

NTNUI Form Management System

TDT4290 Customer Driven Project

Group 6

Jan Burak
Maria Iqbal
Siren Finvik Johansen
Jonathan Linnestad
Erik Liodden
Anders Salvesen
Kristian Bjørn Thoresen

February 11, 2019

Acknowledgements

We would like to thank NTNUI, specifically Even Kallevik and Anders Kirkeby. We would also like to thank the TDT4290 course staff, especially our supervisor Katerina Mangaroska.

Contents

List of Figures	iv
List of Tables	v
1 Introduction	1
1.1 Project Scope	1
1.2 Stakeholders	2
1.3 Report Outline	2
2 Pre-Study	3
2.1 Organization	3
2.2 Existing Workflow of the Organization	4
2.3 Current System - The Member Management System	4
2.4 Planned Solution - The Form Management System	4
2.5 Comparison between the Current System and the Planned System	5
3 Requirement Analysis	6
3.1 User Stories	6
3.2 Functional Requirements	7
3.3 Non-Functional Requirements	7
3.4 Changes in Requirements	8
4 Project Planning	9
4.1 Project Plan Overview	9
4.2 Team Organization	10
4.3 Project Schedule	10
4.4 Communication	11
4.5 Documents	12
4.6 Project Work Organization	12
4.6.1 Integration of Waterfall with other Methodologies	13
4.6.2 Working Hours	13
4.6.3 Meetings	13
4.6.4 Keeping Time	15
4.7 Quality Assurance	15
4.7.1 Time of Response	15
4.7.2 Code Standards	15
4.7.3 Code Review and Version Control	15
4.7.4 Templates and Standards	16
4.7.5 Testing	16
5 Risk Management	17
5.1 Risk Table Legend	17
5.2 Risk Table	18
5.3 Risk Matrix	18
5.4 Risks encountered during the Development Phase	19
6 Methodologies	20
6.1 Adopting the Waterfall Methodology	20
6.2 Waterfall Methodology in the Project	20
6.3 Adaptions to the Waterfall Methodology	21
6.4 Technology and Frameworks	22

7	Tools	24
7.1	Version Control Management	24
7.2	Communication and Documentation	24
7.3	Graphics and Figures	25
8	Architecture and Design	26
8.1	Architectural Drivers	26
8.2	Architectural and Design Patterns	26
8.2.1	Model-View-Template (MVT)	26
8.2.2	State pattern	26
8.3	Architectural Tactics	27
8.3.1	Modifiability Tactics	27
8.3.2	Security Tactics	27
8.3.3	Usability Tactics	28
8.4	Entity Relation Model	28
8.5	The 4+1 Architectural View Model	29
8.5.1	Architectural Views	29
8.5.2	Use Cases	32
9	Security	34
9.1	Security Features in Django	34
9.2	Abuse Cases	34
9.3	Static Security Analysis	34
10	Testing	36
10.1	Unit and Integration Testing	36
10.2	Automated Functional Testing	36
10.3	Usability Testing	37
10.3.1	Test Preparation	37
10.3.2	Test Subjects	38
10.3.3	Test Tasks	39
10.3.4	Test Proceedings	39
10.3.5	Test Analysis	40
11	Final Product	41
11.1	User Guide	41
11.2	Installation Guide	43
12	Group Reflection and Evaluation	44
12.1	Description of Group Dynamics	44
12.2	Reflection on Group Dynamics	44
12.3	Customer Relations	44
12.4	What Future Students Should Know	45
12.5	Feedback on the Course	45
13	Future Work	46
13.1	Modification of Current System	46
13.2	Improvements of the FMS	46
13.3	Suggestions for Future Implementation	47
13.4	Bugs	47
	References	48

A	Evolution in Requirements	i
A.1	Evolution in user stories	i
A.2	Evolution in functional requirements	i
A.3	Evolution in non-functional requirements	ii
B	Testing	iii
B.1	User Test Tasks	iii
B.2	User Test Introduction	vi
B.3	User Test In-depth Analysis	vi
C	Views	vii
C.1	Final product	vii
C.2	Paper prototype	x
C.2.1	Form-owner perspective	xi
C.2.2	Form-signer perspective	xv
D	Reports	xvii
D.1	Customer feedback	xvii
D.2	Summary customer meetings	xviii
D.3	Important status reports	xxii
D.4	Team contract	xxv
D.5	Gantt Chart	xxix
E	Other	xxix
E.1	Working Hours	xxix

List of Figures

1	Work Breakdown Structure of the FMS.	11
2	Estimated distribution of working hours spent on various activities.	14
3	Django’s MVT architecture	27
4	ER-diagram of the FMS	28
5	Logical View	29
6	Flow diagram visualizing the flow of the FMS after a user has received a signing notification.	30
7	Sequence diagram visualizing a successful attempt at filling out and signing a form.	30
8	Development view	31
9	Physical View of the system provided by the customer	31
10	Static security analysis using Bandit	35
11	Code Coverage of Unit and Integration Tests	37
12	Code Coverage of functional testing	39
13	Leader perspective view that allows the user to send out a form to another user.	41
14	View of form text(a), user information input(b) and password verification(c).	42
15	The default view of the regular user perspective	43
16	The main page of the system that was implemented by the customer.	vii
17	View of the regular user perspective containing received forms that have been signed.	viii
18	Leader perspective incoming view	viii
19	Leader perspective view containing outgoing forms that need to be signed by the receiver.	ix
20	Leader perspective view containing incoming forms that were signed by the current user.	ix
21	Leader perspective view containing outgoing forms that were signed by the receiver.	x
22	Paper prototype view that shows the first step in the process of sending a form.	xi
23	Paper prototype view that shows the second step in the process of sending a form.	xi

24	Paper prototype view that shows the third step in the process of sending a form.	xii
25	Paper prototype view that shows the fourth step in the process of sending a form.	xii
26	Paper prototype view that shows the fifth step in the process of sending a form.	xiii
27	Paper prototype view that shows the forms sent and filter functionality.	xiii
28	Paper prototype view that shows the active forms.	xiv
29	Paper prototype view that shows the approved forms.	xiv
30	Paper prototype view that shows the active forms.	xv
31	Paper prototype view that shows the first step in the process of filling a form.	xv
32	Paper prototype view that shows the signing step in the process of filling a form.	xvi
33	Paper prototype view that shows the active forms. The list is empty.	xvi
34	Paper prototype view that shows the approved forms.	xvii
35	Summary of Gantt Chart	xxix
36	Working Hours per team member per week	xxx
37	Estimated Hours the team spent on the different phases.	xxx

List of Tables

1	Stakeholders	2
2	Development team	3
3	A selection of the NTNUI committees	3
4	Comparison between the old and the new system.	5
5	User Stories	6
6	Functional Requirements	7
7	Non-functional Requirements	8
8	Team organization	10
9	Important project milestones	11
10	Risk table legend which describes the columns in Table 11.	17
11	Risk table	18
12	Risk matrix	19
13	Late deliveries	19
14	Technologies and frameworks	22
15	Tools	24
16	Use Case: U1	32
17	Use Case: U2	32
18	Use Case: U3	33
19	Use Case: U4	33
20	Abuse cases for FMS	35
21	Automated functional test 001	36
22	Automated functional test 002	38
23	Test schedule as specified by the customer	38
24	The shortcomings observed during the test and their possible solutions.	40
25	The dates for the changes in user stories	i
26	Evolution of user stories	i
27	Functional requirements version one	ii
28	The dates for the changes in functional requirement version 2	ii
29	Functional requirements version two	iii
30	The dates for the changes in functional Requirements version 3	iii
31	Functional requirements version three	iv
32	The dates for the changes in non-functional requirements	iv
33	Evolution of non-functional requirements	iv

Important abbreviations

CSRF - Cross Site Request Forgery

CSS - Cascading Style Sheets

FMS - Form Management System

FR - Functional Requirement

MMS - Member Management System

NFR - Non-Functional Requirement

NTNU - Norges teknisk-naturvitenskapelige universitet (Norwegian University of Science and Technology)

NTNUI - Norges teknisk-naturvitenskapelige universitets idrettsforening

SiT - Studentsamskipnaden i Gjøvik, Ålesund og Trondheim

US - User Story

WBS - Work Breakdown Structure

1 Introduction

“Digitization”, or more commonly known as “Digitalization”, has become crucial for every organization. Nowadays, digital information is considered the most efficient and effective way of creating, saving and sharing information. From medical data to legal documents, you can find all types of information being used and shared safely online. In this era of technology, many organizations have digitized their operations. However, there are still some organizations that do most of their work manually.

NTNUI is Norway’s largest sports organization which has been operated by students since it was founded in 1910. The organization is still using manual processes and procedures. This is mainly due to insufficient resources, as NTNUI is a voluntary association. The lack of digitization of their organizational operations not only consumes a lot of resources, but the organization also falls behind the modern digital trends. The delay in embracing the novelty has adverse effect on the efficiency and performance of the organization. However, NTNUI SPRINT took the initiative to digitize their processes and procedures. For this purpose, NTNUI SPRINT devised an online platform, the Member Management System (MMS), that manages the members of the organization online. Through this platform, a student can become a member of an associated sports group without physically visiting the office, as well as share information. The online portal is also used by board members to manage their operations.

Although the implementation of the MMS is a great start for digitizing NTNUI’s processes and procedures, there are still some processes that are not yet digitized. Form processing is an important management activity for NTNUI. Today, the form processes are carried out by using paper forms. The paper forms are signed by NTNUI members and then stored in archives. Since the current form processes do not scale well and are tedious work, NTNUI is planning to conduct these processes online. To achieve this objective, the MMS is to be extended with a Form Management System (FMS). The design and implementation of this module was assigned to our group as the topic of the Customer Driven Project. The goal of this project is to develop an FMS where form-owners can instantiate different forms and send them online to form-signers. When the form-signer receives a form, the form-signer can sign the form online and send it back to NTNUI. The form-signer will receive an email notification when a new form is available for signing, and the form-owner receive a notification when a form-signer have signed a form. This report presents the whole process of developing a prototype of a Form Management System for NTNUI.

1.1 Project Scope

We plan to develop an FMS that will address a subset of the goals NTNUI has defined in its strategy. Hence, the scope of our project work includes:

1. The development of an abstract form that the rest of the forms will inherit from, and that can be customized as per the requirement of the form-owner.
2. The abstract form should allow for the digitization of the existing paper forms.
3. The system should allow a form-signer to sign forms.
4. The signed forms should be saved in the database and can be viewed by different users based on credentials.
5. Notifications should be generated for relevant actors in the form of an email.
6. The system should provide an option for saving and signing a form manually if required.
7. In order to ensure the compatibility with the existing MMS, the FMS will be coded in Django.
8. The System should be tested for the defined functionalities.

9. Simple user interface. The interface in the FMS is likely to be changed, as the customer wants a consistent design. Hence, a simple user interface for the FMS will be developed for testing purposes.

The following functionalities was initially part of the scope, but were later taken out (see Section 3.4).

- Signing of Forms with BankID.
- Access control system regarding roles and access rights.

1.2 Stakeholders

The stakeholders are people or a group of people who may affect, be affected by, or perceive themselves to be affected by a decision, activity, or outcome of the project [1]. Stakeholders are all individuals who in some way or another contribute to the success or failure of the project [2].

The identified stakeholders of the following project are shown in Table 1. A list of the group members and their study program is shown in Table 2.

Stakeholders	Description, interest and influence
Development Team	The development team includes seven students that are currently enrolled at NTNU. All members of the team are equipped with knowledge of software development processes and procedures, and are skilled in different fields like web development, application development, software testing and software project management. The team will need to learn Django in order to successfully finish the project. The project will give hands-on experience to all team members regarding the development of a system with real life customers. The team will deliver a fully functional prototype.
Customer	The customer is NTNUI. In order to enhance the efficiency and performance of their organization, they need to reduce tedious manual work. NTNUI opted for digitization of their manual processes. The initiation of the following project is a part of NTNUI's digitization strategy.
Supervisor	The teams's supervisor is a key stakeholder who has a large interest in the project along with crucial influence. The supervisor also monitors the progress of the project and ensures its quality.
Department of Computer Science	The Department of Computer Science is a stakeholder with a large interest in the project but marginal influence. The project is completed as a part of the coursework of TDT4290 Customer Driven Project offered by the department.
End User	The end users of the project are the members of NTNUI who will use the system for accessing and signing forms digitally. These members include board members of NTNUI with greater privileges. The end users' interest is to adopt an effective system that simplifies common organizational processes.

Table 1: Stakeholders

1.3 Report Outline

The report is formulated to document all the processes used for the completion of the Customer Driven Software Project assigned to our group. Sections 1 and 2 include details regarding the introduction of the project and the customer along with the pre-study that is done to develop a complete understanding of the project. Section 3 documents the requirement analysis, which lays the groundwork for project planning in Section 4. In Section 5, the risk management plan, details regarding identified risks, and the plan for mitigating these risk are stated. Section 6 and 7 explain how the group implemented the system and the

Student Name	Program of Study
Jonathan Linnestad	MTDT Computer Science - Databases & Search
Siren Finvik Johansen	MTDT Computer Science - Interaction Design & Game Technology
Kristian Bjørn Thoresen	MTDT Computer Science - Artificial Intelligence
Erik Liodden	MTDT Computer Science - Artificial Intelligence
Jan Burak	MTDT Computer Science - Artificial Intelligence
Anders Salvesen	MTDT Computer Science - Software Engineering
Maria Iqbal	MSINFOSYST Information Systems

Table 2: Development team

usage of methodologies, technologies, frameworks and tools. The overview of the architecture and the design of the system are stated in Section 8 of the report. In Section 9 the security aspect of the project is discussed and in Section 10 the details regarding execution of testing plan are documented. Section 11 describes the final product, including details for the system installation and use. Section 12 is the group reflection which includes details of the group dynamics. Section 13 offer a glimpse of the prospect of the developed system and give suggestions for future work.

2 Pre-Study

2.1 Organization

NTNUI is the sports association of the Norwegian University of Science and Technology (NTNU) and has existed for more than 100 years. Unlike many other sports organizations, NTNUI is operated by students. NTNUI is Norway’s largest and most diverse sports association, with more than 13 000 members. NTNUI organizes sports events and other activities through its sports groups. These groups offer a wide range of over 70 sports, for instance: football, cycling, sailing, swimming, rugby, ice-hockey [3].

