way containing three different phases:

Pre-phase

The first phase of the process was to first observe and research what has already been done on the topic of the research question, and then to conduct semi-structured interviews around the subject. The subjects who were interviewed were fellow students and other people at the university with a technical background. The purpose of the interviews was to test the idea of simplifying the updating process, and to display information to the user through a panel. Each interview took approximately 15 minutes. The subjects were asked open-ended questions, and had to think of solutions on their own relating to different scenarios that they were given. This phase resulted in a deeper look at WordPress and how they are notifying and making the

updates available to their users. An external software called PyUP was also explored and was useful when looking at what kind of information to display to the users.

First iteration:

After the first interviews, the development process started. Based on the feedback from the users, it was clear that they wanted information and for it to be displayed in a good way. The development continued for 2 months. After this period, another round of interviews was done, this time focusing on the solution that was developed. The same persons who were asked in the first round were questioned again, this time in a more structured way. The feedback was evaluated and a plan for the second and final iteration started.

Second iteration

The last step of the process was the second iteration, where the feedback from the interviews after the first development phase was evaluated and considered for implementation. Because of the inverviews, a new design was implemented to display the data in a more convenient way, since the feedback from the users mentioned the design numerous times. After the final implementation, the same four people were questioned once again, and were given the chance to try out the solution.

3.3 WordPress

As a result of the observation part of the pre-phase of the project, WordPress quickly became a good comparison of a similar framework to look at. WordPress is a free and open-source content management system (CMS) based on PHP and MySQL. Their mission is to provide great software, which should work with minimum setup process, so that the users can focus on sharing their story, product, or services freely on the internet [44]. The repository is available online to everyone, meaning that everyone who wants to contribute some functionality or improvements to the software are welcome to do so by either submitting a patch of their changes, or through the discussion community.

The journey of WordPress started in 2003 when Mike Little and Matt Mullenweg wanted an elegant, well-architected personal publishing system. As of today, WordPress is the platform of choice for over 30% of all sites across the world [45] [46].

3.3.1 Security in WordPress

Compared to Django, WordPress has, since 2010, had 174 different vulnerabilities, 146 of them classified on the CVE website [47]. The numbers suggest that WordPress is 3 times more insecure than Django. This can be for a numerous reasons; PHP is very ubiquitous, meaning that it is attractive for an attacker to seek weaknesses in the software. Given the popularity, PHP is used by a lot of novice programmers. This can again lead to more insecure code if the application e.g. includes third party libraries. A figure showing the distribution of vulnerabilties is illustrated in Figure 3.1.

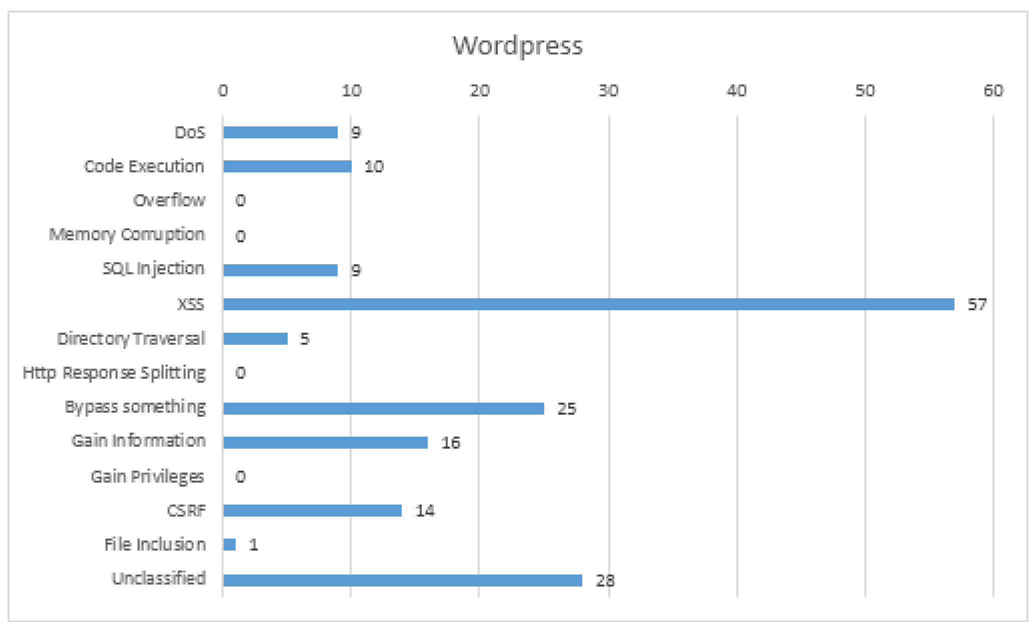


Figure 3.1: Distribution of the vulnerabilities in WordPress from 2010 to 2017 gathered from CVEdetails.com

3.3.2 Updating WordPress

Like other web frameworks, WordPress is also vulnerable to security-related issues and needs to be updated. However, the updating process in WordPress is simplified compared to, e.g., Django. Whenever a new update is available for WordPress, the user receives a notification in the admin panel, as shown in Figure 3.2.

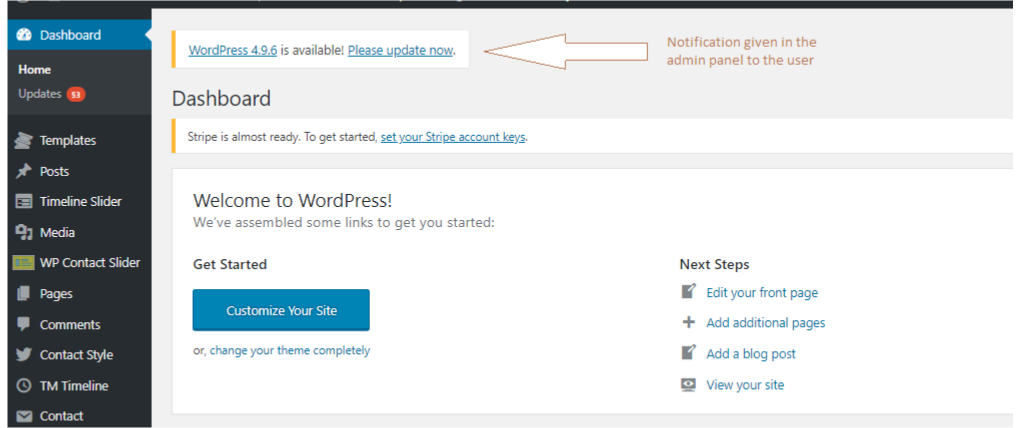


Figure 3.2: Notification given to the administrator in WordPress

By getting this notification, the administrator can do a "one-click update" with a push of a button. The updating process then starts, and the site is going to be temporarily unavailable. The framework also gives the user instructions on how to back up the files and database of the application before performing the update, in case something breaks. This is illustrated in Figure 3.3.

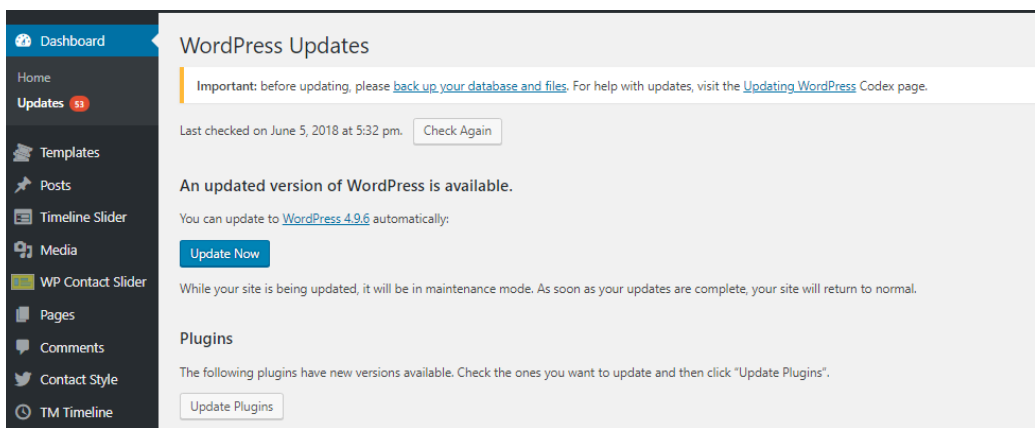


Figure 3.3: One-click update in the WordPress admin panel

3.4 PyUp-Django

Along with discovering WordPress as a similar framework, the focus shifted slightly towards finding a solution that might be implemented into Django. It was at this time that I came across the solution PyUP.

PyUP is a developed software that can help with automated security and dependency updates in Python []. PyUP consists of two main parts: the bot and Safety. They, combined, can notify users whenever a dependency or security update is available. The bot keeps the dependencies updated, and Safety warns the user about insecure dependencies.

PyUP can be configured and authenticated against a GitHub repository. Whenever Safety detects an outdated dependency, it notifies the bot, which then creates a pull request on the repository suggesting the updates.

PyUP-django is a package designed to be used in a Django project. The package can be installed through PIP and added in the INSTALLED_APPS dictionary inside the settings file of the Django project. PyUP-django is an extension of the Safety module provided in PyUP and displays a warning in the administrator panel if the Django dependency is outdated. This is shown in Figure 3.4. The image is taken from the documentation on GitHub [].

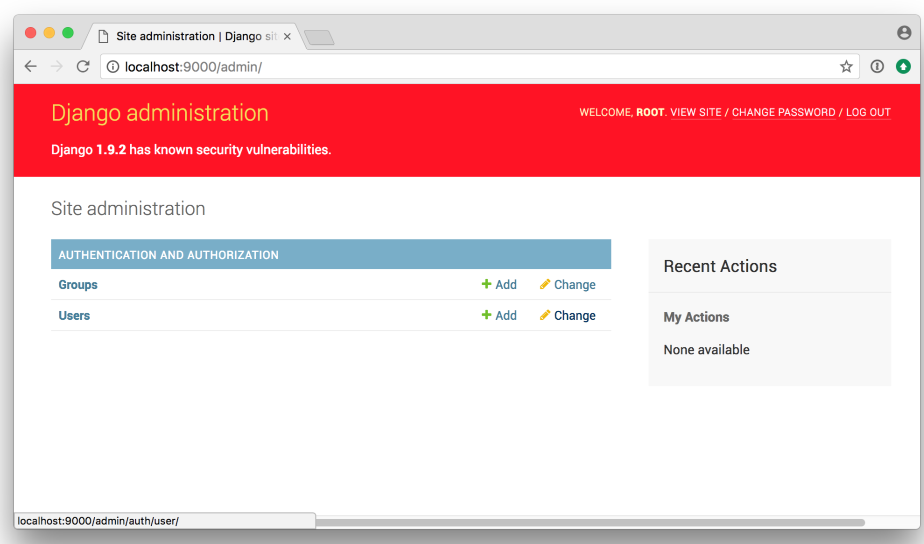


Figure 3.4: PyUP-Django - Example of the administrator panel

PyUP can also be used to generate reports about the vulnerabilities of the outdated dependencies in the project. This is done by installing Safety in the project, then running:

```
1 safety check --full-report
```

The limitation of PyUP is the pricing of the software; the free version is only limited to one repository, and it does not provide all of the features. A user pays between 15 and 50 dollars a month, depending on whether the software is used on a personal basis or in an organization. Another limitation of PyUP is that it will recommend the latest version of all dependencies, and disregards the possibility that, for example, one version of Django requires a specific version of dependency X, while another version of Django requires another version of the same dependency X: It will only recommend the latest version of dependency X.

Chapter 4

Results of the Observations and Interviews

The purpose of this chapter is to give a summary of the interviews before moving onto the details of the implementation.

4.1 Pre-phase Interview

The result of the pre-phase was an understanding of how WordPress presented and notified their users of a new update. WordPress also had an effortless way of installing the latest updates to keep their software up-to-date. It was also interesting to see the solution provided by PyUP, and how they decided to give the users a feedback straight into the administrator view if the application was outdated. PyUP also provided a detailed level of information about each vulnerability gave an idea for a solution of displaying the information easier to the end users. After learning about the existing solutions, the interviews were performed. As mentioned previously, the interviews were conducted among 4 candidates who all were students in the Department of Computer Science at NTNU. The interviews were held in an unstructured way, in which the topic was based on the research questions. None of the recipients had any experience using Django as their framework; however, they had all taken a security and software development courses at NTNU and were using devices and applications that received regular updates.