Organizational Structure

The main board of the organization consists of a President, Deputy, Manager of Finance, Main Treasurer, Manager of Information and contact for the student games, Material Responsible, Event Manager, Secretary and Leader of Baneutvalget, Head of HR and open practices, PR- and Sponsor responsible and Chief editor for Klubbavisa. Along with these board members, the organization also has committees working for various purposes. A selection of these committees is shown in Table 3.

Name of the Committee	Purpose
Bumerang	Lending sports equipment for free to all students
The Student Games	Responsible for organizing the biggest student sports festival in Trondheim
Hyttestyret	Responsible for managing and maintaining Studenterhytta
Treningsløftet	A cooperation between NTNUI, NTNU, SiT and The Student Welfare Association aiming to improve the fitness of students
Koiegrupper	Maintains and manages 23 ”Koier” all around Trønderlag
NTNUI SPRINT	Works with software development in cooperation with the groups.

Table 3: A selection of the NTNUI committees

Along with these committees, the organization has about 61 groups offering training and organizing competitions for a wide range of sports [4]. These groups are usually divided into subgroups, each one for a specific

degree of skill, e.g. "Rowing - Beginner", "Rowing - Intermediate" and "Rowing - Advanced".

Becoming a Member

Any student at NTNU can become a member of NTNUI by going to the reception of a SiT sports center. However, if a student needs to join a group that uses one of SiT's sports centers, they have to acquire a SiT membership as well. Once a person is a member of NTNUI, the person is able to join a NTNUI group. The processes of joining a group and becoming a board member are done manually. The SPRINT committee is working on developing a platform that enables carrying out these processes online.

2.2 Existing Workflow of the Organization

At NTNUI, most processes and procedures are carried out manually. From a management perspective, these procedures require a lot of time. There are several reasons why the organization lags behind in adopting digital technologies:

- Availability of funds
- Lack of resources for in-house development of a management system.
- Regularization of processes and procedures for management tasks.

Some of the most important organizational processes include:

- *Becoming a member of a sports group:* To become member of a sports group, students have to visit one of SiT's sports centers and submit a request using a paper form.
- *Request forms:* To place a request for various purposes, e.g. issuance of equipment or to release funds from a sports group, students have to sign a paper form that is available at the sports center.
- *Signing and storing forms:* All forms are paper based and need to be signed either by group members or board members. The signed forms are stored physically.
- *Changing roles and access rights using signed forms:* Forms are used by board members to change the board structure or to alter a member's access rights.

2.3 Current System - The Member Management System

NTNUI SPRINT is currently working on the MMS. The users of the MMS can log into the system as regular users or administrators. The MMS allows users to view and join groups, view events, and access details regarding groups and group members. The administrators have exclusive access to an admin panel that allows them to administrate the database. The system is developed in Django and uses HTTP requests to GET, POST, DELETE and PUT data. The MMS includes a database architecture that support the aforementioned functionality.

2.4 Planned Solution - The Form Management System

The planned solution will extend the current MMS with a FMS. The FMS will be accessible through the "Members portal" after the authentication of the user according to the access rights granted by the organization.

The two main categories of users are:

User: Form-Owner

After logging in to the system, the user will be able to:

1. Instantiate a form.
2. Assign a signer to the form.
3. The system will notify the form-signer.
4. View the status of instantiated forms

User: Form-Signer

After logged in to the system, the user will be able to:

1. receive a notification, through email, when a new form is available for signing.
2. Sign a form with password.
3. The system will notify the form-owner when the form is signed.

The Flow diagram visualizing these functionalities is shown in Figure 6.

Learning Requirement for the Planned Solution

The MMS is built using the Django framework, so the planned solution needs to also be implemented in Django. To accomplish this, the project team had to acquaint itself with the framework. The team was already familiar with Python, which Django is based on. This shortened the time required for acquiring the competence necessary for the project. The team learned to work with the Django framework using the official tutorial [5].

2.5 Comparison between the Current System and the Planned System

Table 4 shows a comparison between the current processes regarding form management and the execution of these processes using the FMS.

Existing workflow of NTNUI	Workflow integrated with FMS
A board member (form-owner) has to contact the relevant users (form-signers) when a user has to sign a form. The user has to meet the board member to sign the form and ask for a copy if it is desired.	The form-owner can instantiate a form and the system will notify the chosen form-signer by email. The form-signer can then sign the form online. The form will be saved in the database and the form-signer can view the form when desired.
NTNUI has to manage a wide range of forms, which requires printing and physical storage.	Through the FMS all the forms will be saved digitally, eliminating the requirement for space and physical resources.
Retrieval of signed forms requires time and resources	Signed forms stored in the FMS can be viewed online by authorized members, saving time and resources.
To check the status of forms, signed or unsigned, requires a lot of time and resources	The status of the forms can be easily checked in the FMS. Relevant users will be notified when a new form is available for signing or when a form has been signed.

Table 4: Comparison between the old and the new system.

3 Requirement Analysis

3.1 User Stories

The user stories were created by the development team after several meetings with the customer. The user stories are defined in Table 5. Their purpose was to understand the customer’s needs and for the customer to see whether the team understood the customer’s vision. The user stories were used to create the functional requirements, see Section 3.2, which were used to develop the system.

ID	User stories	Acceptance criteria
US1	As a form-signer, I should be able to fill out a form in Norwegian.	Given a user is logged in, when the user requests a form, the form should be returned in Norwegian.
US2	As a form-owner, I can require that a form needs external approval to be completed.	The system allows form-owner to choose external approves if it is required by the form.
US3	As a user with appropriate privileges, I should be able to see both signed and unsigned forms.	A user can see forms they have initiated themselves. Not others, even with appropriate privileges.
US4	As a form-owner with form instantiation privileges, I should be able to instantiate a form .	Given that a user is on the "send skjema" page, the system allows the user to instantiate a form.
US5	As a form-owner, I should be able to create a new form.	Given a form-owner with the right privileges, the system should allow the form-owner to create a new form and add it to the database.
US6	As a form-owner, I should be able to make a form available for signing for relevant form-signer.	Given that a form-owner is the president of a group, the system allows the form-owner to choose a form for signing and a form-signer to sign the form.
US7	As a form-owner, I should be able to see all my instantiated forms and their status.	Given that a form-owner has sent out forms, the system allows the form-owner to view all the form that they own and their status.
US9	As a form-signer, I should be able to view and fill out content in a form.	Given a form-signer has received a form, the system allows the form-signer fill out content in the form.
US10	As a form-signer, I should be notified when a form is available to be signed.	Given that a form-owner has sent out a form, the system should notify a form-signer when a new form is available for signing.
US11	As a form-owner, I should be notified upon form completion.	Given that a form-owner has sent out forms, the system should notify the form-owner upon form completion.
US12	As a form-signer, I should be able to sign a form using my password when required.	Given that a form-signer has a form to sign, the system should allow a user to sign a form with their password if it is required by the form.
US13	As a form-signer, I should be able to sign a form using BankID when required.	Given that a form-signer has a form to sign, the system should allow a user to sign a form with BankID if the form require higher order signing.
US14	As a form-signer, my access-rights to the system should change upon the completion of specific forms.	The system should change a form-signers access rights if it is specified in the form.

Table 5: User Stories

User stories US2, US5, US13 and US14 were taken out of scope, so they are not linked to functional requirements. This is because they were not prioritized by the customer and that their implementation would require too much time, given the team’s time constraints. However, the team kept the customer vision in mind in order to develop a modular system that can be expanded in the future to include these user stories. US8 has been removed altogether from Table 5 due to being covered by the modified version of US7. See Appendix A.1 for details.

Due to the customer’s technical competence, the team used the functional requirements instead of the user

stories to determine when the project was complete and to measure progress. This was approved by the customer (see Appendix D.1).

3.2 Functional Requirements

The final version of the functional requirements are shown in the Table 6 and are sorted by priority. The customer prioritized the requirements from low to high. This was so that the team knew in which order the customer wanted the functional requirements implemented. The functional requirements were created based on the user stories in Table 5. FR3 was replaced by FR13, and FR9 was removed as explained in Section 3.4.

After impediments caused by the customer during the development phase, see Risk ID 2 in Section 5.4, the customer decided that all functional requirements with a low priority should be considered future work. However, these requirements were also implemented, because the team had time to complete these requirements.

All the functional requirements are considered done by the customer, see Appendix D.1 for customer feedback.

ID	Functional Requirements	Priority	User stories	Done
FR1	The system should contain a modular abstract form that can be inherited by other forms.	High	-	Yes
FR2	The system should save signed forms in the database.	High	-	Yes
FR13	Presidents/Leaders in a group should be able to instantiate forms to be signed by members in that group.	High	US4	Yes
FR4	The system should contain functionality to sign forms using password.	Medium	US12	Yes
FR5	The form-owner can specify form-signers of a particular form.	Medium	US6	Yes
FR6	The form-owner should be able to view all the forms they have instantiated and the status of these forms.	Medium	US7, US3	Yes
FR7	Upon signing a form, the system should notify the form-owner(s) as defined by the form's notification policy.	Medium	US11	Yes
FR8	The forms can be customized for higher order authentication.	Low	-	Yes
FR10	Upon receiving a sign-request, the system should present the form to form-signer along with the ability to sign the form.	Low	US9, US12	Yes
FR11	The system should be able to notify actors on form instantiation if specified in the form's notification policy.	Low	US10	Yes
FR12	The system should contain a general "Team contract"-form (Team kontrakt) in the database.	Low	-	Yes

Table 6: Functional Requirements. All functional requirements are considered as done and completed by the customer as of 23-Oct-2018. See Appendix D.

3.3 Non-Functional Requirements

Non-functional requirements are shown in Table 7. The non-functional requirements were created in collaboration with the customer.

ID	Non-functional requirement	Done
NFR1	The system must be written as a Django-app. (Django 2.1)	Yes
NFR2	The code should have one or two integration tests and pass every test.	Yes
NFR3	The system should be user tested.	Yes
NFR4	The code should be documented, both in code and at GitHub wiki.	Yes
NFR5	The system should be available in Norwegian	Yes
NFR6	UIKit should be the main CSS-library for styling.	Yes
NFR7	All forms should use a Cross-Site Request Forgery (CSRF) token.	Yes

Table 7: Non-Functional Requirements. All non-functional requirements were considered done by the customer as of 23-Oct-2018. See Appendix D.

3.4 Changes in Requirements

The functional and non-functional requirements evolved throughout the project. The changes were due to the customer reprioritizing and removing some requirements, see Appendix A for details about the evolution of requirements.

We initially planned to implement the ability to sign a form with a higher level of authentication using BankID, see user story US13 in Table 5. The customer realized that they did not need BankID as much as they initially thought, and thus the requirement was taken out of scope. The system contains the ability to specify that a form requires higher order type of signing (see FR8 in Table 6) such as BankID or a paper signature. The FMS prevents a user from signing a form using their password (a *low-level* type of signature) if the form requires a higher order type of signing. However, the implementation of the *high-level* type of signature is considered future work, see Section 13.

The customer had a limited access control system, and the functional requirements regarding changes in access rights were considered out of scope and removed. After discussion with the customer, FR3 was agreed to be outside the scope and was replaced by FR13. FR9 was removed completely (see Table 31 in Appendix).

4 Project Planning

This section includes details of the overall plan that will be used in the project. The devised plan is based on the requirements defined by the customer. The main objective of this project plan is to control the project execution and to define the approach for the development of the software.

The people who will refer to this plan are:

- *Project Manager / Team Leader*: To plan the project schedule and resource needs, and to track the progress against the schedule.
- *Project Team*: To have a reference for what they need to do and when they need to do it, and to see if there are any dependencies among the activities.

This project plan contains the following information:

- *Overview of the project plan*: Provides a description of the project's purpose, scope, and objectives. It also defines the deliverables of the project.
- *Team Organization*: Describes the organizational structure of the project team.
- *Management processes*: Estimates the schedule, defines the major phases and milestones for the project, documents, and tools used for communication.
- *Applicable Plans and Guidelines*: Provides an overview of the software development process, including methods, tools and techniques to be followed, along with details regarding the quality assurance plan.

4.1 Project Plan Overview

The project plan was initiated at the start of the project after the first meeting with the customer. The customer introduced the existing system and defined requirements for future development.

The important phases in the project are as follows:

Pre-Study and Planning Phase

This phase of the project commences as soon as the scope of the project is defined. This was done in collaboration with the customer. One of the main customer requirements was for the system to be developed in Django. Therefore, the team has allocated time for a learning phase to become familiar with the framework.

Extensive time was allocated to the planning phase. The customer requirements were refined during this phase, and a thorough understanding of the existing system was developed. For this purpose, meetings were organized where the customer was requested to demonstrate the existing code. After the customer requirements were finalized, the system was designed, and a plan was devised for the implementation and testing phases.

Implementation Phase

The implementation phase will be initiated as soon as the major system requirements are finalized. The process begins with the development of the **AbstractForm**, which is a major functional requirement defined by the customer, see Section 3.2. This phase includes the development of the user interface and functionalities such as notifications and signing of forms using password authentication.

Testing Phase

The testing phase of the project comprises of user testing, unit testing and integration testing. The FMS developed in the implementation phase will be tested against the requirements. The results are documented in Section 10.

Presentation Phase

This phase of the project initiates one week before the deadline of the project. In this phase, the project presentation and the video will be prepared for the final examination.

4.2 Team Organization

The team organization structure is shown in Table 8.

Roles	Team Members	Responsibilities
Team Leader/ Project Manager	Siren Finvik Johansen	<ol style="list-style-type: none">1. Planning and coordinating team activities.2. Providing feedback about team progress to the supervisor.3. Motivating team members.4. Sharing internal reviews of the items made by the team.
Quality Manager	Jan Burak	<ol style="list-style-type: none">1. Ensure the quality of the end product and the overall process.2. Check that all project documents are consistent3. Arrange internal and external reviews.4. Monitor and review all testing activities.
Product Owner/ Customer	NTNUI	<ol style="list-style-type: none">1. Define all of the requirements. Check that the requirements are met.2. Ensure that resources required for the development are provided.
Test Manager	Jonathan Linnestad	<ol style="list-style-type: none">1. Ensure that the system is sufficiently tested according to the non-functional requirement.2. Perform frequent regression testing
Development Team Members	Jonathan Linnestad Siren Finvik Johansen Erik Liodden Jan Burak Anders Salvesen Maria Iqbal Kristian Bjørn Thoresen	<ol style="list-style-type: none">1. Assisting the Team Leader by signaling problems in an early stage.2. Executing plans made by the Team Leader.3. Keeping track of the time spent on various tasks.4. Following procedures and plans.