As the discussion continued, it quickly become apparent that this field of research was necessary, and by explaining the current situation of Django, some good discussions were had.

The first topic that had been discussed was their preference on notifications. As all of the students had smartphones, they were quick to mention the push notification they received when their smartphone needed to be updated. However, none of them wanted another app on their phone for receivnig notifications from their web application. They were positive toward the idea of receivnig an email containing information whenever their application had a possible vulnerability. During the interview, I mentioned the public mailing list that a user could subscribe, to and they reacted positively to this list. Despite the positivity, 2 out of the 4 recipients wanted a more targeted information concerning their application, rather than getting the public statement from Django.

Regarding the topic of performing the update, all of them pointed out that they would probably not update their application very often if they had to log on to the server and do the steps involved by pulling the software into a local environment and performing the update there. In one of the interviews, the other person asked why not just have the update installed automatically. After discussing this back and forth, the student concluded that by having an update done automatically, without the guarantee of a successful installation, the application might become unstable and control would be lost over the update. The resulting solution of the other discussions was to have a button one could press that would do the update itself at the administrator's request.

4.2 Design

As the process for this thesis have been an iterative process, this section shows the evolution of the development and feedback from the users after each of the two iterations.

4.2.1 First iteration and interviews

After the development had gone on for approximately 2 months, the interviewees were questioned a second time. This time the results from the first interviews was evaluated and a temporary solution was presented to the students. The interface that the candidates was presented, is shown in Figure 4.1

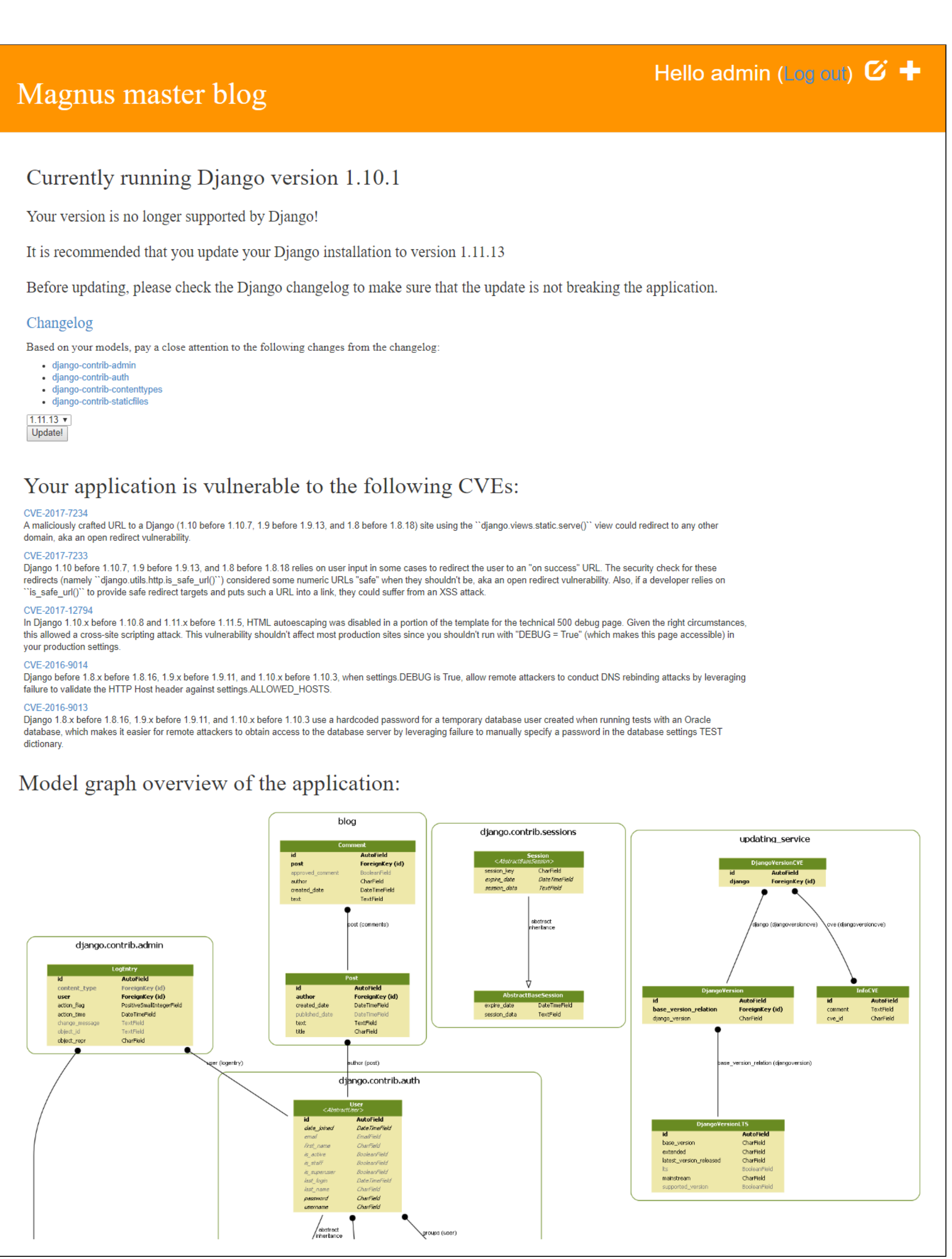


Figure 4.1: Old design of the application

In the old design, the view is divided into three main parts: Information about current version and recommendation for updating to a newer version along with the changelog and a button for doing the installation. Information about the CVEs related to the current installed version of Django. And finally, a model graph overview of the models that are used in the application. The idea was, based on the feedback from the pre-phase, to display all relevant information to the administrator.

The top part of the view shows information about the current installed version of Django, and which version that it is recommended updating to. It also includes a link to the different critical parts of the Django core that is going to be updated when the recommended version is installed. When clicking on the link, it opens a new tab displaying the changelog released from Django and documentation of that module. Under the changelog is a small button for installing the recommended version when clicked on.

Middle part of the site displays information about the exposed vulnerabilities that is associated with the installed version. Each of the CVEs could be clicked on, and a new tab would lead to the CVEDetails website where more information about the vulnerability could be found.

The final part of the view is the overview of the model graph. This section contains a picture with the relations among the models that is been used in the application. This picture is generated based on the *INSTALLED_APPS* variable inside the *settings.py* file of the project.

As the interviewees was given the opportunity to play around with the solution, and to see the flow of it, they provided some useful pointers that has been addressed in the next development phase. Key points of the feedback are as follows:

- The webpage displayed too much information at once.
- The separation of the various parts should be more distinct.
- Would like to see the information about changes in the changelog.
- Should be able to view the changes in the code caused by fixing the CVEs.
- Updating button is to small.
- Not sure of how to use the model graph since its too small.

After evaluating and discussing with the interviewees how to cope with the challenges and feedback that they gave, the development continued in improving the solution too satisfy the users as much as possible.

4.2.2 Second iteration and interviews

When evaluating the feedback from the second interview, it looked like it would improve the solution a lot if the design was better, and the information was clearer. Because of the feedback, a process of finding a clever design started, and a choice of using Bootstrap as an external framework on front-end was made. After the development process was completed, the finishing view that was presented to the users in a final interview is presented in Figure 4.2.

As the figure shows, a more distinct separation of the various parts has been made. At the top of the view a summary of the most key details is displayed. Here the users get an overview of what version they currently have installed, how many vulnerabilities that is associated with the current version, whether the current version is supported by Django, a recommendation for an updated version to be installed, and the updating button displayed with a greater size than after the first iteration.

The second part of the developed web interface is displaying the CVEs. Since the interviewees pointed out that the site contained too much information, I decided to place each of the CVEs as an entry in a list, and to update the main section of that row whenever a new CVE is selected in the list at the right-hand side. As requested by the interviewees, a possibility for looking up the fixed code was added as a link to the fixing commit at GitHub.

Feedback from the last interview suggested that the relevant information for critical modules, when concerning about installing a new update, is displayed, the section below the CVEs follows the same principle as the CVE section. The critical modules are looked up in the change log and the information is extracted and placed into the view. A link to the complete changelog is also provided.

The last part containing the model graph has not changed that much since the second interviews. However, the interviewees complained about the image been too small. This is now improved in a way that allows users to click on the image and the image will enlarge and fill the whole screen.

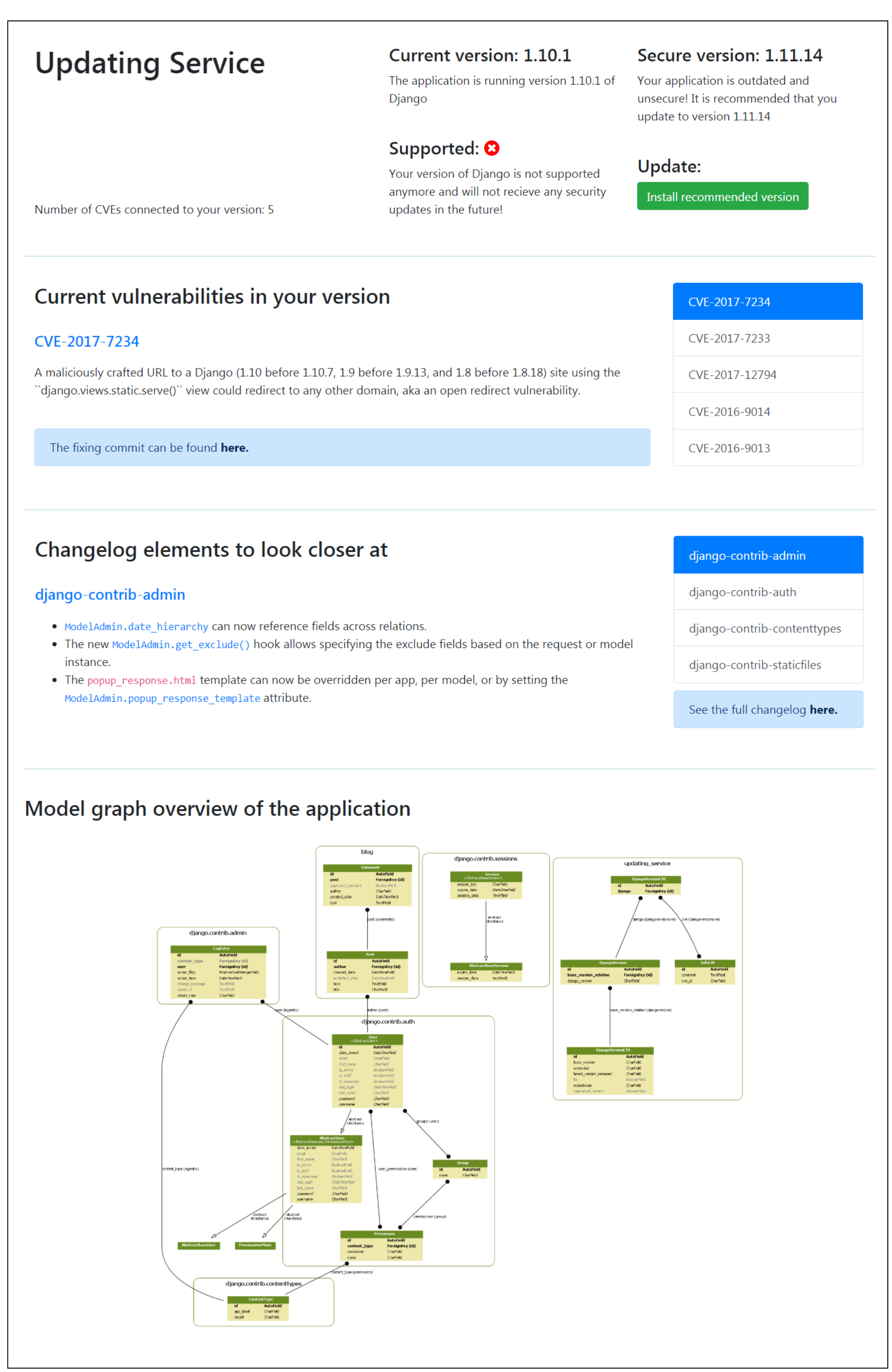


Figure 4.2: New design of the application

The overall feedback from the users were good, and all the interviewees manage to install a new update. Here is a list of feedback that can be taken from the interviews:

- The application does not have the support of been displayed on a mobile device.
- The model graph is not showing the direct link of a critical area of the code.
- It would be helpful if the components listed in the changelog section were linked to the changes been made in the code. A simular solution as the one that is provided to the CVEs.
- No feedback after one clicks the button to initiate the update. Should be a progress bar or similar graphics.

Chapter 5

Implementation

This chapter gives an overview of the developed application, assumptions made before developing, and how the system is operating on a real server.

5.1 Assumptions

Before beginning the development process, some assumptions were made to simplify the example application and the new updating application. This section elaborates on these assumptions, gives a brief overview of the consequences, and justifies why the assumptions were made in the first place.

5.1.1 Python version

As of today, there are two main versions of Python used: Python 2 and Python 3. In 2016, JetBrains, a software development company whose tolls are targeted towards software developments and project managers [48], distributed a survey to the community. This survey was conducted among more than 1,000 Python developers to identify the latest trends and gather insight into what the Python development world looks like today [49]. One of the results from this survey was the distribution of the different versions of Python used by developers. The results are presented in Figure 5.1, and reveal that Python 2 was used by 60%, while Python 3 was used by 40%. The distribution was confirmed with a correlation with external research and their own PyCharm internal statistics.

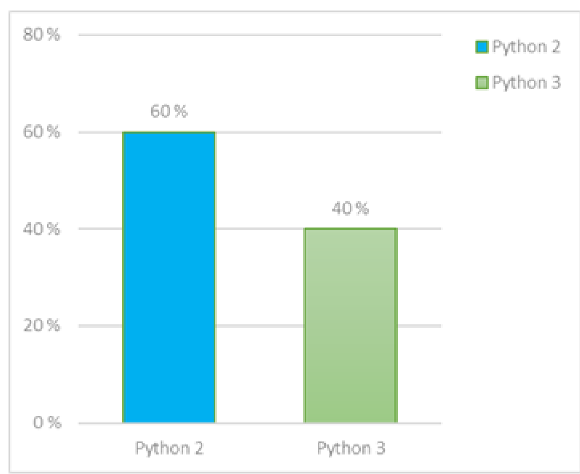


Figure 5.1: PyCharm survey from 2016

Based on the results of both the survey and their internal statistics, JetBrains saw rapid growth in the usage of Python 3, and expected it to overtake Python 2 in the near future.

One year later, in late 2017, JetBrains performed another survey out to the community. This survey got over 9,500 developers from over 150 different countries to participate, and was useful for mapping out an accurate landscape of the Python community [50]. In an evaluation of the results, the interesting part was that Python 3 had overtaken most of the users by a stunning 75%, while 25% of the respondents used Python 2 the most, as shown in Figure 5.2. This is likely to be due to the decrease in support for Python 2. The version is not going to get new features, will not actively develop, and its maintenance will be stopped in 2020 [51].

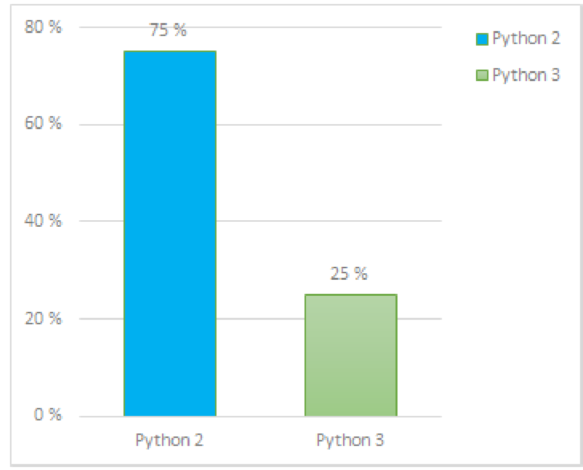


Figure 5.2: PyCharm survey from 2017

By looking at the results from the two surveys and the fact that the support for Python 2 is going to be dropped in 2020, the application that has been developed in this thesis is using Python 3 as the supported version.

5.1.2 Dependencies

By using PIP in the developing process in Django, users can easily install and manage software packages written in Python. many of these packages can be found, installed, and updated using PIP. These packages are sometimes outdated and need to be updated to work properly.

In the example application all of the external packages and their dependencies are left untouched. By performing an upgrade in Django, some of the dependencies must be updated. However, scanning through every dependency of the application is likely to be a cumbersome process that will take much time to develop and perform. The application relies on the user to update the necessary packages and to make sure that the usage of these packages is correct.

5.1.3 Django version base

Django has, since their beginning, released several new versions and releases of their framework. The releases follow an 8-month interval where a new subversion of the framework is being released. This is not to be confused with security patches for any given version, but the release might introduce new or deprecate old features in the framework. In some cases, Django has

released a completely new version, going from, e.g., Version 4.7 to Version 5.0. Common for these updates is a change in syntax or structure of the framework. These changes are major, and in the example application, this major update has not been prioritized and is likely to cause an error if used to perform this kind of update.

5.2 Example application

To test the implementation of the Updating Service application, I created a simple test application. The goal is for it to be used in production to see the update in a live environment. The application is a simple blog where the administrator can create new, edit, and delete posts. It has a simple model structure with a post and comment table. As showed in Figure 5.3, the comment has a foreign key in post. The relation between *Post* and *Comment* is a one-to-many, where as one Post can have many Comments.

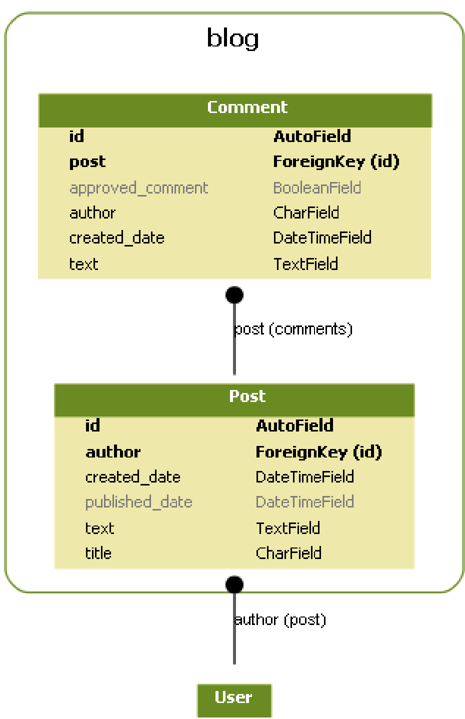


Figure 5.3: Models in the example application

The blog application is installed through the INSTALLED_APPS attribute in the settings file and is migrated with the database. Only a user with pub-

lishing rights should be able to create a new Post. This is done by providing an extra layer of security in which the user must log in with a username and password to publish a new Post.

Every post has an "author", which is related to their username. The author field has a foreign key '*auth.User*' which is the built-in database handling the userbase. This module makes a relation between the logged-in user and the creator of the Post whenever a new Post is created. The example application was created based on a tutorial provided by Django Girls as inspiration, and it consists of two different pages. The main page, which is directly accessed by the root of the URL, is shown in Figure 5.4.

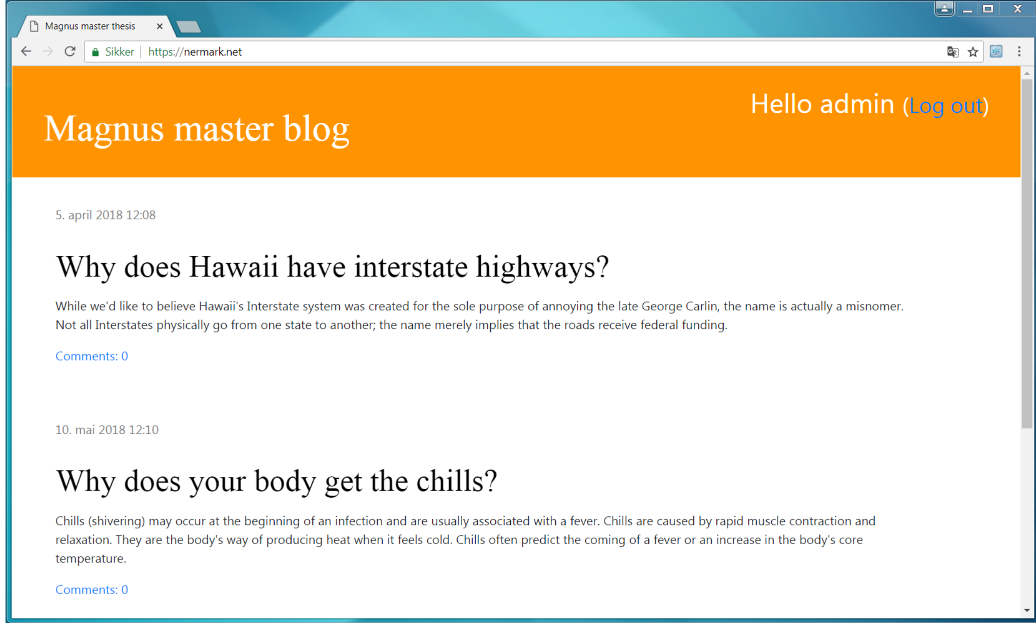


Figure 5.4: Main page of the example application

The second page containing the blog post itself is shown in Figure 5.5. This page also contains the comment section for users to interact, discuss, and leave feedback on a specific post.

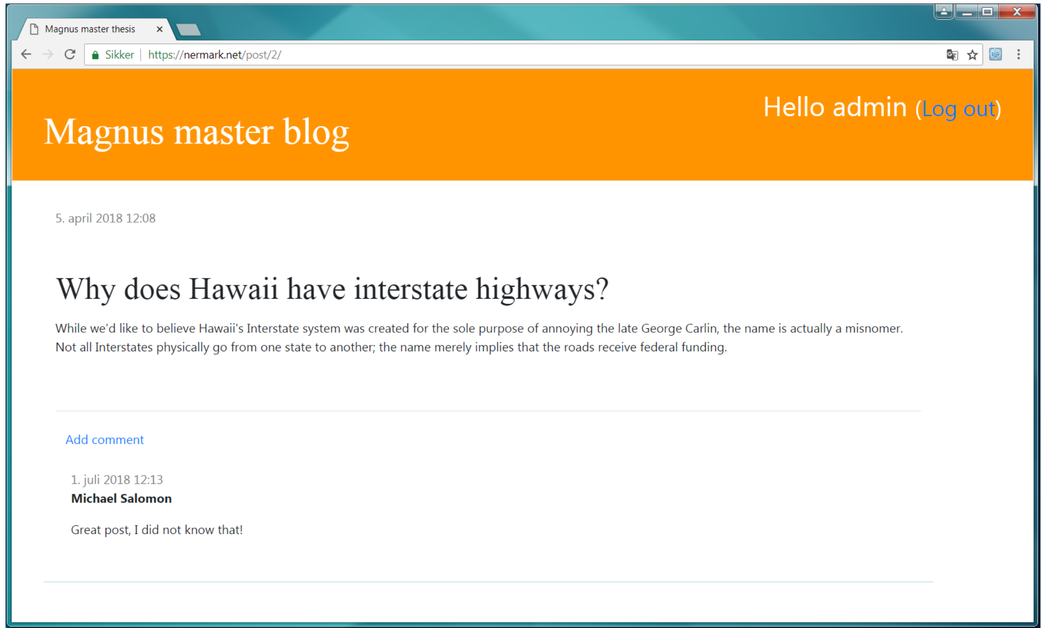


Figure 5.5: A typical blog post page of the example application

5.3 Updating service application

This section explains the updating service application, how it was created, and how it is intended to be used. The updating service application consists of two parts to cope with the research questions mentioned in section 1.3. The first part is related to RQ1, and the second part is related to RQ2 and RQ3.

5.3.1 update-script.py

Based on feedback gathered from the first interviews, the subjects mentioned that they would prefer an alternative notification method more specified to their solution. The suggestion of an app for this notification part was quickly tossed away, and they felt that an email would suite this purpose well. The implementation of an email notification method is described in this section. This is a result of the first research question, RQ1.