Table 8: Team organization

4.3 Project Schedule

A project schedule is devised using scheduling tools like the Work Breakdown Structure (WBS) and a Gantt Chart. The important milestones of the project are identified, and for ensuring the milestones are reached, the WBS approach is used.

Milestones Plan

Milestones planning is used to identify the important milestones in a project and does not include any detailed description of how the results will be achieved. This is result oriented planning rather than activity oriented planning[2].

The important milestones for the project are shown in Table 9.

Sr.No	Milestones	Important Dates
1.	Project Kick off and Task Assignment	August 28, 2018
2.	Finalization of requirements	September 18, 2018
3.	Setup code base (Abstract form)	October 2, 2018
4.	Mid Project Feedback	October 9, 2018
5.	Implementation of all functionalities	October 23, 2018
6.	Testing	November 01, 2018
7.	Last Report Draft	November 14, 2018
8.	Presentation	November 22, 2018

Table 9: Important project milestones

Work Breakdown Structure (WBS)

WBS is used to break down the project into small autonomous, controllable tasks with minimal dependency between them. The WBS exhibits important milestones of the project, along with the details of the deliverables and the work packages. The WBS is shown in Figure 1.

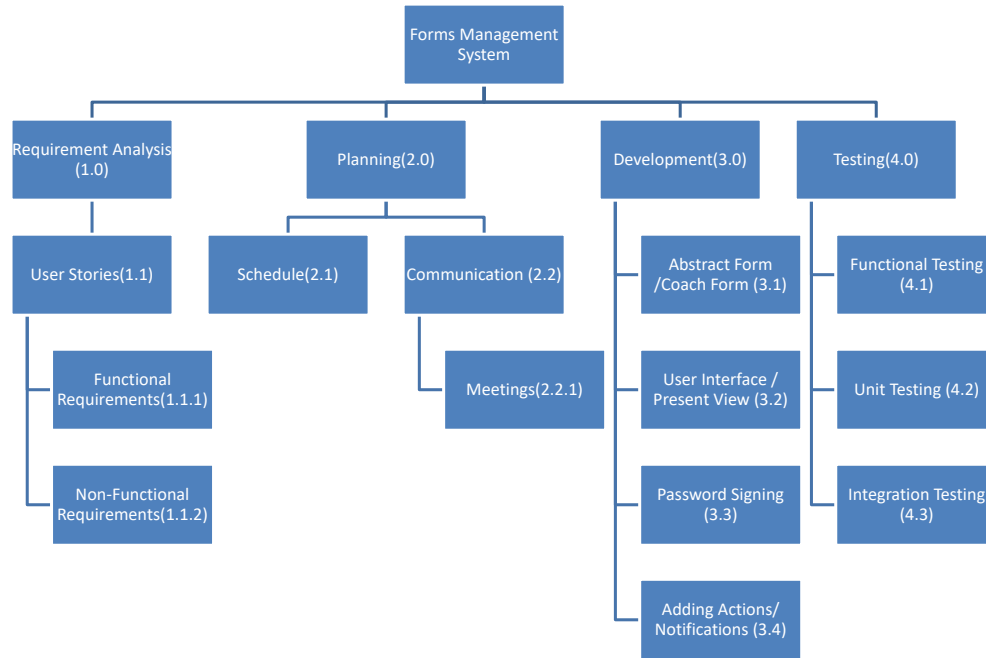


Figure 1: Work Breakdown Structure of the FMS.

Gantt Chart

For further details regarding the project schedule, the Gantt Chart has been used. The Gantt Chart includes tasks to be performed for the project and the duration required for the completion of the tasks. The Gantt Chart is attached in Appendix D.5.

4.4 Communication

Effective communication is the key for success in every project. As agreed with the customer, the team will use the following modes of communication to avoid any delays or uncertainty:

Customer meetings

Weekly meetings with the customer are organized in which the scope of the project was defined. The customer will be updated on the development's progress and regular feedback was received from the customer.

Weekly team meetings

The team will organize weekly team meetings, usually on Tuesday and Thursdays, in which the overall progress will be discussed along with the individual work progress.

Status Meetings

Status meetings are held with the supervisor in which the supervisor is updated on the development and report progress.

Slack and Asana

Slack and Asana are team collaboration tools suggested by the customer for the communication between the team and the customer. Both platforms are used for messaging, file sharing and planning.

Facebook Messenger

Facebook Messenger is also used during the project for internal communication among the team members. The team members uses messenger for the communication of internal queries, meeting times and other relevant information.

Google Team Drive

Google Team Drive is another important cloud platform that is used by the team and the customer for online sharing of documents. Google Drive is also used to keep track of all meetings notes, schedules and a repository for documents related to the project.

4.5 Documents

As a result of software project management and web application development, the following documents are created:

- Source Code
- Project Report
- Minutes of Meetings
- Data Flow Diagram
- Requirement Specification Document
- User Stories and other relevant documents

Most of the documents are available to the team on Google Team Drive for ease of access and security. Documents are also visible to the customer and the supervisor. The source code of the project is saved on GitHub. For project report writing and saving, Overleaf is used.

4.6 Project Work Organization

For the organization of the project work, the Waterfall methodology was adopted as the requirements of the project work were defined by the customer. However, Agile Methodology was incorporated into the requirement analysis phase to accommodate changes in the functional requirements. Along with this, different methods were used to ensure progress and for monitoring and control purposes. These methods include allocation of working hours, keeping time, and regular meetings that can help to oversee the progress of the project.

4.6.1 Integration of Waterfall with other Methodologies

As per the Waterfall model, the project plan focuses on the logical progression of the processes defined for Software Development Life Cycle[6]. This sequential and straightforward model includes step-by-step methods for the completion of the project. The phases to be used for the development of the FMS are as follows:

- *Requirements Gathering*: This phase involves the identification of user requirements through user stories. This process is initiated at the first meeting with the customer. However, due to the continuous refinement of the requirements, the process is kept agile.
- *Analysis and Design*: This phase includes analyzing and identifying the technical requirements. The development is based on the blueprints devised in this process in the form of an ER diagram and a class Diagram.
- *Implementation / Coding*: The actual development of the system usually takes place in this phase. Since the team will learn a new framework, pair programming techniques are intended to be integrated in this phase.
- *Testing*: Testing is an important Waterfall process which validates and verifies the developed system against the defined requirements and ensures quality.

4.6.2 Working Hours

There are a total of 7 members in the team, and according to the compendium [7], a team of 6-8 people should be spending 24 hours each per week on the project. However, in the initial phase, the team will allocate 16-18 hours per week for the customer meetings, team meetings, and meetings with the supervisor. Due to the requirement that the system should be developed in Django, time will also be spent on learning the framework.

Initially, the working hours per person were low due to:

- Uncertainty in requirements
- Delay in documentation and resources required from the customer

To accommodate this, the team members contributed more during the later phases of the project. The time spent on the various activities is shown in Figure 2. Raw data can be found in Appendix E.1.

4.6.3 Meetings

To monitor the progress and ensure the execution of the plan, regular meetings will be scheduled throughout the project. These meetings include:

Customer Meetings

Meetings with the customer are scheduled on every Tuesday throughout the project. Two to three hours have been allocated to these meetings. The most important meeting minutes are available in Appendix D. The purpose of these meetings is to:

- To define the requirements in a clear and concise way.
- To identify customer resources that are needed by the team.
- To understand the system and the database architecture used at the customer end.
- To verify and validate the project requirements.

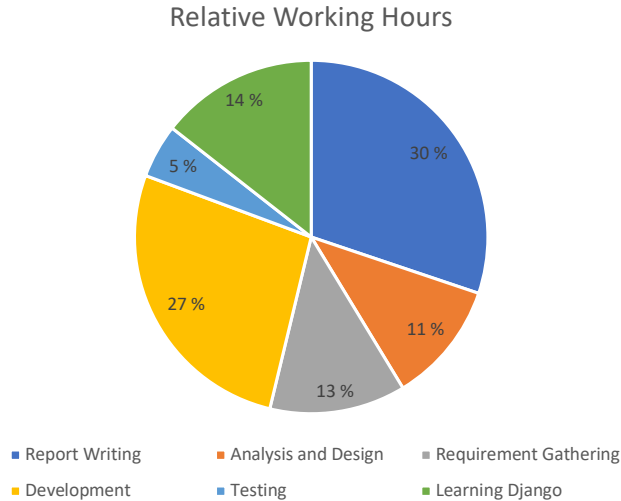


Figure 2: Estimated distribution of working hours spent on various activities.

Team Meetings / Status Meetings

Team meetings are organized on a regular basis to monitor and control the project progress. These meetings will be scheduled on every Thursday. The main objectives of these meetings are:

- Discussing the project status
- Presenting the individual work progress
- Discussion of the technology to be used
- Prioritization of tasks
- Feedback on tasks completed or to be completed
- Discussion of the agenda for the next customer meeting
- Assignment of tasks for the next week
- Status report for each week (These reports are available for the customer on Google Drive)

Meetings with the Supervisor

For meetings with the supervisor, a request should be sent a few days in advance. The meetings with the supervisor may include discussions of:

- Relations with the customer
- Feedback on the progress
- Clarification of course requirements
- Feedback on the project report
- Recommendations for future work

Team Leader Meetings

The team leader meetings are organized by the department almost every two weeks. The details regarding these meetings and the minutes of meetings should be available to all team members on Google Drive.

4.6.4 Keeping Time

For tracking working hours of the team members, a spreadsheet is used and maintained in Google Drive by the team leader. The objective of tracking time is to ensure that all group members are working on the project and devote sufficient time for the completion of tasks. The spreadsheet can be found in Appendix E.1.

It is not possible to avoid every hazard that can result in the loss of working hours. However, to cope with such situations, it was decided that in case a team member was not able to attend a meeting or could not work for certain hours, they have to take responsibility for making up the missing time. It is mandatory for team members to inform the team leader in such scenarios. Also, it should be made feasible for all team members to access the documentation, minutes of meetings or relevant material remotely, so that they are able to work or keep themselves updated.

4.7 Quality Assurance

These sections include the details of the quality assurance procedures the team plan to perform during the project, and how these processes will be implemented.

4.7.1 Time of Response

The customer has agreed to respond to queries through Slack and Asana within 24 hours. However, regular meetings with the customer are also held on a weekly basis to resolve issues without delays. The customer agreed to respond for:

- Approval for functional requirements – 24 hours
- Approval for prioritization of the requirements to initiate development – 24 hours
- Answer questions through Asana / Slack – 24 hours
- Providing required resources – 1 week
- Providing a test server – at the start of the project, however, it was not required for the project.

4.7.2 Code Standards

Since the FMS has to be integrated with the existing MMS, "Best Practices" will be used. The customers existing code has also been used as reference. It is important that the code can function as a basis for future development. For this purpose, common design and code conventions, which ensure consistency in the code, will be used. Comments should be added for improved readability of the source code and for future reference. Along with this, pair programming and peer reviews will be used during the development.

4.7.3 Code Review and Version Control

Code review is a process that is required for quality assurance purposes. With code reviews and version control, the team ensures that the source code and the documentation are following the standards required by the customer.

An internal/external approval approach will be adopted, along with the peer review method. Small atomic tasks will be allocated among the team and each task developed, reviewed and then added to the main git branch. This approach reduces potential problems that could arise during development. The reviews should

be made with an explanation, comments and annotations for future reference. Peer review is done through GitHub.

4.7.4 Templates and Standards

For templates and standards of documentation, the standard predefined templates are mostly used. For the report writing and other documentation, the conventions defined in the Compendium are being followed. However, for internal documentation like that of Minutes of Meetings, Weekly Status Reports, Requirement Tracking different templates are used that are already defined.

For the organization of files, Google Drive will be used as the main repository. For communication with the customer, Slack and Asana are used.

4.7.5 Testing

Testing is an essential part of Quality Assurance that not only ensures the quality of the product but also validates its usability. Even though the customer did not insist on extensive testing of the product, the project team has devised a testing plan for the validation and verification of the functional requirements defined in Section 3.

The testing practices used by the team are further described in Section 10.

Manual Testing

- *Usability Testing / Functional testing*: For user testing, the system should be tested for all the requirements defined in Section 3.2 and 3.3. It is a form of black-box testing, as it does not concern itself with the actual code, but focuses on functionality. It should be carried out after the implementation of the requirements to verify that the system works as intended and is user friendly.

Automated Testing:

- *Automated Functional Testing, black box*: Simulates a user flow and ensures that every step results in the expected output. It concerns itself on testing from a user perspective and not the code itself, as such it is a form of black-box testing. It will be based on the use cases defined in Section 8.5.2. It should be implemented on requirement completion, before pushing to production.
- *Unit and Integration Testing, white box*: In the unit tests, individual modules of the system are tested for associated functionalities. Integration tests consist of testing different modules against each other. They test the code itself, and as such are a form of white-box testing. They should be implemented while coding and should run before each pull-request.

5 Risk Management

This section will identify the different risks that may arise while working on the project and provide an analysis of the different risks. By doing proper risk management, the team will be prepared for handling potential risk impediments.

5.1 Risk Table Legend

Table 10 describes the meaning of each column in Table 11. The reason the team chose this format for the risk table is that this format was the one proposed in the compendium for TDT4290 Customer Driven Project [7].