A full overview of the class can be found in section A.1

Finding all CVEs related to their version

The first step involved in the notification system is to gather information about CVEs that is related to the currently installed version. This is done by using an external library called pycvesearch. This library makes a request to the CVE server and returns a JSON file with the associated data that is connected to the unsecure version. A list containing the CVE id and a summary of the vulnerability is returned by the class:

```
1 def get_all_cve_on_given_version(VERSION_NUMBER):
2     cve = CVESearch()
3     returned_data = cve.search('djangoproject/django
        :' + VERSION_NUMBER)
4     cve_list = []
5
6     for element in returned_data:
7         cve_list.append(element.get('id'))
8         cve_list.append(element.get('summary'))
9
10    return cve_list
```

A help class was also created for gathering information about the current version of Django running on the server:

```
1 def get_current_django_version():
2     version_number = str(django.VERSION).strip('()').
        strip(' ').split(', ')[:3]
3     return version_number[0] + "." + version_number
        [1] + "." + version_number[2]
```

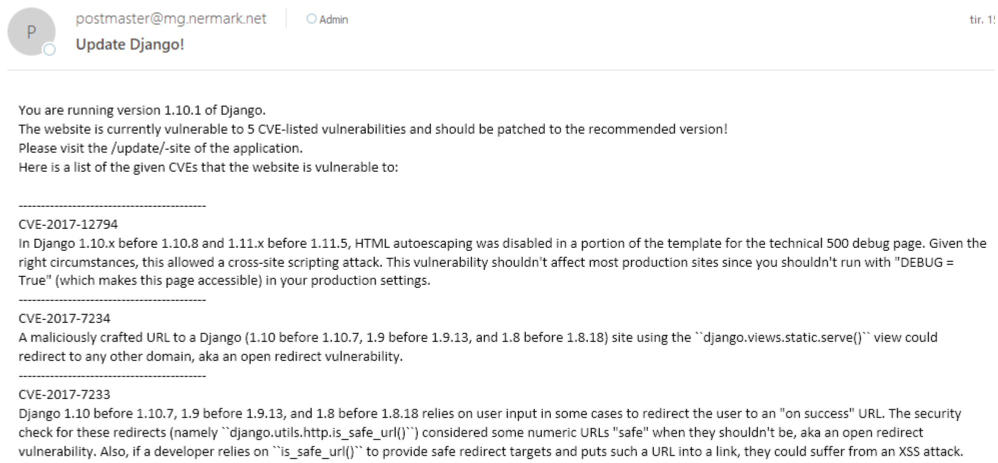
Sending email to the administrator

After gathering the CVEs related to the installed version, the application checks whether there are any CVEs in the list that have been returned. If there are CVEs in the list, then the method starts to generate the message. Finally, the message is sent by posting a request to the Mailgun service as shown in section 2.11. Due to privacy concerns, the API key and correct mail server are not provided in the code examples:

```
1 def find_outdated_version_and_send_email():
2     #CHECK IF THERE IS ANY CVEs
3     .
4     .
```

```
5       # GENERATE MESSAGE
6       .
7       .
8
9       return requests.post(
10          "https://api.mailgun.net/v3/
            sandboxc176XXXXXXXXXXXXXXXX.mailgun.org/
            messages",
11          auth=("api", "key-8866-YOUR-API-KEY-
            HEREXXXXXXXXXXX"),
12          data={"from": "Updating Service <
            postmaster@sandboxc176XXXXXXXXXXXXXXXX.
            mailgun.org>",
13            "to": "Admin <YOUR@EMAIL.COM>",
14            "subject": "Update Django!",
15            "text": text_message})
```

In Figure 5.6 is shown an example of an email that is been sent to the administrator if a vulnerability is detected. In order to use the sending email part, it is necessary to register an account at Mailgun, and provide the mailbox-details that is given to you by them into the script above.


postmaster@mg.nermark.net | Admin
Update Django!

You are running version 1.10.1 of Django.
The website is currently vulnerable to 5 CVE-listed vulnerabilities and should be patched to the recommended version!
Please visit the /update/-site of the application.
Here is a list of the given CVEs that the website is vulnerable to:

CVE-2017-12794
In Django 1.10.x before 1.10.8 and 1.11.x before 1.11.5, HTML autoescaping was disabled in a portion of the template for the technical 500 debug page. Given the right circumstances, this allowed a cross-site scripting attack. This vulnerability shouldn't affect most production sites since you shouldn't run with "DEBUG = True" (which makes this page accessible) in your production settings.

CVE-2017-7234
A maliciously crafted URL to a Django (1.10 before 1.10.7, 1.9 before 1.9.13, and 1.8 before 1.8.18) site using the ``django.views.static.serve()`` view could redirect to any other domain, aka an open redirect vulnerability.

CVE-2017-7233
Django 1.10 before 1.10.7, 1.9 before 1.9.13, and 1.8 before 1.8.18 relies on user input in some cases to redirect the user to an "on success" URL. The security check for these redirects (namely ``django.utils.http.is_safe_url()``) considered some numeric URLs "safe" when they shouldn't be, aka an open redirect vulnerability. Also, if a developer relies on ``is_safe_url()`` to provide safe redirect targets and puts such a URL into a link, they could suffer from an XSS attack.

Figure 5.6: Example of an E-mail sent to the administrator

5.3.2 Models

When it comes to Django, a model is the single, definitive source of information about one's data. It contains the essential fields and behaviors of the

data being stored. Generally, each model maps to a single database table [52]. The application for doing the update has been developed with four models that are interconnected. The four modules are as follows:

- InfoCVE - Table that contains the information of a given CVE.
- DjangoVersionLTS - Table that holds the information of a base version of Django.
- DjangoVersionCVE - Relation table between a given CVE and a Django version.
- DjangoVersion - Relation table between a specific version and a base version in Django.

The relations between the different models are displayed in Figure 5.7.

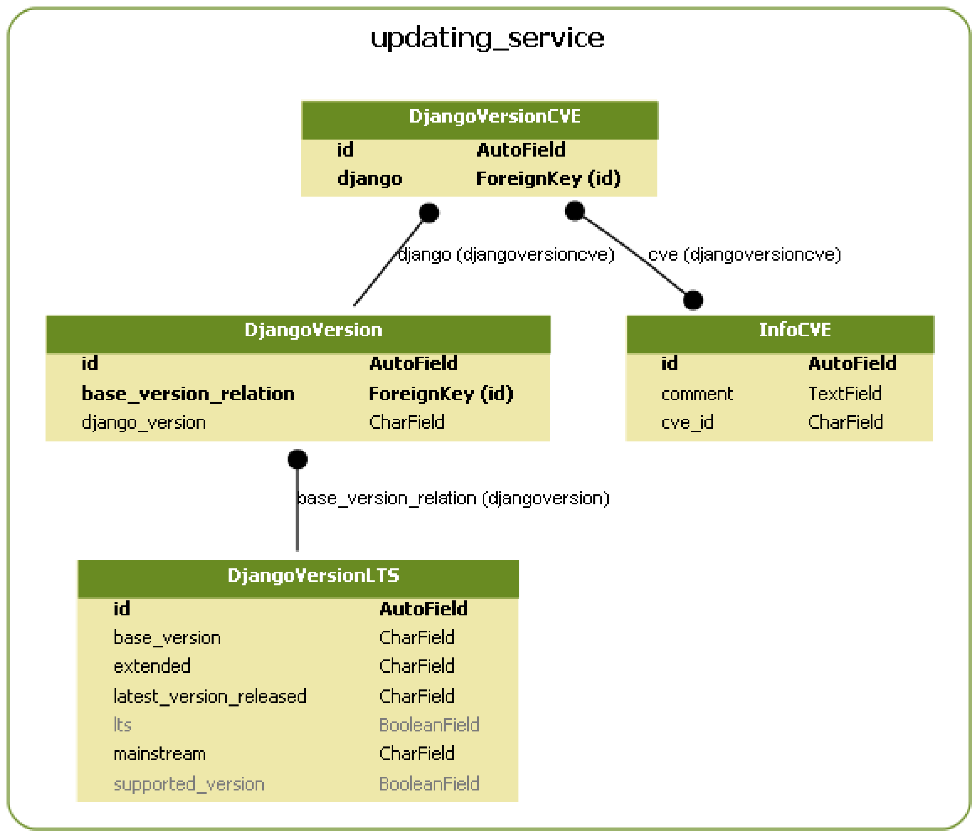


Figure 5.7: Model overview of the updating application

InfoCVE:

This table in the database holds the information about a given CVE. The CVE_ID is the actual ID collected from the CVE website, e.g., CVE-2016-12345. Along with the CVE_ID, the comment regarding the summary of that given CVE is also added to the table. This is the comment attribute selected to be a TextField.

DjangoVersionLTS:

The DjangoVersionLTS table contains information about the base version released by Django. The *base_version* and *latest_version_released* are as their names suggest, the base version of Django, e.g., Version 4.5, and the latest version that has been released within that base version, e.g., 4.5.11. Extended and mainstream are when the ended mainstream and extended support will end, meaning when the version is cut by the Django team. The last two fields in the table, *lts* and *supported_version*, are both Boolean fields, which indicates whether a version is supported, and whether that version is a long-time supported version of Django.

DjangoVersion:

The DjangoVersion table is used to relate a given version of Django, e.g., version 4.5.9, to the correct base information about that release. This is to prevent attributes, such as the LTS and end of mainstream support from being stored on each subversion of a base version. The relation between this table and the DjangoVersionLTS is a one-to-many relation in which a base version can have many DjangoVersions.

DjangoVersionCVE:

The final table is a relation table to connect a specific version of Django to a given CVE_ID. The relation table has two relations, including a one-to-many relation with DjangoVersion in which a given version can only have one Django version. The other relation is a many-to-many-relation in which a CVE can be related to many Django versions.

5.3.3 Views

In Django, a view is a callable which takes a request and returns a response. The callable can be more than just a function, and is accessed with a call from a URL request. Django provide examples of some classes which can be used as views [53]. In the Updating Service application, there are three different URL endpoints that can be accessed by the user. They are as follows:

- /update/ - Displays the interface presented in Figure 4.2
- /update/perform/ - An endpoint for initiating and starting the installation of a new update
- /update/pushdata/ - An endpoint to ensure that the database is updated

The view class is quite complex and is using many different methods with in it. An overview of the class is given in Figure 5.8.

/update/

The */update/* page is the main page that the user will be interacting with. It combinds and covers the second and the third research questions. When a request is sent to the /update/ site, this triggers the function *update_page* in the views. This function will first try to access the available CVEs for the current version of Django and will later return the list of CVEs to the template and show this to the user. This is completed with the following code:

```
1 cves = DjangoVersionCVE.objects.get(
      django__django_version=get_current_django_version
      ()).cve.all().order_by('cve_id').reverse()
```

If no CVEs exist for that given version in the database, a help function is applied to gather the CVE information from the CVE website and pushes this to the database. The function *get_cves_for_version(request)* is the same function that is used when pushing data to the database. This one is described in the */update/pushdata/* section.

A few other variables such as the *base version*, *current version*, and *changelog-data* are collected and returned with the CVE-list to the template and displayed to the user. The list of returned variables is displayed in Table 5.1.

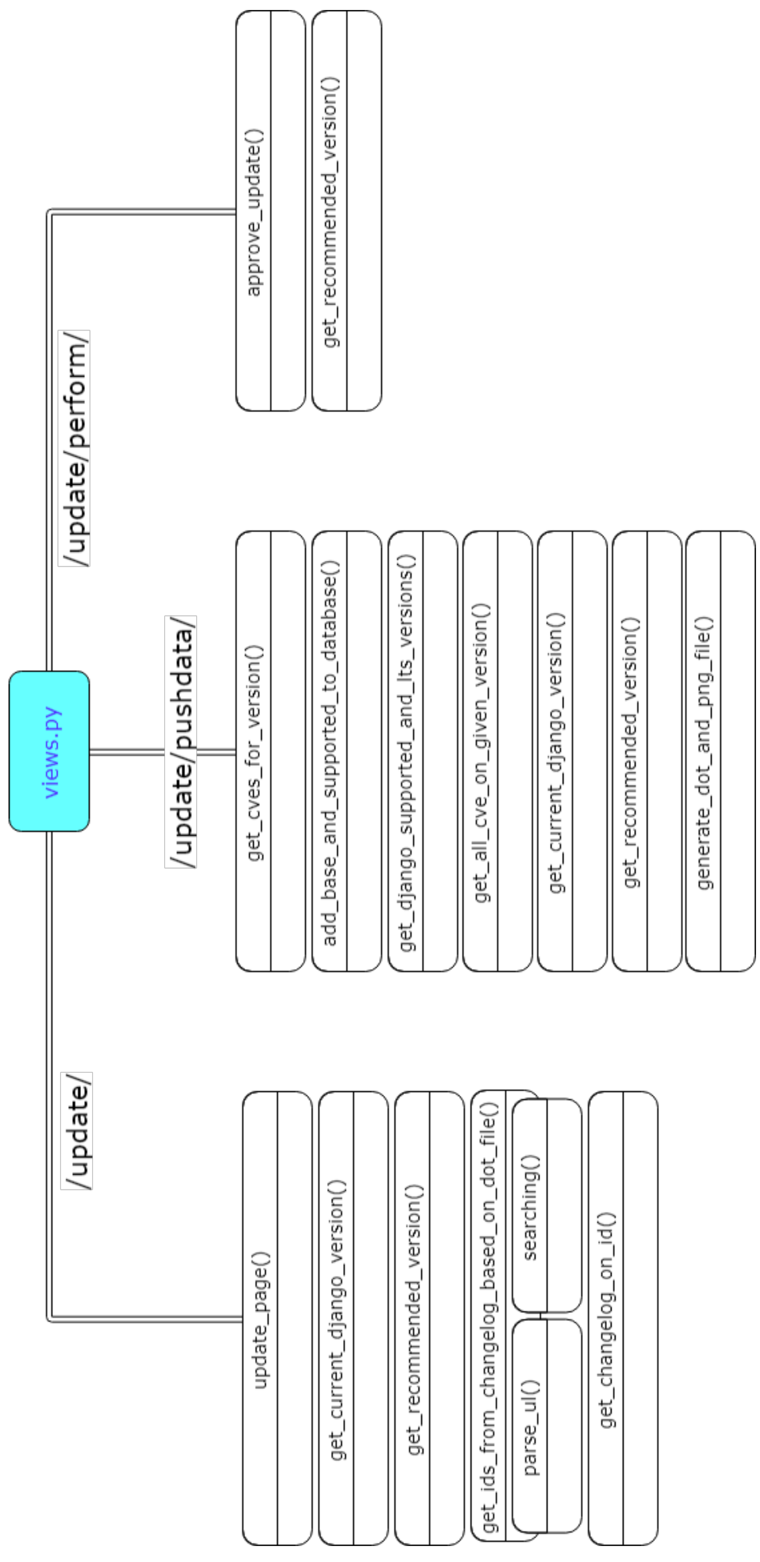


Figure 5.8: Overview of views.py

Variable-name	**Value**
'cves'	cves
'django_version_info'	django_version_info
'current_django_version'	get_current_django_version()
'ids_to_look_closer_at_changelog'	ids_to_look_closer_at_changelog
'recommended_version'	get_recommended_version(django_version_info)
'recommended_base_version'	recommended_base_version
'changelog_info'	changelog_info
'changelog_url'	changelog_url

Table 5.1: Variables and values returned to the template

Variable description of Table 5.1:

'cves'

'cves' is a list of all of the CVEs related to the given version of Django that the application is running.

'django_version_info'

'django_version_info' is a list of all of the Django versions that are listed on their website. Each object in the list contains information on whether the version is supported, when the support is scheduled to be dropped, and the version number.

'current_django_version'

'current_django_version' is a string containing the current version of Django that is installed and running the application.

'ids_to_look_closer_at_changelog'

ids_to_look_closer_at_changelog contains a list of different classes that are recommended to look closer at when doing an update.

'recommended_version'

'recommended_version' contains the version number that is recommended based on a long-time supported version, and is still getting updates whenever a new CVE is introduced. The value is returned as a string.

'recommended_base_version'

'recommended_base_version' is a string containing the base of the recommended version. This variable is used when linking to the different changes in the documentation of Django.

'changelog_info'

'changelog_info' contains a list of information based on the *'ids_to_look_closer_at_changelog'* list. It is used to display the changes that have been made to the specific class in the core code of Django.

'changelog_url'

'changelog_url' is a string containing the changelog URL to link to the changelog in my application.

A full overview of the class can be found in subsection A.2.2. The usage of these variables is shown in Figure 4.2.

/update/perform/

The sub-URL /perform/ is the endpoint that the user is accessing when he or she wants to do the update. This endpoint makes it possible for the user to perform an update, and is an answer to the third and final research question. As demonstrated in Figure 4.2, the user can perform an update of the framework with the click of a green button button. When the user clicks the button, a request is made to the endpoint, which triggers the function *approve_update*. If the request is successful, the installation will then be initiated with the recommended version that the application has proposed.

To perform the update, the subprocess package in Python is used. The package makes it possible to do system calls as a subprocess. Since the installation is encapsulated inside a virtual environment, we can do the following call to perform the update:

```
1 subprocess.call([sys.executable, '-m', 'pip', '
      install', 'django=='+version_number])
```

This is the equivalent of performing the update inside a shell with the following command:

```
1 $ pip install django==version_number
```

The Version number is decided by the application with the following command:

```
1 version_number = get_recommended_version(
      DjangoVersionLTS.objects.all().order_by('
      base_version'))
```

After doing this, the update will begin in the background and the site will be up-to-date in a couple of minutes. Since the application is running on top of an instance of Apache, the site will not go down as Apache caches the site. However, when the application is updated, a crown job will restart the Apache instance and the cache will be reset within the next 20 minutes.

/update/pushdata/

The pushdata URL is an endpoint used for gathering information and updating the database with the newest data pulled from various sources. This includes checking the current version up against the CVE database, updating information such as the current support for the currently installed Django version. This section breaks down the calls and gives an overview of the various processes involved when gathering the data and putting it to the database.

Pushdata is an endpoint makes makes it possible for the cron job to access the endpoint as a logged-in user. The URL is hidden behind a login to prevent other users from triggering the event of updating the database with the function *get_cves_for_version(request)*. The function contains numerous other functions, and as the cron job executes, the first method that it executes is the *add_base_and_supported_to_database()*. This method triggers the function *get_django_supported_and_lts_versions()*, which makes a request to the Django download page. By using the library of BeautifulSoup to parse the request, I was able to crawl the web page to gather information about the different versions of Django and see whether they were supported versions.

```
1 def get_django_supported_and_lts_versions():
2     req = Request("https://www.djangoproject.com/
          download/", headers={'User-Agent': 'Mozilla
          /5.0'})
3     webpage = urlopen(req)
4     soup = BeautifulSoup(webpage, 'html.parser')
5     table = soup.find('table', attrs={'class': '
          django-supported-versions'})
6     version_support = {}
7     for row in table.findAll('tr', {'class': ''}):
8         # logic for finding information in the html
              and adding it to version_support
9         .
10    for row in table.findAll('tr', {'class': '
          unsupported'}):
11        # logic for finding information in the html
              and adding it to version_support
12        .
13    return version_support
```

get_django_supported_and_lts_versions() returns a dictionary containing the version number as the key, in addition to the following values:

- information about whether the version is supported
- whether the version is an LTS version
- information about the latest released version
- when the mainstream support is scheduled to end
- when the extended support is scheduled to end

The dictionary is returned to add_base_and_supported_to_database(), and added to the database.

After updating the database with the Django versions, the next step is to gather all of the CVEs for the given version of Django. This is done by calling the method get_all_cve_on_given_version(VERSION_NUMBER). By using the library pycvesearch and importing CVESearch(), a request could be made to search for the CVEs to a specific version by applying the method *CVESearch().search('project:version')*.

```
1  def get_all_cve_on_given_version(VERSION_NUMBER):
2      cve = CVESearch()
3      returned_data = cve.search('djangoproject/django
           :' + VERSION_NUMBER)
4      cve_list = []
5
6      for element in returned_data:
7          cve_list.append(element.get('id'))
8          cve_list.append(element.get('summary'))
9
10     return cve_list
```

Every CVE gathered from the search is added to a list, returned to *get_cves_for_version(request)*, and stored as a local variable.

When requesting */update/pushdata/*, the model graph is also regenerated with the following method:

```
1 def generate_dot_and_png_file():
2     subprocess.call([sys.executable, './manage.py', '
         graph_models', '-a', '-g', '-o', '
         updating_service/static/img/
         my_project_visualized.png'])
```

5.3.4 Templates

The templates developed for this project contains two files. First out is the base.html file. This file loads the skeleton of an HTML following by the HEAD and BODY tag. The goal of this file is to load all the internal and external static files such as the styling in the .css file, and the programmable JavaScript in the .js file of the project. Here is a snippet of the base.html file:

```
1 {% load staticfiles %}
2 <!doctype html>
3 <html lang="en">
4     <head>
5         <meta name="viewport" content="width=device-
             width, initial-scale=1, shrink-to-fit=no">
6
7         <title>Django Updating Service</title>
8         <link rel="stylesheet" href="..." >
9         <link rel="stylesheet" href="{% static 'css/
             updating_service.css' %}">
10    </head>
11    <body>
12        {% block content %}
13        {% endblock %}
14
15        <script type="text/javascript" src="..."></
             script>
16        <script type="text/javascript" src="{% static
              'js/javascript.js' %}"></script>
17
18    </body>
19 </html>
```

In the process of improving the interface of the application, a decision of using bootstrap as an external library was made. Bootstrap helps to divide the page into sections and have a grid system where content can be presented in. Bootstrap provides styled buttons and lists, which has been used when improving the layout.

The second, and most important part of the template is the information that is put into the *body* tag of the *html* tag. This is done in a Django way where all the content located in the *info_page.html* is loaded into the tag *{% block content %}* as displayed in the example abobe.

The idea is to use the variables that is rendered by the *update_page()* method, and extract the information from the returned variables. One example of this is the extraction of CVEs from the returned value *'cves'* that is available from the view:

```
1 {% if cves %}
2     {% for cve in cves %}
3         <h5><a target="_blank" href="https://www.
              cvedetails.com/cve/{{ cve.cve_id }}">{{
              cve.cve_id }}</a></h5>
4         <p>{{ cve.comment|linebreaksbr }}</p>
5     {% endfor %}
6 {% endif %}
```

In the example above, the *cves*-variable is treated as a list, and iterated over in a Django fashion way using *{% for cve in cves %}*. This is equivalent of a *for each*-loop in any programming language. In this case; for each *cve* element inside the *cves* list, the example is extracting the information from each *cve* object that exists in the *cves* list.

As the *updating_service.html* file contains too much configuration, it is not to be shown in the thesis. Please have a look at the source code for a further inspection of the code and how the different elements are displayed in the view.

5.4 Requirements

This project has made use of external libraries. A full overview of the requirement file can be found in section A.4. Please note that the file contains an insecure version of Django. In Table 5.2, the external libraries that have

been used are listed:

Requirement	Description
beautifulsoup4	Used for crawling web pages by creating a collection of the content on the web page
Django	Web framework used on the web server
django-extensions	Package for generating the model graph
pycvesearch	A library in Python that returns a list of CVEs when queried
pydotplus	Makes it possible to generate .DOT files and .PNG files
requests	Used to create a request with a web server
urllib3	Library for making requests with beautifulsoup4 in Python 3

Table 5.2: Brief description of requirements in the project

5.5 Development Environment

By following the steps in this section, you will be able to set up and run the application in a development environment from scratch. The setup is completed on a Windows computer with a Linux-based terminal. The setup process should be fairly similar to the Windows installation, with a few modifications on Linux. A summary of the commands that are used for setting up the local development environment is listed in Table 5.3.