Table column	Description
ID	A number used to identify different risks
Activity	Which of the activities of the project are affected
Risk factor	The name of the risk factor
Consequence	The impediments that a risk factor will cause. The impediments is categorized by the degree of impact they have on the project: <ul style="list-style-type: none"> • H (High impact) - The impediment will cause great damage to the project • M (Medium impact) - The impediment will do some damage to the project • L (Low impact) - The impediment will do little damage to the project
Prob. (Probability)	The probability that the risk impediment will occur. The probabilities will be divided into: <ul style="list-style-type: none"> • H (High probability) - The risk will most likely occur • M (Medium probability) - Equal chance the risk occurs or not occurs • L (Low probability) - The risk will most likely not occur
Strategy and actions	The strategies and actions the group can do in order to prevent the risk impediment to occur. The strategies are divided into: <ul style="list-style-type: none"> • Avoid - Change plans to avoid the problem • Transfer - Transfer risk management to another party • Reduce - Reduce likelihood of risk impediment by defining preventive actions • Accept - Accept the risk and do nothing about possible impediments
Deadline	The deadline says in what project phase the risk impediment may occur. Abbreviations used: <ul style="list-style-type: none"> • Cont. - Continuously • Dev. - Development
Resp. (Responsible)	The person or group responsible for preventing and handling risk impediment

Table 10: Risk table legend which describes the columns in Table 11.

5.2 Risk Table

Table 11 identifies the risks, the consequence of them happening, the probability for them to happen, what to do to prevent the risk from happening, in which phase the risk can happen and who is responsible to prevent the risk. A description of the meaning of each column is shown in Table 10.

ID	Activity	Risk factor	Consequence	Prob.	Strategy and actions	Deadline	Resp.
1	All	Group member arrives late to the group working session	M: The quality of the project results will decrease	H	Reduce: Define in the team contract penalties for arriving late.	Cont.	All
2	All	The existing system does not provide sufficient functionality to fulfill the requirements within time and skill boundaries	H: The group will have to do work that is not defined in scope and this will make the original scope unfeasible	L	Reduce: When discussing the scope of the project with the customer, ask the customer clearly if the existing system provides everything needed to implement the requirements agreed upon. Transfer: This impediment should be handled by the customer, any feature requirements dependant on non-existing functionality from the existing system should be removed.	Dev.	All
3	All	Late deliveries from the customer	H: The group will be blocked by the customer and person-hours will be lost	H	Reduce: Have a good dialogue with the customer throughout the project and send reminders when due date is approaching. Transfer: This impediment should be handled by the customer, any feature requirements dependant on non-existing functionality from the existing system should be removed.	Cont.	All
4	Dev.	Customer demands more features	L: The feature requests will be rejected	H	Reduce: Be clear in the planning phase with the customer that the scope agreed upon is not expandable.	Cont.	Group leader
5	All	Data loss	H: Loss of person-hours and the group members will need to redo the work	M	Reduce: Make sure all data produced by the group is uploaded and backed up in the cloud (e.g. Github, Overleaf, Google Drive etc.) as often as possible.	Cont.	All
6	All	Redundant work	H: Loss of person-hours and the involved group members will need to sort out which version to use	L	Reduce: Make sure the tasks are clearly defined and that group members update the other group members when a task's status has changed. There should also be Weekly stand-ups where each group member talks about their progress since last the stand-up meeting, this is done to discover redundant work early.	Cont.	All
7	All	Group member is sick	L: Loss of person-hours	M	Accept: Everybody gets sick once in a while.	Cont.	All
8	All	Group member has to learn new frameworks or tools	M: Loss of person-hours	M	Transfer: If the customer wants the developers to use frameworks or tools that they are unfamiliar with the customer has to take into account that the developers have to use time on learning the unfamiliar frameworks and tools.	Cont.	All

Table 11: Risk table. The risk table describes different aspects of each risk. A description of the columns can be found in Table 10.

5.3 Risk Matrix

The risks described in Table 11 are categorized in a risk matrix in Table 12.

Frequency/Consequence	Very Unlikely	Remote	Occasional	Probable	Frequent
Catastrophic		2			
Critical			3, 5		
Major		6		8	
Minor			1, 7	4	

Table 12: Risk matrix that relates a risk to its probable frequency and consequence. Green, yellow and red corresponds to low, medium and high severity respectively. The numbers in the risk matrix refers to the risks IDs in Table 11.

5.4 Risks encountered during the Development Phase

No matter how much one tries to prevent risk impediments, they still occur to some degree. The risk impediments the team encountered had big consequences for the project and are discussed in more detail below.

Risk (ID 2): *"The existing system does not provide sufficient functionality to fulfill the requirements within time and skill boundaries"*. Initially the team was to implement a feature, see Appendix A, that changed user's privileges when a user signs certain types of forms. E.g. a regular member that has signed a coach form should be granted rights for sending a form to another member. In order to implement this functionality we were dependent on a proper access control system in the existing system. The existing system contained an access control system to some degree, but it was not implemented sufficiently for the sake of the project. This resulted in that the functional requirements that require a proper access control system would not be possible to implement. Since creating a proper access control system was not a part of the project scope, we did not consider implementing it. The customer acknowledged that the existing access control system was inadequate for implementing FR3 and FR9 (see Table 31 in Appendix). Therefore, the customer agreed to remove these functional requirements from the scope. Although implementing a proper access control system was not a part of the project scope, thoughts on how a potential implementation would look like are presented in Section 13.

Risk (ID 3): *"Late deliveries from the customer"*. During the planning and development phase the customer failed several times to provide the promised deliverables on time. Table 13 highlights the late deliveries and the consequence of them. After multiple late deliverables from the customer, the customer decided that all functional requirements with a priority *low* will be considered as "future work".

Late deliverable	Consequence
Fork of customer's GitHub repository	In a customer meeting we requested that they would create a fork immediately.
Test server	The team agreed with the customer that they would provide mock data users instead of a test server.
Access to forms	The forms implemented are "example forms", and need to be modified by the customer
Mock data	Limited mock data was created by the team.

Table 13: Late deliveries

Risk (ID 6): *"Redundant work"*. The team experienced some redundant work during the learning phase. To make the Django learning process more useful, some group members started to implement parts of the system as a way to learn Django. Some of the code created by the different group members during this phase was decided to be part of the FMS. The overlapping code had to be merged, but this was done very quickly as the amount of overlapping code was not extensive. Since this happened in the learning phase, it did not have a big impact on the project progress. Redundant work occurred only in the learning phase. During

development, we defined the tasks clearly. Redundant work during this phase was not a problem.

6 Methodologies

This section will discuss the different software and development methodologies used in the project.

6.1 Adopting the Waterfall Methodology

After the initial meetings with the customer, the team decided to use the Waterfall methodology for the development of this project. This methodology suits the project best for the following reasons:

Defined Structured Organization

All the aspects of the project, which include requirements (Section 3), modes of communication, development mode and testing requirements (Section 6.4 and Section 7), were defined by the customer at the beginning of the project. In order to comply with the customer requirements, the team chose to follow a disciplined approach of Waterfall methodology. The orderly processes of the methodology include design, development, testing and implementation.

Possibility to Integrate Changes at Earlier Stages

The Waterfall methodology allows incorporating changes at earlier stages during the design process. This was important for the project as there were some uncertainties in the customer requirements. Due to changes in the requirements with the passage of time, the design of the project evolved. The record of the requirements' changes can be viewed in Appendix A. The team and the customer had several meetings to finalize the requirements, especially for using an API for BankID and for defining the scope of the project given the system available at their end.

Definite and limited scope of the project

The FMS the team developed has to be integrated with the existing MMS which is still under development. Therefore, the scope of the project was limited from the start. It also meant that the customer had definite requirements regarding development. These requirements included using the Django framework, as well as implementation of an "Abstract forms" that ensured modularity for implementing future forms. To accommodate these requirements and meet the scope of the project, a linear approach of the Waterfall methodology has been adopted.

6.2 Waterfall Methodology in the Project

As discussed in Section 4.6, the project has been completed following the Waterfall methodology. The details regarding integration of the Waterfall methodology are as follows:

Requirements gathering and Planning

Detailed planning is the core process of the Waterfall methodology and for this process it is mandatory that the requirements are defined and clear to every stakeholder. However, in the beginning, the customer requirements were not certain as some functional requirements required reconsideration by the customer.

Requirements gathering using Agile

To define the requirements and scope of the project, several meetings with the client were scheduled. Some of the requirements were a cause of uncertainty such as:

- Signing of a form using BankID: For implementation of this requirement, a BankID API has to be integrated. This could be expensive as there is a fee on each use of BankID. Alternatives were discussed in different meetings which led to the agreement to use different levels of signatures.

- The current system lacks a proper access control system. This meant that the changing of access rights upon form completion could not be implemented. Therefore, to accommodate this change the scope of the project was redefined with the consent of the customer.

After meetings and discussions, both within the team and with the customer, the requirements were changed and logged in Appendix A. A decent portion of time was dedicated to requirement gathering and refining, as the rest of the project was highly dependent on the requirements.

Analysis and Design

With the conclusion of the requirement gathering process, the analysis and design phase was initiated. As the result of the established requirements, the system's technical design was devised. According to the Waterfall methodology no coding takes place at this point. However, the technical requirements were identified such as programming language, requirements for user interface and database infrastructure. During this process the component diagram was defined along with an ER diagram. The ER diagram and component diagram are shown in Figure 4 and 5 respectively.

Implementation and coding

The blueprint for implementation and coding is provided by the analysis and design phase. As per the non-functional requirement NFR1 in Table 7, the FMS has to be developed using the Django Framework. Thus, Django was learned and used in this phase.

To develop a functional prototype, practises from Extreme Programming and Scrum were adopted and used. See Section 6.3 for details.

Testing

Testing is the final phase in the Waterfall methodology. The main objective of the phase is to verify and validate the product. For this purpose, a thorough usability test was conducted in which the prototype was tested for use cases 1, 2 and 3 described in Section 8.5.2. Minor bugs were reported and fixed during the testing process. The prototype was also unit- and integration tested. This ensured that the individual modules were working and free of bugs. The details of the testing phase are documented in Section 10.

6.3 Adaptions to the Waterfall Methodology

During the implementation phase, some practices from Extreme Programming and Scrum were used. Extreme Programming values communication, feedback and respect [8]. The chosen practices of Extreme Programming and Scrum are:

Pair Programming

Pair programming allows the team members to work together in pairs to enhance learning and tackle bigger tasks together. This was useful since none of the team members had experience with Django and the team members could learn Django together. Pair programming was also useful to use each others expertise and help each other during the project. Using Pair Programming enhanced learning among the team members and helped the members to get a better understanding of the code base.

Coding Standard

Coding standard ensures that the team members agree upon a set of rules. The rules included that the team should, to the best of their ability, use the same coding standard as the customer. This is to have consistency through the whole system, and have the same file structure as the customer.

Collective Code Ownership

Collective code ownership makes each team member responsible for the code and everyone is allowed to change any part of the code. This is so that each team member has knowledge about most of the code. This was managed to some extent, so if a team member was sick or was unable to work, others could take over and continue the work. Team members could contribute if there were any problems with the given task or help with understanding of the task.

Continuous Integration

The team used git for version control and Github to share the repository. To avoid "integration hell" [9] the team worked on separate branches that were continuously updated to reflect changes done to the main branch. This ensures that the developers work on the same or similar stage of the code. The tasks were divided among developers into smaller atomic tasks. Upon completion, the code was reviewed and then added to the main branch. All the functional requirements were completed as a deliverable to the testing phase.

Weekly Stand-ups

A practice that was adopted from Scrum was Daily Stand-ups, but since the team did not meet daily, it was changed to Weekly Stand-ups. The Stand-ups were held to inform the other team members what had been done since the last Stand-up, what problems they may have encountered and what they were planning on doing next. See Appendix D for a summary of some important Weekly Stand-ups.

6.4 Technology and Frameworks

The current system is built as a web application using Django 2.1. Since the team was expanding an existing system (MMS), the team was given little flexibility in the choice of language and framework to use. The main technologies and frameworks required by the customer to be used for developing the product are shown in table 14.

Technology type	Technology name	Description
Web Framework	Django 2.1	Already in use by the existing system
Programming language	Python 3	Required by Django 2.1
Build Environment	Docker	Create an identical environment for every developer
Front-end styling	UIKit	Required by the customer as a non-functional requirement. See NFR6 in Table 7

Table 14: Technologies and frameworks used for developing the product.

Details regarding other technologies and frameworks used are as follows:

Python

Python is an interpreted high-level general purpose programming language [10]. Django 2.1 requires Python 3 [11], and since Django 2.1 was required by the customer, the FMS was developed in Python 3.

Django

Django is a free and open source web framework written in Python. Django handles the database setup and interaction, the logic and the visual presentation of a web application [12]. The MMS developed by the customer is implemented with Django, and the customer wants the team to develop a Django module to expand their system.

Docker

Docker performs operating-system-level virtualization, and can be used to make sure that all the developers have identical environments, separated and generated isolated for this project. This has several benefits, including an increase in developer productivity and faster issue resolution [13]. The existing system that the team got from the customer as a starting point were configured to use Docker. However, team members developing on machines running Windows Home had issues with the Docker-setup. This issue was solved manually by modifying the global environment to reflect the project's environment. Although this solution works, it is up to the developer to make sure that the environment stays consistent. Another benefit of docker is that the production and testing environment mirrors the development environment. This means that tests

are run in a identical environment as production. Other benefits with Docker are easier deployment through containers.

Travis-CI

The customer uses Travis-CI [14] for automatic unit- and integration testing. The repository set up for the team for this project included a configuration of Travis-CI, but the team chose to disable continuous integration due to a bug in the configuration given by the customer that the team had insufficient permissions to correct.

UIKit

UIKit is a CSS library that simplifies and standardize the front-end styling. Although our job was primarily back-end development, some front-end development had to be done. The customer uses UIKit for styling of the other parts of the system, and the customer specified the use of UIKit as a non-functional requirement (see NFR6 in Table 7).

Selenium

Selenium is a browser simulator most commonly used to test the functionality of a web page. The team used Selenium to create automatic tests which simulates a user clicking through the application. Using the django-nose test-suite the tests written assert if a given state is as expected. We have used Selenium as it has strong python support and the customer had already set up a working environment.

Bandit

Bandit [15] is a program designed to find common vulnerabilities in Python code. It does so by parsing each file. It logs a report showing the vulnerabilities and the severity of each one. It was used to make sure that there were no severe security vulnerabilities.

7 Tools

The customer suggested that the team should use many of the tools that they already use for the project in their in-house developing team. The customer provided most of the tools used for this project, but allowed the team to suggest and use other tools as well if there was a need for it. The team decided to use the tools provided, as many of the team members were familiar with those and the tools were either open source or well known.