List of terminal commands

Command	Description
python --version	Shows the Python version installed on the system (or virtual environment)
py -3 -m venv env	Creates a new virtual environment inside the env-folder
env\Script\activate	Activating the virtual environment located in *env* folder of the project
django-admin startproject mysite	Initiates a new Django project inside the *mysite* folder
python manage.py makemigrations	Prepares the migrations of the Django models if any changes have been made
python manage.py migrate	Migrates the changes of the models in the database
python manage.py createsuperuser	Creates a superuser used for logging into the admin panel of the application
python manage.py runserver	Runs a local server for testing the application. The default location of the development server is at *http://127.0.0.1:8000/*

Table 5.3: Useful terminal commands

5.5.1 Installing Python

The first thing we need to do is to download and install Python on our computer.

Step 1: Go to *https://www.python.org/downloads/* and download the latest version of Python 3.

Step 2: Install it to the default location suggested by the installer.

Step 3: Make sure to select the option "to add Python to PATH variable"

Step 4: Restart the computer if necessary.

Step 5: Ensure that the installation was successful by opening the terminal and typing *python -V*. The response should look something like this:

```
c:\>python -V
Python 3.7.0
```

5.5.2 Setting up virtual environment

Now that Python is installed it is time to setup the virtual environment.

Step 1: Open the terminal.

Step 2: Navigate to where you want to store your project, e.g.:

```
1 c:\myproject>
```

Step 3: Create the virtual environment by entering the following command in the terminal:

```
c:\myproject>py -3 -m venv env
```

This will create a new folder named env containing the virtual environment and based on the Python version we just installed.

Step 4: Activate the virtual environment by typing the following in the terminal:

```
c:\myproject>env\Script\activate
(env) c:\myproject>
```

5.5.3 Installing the requirements

Step 1: Download or import the Updating Service application either from the Zip file or from GitHub [54]. The folder structure should look something like this:

Step 2: Navigate into the Updating Service folder:

```
1 (env) c:\myproject\>cd updating-service
2 (env) c:\myproject\updating-service>
```

Step 3: Install the requirements located in the folder:

```
1 (env) c:\myproject\updating-service>pip install -
    r requirements.txt
```

Step 4: Install PyCVESearch package:

```
1 (env) c:\myproject\updating-service>cd
     PyCVESearch
2 (env) c:\myproject\updating-service\PyCVESearch>
     pip install .
```

5.5.4 Setting up a Django Project

If you already have a Django project, please feel free to skip this step. Now it is time to initiate a new Django project.

Step 1: Make sure that you are inside the virtual environment.

Step 2: Install Django through PIP. This will install an unsecure version of Django:

```
1 # UNSECURE VERSION FOR TESTING PURPOSES
2 pip install Django=="1.10.1"
```

Step 3: Start a new Django project and navigate into the newly created project:

```
1 (env) c:\myproject>django-admin startproject
     mysite
```

Step 4: Navigate into the root folder of the Django project that was created in the previous step:

```
1 (env) c:\myproject>cd mysite
2 (env) c:\myproject\mysite>
```

Step 5: Migrate the installed apps:

```
1 (env) c:\myproject\mysite>python manage.py
     makemigrations
2 (env) c:\myproject\mysite>python manage.py
     migrate
```

Step 6: Create a superuser:

```
1 (env) c:\myproject\mysite>python manage.py
     createsuperuser
```

Step 7: Run the server and try the login at *127.0.0.1:8000/admin*.

5.5.5 Installing the Updating Service application

Step 1: Download or import the Updating Service application either from Zip file or from GitHub [54].

Step 2: Add it to the INSTALLED_APPS dictionary in the *settings.py* file:

```
1 INSTALLED_APPS = [
2       ..
3       'updating_service',
4       ..
5 ]
```

Step 3: Make migrations and migrate:

```
1 (env) c:\myproject\mysite>python manage.py
     makemigrations
2 (env) c:\myproject\mysite>python manage.py
     migrate
```

Step 4: Add the service to the urls.py-fil:

```
1 from django.conf.urls import include, url
2
3 urlpatterns[
4       url(r'^update/', include('updating_service.
           urls')),
5 ]
```

Step 5: Run the server and check that the service is up and running by visiting *127.0.0.1:8000/update* in your web browser.

```
1 (env) c:\myproject\mysite>python manage.py
     runserver
```

5.6 WebFaction

As a proof of concept, the example application, along with the developed updating service application, was installed in a live production environment on a web server served by WebFaction. This section gives an overview of how the application is set up on the server and what is required to have the application up and running.

WebFaction is a complete web hosting service based in the UK. They provide everything a user needs when it comes to Web Hosting, Email, Database Hosting, Backups, Monitoring and System Administration, and Support. The team at WebFaction has done an amazing job of simplifying the hosting experience for their users and documenting their service in a good fashion. They bring everything together into one complete system for their users, and have three data centers available for usage: one in the U.S., one in Singapore, and one in Europe. The data center in Europe has been used in this case.

Since WebFaction was already hosting a website for me in a personal project, the choice of hosting the example application along with my personal page was made. By doing this, it was possible to prove that this updating method would not influence or conflict with other applications that the server might run; additionally, I did not have to pay for another hosting plan. Other services that provide Django hosting, such as BlueHost [55], HostGator [56], LiquidWeb [57], and DigitalOcean [58], should also be good alternatives to WebFaction.

5.6.1 VirtualEnv

As mentioned in section 2.4, by creating a virtual environment on the server, I could keep the installation of Django and its dependencies separate from the rest of the server. Since WebFaction does not do this automatically when creating a new Django application in the Dashboard, I had to set up the installation manually. The following installation process is based on a blog post published by Michal Karzynski [59] and is summarized with the key points in this section.

The first step was to create a new application through the WebFaction control panel as a generic mod_WSGI application running Python 3.6. Since this was the newest version of Python, virtual environment was already installed on the server and was ready to be set up.

After verifying that the mod_WSGI application, hereby called *master_app*, was set up and running, it was time to SSH into the server and set up the virtual environment.

```
1 $ cd ~/webapps/master_app
2 $ python3.6 -m venv env
```

This created a virtual environment inside the folder env. Note that the Python version specified should match the version of Python used when created the mod_WSGI application. By activating the virtual environment by the following line:

```
1 $ source env/bin/activate
2 (env) $
```

Every new installation of libraries was now installed inside the environment. PIP was also installed automatically inside the environment. I was then able to install the requirements for the Django application by importing a requirements.txt file and installing it through PIP:

```
1 (env) $ pip install -r requirements.txt
```

A detailed description of the requirements file can be found in section 5.4.

Since the requirements file contained the specific Django version that we wanted for the example application, Django is now ready to be used inside the virtual environment. By navigating to the root of the project, I could create a new Django project named mysite:

```
1 (env) $ django-admin startproject mysite
2 (env) $ chmod +x mysite/manage.py
```

The Django-project was now created, and I made sure that it was possible to execute the manage.py file so that the application could run.

After creating a Django project and ensuring that the files were executable, it was time to set up a virtual host to serve the site. This was done by editing the httpd.conf file located in /webapps/yourapp/apache2/conf/httpd.conf. The first step was to remove DirectoryIndex, DocumentRoot, and <Directory>:

```
1 DirectoryIndex index.py
2 DocumentRoot /home/USERNAME/webapps/master_app/htdocs
3 <Directory /home/USERNAME/webapps/master_app/htdocs>
4     Options +ExecCGI
5     AddHandler wsgi-script .py
6 </Directory>
```

Then, a virtual host was created and the log settings, along with WSGI-variables, were put inside the virtual host as follows:

```
1 <VirtualHost *>
2       ServerName USERNAME.webfactional.com
3
4       # Logging config
5       LogFormat ...
6       CustomLog ...
7       ErrorLog  ...
8
9       # Django WSGI settings
10      WSGIDaemonProcess master_app processes=2 threads
          =12 python-path=/home/USERNAME/webapps/
          master_app/mysite:/home/USERNAME/webapps/
          master_app/env/lib/python3.6/site-packages:/
          home/USERNAME/webapps/master_app/env/lib/
          python3.6
11      WSGIProcessGroup master_app
12      WSGIScriptAlias / /home/USERNAME/webapps/
          master_app/mysite/mysite/wsgi.py
13 </VirtualHost>
```

Note that the WSGIDeamonProcess originally pointed to the lib that WebFaction first created, which does not contain anything. I had to update this to point to the root of the project and to the lib-folder inside the virtual environment. The WSGIScriptAlias directive was also added to match the wsgi.py-file inside the Django project.

5.6.2 Let's Encrypt

The principal goal of Let's Encrypt [60] and the ACME protocol [61], [62] is to make security accessible to everyone. The strategy for doing so is a service for websites to obtain a browser-trusted certificate without any human intervention by the Certificate Authority, and with minimal effort from the server's administrator. This occurs by running a certificate management agent on the web server. The procedure happens in two steps. First, the agent proves to the CA that the web server controls a domain. Then, the agent can request, renew, and revoke certificates for that domain [63].

The example application and updating service app is running HTTPS on WebFaction. This is done by using an open source project on GitHub named *acme-webfaction* [62]. *acme-webfaction* is easy to set up and works perfectly.

The process of installing Let's Encrypt on the server involved the following steps given in the repository:

1. First, we need to install acme.sh on our server:

```
1 $ curl https://get.acme.sh | sh
```

2. Then, we need to download the tool provided in the repository, place it in the correct location, and make sure we are able to run the file:

```
1 $ wget https://raw.githubusercontent.com/
      gregplaysguitar/acme-webfaction/master/
      acme_webfaction.py
2 $ cp ./acme_webfaction.py ~/bin/
3 $ chmod +x ~/bin/acme_webfaction.py
```

3. Finally, we can issue the certificate:

```
1 $ acme.sh --issue -w /path/to/webroot -d example.
      com -d www.example.com
```

A certificate from Let's Encrypt is only valid for 90 days before a new certificate must be issued. As the repository suggests, the renewal should be set as a cronjob and by using the *acme_webfaction.py* file.

The application instance on WebFaction is also set to serve the site as HTTPS. This was done in the user dashboard on WebFaction.

5.6.3 Cron

As mentioned in section 2.12, a cron is used when one wishes to execute a file at a given time. In this project, the following cron is added to the cron table to execute the script for sending out an email notification if there are any CVEs connected to the currently installed version of Django. The following cron job is set up:

```
1 20 13 * * Tue "/home/magnusnn/webapps/master/env/bin/
      python" "/home/magnusnn/webapps/master/update-
      script.py"
```

This means that the file *update-script.py* is executed every Tuesday at 13:20.

Two more entities were added to the cron tab as well to ensure that the application is restarted frequently and to renew the certificate provided by Let's Encrypt. They are listed in section A.3

Chapter 6

Evaluation and further development

This chapter contains the evaluation of the methodology used in this project, evaluation of the solution as well as a recommendation for further development of the solution presented.

6.1 Methodology

The methodology for performing this research has been to observe which solutions that are out on the market today, followed by conducting unstructured and semi-structured interviews, and using the feedback in an iterative process to continuously improve the solution. Looking at the process with a retrospective view, I feel that the interviews should have had a bit more structure to it. If the interviews have had predefined questions, it would have been easier to evaluate each resonance against each other. With the lack of structure, the process after the interviews got a lot harder than it should have been as the results was so different.

The number of interviewing subjects should have been much higher than it was for this project. As mentioned earlier, the interviews were conducted with 4 students at the Department of Computer Science. By only testing the application on this small user group, one would not know whether the application is good enough for used in the Django community or not. However, the lack of network and connections prevented the process of interviewing different audiences that might have an interest of using this product.

Another point one can learn from this process is the importance of eval-

uating the interviews right after it happened. The interviews should also have been recorded to make post processing a lot easier. If a recording device had been used, one could after the interviews, transcripted the dialog, and had the opportunity to catch vital details that might have been missed in this process. Finally, but not least, the application should have been user tested with the students testing out the application in a task-driven way. The interviewing subjects should have been given the same set of tasks to evaluate if the application is confusing the users, or the users get lost when using it. If a set of tasks have been given during the interview, it would have been easier to see if the user got stuck while trying to perform a task.

6.2 Evaluation of solution

As mentioned in the previous section, the solution was tested on a small audience and to give a good evaluation on how well the solution is performing can be difficult. However, when concerning about the research questions, the developed solution is fulfilling the questions in a way that the users would prefer. As we have seen by the results of the interviews, the preferred notification method in this case was by email. An email notification system was developed, and feedback from the students were positive. The simplicity as well as the targeted information related to their application was a key point for the students.

When looking at the research question regarding the impact analysis, the provided solution helps the user understand critical parts that might not function properly after been updated. However, since none of the interviewees had any experience in using Django, they were not familiar with structure of the software and it was difficult for them to find the critical parts of the application that had a considerable risk when installing an update. I do believe that this part of the application can be improved to help users with a lower knowledge of the Django domain. The suggested improvements are listed in the next section.

Finally, when looking at the final part of this research concerning simplifying the updating process, the application allows for users to install and patch their system with a click of a button. When questioned about whether the users would use this feature, they all replied that they believed so as this simplified the process a lot. One limitation of this solution is the lack of feedback while doing an update, and the lack of backup done before an update is started. An idea for a solution is provided in the next section.

6.3 Further development

Since the time scope of the thesis was quite limited, and a time-consuming process with interviews was followed the solution might not be ready for release quite yet. However, based on the user feedback and tests, I believe that the finishing product is not fair away with some minor changes.

Error handling

One of the first thing to mention when talking about making the final touches on the solution is the error handling when installing an update with a click of a button. There is no backup solution in case the installation fails, or the installation causes the application to fail. The implementation of such a solution was not prioritized, because it would take some time to understand which errors that usually might occur when installing new updates.

An idea for such a solution could be to make a copy of the existing virtual environment, and performing the update in the duplicated environment. If the update is successful, the original virtual environment is replaced with the newly updated one. Time will show if this is possible in practice, or if there is a better solution suited for this purpose.

Feedback when installing an update

This is partially related to the previous problem of error handing. There feedback to the administrator should be improved when performing an update. When a user clicks the button, no feedback is given, and the user might get confused whether the update is installing. If the installation fails, there is no way of showing this to the user.

There should be implemented a log or progress bar displayed whenever an update is initiated to show to the user that the application is currently installing a new update. The updating-button should also be deactivated while a new update is being installed.

API request instead of individual databases

As the solution developed based on the research of how to notify, perform and evaluate impact of an update in Django was created, it has come to the attention that the application might have a drop in performance when used at a low network bandwidth. This is because whenever the application is displayed in the browser, a new request is made to multiple websites and

services to gather information and display this to the users. It is also unnecessary for every parties that is using the provided updating application to have their own database containing the information about the Django release information and the CVEs related to that version. A solution for this to create an API endpoint which users can query to retrieve the necessary information about their version. This might also ensure that the data stored at the API endpoint is up to date. Another benefit from this is the saved computational power needed on each of the application, this will probably also speed up the loading time of the application.

Use internal mail service

Since the project started out from scratch, the choice of using an external mail provider was made. However, as the project evolved, it soon became noticeable that the users would have more trust in the system if the email came directly from the application, and not through Mailgun as the solution does now. Before the final interviews, I made a custom setup for sending mails through Mailgun using a custom mail server located on Webfaction. As it turns out, the users felt more confident in receiving notification from this mail server.

Another argument for getting rid of Mailgun in the process of sending out emails is the limitation of the service. In the free plan that this project is using, the restriction of only sending out one email at the time could cause a problem when wanting to notify many system administrators at once. It also turns out that the process of setting up the mail serve provided by Python is not that hard, however, for testing purposes it was more convenience to use Mailgun, especially in the local environment.

Design

When it comes to design, the solution provided has not been tested for a usability score. The feedback from the interviews were good; however, the interviews did not follow a usability testing process to detect design flaws of the application. Another aspect when it comes to design is the model graph. The implemented solution generates a .PNG picture based on a .DOT file, however, there should be some indication with either colors or arrows to help the administrator to locate critical parts of the application that might cause an error when the update is installed. By coloring the path between critical parts of the models, I believe that the administrator would be able to look at the model graph alone and decide if the update is worth doing.

Chapter 7

Conclusion and further work

This chapter summarize and conclude this thesis and has a recommendation for further research in this field.

7.1 Conclusion

As frameworks are more and more commonly used when developing a web application, the importance of an easy and quick way of patching the framework for the end users is important. The updating process of Django is a cumbersome and time-consuming process. The research of this thesis has shown through interviews and observations that an easier and more convenient way of patching a Django application is possible and preferred.

A developed application for this purpose is provided, and has been tested on end users. The evaluation of the developed software shows that the application is in an early stage of development and would need some modifications before been ready for a release. It would need to be tested on a broader audience, and a code refactoring process for optimizing the application would be much valued before a release is provided.

7.2 Further Work

One recommendation for further research besides improvements of the application is to evaluate the statement coming from Django itself where they ensure that a security update will have no impact on the already existing application when installing a patch. It would be interesting to see if this statement is valid, and to see if the improvements that has been made might cause a security vulnerability further down the line.

Bibliography

[1] Django Software Foundation. *Django Upgrading process.* URL: https://docs.djangoproject.com/en/2.0/howto/upgrade-version/.

[2] Django Software Foundation. *Who Receives Advance Notification.* URL: https://docs.djangoproject.com/en/dev/internals/security/#who-receives-advance-notification.

[3] V Okanović and T Mateljan. "Designing a new web application framework". In: *MIPRO 2011 - 34th International Convention on Information and Communication Technology, Electronics and Microelectronics - Proceedings.* 2011, pp. 1315–1318. ISBN: 9789532330670. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-80052287599&partnerID=40&md5=529f2409502f80b4ed25c7dc39ca9450.

[4] Mohamed Fayad and Douglas C. Schmidt. "Object-oriented application frameworks". In: *Communications of the ACM* 40.10 (1997), pp. 32–38. ISSN: 00010782. DOI: 10.1145/262793.262798. URL: http://portal.acm.org/citation.cfm?doid=262793.262798.

[5] David A. Wheeler. "Why open source software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!" In: *Challenges* (2014), pp. 1–145.

[6] Glenn E Krasner and Stephen T Pope. "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System". In: *Journal of object oriented programming* (1988). ISSN: 0896-8438. DOI: 10.1.1.47.366.

[7] M F Sanner. "Python: a programming language for software integration and development." In: *Journal of molecular graphics & modelling* 17.1 (1999), pp. 57–61. ISSN: 1093-3263. DOI: 10.1016/S1093-3263(99)99999-0.

[8] David Robinson. *The Incredible Growth of Python.* 2017. URL: https://stackoverflow.blog/2017/09/06/incredible-growth-python/.

[9] Django Software Foundation. *Djangoproject*. 2017. URL: https://www.djangoproject.com/.

[10] PePy. *Django Download Stats*. URL: http://pepy.tech/project/django.

[11] Asad Jibran Ahmed. *Django Project Blueprints*. Packt Publishing, 2016. ISBN: 1783985429, 9781783985425.

[12] Django Software Foundation. *FAQ: General*. URL: https://docs.djangoproject.com/en/2.0/faq/general/#why-does-this-project-exist.

[13] Python Wiki. *Django*. URL: https://wiki.python.org/moin/Django.

[14] Django Software Foundation. *Why Django?* URL: https://www.djangoproject.com/start/overview/.

[15] Django Software Foundation. *Django in use at washingtonpost.com*. URL: https://www.djangoproject.com/weblog/2005/dec/08/congvotes/.

[16] Bala Kumar. *Django Archirecture*. 2013. URL: https://www.slideshare.net/balakumarp/django-framework/6.

[17] Django Software Foundation. *Django's security policies*. URL: https://docs.djangoproject.com/en/dev/internals/security/.

[18] Tim Graham. *Django's Roadmap*. URL: https://www.djangoproject.com/weblog/2015/jun/25/roadmap/.

[19] CVE Details - The ultimate security vulnerability datasource. *CVEdetails.com*. 2017. URL: https://www.cvedetails.com/.

[20] Kenneth Reitz. *Requests: HTTP for Humans*. URL: http://docs.python-requests.org/en/master/.

[21] Python Software Fundation. *Virtual Environments and Packages*. URL: https://docs.python.org/3/tutorial/venv.html.

[22] Mark Lutz. *Programming Python*. O'Reilly Media, Inc., 2006.

[23] Python Software Foundation. *PiPY - the Python Package Index*. URL: https://pypi.org/.

[24] T. Ylonen and C. Lonvick. *RFC 4252: The Secure Shell (SSH) Authentication Protocol*. 2006.

[25] Robin Seggelmann, Michael T??xen, and Erwin P. Rathgeb. "SSH over SCTP - Optimizing a multi-channel protocol by adapting it to SCTP". In: *Proceedings of the 2012 8th International Symposium on Communication Systems, Networks and Digital Signal Processing, CSNDSP 2012*. 2012. ISBN: 9781457714733. DOI: 10.1109/CSNDSP.2012.6292659.

[26] D J Barrett and R Silverman. *SSH, The Secure Shell - The Definitive Guide*. 2001. ISBN: 9780596008956. DOI: 10.1016/S1361-3723(05)00151-X.

[27] Nicholas Rosasco and David Larochelle. "How and Why More Secure Technologies Succeed in Legacy Markets: Lessons from the Success of SSH". 2003.

[28] Berkeley University of California. *Securing Network Traffic With SSH Tunnels*. URL: https://security.berkeley.edu/resources/best-practices-how-articles/system-application-security/securing-network-traffic-ssh.

[29] mitre.org. *Common Vulnerabilties and Exposures*. URL: https://cve.mitre.org/about/.

[30] Owasp. "OWASP Top 10 - 2013". In: *OWASP Top 10* (2013), p. 22. ISSN: 13514180. DOI: 1. URL: http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf.

[31] Eloisa Vargiu and Mirko Urru. "Exploiting web scraping in a collaborative filtering- based approach to web advertising". In: *Artificial Intelligence Research* (2012). ISSN: 1927-6982. DOI: 10.5430/air.v2n1p44.

[32] Leonard Richardson. *Beautiful Soup documentation*. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[33] MyDataCareer. *Python Web Scraping With BeautifulSoup: A How To Guide On Web Scraping*. URL: https://mydatacareer.com/pythonwebscraping/.

[34] wiseGEEK. *What Is a Notification System*. URL: https://www.wisegeek.com/what-is-a-notification-system.htm.

[35] Chris Gayomali. *The text message turns 20: A brief history of SMS*. URL: https://theweek.com/articles/469869/text-message-turns-20-brief-history-sms.

[36] Craig Partridge. "The technical development of internet email". In: *IEEE Annals of the History of Computing* (2008). ISSN: 10586180. DOI: 10.1109/MAHC.2008.32.

[37] Anas Baig. *91% of cyber attacks start with a phishing email.* 2017. URL: https://digitalguardian.com/blog/91-percent-cyber-attacks-start-phishing-email-heres-how-protect-against-phishing.

[38] CogNiTioN. *Newbie: Intro to cron.* 1999. URL: http://www.unixgeeks.org/security/newbie/unix/cron-1.html.

[39] Colin Robson. "Real world research". In: *Edition. Blackwell Publishing. Malden* (2011). ISSN: 08954356. DOI: 10.1016/j.jclinepi.2010.08.001.

[40] Ole T. Berg. *Spørreskjemametode.* 2012. URL: https://snl.no/sp%C3%B8rreskjemametode.

[41] Frode Svartdal. *Test: psykologi.* 2018. URL: https://snl.no/test_-_psykologi.

[42] Robert Arnold and Shawn Bohner. *Software Change Impact Analysis.* Wiley-IEEE Computer Society Press, 1996, p. 392. ISBN: 978-0-8186-7384-9.

[43] Steve Krug. *Don't Make Me Think!* 2006. ISBN: 0321344758. DOI: 10.1098/rspb.2009.1614.

[44] WordPress. *About.* URL: https://wordpress.org/about/.

[45] Glenn Leibowitz. *This CEO Runs a Billion-Dollar Company With No Offices or Email.* 2016.

[46] W3Techs. *Usage of content management systems for websites.* URL: https://w3techs.com/technologies/overview/content_management/all/.

[47] MITRE Corporation. *Wordpress: CVE security vulnerabilities, versions and detailed reports.* URL: https://www.cvedetails.com/product/4096/Wordpress-Wordpress.html?vendor_id=2337.