The tools provided by the customer that the team used for management of the project, communication between the team and the customer, and source control management are summarized in table 15.

Type of Tool	Tool used	Description
Communication	Slack	Main communication channel between the team and the customer.
Version Control	Git	Well known tool and already used by NTNUI.
Document Sharing	Google Drive	Versatile tool for document sharing, creation and collaboration. Used by the team for meeting agendas, reports and a place to share important documents for the project that were not part of the code.
External hosting of source code	GitHub	Well known service for hosting git repositories and already used internally by NTNUI.

Table 15: Tools for Management, Communication and Version Control provided by the customer and used by the team.

7.1 Version Control Management

The current system used `Git` for version control management. `Git` is open source distributed version control system and is used by many well known and large open source projects around the world [16]. Most of the team members are familiar with `Git` and it was therefore natural for the team to continue to use this tool during development.

Branch Strategy

`Git` allows frequent branching and merging of code, and the team used this approach during development. The code had a `dev` branch that was always a working version of the product. All features were added by first branching of `dev` to a separate `feature` branch that was later merged into `dev` after internal approval and review by at least one other team member. This setup made it easy for the developers to work on different features in parallel with each other.

Merge Strategy

A `feature` branch is merged into `dev` after an approved pull-request of the `feature` branch and after the feature is working and complete. If a merge conflict occurs, the owner of the pull-request and the owner of the conflicting part of the code resolve the conflict together during the merge.

7.2 Communication and Documentation

GitHub

The source code of the current system is hosted at GitHub as a private repository only visible to the customer and the team members. The customer created a new repository for the team containing a fork of the current

system. The team decided to continue to use this repository as a remote hosting of the project during development. GitHub provides a good way to look at commit history and to discuss pull-requests before they are merged with the main branch [17].

Slack

The customer suggested to use Slack as the main communication channel between them and the team. Slack has the advantage that communication is confined in different channels and threads and it is therefore easier to organize and navigate in the communication history [18]. Slack was used to communicate with the customer, clarify unforeseen issues during development, and schedule meetings. Two channels were created: one for communication between the team and the customer and one for internal communication between the team members. However, Facebook Messenger was more convenient for the internal discussions in the team as team members answer more quickly there, and the internal slack-channel was rarely used.

Google Drive

Google Drive is a versatile cloud platform for file and document sharing, creation and collaboration [19]. It is free to use and all team members are familiar with it. The customer provided the team with access to a shared *Team Drive* and suggested that we used this as the main tool for storing and sharing files not directly related to the code. Both the team and the customer had access to the *Team Drive*.

Overleaf

Overleaf is a free online tool for creation and collaboration of documents written using L^AT_EX[20]. The team chose to use Overleaf for the technical report as it would make it easier to write different chapters in parallel and because LaTeX allows full control over the layout and formatting of the report. This includes automatic table and figure references in the text and linked chapters in the compiled PDF file. NTNU has a premium licence for its students, which makes all team members able to work on the same document at the same time.

7.3 Graphics and Figures

Sketch

Sketch is a vector graphics editor that allows users to easily create paper prototypes for software systems. The application also provide the functionality to link the paper prototypes together so that it's possible to run a paper prototype demo within the application. The team chose to use this tool because a team member was familiar with the software and owned a license. In this project, the team used Sketch to create paper prototypes (see Appendix C.2) for the system, showcase the paper prototypes and share the paper prototypes through Sketch's cloud service.

Draw.io

Draw.io [21] is an online tool for creation of various diagrams including flow diagram, class diagram, etc. The team chose to use draw.io because it is free, easily accessible and works well for our purpose. It also supports collaboration.

8 Architecture and Design

This section explains the main architectural drivers, namely technical constraints and quality attributes. The quality attributes will be linked to relevant functional and non-functional requirements (see Table 6 and Table 7). The architectural patterns and tactics used are also presented. Lastly the 4+1 architectural view-model is used to describe the architecture from the different viewpoints of various stakeholders.

8.1 Architectural Drivers

A significant technical constraint is that the application needs to be developed using the Django framework (NFR1). This limits the group's architectural decisions, as further development needs to adhere to the architecture Django is built upon. Another technical constraint is that all styling should use the UIKit CSS-library (NFR6). This may have some implications for the usability of the product.

The customer explicitly asked for a modular forms app within their existing MMS. Modifiability, as it is closely related to modularity, is therefore the primary quality attribute when designing the FMS. The FMS should be as small and modular as possible. It should be easy for programmers to add new forms to the system without replicating code. This quality attribute can be linked to FR1 and due to this requirement's priority, it is regarded the most important.

Security is a secondary quality attribute. Data confidentiality and data integrity is especially important. The FMS will potentially collect vast amounts of data, and it is important to ensure that this data is protected against unauthorized access as well as unauthorized manipulation [22, page 147]. This quality attribute can directly be linked to FR4 and NFR7. It can implicitly be linked to FR6, FR13, as these requirements relate to access control.

Usability is another secondary quality attribute. The different views created should mirror the existing solution as much as possible in terms of visual design. The user should be presented with views that are intuitive and easy to interact with. This quality attribute is linked to FR6 and NFR6.

8.2 Architectural and Design Patterns

8.2.1 Model-View-Template (MVT)

Django is a python web framework and is built around the MVT-pattern. MVT is slightly different from the Model-View-Controller (MVC) pattern in that the controller part is handled by the Django framework itself. As illustrated in Figure 3, the *template* layer is the presentation layer, defining the layout and structure of the design, and is usually a collection of HTML files containing both HTML and Django Template Language (DTL). The *model* is the database layer, and one model usually corresponds to a table in the database. In other words, this layer contains the data and handles how the data should be accessed and validated. The *View* functions as a bridge between the model and template. The view receives HTTP requests, accesses the relevant data in the database, and returns an HTTP response while delegating the formatting of these responses to the templates [23, 24].

8.2.2 State pattern

The state pattern is a behavioural design pattern that has been used in order to make the FMS as modifiable as possible. Each form will have a list of actions that the system should perform and the state of each form is stored. Defining the actions is the responsibility of developers and form creators. Examples of such actions can be to *notify* the form-signer and to *change access rights* when a form has been signed or approved.

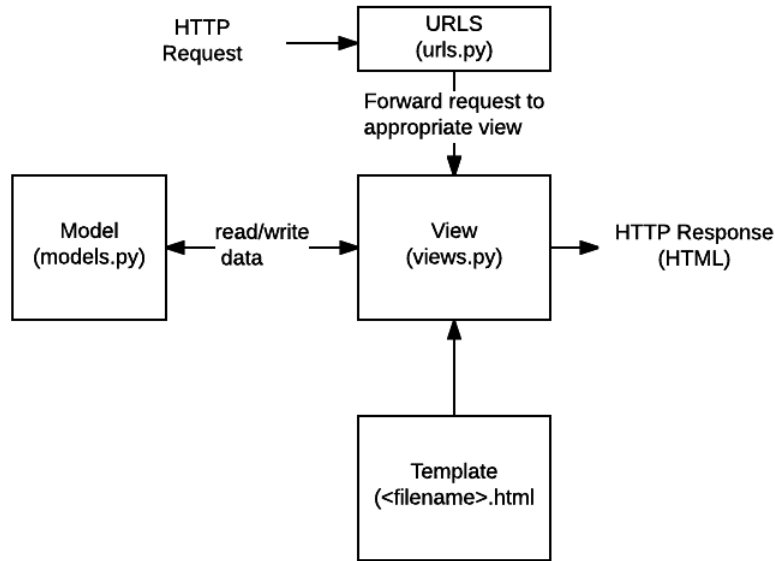


Figure 3: Django’s MVT architecture [23]

The actions in a form’s action list will be performed by the system sequentially until the form is marked as complete. It should be easy for developers to add new actions if needed, and these actions should not affect the existing actions.

8.3 Architectural Tactics

8.3.1 Modifiability Tactics

One of Django’s fundamental goals is *loose coupling* and *high cohesion*. In Django, coupling is reduced by restricting dependencies and having different layers which are independent of each other wherever possible. This means that the various layers should not be aware of each other unless it is absolutely necessary. For example, the view layer operates independently from the template layer, and the model is not concerned with how the data is presented to the user [25].

While developing the FMS, the team tried to build upon the advantages of Django’s architecture by reducing coupling and increase cohesion within the FMS. This includes *increasing semantic coherence* by ensuring that all form-related logic is contained within the forms application, and that its responsibility is clear and concise. Loose coupling within the FMS has been maintained by ensuring that the different *views* are properly decoupled from the various templates they are rendering.

8.3.2 Security Tactics

The security tactic *Authenticate actors* has been used in order to ensure that the user accessing the system actually is who they purports to be. A user needs to be logged into the system in order to get access to the FMS. Upon signing a form, the the password needs to be confirmed by the user. After login, the security tactic *authorize actors* has been used to ensure that the user has the right credentials to perform certain tasks. Only a user with the role ”president/leader” is able to instantiate a new form, and a user is only

permitted to fill out a form if it is part of the "signers-list" specified on the form, and has not previously filled out the form [22, page 152].

8.3.3 Usability Tactics

To facilitate efficient use of the system, as well as minimize the impact of errors and increase overall user satisfaction, one can make use of some usability tactics [22, page 177-180]. Django automatically handles form validation, and gives the user the appropriate error messages when forms are filled out incorrectly. In addition, the FMS supports user initiative by having the ability to *pause/resume*, meaning that when a user is filling out a form, the input should not be lost if the user decides to continue filling out the form at a later time. In addition, the visual layout presented to the user is developed to seem as similar to the existing apps as possible in order to make a consistent interface the user can interact with.

8.4 Entity Relation Model

The ER-diagram of the database structure of the FMS that was created by the team is shown in Figure 4. The entities `UserModel` and `GroupModel` are parts of the MMS and connects the FMS to MMS. The `FormTextModel` is an entity-class where static form information is stored. The various form types in the FMS should be disjunct sub-classes of the `AbstractFormModel`.

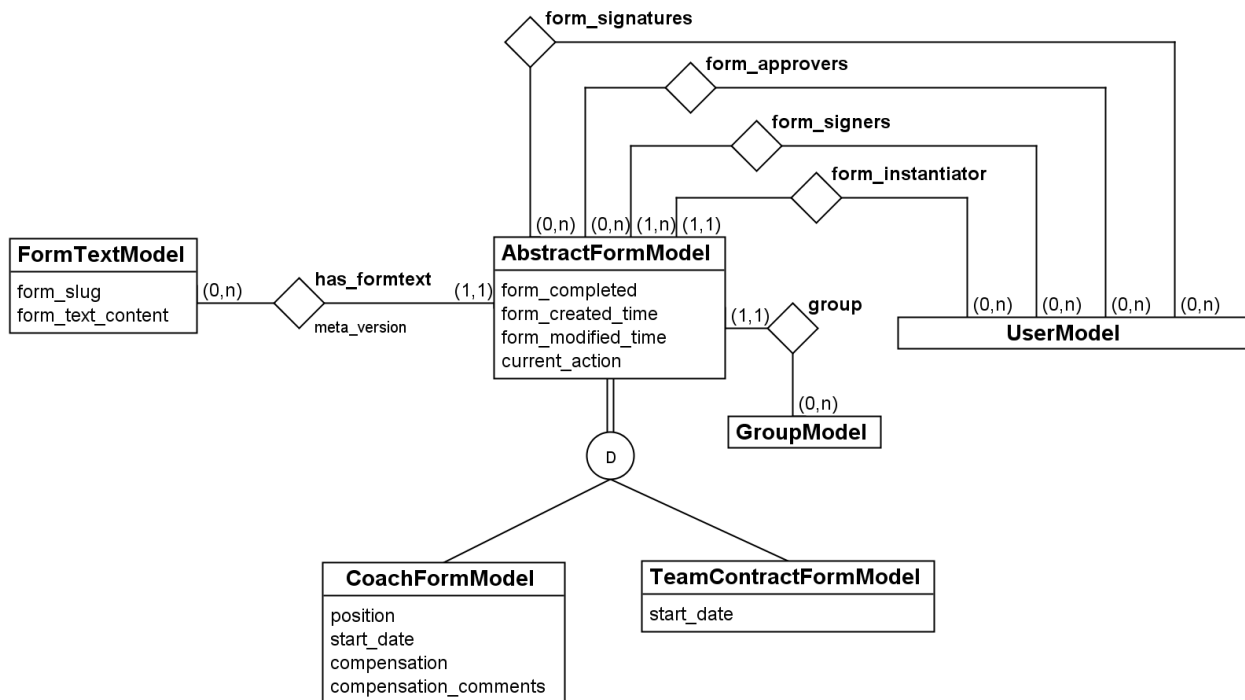


Figure 4: ER-diagram of the FMS. Note that `UserModel` and `GroupModel` entities are part of the MMS.

8.5 The 4+1 Architectural View Model

8.5.1 Architectural Views

The 4+1 architectural view model consists of a logical view, process view, development view and physical view, as well as use cases. The advantages in using this model is that it presents the architecture in different ways to different stakeholders. The *logical view* is illustrated in Figure 5 and is an abstraction of the most important functionality provided to the end users. The *process view* is illustrated with a flow diagram and sequence diagram, presented in Figure 6 and 7 respectively. This view describes the workflow and processes of the system, the communication among them, and how specific tasks are performed. The flow diagram illustrates the flow of the FMS after a form signer has received a notification in-page or by email. Note that this diagram encapsulates some future work, such as different signature levels (see Section 13.3). The sequence diagram visualizes a successful attempt to fill out and sign a form after clicking a form link. The *development view*, presented in Figure 8, is a more detailed version of Figure 3. This view shows the software module organization and illustrates the FMS from a developer's perspective. The *physical view* is shown in Figure 9. This view is provided by the customer as deployment of the system has not been a part of the project scope.

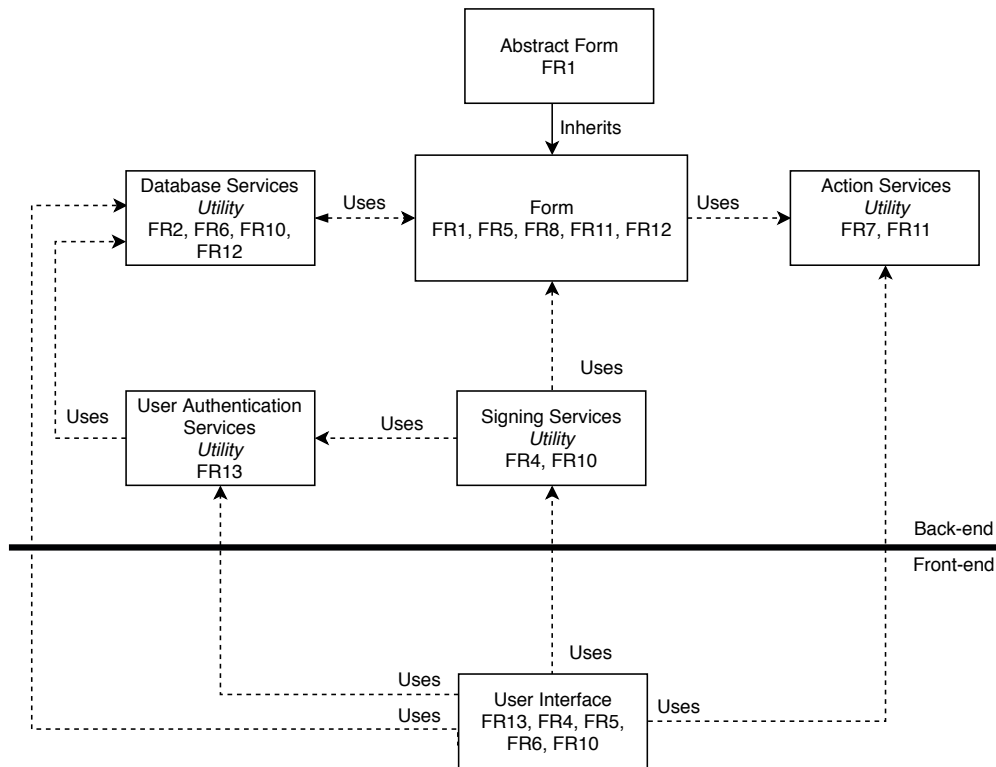


Figure 5: Logical view of the FMS components. The figure shows how the system components support the functional requirements. E.g. FR13, from Table 6, is fulfilled by the User Interface and the User Authentication Services. FR3 and FR9 were taken out of scope, see Section 3.4.

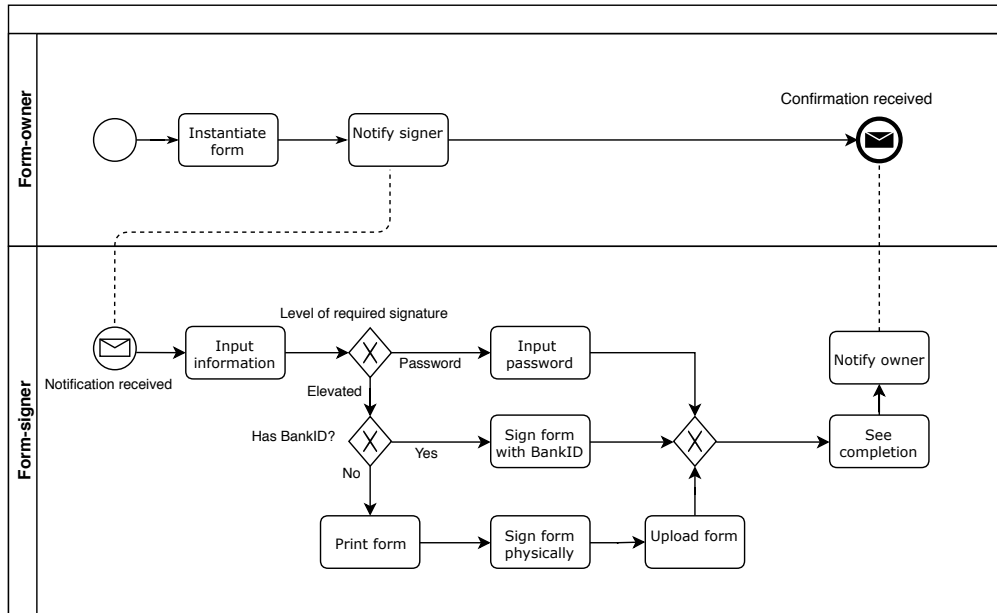


Figure 6: Flow diagram visualizing the flow of the FMS after a user has received a signing notification.

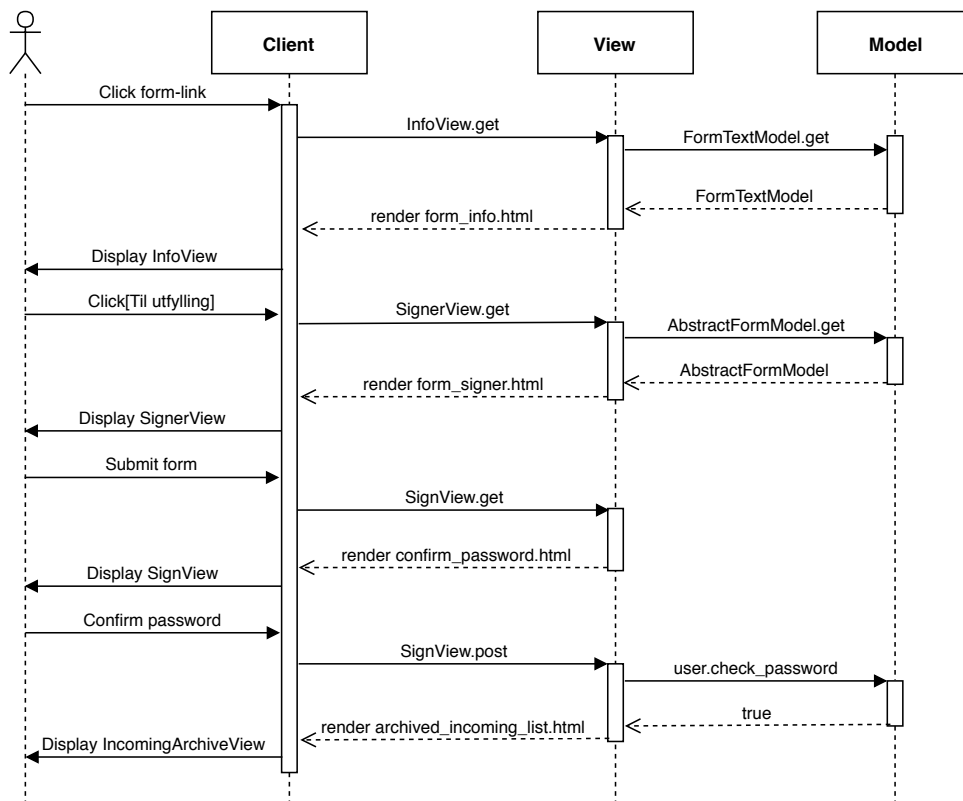


Figure 7: Sequence diagram visualizing a successful attempt at filling out and signing a form.

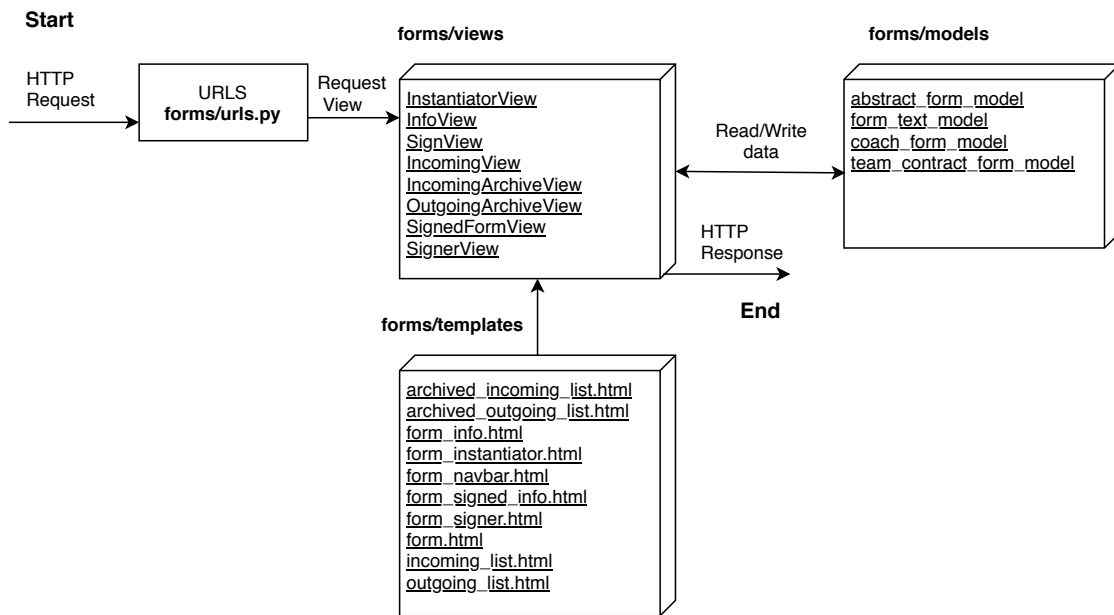


Figure 8: Development view of the forms application: A more detailed view of Figure 3.

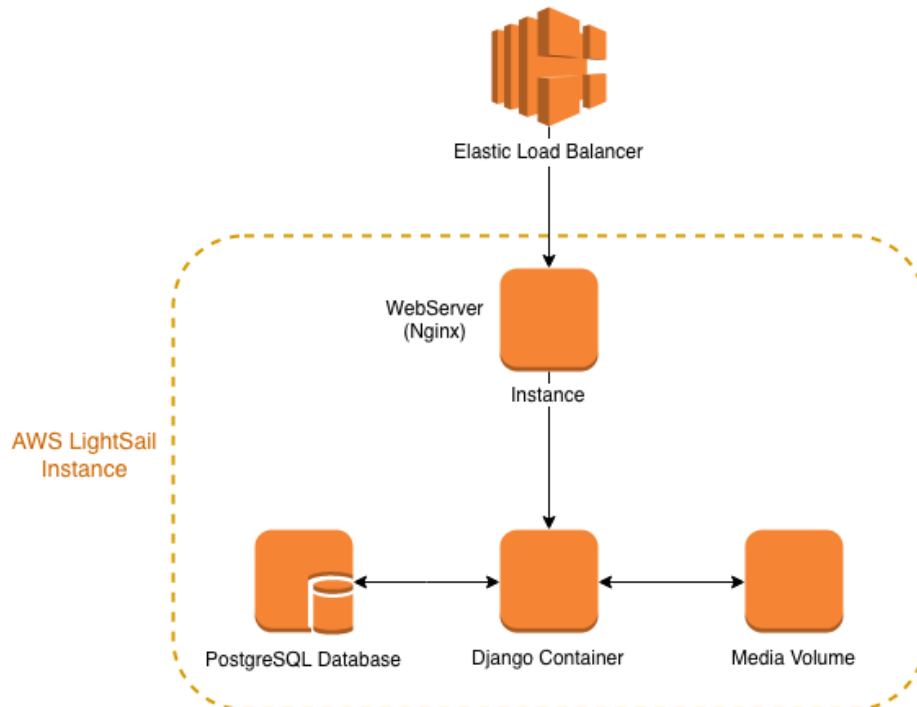


Figure 9: Physical View of the system provided by the customer

8.5.2 Use Cases

Use cases are important to get an overview of the normal progression in the most important parts of a system and to illustrate and validate architectural decisions [26]. The following use cases, presented in Table 16, 17, 18 and 19, are created to illustrate how the FMS works for different stakeholders. The use cases are related to the functional requirements described in Table 6.

Use case ID	U1
Priority	High
Actors	User
Use case summary	The user will sign a form using password
Pre-condition	The user is logged in to the system, and is a member a group. The user is on the "Skjema" page
Normal Course of Events	Alternate Path
1. The user will view available forms for signing on "Mottatte skjema" (received form) page	2a. If there are no forms available this will be display a "no forms available" message
2. The user clicks on a form link and can view the content of the form	
3. The user clicks "Til utfylling" (to fill out) button	
4. The user can fill out content that is specified by that form	
5. The user clicks "Til Signering" (to signing) button	
6. The user writes its password to sign the document	4a. If the user enters wrong password then "wrong password" message will be shown.
Post Conditions	The user will successfully have signed a form

Table 16: Use case for where a member of a group is signing a form. This use case is linked to the functional requirement FR4 and FR10, which is an important part of our system.

Use case ID	U2
Priority	High
Actors	User that is Leader/President of a group
Use case summary	The user will instantiate and send out a form to form-signer
Pre-condition	The user is logged in to the system, and is a member a group. The user is on the "Skjema" page.
Normal Course of Events	Alternate Path
1. The user clicks on "Send skjema" (send form) page to instantiate a new form.	
2. The group the user is president/leader is chosen automatically	2a. If the user is president/leader in multiple groups. The user can choose which group to send the forms from.
3. The user selects a form-signer from the group he have selected	
4. The user selects the form the user wants the form-signer to sign	
5. The user clicks send	
Post Conditions	The user will successfully have instantiated and sent a form to a form-signer

Table 17: Use case for where a president/leader of a group is instantiate a form. This use case is linked to functional requirement FR4 and FR5.

Use case ID	U3
Priority	Medium
Actors	User that is a member of a group
Use case summary	The user will retrieve a form that has previously been signed.
Pre-condition	The user is logged in to the system, and is a member a group. The user is on the "Skjema" page.
Normal Course of Events	Alternate Path
1. The user clicks on "fullførte skjema" (finished forms) page to look forms they have signed.	
2. The user can view old forms the user have signed	2a. If the user never has signed a form. This will be empty
3. The user click on the signed form	
4. The user can view the form the user have signed	
Post Conditions	The user will successfully have looked at a previously signed form

Table 18: Use case for where a member of a group is finding old signed forms. This use case is partly linked to functional requirement FR6 and FR14.