[48] JetBrains s.r.o. *JetBrains: Developer Tools for Professionals and Team.* URL: https://www.jetbrains.com/.

[49] JetBrains. *Python Developers Survey 2016: Findings.* 2016. URL: https://www.jetbrains.com/pycharm/python-developers-survey-2016/.

[50] JetBrains. *Python Developers Survey 2017: Findings.* 2017. URL: https://www.jetbrains.com/research/python-developers-survey-2017/.

[51] Python Software Fundation. *PEP 373 – Python 2.7 Release Schedule*. 2008. URL: https://www.python.org/dev/peps/pep-0373/.

[52] Django Software Foundation. *Django Models*. URL: https://docs.djangoproject.com/en/dev/topics/db/models/.

[53] Django Software Foundation. *Class-based views*. URL: https://docs.djangoproject.com/en/dev/topics/class-based-views/.

[54] Magnusnn. *GitHub Repository*. URL: https://github.com/magnusnn/django-updating-service.

[55] BlueHost Inc. *BlueHost*. URL: https://www.bluehost.com/.

[56] HostGator.com LLC. *HostGator*. URL: https://www.hostgator.com/.

[57] LiquidWeb LLC. *LiquidWeb*. URL: https://www.liquidweb.com/.

[58] DigitalOcean LLC. *DigitalOcean*. URL: https://www.digitalocean.com/.

[59] Michal Karzynski. *Setting up Django in Virtualenv on WebFaction's Apache with mod_wsgi*. 2013. URL: http://michal.karzynski.pl/blog/2013/09/14/django-in-virtualenv-on-webfactions-apache-with-mod-wsgi/.

[60] *Let's Encrypt*.

[61] *Acme protocol*. URL: https://tools.ietf.org/html/draft-barnes-acme-04.

[62] *Acme git repository*. URL: https://github.com/letsencrypt/acme-spec.

[63] Adwiteeya et al. "Shedding Light on the Adoption of Let's Encrypt". In: (2016). URL: http://arxiv.org/abs/1611.00469.

Appendix A

Code-Classes

A.1 update-script.py

```
1 import requests
2 from pycvesearch import CVESearch
3 import django
4
5
6 def get_all_cve_on_given_version(VERSION_NUMBER):
7     cve = CVESearch()
8     returned_data = cve.search('djangoproject/django
        :' + VERSION_NUMBER)
9     cve_list = []
10
11    for element in returned_data:
12        cve_list.append(element.get('id'))
13        cve_list.append(element.get('summary'))
14
15    return cve_list
16
17
18 def get_current_django_version():
19    version_number = str(django.VERSION).strip('()').
        strip(' ').split(', ')[:3]
20    return version_number[0] + "." + version_number
        [1] + "." + version_number[2]
21
22
```

```
23 def find_outdated_version_and_send_email():
24     ALL_CVE_ON_GIVEN_VERSION =
          get_all_cve_on_given_version(
          get_current_django_version())
25     if ALL_CVE_ON_GIVEN_VERSION:
26         counter = 0
27         text_message = "You are running version " +
              get_current_django_version() + " of Django
              .\n" \
28         "The website is currently vulnerable to " +
              str(int(len(ALL_CVE_ON_GIVEN_VERSION)/2))
              + " CVE-listed vulnerabilities and should
              be patched to the recommended version!\n"
              \
29         "Please visit the /update/-site of the
              application.\n" \
30         "Here is a list of the given CVEs that the
              website is vulnerable to:\n\n"
31         for element in ALL_CVE_ON_GIVEN_VERSION:
32             if(counter%2 == 0 ):
33                 text_message +=
                      '-----------------------------\n'
34             text_message += element + "\n"
35             counter += 1
36
37         return requests.post(
38             "https://api.mailgun.net/v3/
                  sandboxc176XXXXXXXXXXXXXXXXX.mailgun.
                  org/messages",
39             auth=("api", "key-8866-YOUR-API-KEY-
                  HEREXXXXXXXXXXX"),
40             data={"from": "Updating Service <
                  postmaster@sandboxc176XXXXXXXXXXXXXXXXX
                  .mailgun.org>",
41                   "to": "Admin <YOUR@EMAIL.COM>",
42                   "subject": "Update Django!",
43                   "text": text_message})
44
45
46 find_outdated_version_and_send_email()
```

A.2 views.py

A.2.1 imports

```
1 import django
2 import sys
3 import subprocess
4 from pycvesearch import CVESearch
5 from django.shortcuts import render, redirect,
    HttpResponse
6 from django.contrib.auth.decorators import
    login_required
7
8 from bs4 import BeautifulSoup
9
10 from urllib.request import Request, urlopen
11
12 from .models import DjangoVersion, InfoCVE,
    DjangoVersionCVE, DjangoVersionLTS
13 from mysite.settings import INSTALLED_APPS as
    INSTALLED_APPS
```

A.2.2 update_page(request)

```
1 @login_required
2 def update_page(request):
3     try:
4         cves = DjangoVersionCVE.objects.get(
              django__django_version=
              get_current_django_version()).cve.all().
              order_by('cve_id').reverse()
5     except DjangoVersionCVE.DoesNotExist:
6         get_cves_for_version(request)
7         cves = DjangoVersionCVE.objects.get(
              django__django_version=
              get_current_django_version()).cve.all().
              order_by('cve_id').reverse()
8
9         django_version_info = DjangoVersionLTS.
              objects.all().order_by('base_version')
10    current_django_version =
          get_current_django_version()
11    base_django_version = current_django_version.
          split('.')[0] + "." + current_django_version.
          split('.')[1]
12    current_base_version_info = DjangoVersionLTS.
          objects.get(base_version=base_django_version)
13    recommended_version = get_recommended_version(
          django_version_info)
14    recommended_base_version = str(
          recommended_version.split('.')[0] + "." +
          recommended_version.split('.')[1])
15    ids_to_look_closer_at_changelog =
          get_ids_from_changelog_based_on_dot_file(
          recommended_version, recommended_base_version)
16    changelog_info =  get_changelog_on_id(
          ids_to_look_closer_at_changelog,
          recommended_base_version)
17    changelog_url = "https://docs.djangoproject.com/
          en/" + recommended_base_version + "/releases/"
           + recommended_base_version + ""
18
```

```
19      return render(request, 'updating_service/
           info_page.html', {
20          'cves': cves,
21          'django_version_info': django_version_info,
22          'current_django_version':
              get_current_django_version(),
23          'current_base_version_info':
              current_base_version_info,
24          'ids_to_look_closer_at_changelog' :
              ids_to_look_closer_at_changelog,
25          'recommended_version':
              get_recommended_version(
              django_version_info),
26          'recommended_base_version':
              recommended_base_version,
27          'changelog_info': changelog_info,
28          'changelog_url': changelog_url,
29      })
```

A.2.3 get_current_django_version()

```
1 def get_current_django_version():
2     version_number = str(django.VERSION).strip('()').
         strip(' ').split(', ')[:3]
3     return version_number[0] + "." + version_number
         [1] + "." + version_number[2]
```

A.2.4 get_all_cve_on_given_version(VERSION_NUMBER)

```
1 def get_all_cve_on_given_version(VERSION_NUMBER):
2     cve = CVESearch()
3     returned_data = cve.search('djangoproject/django
         :' + VERSION_NUMBER)
4     cve_list = []
5     for element in returned_data:
6         cve_list.append(element.get('id'))
7         cve_list.append(element.get('summary'))
8     return cve_list
```

A.2.5 add_base_and_supported_to_database()

```
1 def add_base_and_supported_to_database():
2     version_list =
         get_django_supported_and_lts_versions()
3     for element in version_list:
4         try:
5             django_version_lts_object = (
                 DjangoVersionLTS.objects.get(
                 base_version=element))
6             django_version_lts_object.
                 supported_version = (version_list[
                 element]['supported'])
7             django_version_lts_object.lts = (
                 version_list[element]['LTS'])
8             django_version_lts_object.
                 latest_version_released = (
                 version_list[element]['latest'])
9             django_version_lts_object.mainstream = (
                 version_list[element]['mainstream'])
10            django_version_lts_object.extended = (
                 version_list[element]['extended'])
11            django_version_lts_object.save()
12        except DjangoVersionLTS.DoesNotExist:
13            django_version_lts_object =
                 DjangoVersionLTS.objects.create(
14                base_version=element,
15                supported_version = version_list[
                     element]['supported'],
16                lts = version_list[element]['LTS'],
17                latest_version_released =
                     version_list[element]['latest'],
18                mainstream = version_list[element]['
                     mainstream'],
19                extended = version_list[element]['
                     extended']
20            )
21            django_version_lts_object.save()
```

A.2.6 get_django_supported_and_lts_versions()

```
1 def get_django_supported_and_lts_versions():
2     req = Request("https://www.djangoproject.com/
         download/", headers={'User-Agent': 'Mozilla
         /5.0'})
3     webpage = urlopen(req)
4     soup = BeautifulSoup(webpage, 'html.parser')
5
6     table = soup.find('table', attrs={'class': '
         django-supported-versions'})
7
8     version_support = {}
9
10    for row in table.findAll('tr', {'class': ''}):
11        columns = row.find_all('td')
12        counter = 0
13        column_info = []
14        version_number = None
15        for column in columns:
16            column_info = column.get_text()
17            print(column_info.split(' '))
18            if(counter == 0):
19                version_number = column_info.split('
                     ')[0]
20                if len(column_info.split(' ')) >= 2
                    and column_info.split(' ')[1] == '
                    LTS':
21                    version_support[version_number] =
                         {'supported': True, 'LTS':
                        True, 'latest': None, '
                        mainstream': None, 'extended':
                         None}
22                else:
23                    version_support[version_number] =
                         {'supported': True, 'LTS':
                        False, 'latest': None, '
                        mainstream': None, 'extended':
                         None}
24                # break
25            elif(counter == 1):
```

```
26                  version_support[version_number]['
                        latest'] = column_info
27              elif(counter == 2):
28                  version_support[version_number]['
                        mainstream'] = column_info
29              elif(counter == 3):
30                  version_support[version_number]['
                        extended'] = column_info
31              counter += 1
32
33      for row in table.findAll('tr', {'class': '
            unsupported'}):
34          columns = row.find_all('td')
35          counter = 0
36          column_info = []
37          version_number = None
38          for column in columns:
39              column_info = column.get_text()
40              print(column_info.split(' '))
41
42              if(counter == 0):
43                  version_number = column_info.split('
                        ')[0]
44                  if len(column_info.split(' ')) >= 2
                        and column_info.split(' ')[1] == '
                        LTS':
45                      version_support[version_number] =
                             {'supported': False, 'LTS':
                            True, 'latest': None, '
                            mainstream': None, 'extended':
                             None}
46                  else:
47                      version_support[version_number] =
                             {'supported': False, 'LTS':
                            False, 'latest': None, '
                            mainstream': None, 'extended':
                             None}
48
49              elif(counter == 1):
50                      version_support[version_number]['
                            latest'] = column_info
```

```
51                  elif(counter == 2):
52                      version_support[version_number]['
                           mainstream'] = column_info
53                  elif(counter == 3):
54                      version_support[version_number]['
                           extended'] = column_info
55                  counter += 1
56
57          return version_support
```

A.3 Crontab

```
1 16,36,56 * * * * ~/webapps/master/apache2/bin/start
2 16,36,56 * * * * ~/webapps/master/apache2/bin/restart
3
4 20 13 * * Tue "/home/magnusnn/webapps/master/env/bin/
     python" "/home/magnusnn/webapps/master/update-
     script.py"
5 45 0 * * * "/home/magnusnn/.acme.sh"/acme.sh --cron
    --home "/home/magnusnn/.acme.sh" >> /home/magnusnn
     /log/cert.log 2>&1
```

A.4 requirements.txt

```
1  beautifulsoup4==4.6.0
2  certifi==2018.4.16
3  chardet==3.0.4
4  Django==1.10.1
5  django-extensions==2.0.7
6  idna==2.7
7  pydotplus==2.0.2
8  pyparsing==2.2.0
9  requests==2.19.1
10 six==1.11.0
11 urllib3==1.23
```