Use case ID	U4
Priority	High
Actors	Developer
Use case summary	The developer will create a new form to be stored in the database, ready for instantiation.
Pre-condition	The developer has access to the source code and all needed technology.
Normal Course of Events	Alternate Path
1. The developer creates static form information as HTML and stores it in a <code>FormTextModel</code> in the <code>database/fixtures</code> directory.	
2. The developer creates a <code>model</code> for the form, inheriting from <code>AbstractFormModel</code> . The developer also specifies form name, sign type, form slug and form specific user input-fields.	
3. The developer specifies a list of <code>actions</code> that should be undertaken after a form is instantiated	3a. The developer creates a new <code>action</code> if the form requires an action that has not yet been implemented, and adds this to the <code>actions</code> -list.
4. The developer creates a form inherited from <code>ModelForm</code> and defines the <code>model</code> that is to be used, and specifies the fields that are to be presented to the form signer.	
5. The developer registers the form model created in <code>form_types.py</code>	
Post Conditions	The developer has successfully created a new form ready to be instantiated

Table 19: Use case for where a developer creates a new form to be stored in the database ready for instantiation. This use case is linked to functional requirement FR1.

9 Security

The FMS that the team developed used Django Open Source Libraries and was integrated with the existing databases of the customer. The only requirement by the customer regarding security is that the FMS should be developed in Django, as this framework caters various security issues by itself.

9.1 Security Features in Django

Some security features offered by Django for Web applications are:

Cross Site Scripting (XSS)

The template system of Django offers protection from XSS attacks in which an attacker can inject client side scripts through the website into the browser of other users [27].

Cross Site Request Forgery Protection (CSRF)

To prevent CSRF attacks, Django generates a browser/user specific key and rejects a POST request if the form does not contain the key, thus blocking CSRF attacks in which a malicious user can execute actions using credentials of other users without their knowledge [27].

Enforcing SSL/HTTPS

HTTPS protection enabled in Django encrypts all the traffic between client and server which also includes authentication credentials like Passwords and Username [27]. HTTPS ensures protection of the privacy and integrity of the transmitted data, and prevents man-in-the-middle attacks where a malicious user can relay or alter communication between a client and the server.

Using HTTPS in Django will also offer security measures that include:

- SECURE_PROXY_SSL_HEADER
- SECURE_SSL_REDIRECT
- HTTP Strict Transport Security (HSTS)
- Secure cookies
- Host header validation

Denial-of-Service attacks

For protection against DoS attacks, uploads can be restricted in the web server configuration to a reasonable size. However, the FMS does not deal with file uploads, so this can be done at the server end where forms will be stored.

9.2 Abuse Cases

Table 20 depicts the abuse cases for our project. The team has found several abuse cases by going through the functional requirements, as well as looking at the Django framework [12]. The team used STRIDE [28] to identify the threat associated with each abuse case. The team found countermeasures to each abuse case to reduce or eliminate the security risks.

9.3 Static Security Analysis

Bandit[15] was used as a static security analysis tool. The FMS had a low severity security issue. This was due to a try-statement with a passing exception. The only thing this ensures is that the program continues to run on the error. The results from Bandit are depicted verbatim in Figure 10.

Abuse case	Counter measures	Threat
An user attempts to log in as another user	Django's built in authentication handles password and username. Could implement two-factor authentication for stronger security.	Spoofing
A hacker looks at the intermediate nodes in the network to get user credentials	Out of scope. Customer responsibility due to being at the server architecture level. At the server architecture level, SSL-encryption should be used to encrypt the packages sent between the client and the server.	Spoofing
A hacker uses Cross Site Request Forgery to force a state change in the system	Django has built in CSRF protection enabled by default. The system code also uses the django recommended tags in templates for doing POST-requests.	Tampering
An user tries to access another users form, by going directly to the url	The system verifies that the given user has access to the form before sending the form information.	Information disclosure
A malicious user tries to prevent the access to the system by requesting a lot of data	Out of scope. Customer responsibility due to being at the server architecture level. The system should use load-balancing and monitor unusual requests, as well as blacklist IP-addresses with an unusual amount of requests.	Denial of service
An user with administrative privileges changes the content or input of a form after it has been signed	Not safeguarded against. There is no way to prevent this abuse at the moment. A solution would be to hash the input, the form content and the time to verify at signing. This could be used to verify that a form has not been changed at a later time.	Tampering

Table 20: Abuse cases for FMS

```

Code scanned:
Total lines of code: 728
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
Undefined: 0.0
Low: 1.0
Medium: 0.0
High: 0.0
Total issues (by confidence):
Undefined: 0.0
Low: 0.0
Medium: 0.0
High: 1.0
Files skipped (0):

```

Figure 10: Static security analysis using Bandit

10 Testing

The following section provides an overview of the testing of the FMS. Each section will go through the specific testing method and the results from each method.

10.1 Unit and Integration Testing

The customer did not weigh the importance of unit tests and instead wanted integration testing. The team did, however, write several unit tests to verify and confirm our own understanding of the code as well as to ensure quality of the code. This also acts as documentation, as future developers working on the same code can use the unit and integration tests to understand the functionality and properties of the classes and methods involved. It will also ensure that future work does not overwrite intended functionality accidentally as the tests would fail. Not all lines of code were tested, as parts of the code consists of simple database-models or had limited amount of functionality.

Figure 11 shows the verbatim output when running the django-nose test-suite with the `--code-coverage` flag. The `django-models` have not been tested due to the lack of any specific functionality.

10.2 Automated Functional Testing

A big part of the automated testing comes from the functional tests. These are done using the browser simulator Selenium (section 6.4). Selenium allows us to simulate the flow of the user in a browser. The functional tests are written using the DOM-elements on the web page, and running events on these, such as clicking a button, or writing in an input field. By clicking through the application simulating a user, the test shows both whether or not the application works and gives predictable results, but it also verifies whether or not changes in code change the functionality and usability of the application. An example would be a change in routing, a part that is not often tested, but one that can easily go unnoticed after a single typo. Browser testing is an insurance against these types of errors going into production. The functional tests encompass the entire user-flow: from log in, to creating a form, and signing it.

The functional tests follow the flow of use cases U1 (Table 16) and U2 (Table 17). They continuously check that each step of the flow is as predicted. See Table 21 and 22 for overview of tests.

ID	001
Description	Create form
Task	1. Log in 2. Go to forms 3. Go to create form 4. Input group, signer and formtype 5. Create form
Result	Form is created and added to tab for sent forms

Table 21: Automated functional test 001

By comparing Figure 11 to Figure 12 , the code coverage increases from 38% to 73%. This is expected, as the functional test cover the entire flow of the user, and more parts of the system are executed.

Name	Stmts	Miss	Cover
forms/__init__.py	0	0	100%
forms/actions/__init__.py	1	0	100%
forms/actions/action_abstract_class.py	6	0	100%
forms/actions/actions.py	16	5	69%
forms/actions/notify.py	27	4	85%
forms/admin.py	7	7	0%
forms/form_types.py	4	0	100%
forms/forms.py	22	11	50%
forms/migrations/0001_initial.py	7	0	100%
forms/migrations/__init__.py	0	0	100%
forms/models/__init__.py	3	3	0%
forms/models/abstract_form_model.py	26	25	4%
forms/models/coach_form_model.py	19	19	0%
forms/models/enums.py	2	2	0%
forms/models/form_text_model.py	9	9	0%
forms/utils/form_utils.py	22	18	18%
forms/views/__init__.py	7	0	100%
forms/views/archived_list_view.py	16	8	50%
forms/views/info_view.py	17	11	35%
forms/views/instantiator_view.py	62	46	26%
forms/views/list_view.py	16	8	50%
forms/views/sign.py	36	26	28%
forms/views/signed_form_view.py	16	8	50%
forms/views/signer_view.py	26	17	35%
TOTAL	367	227	38%

Ran 7 tests in 0.058s
OK

Figure 11: Code Coverage of Unit and Integration Tests. Stmts is short for statements. Miss is amount of lines not covered. Cover is the percentage of code covered by the tests

10.3 Usability Testing

The views referred to in this section can be found in Section 11 and Appendix C.1. The leader perspective contains the views in figures 18, 19, 13, 20 and 21. The regular user perspective contains the views in figures 15 and 17. Both perspectives share form signing views in figures 14a, 14b and 14c.

10.3.1 Test Preparation

One of the non-functional requirements specified by the customer was that the product should be user tested. The team has arranged the test to be conducted on 01.11.2018 in IGB-Ø2 in Berg, Gløshaugen. On 16.10.2018 the customer agreed to provide 6 to 10 test subjects. The purpose of the test was twofold. For the sake of this project, the test was mainly summative - to gauge the usability of the product as a finished system. This was done with respect to form management aspects that are common for all types of forms, which will be relevant for all NTNUI members once the system is deployed. The formative aspects of the test will be used by the customer to further develop the system. These aspects support some parts of the future work presented in section 13. Some of the smaller adjustments were implemented by the team.

ID	002
Description	Sign form
Task	<ol style="list-style-type: none"> 1. Log in 2. Go to forms 3. Go to received forms 4. Input form data 5. Sign form
Result	Form is signed, and is updated accordingly

Table 22: Automated functional test 002

Considering we were tasked with developing a back-end for the forms system with minimal front-end as per section 1.1, the main purpose was to test the flow of the form instantiation and signing process. Nevertheless, the front-end was tested to lay the groundwork for future work regarding the user interface.

The test was carried out by Siren, Kristian and Jan. A document describing the details of the test was prepared in advance of the actual test, parts of which were either incorporated into this report or attached in appendix B. In advance of the test, a preliminary pilot test was conducted to prepare the team and discover potential errors with the layout.

10.3.2 Test Subjects

The team agreed with the customer on 16.10.2018 that they will provide 6 to 10 test subjects. This resulted in the customer providing us with a schedule containing six participants including their names, time of arrival, group membership and group roles. A selection of this information is presented in Table 23. The six given participants allowed the team to perform a proper qualitative analysis with regard to the scope of the system. According to Jakob Nielsen’s research regarding quantity of test subjects [29, page 389], the test should have discovered around 80% of usability problems assuming high test validity.

Test subject	TS1	TS2	TS3	TS4	TS5	TS6
Time	12:15	12:30	12:45	13:45	14:15	14:30
Role	Cashier	Vice president	Webmaster	President	Webmaster	President

Table 23: Test schedule as specified by the customer

The subjects were a random selection of NTNUI members. This meant that the subjects are 18-25 years old students and have partial higher education. Because of that we expected the subjects to have basic technological understanding. Having NTNUI members as test subjects meant that we tested the system with potential future users of the product, which contributes to the test validity. Considering that the release date of the full system is unspecified, it is not possible to ascertain whether the test subjects will be among the actual future users. Some of the system functionality is reserved for group leaders which represent a minuscule subset of NTNUI members. The reserved functionality was tested with basic members anyway to gauge how intuitive the user interface is. All of the participants had roles like *leader*, *cashier* or *webmaster* in their respective sport groups under NTNUI, so there was a bias for people who are significantly engaged in various organizations. This meant that the system was not tested with less engaged people, like the ones who are members of NTNUI just for the sake of exercising. The user experience of less engaged members would probably differ, but this could not be tested given the kind of test subjects provided.

Name	Stmts	Miss	Cover
forms/__init__.py	0	0	100%
forms/actions/__init__.py	1	0	100%
forms/actions/action_abstract_class.py	6	0	100%
forms/actions/actions.py	16	0	100%
forms/actions/notify.py	27	1	96%
forms/admin.py	7	7	0%
forms/form_types.py	4	0	100%
forms/forms.py	22	0	100%
forms/migrations/0001_initial.py	7	0	100%
forms/migrations/__init__.py	0	0	100%
forms/models/__init__.py	3	3	0%
forms/models/abstract_form_model.py	26	24	8%
forms/models/coach_form_model.py	19	19	0%
forms/models/enums.py	2	2	0%
forms/models/form_text_model.py	9	9	0%
forms/urls.py	6	0	100%
forms/utils/form_utils.py	22	3	86%
forms/views/__init__.py	7	0	100%
forms/views/archived_list_view.py	16	4	75%
forms/views/info_view.py	17	1	94%
forms/views/instantiator_view.py	62	13	79%
forms/views/list_view.py	16	0	100%
forms/views/sign.py	36	6	83%
forms/views/signed_form_view.py	16	8	50%
forms/views/signer_view.py	26	2	92%
TOTAL	373	102	73%

Ran 12 tests in 36.713s
OK

Figure 12: Code Coverage of functional testing. *Stmts* is short for statements. *Miss* is amount of lines not covered. *Cover* is the percentage of code covered by the tests

10.3.3 Test Tasks

The tasks presented to the participants are shown in the appendix B.1. They were translated to Norwegian before the test, because all of the participants were Norwegian. The first three use cases presented in Section 8.5.2 were used as a starting point for the test formation. Furthermore, the tasks were designed in a way that encompasses every aspect of the user interface - every implemented view was tested. The tasks were divided based on the two types of users: regular users and leaders. This necessitated some intervention during the test to adjust the environment.

10.3.4 Test Proceedings

The team decided in advance the test-related duties of every present team member. Siren was responsible for taking notes and measuring the task time. Kristian was responsible for presenting the test and the tasks to the subjects. Jan was also responsible for taking notes, as well as for the maintenance of the test environment. Every team member was provided with an overview of what the participants should know before the start

of the test, team’s conduct under the test, and follow-up questions. The overview was written in Norwegian, but the basis for it was written in English and can be found in the appendix B.2. The team also had seven copies (one extra) of consent forms that the participants signed to let us use their data for the sake of the test. Every team member had seven copies of the overview of the tasks with extra space for taking notes during the test.

The customer gave minimal information to the participants when scheduling the test, so some of them were somewhat confused after coming into the test room. Once in the test room, the participants were told about the purpose of the test and other formalities which clarified any confusion. The information presented was based on the overview B.2 as described above. After that the participants were presented with and completed the tasks one by one. When all the tasks were finished, the team discussed the product with the participants.

Due to minor recurring task slowdowns, the team added small clarifications to the user interface after the first three participants were finished. They consisted of adding navigation links to list help text and changing button text in views related to form signing (figures 14a and 14b).

10.3.5 Test Analysis

After the test was finished, the team collated the notes for analysis. The product proved its function as a form management system based on the opinions of the participants. The consensus was that this kind of system would improve the bureaucratic processes of NTNUI. The validity of this statement is mediocre due to having only six participants as provided by the customer.

The test showed that the regular user perspective was very intuitive. This is important because regular users do not handle forms often, so a convoluted system would require constant relearning. The leader perspective was more complicated and not as intuitive at first. This is compensated by the fact that leaders will spend more time managing forms than regular users, so they will have to take a more structured approach when exploring the views. This is supported well by the help text present in every view. Table 24 presents observations regarding possible shortcomings and suggests how the system can be improved in the future. For the sake of the conciseness of the report, an in-depth analysis of the test is presented in appendix B.3 that might be valuable to the customer.

Observation	Possible solution
Participant navigates to the group view instead of the forms view when on the main page.	Add a button navigating to forms view in group view.
Participant is confused about the contents of the current form list view.	Improve the help texts in the form list views.
Participant wants to find a form more easily.	Add sorting and querying functionality to the form list views. A suggested solution can be seen in Figure 27.
Participant expects finished forms to be expired.	Add expiration date to the abstract form model and create a view that shows expired forms.
Participant doesn’t know where the form they are trying to find is.	Show signed and non-signed forms in the same view with non-signed forms at the top prior to sorting (if sorting is implemented).
Participant doesn’t see the status of a form.	Make the status more visible by using a separate column and colors, icons, or decorated text.
Participant has trouble with the date picker.	Find an alternative/addition to UIKit that supports a date picker.

Table 24: The shortcomings observed during the test and their possible solutions.

11 Final Product

The final product that has been developed works as required by the functional requirements. It is a minimal viable product and as such fills the requirement set by the course TDT4290. The product allows for user signing, instantiating of forms, notification through email, and a view of forms. The code-base of the product is fully integrated with the code-base of the existing member management system. It follows a similar design to the MMS. The developed system eliminates the use of paper forms in most cases. It enables the signing of forms to happen anywhere and anytime, and the need for a physical exchange of forms is greatly reduced.

11.1 User Guide

To access the FMS, the user has to click on the 'Skjema' menu item. If the user has the role 'President/Leader' in at least one sports group, the user will be presented with four different tabs:

- *Send Skjema*: Instantiate and send out new forms.
- *Utsendte Skjema*: View status of the delivered forms that have not yet been completed.
- *Mottatte Skjema*: View received forms that are to be signed by the user.
- *Fullførte skjema*: Archive of completed forms, both *Mottatte Skjema* and *Utsendte Skjema*.

A regular user has access to the following tabs:

- *Mottatte Skjema*: View received forms that are to be signed by the user.
- *Signerte Skjema*: View forms that have been signed and completed.

The 'Send Skjema' view is shown in Figure 13. A user with the role 'President/Leader' can configure a new form and send it to a user that has to fill out and sign it.

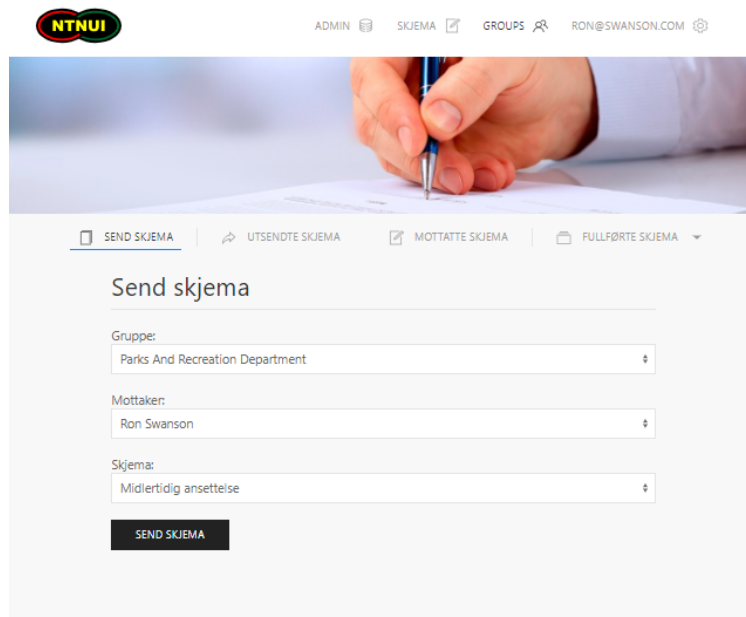
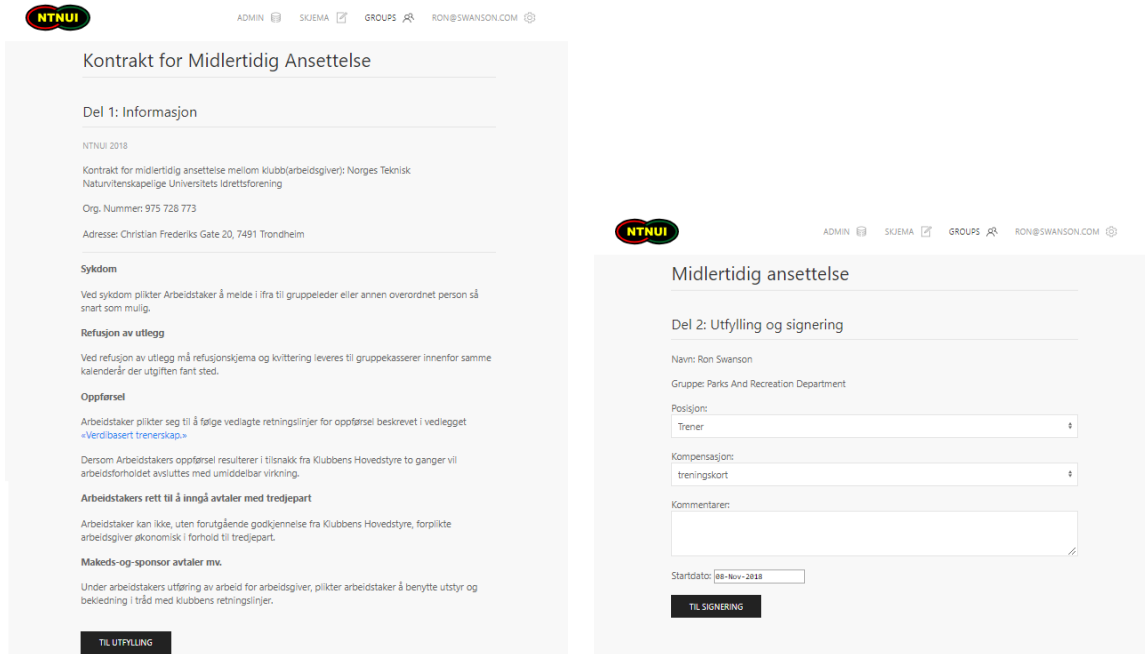


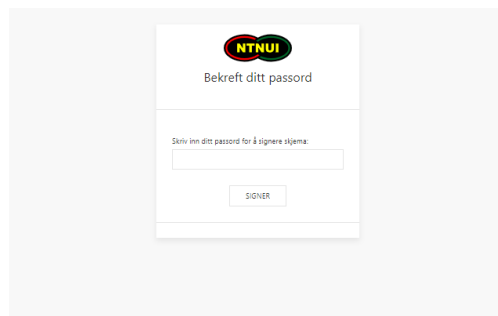
Figure 13: Leader perspective view that allows the user to send out a form to another user.

After the form is sent, the form-signer receives an email that includes information about the form and a direct link to that particular form. If the form-signer accesses the link, he/she is presented with static information about the form shown in Figure 14a. This information page can also be accessed through the form-signer's 'Mottatte Skjema' view, Figure 15. By clicking the button 'Til Signering' at the bottom of the page, the user is taken to the view shown in Figure 14b. Here, the user can fill out the information required by the form, and the information will be stored by the FMS for later retrieval. After the user clicks 'Til Signering', the user has the possibility to sign the form. The 'sign with password' view is shown in Figure 14c.



(a) The view containing form info shown to the user before signing.

(b) The view containing form input fields shown to the user before signing.



(c) The final view in the signing process requesting the user to confirm their password to finish signing the form.

Figure 14: View of form text(a), user information input(b) and password verification(c).

After the signature has been submitted correctly, the user is redirected to 'Signerte Skjema', and the 'President/Leader' that initiated the form receives an email notifying him/her that the form has been signed and completed.

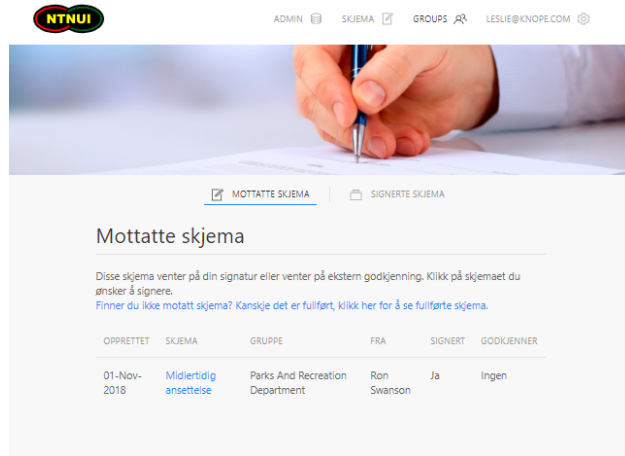


Figure 15: The default view of the regular user perspective. Contains received forms that are awaiting a signature.

Additional screenshots of the final product can be found in Appendix C.1.

11.2 Installation Guide

The system is easily built and run using Docker on Linux, MacOS, Windows Pro and Windows Enterprise. To build and run the system, follow these steps:

1. Make sure Docker is installed and working. See <https://www.docker.com/products/docker-desktop> for details.
2. download and extract the source code from <https://github.com/kapteinstein/tdt4290>
3. in the source code folder run:

```
make build && make dev_clean_install
```

Docker will take care of the dependencies and environment. Access the site at <http://localhost:8000> and login using one of the mock-users: username `mock@user.com` and password `locoloco`.

12 Group Reflection and Evaluation

12.1 Description of Group Dynamics

The first lecture of the course TDT4290 “Customer Driven Project” was about “Group Dynamics” conducted by Christine Holm Berntzen and Marianne Ingeborg Karlsen from Sit Råd. As defined in the lecture, the “Group Dynamics” is all about how a group functions, how team members can cooperate and communicate with each other, and how conflicts can be resolved within the group.

The lecturers defined four stages of group dynamics which include Forming, Storming, Norming and Performing [30]. The lecture also included an activity in which the goal was to identify which type of personality we had as individuals and how different type of personalities can create a performing group. The activity also enables us to appreciate the differences in personalities we have and to use these differences for better performance of the group.

12.2 Reflection on Group Dynamics

Our team consists of seven people with varied background and experiences. The most important factor for working as a team is that we have to develop a strong communication among all team members. For this purpose, the team shared their contact information and used all channels of communication for prompt response from each other.

The next step was to develop a mutual understanding of the project and to use individual expertise of group members in their respective areas. During the planning phase of the project, all team members provided their input for allocation of task and devising schedule for completion of the assigned task. For the implementation and documentation phase the team marked some obligatory hours. Along with this, meetings were organized twice a week so that the team members could communicate with each other and convey their progress and problems, if any. In case of late arrival of a group member or if the member is absent, the other group members carry on their tasks and communicate the progress to the members through email or Google Docs where the status of the project is updated on a regular basis. In addition to the organized group meeting, the team members have also spent individual working hours on their assign tasks and shared the work online so that other team members could comment or review the work.

The team did not face any major conflicts. In case of any issues, the issue was solved through open discussion. The communication among group members was improved throughout the project. In general, the team has worked in a cooperative and cohesive way and the overall learning experience of team members have been positive.

12.3 Customer Relations

Our customer, NTNUI, is an important stakeholder of the project. The representative of NTNUI have played a significant role in the successful completion of the projects. Initially the requirements defined by the customer were not very discrete and scope of the project was not clearly defined. After regular meetings with the customer the scope of the project was redefined. Requirements were elucidated taking into account the available resources and the system development at the customer end.

In terms of communication, the customer provided channels of communication that includes Slack and Asana. The project team used these channels to communicate with the customer whenever it was required. During the development phase the customer did not only go through the developed code, but also provided valuable feedback and tips for progressive development of the system. The customer remained supportive throughout the project and shared a great deal of knowledge to the team members.

12.4 What Future Students Should Know

The most important thing that future students of TDT4290 should know is that this is not only a course, but a complete learning journey that can help them to gain practical knowledge of coping with a large project and customers in real life. However, one difficulty that certain students may experience in this course is the differences in the expertise and knowledge level among students from different academic and geographical backgrounds. This is also a major aspect that can help them enhance their knowledge significantly and acquire new skills.

Students should not refrain themselves from asking questions to other group members as this course intends to impart practical knowledge between students. Students should not be afraid of telling what they do not know or understand.

Another aspect of the course that students should be aware of is that sometimes they may be limited to use specific technologies defined by the customer requirements. However, this restriction can help students learn new technologies in more practical ways using the time available for the project.

The lectures conducted during the course should be attended so that all students can achieve a certain level of understanding of relevant topics, especially software project management. Lectures given during the course helped us understand the project management aspect of the project in a more concise way.

Our assigned supervisor for the project was of great help throughout the project. Therefore, every team member should attend meetings with their supervisor and convey their problems and concerns openly.

12.5 Feedback on the Course

The course “Customer Driven Project”, TDT4290 is a practical course that require deep involvement of students in terms of time and effort. The course offers a thorough learning experience in which all team members have learned new things while performing the assigned tasks. The course not only imparts theoretical knowledge of project management in form of lectures, but also implied practical knowledge of software development methodologies and techniques regarding project development.

The learning process and practical experience are the most important aspects of the course. Along with this, dealing with a customer in a real-life scenario is also a significant aspect of software development projects. The project gives insight to students regarding how different customers can come up with their diversified requirements and what can be done to handle these requirements with limited time and resources. The experience also enables students to understand how important the involvement of all stakeholders in the project is, and how vital customers engagement in the project is for a successful completion of the project.

13 Future Work

The developed FMS lays the groundwork for easy additions of new forms. The FMS has a general work flow for instantiating, viewing and signing various forms. All specific forms inherit from a general `AbstractFormModel`, see ER-diagram in Figure 4. By using the `AbstractFormModel`, the other forms NTNUI uses that are not implemented can easily be created and added to the system. See `forms/models/coach_form_model.py` in the source code for an example.

13.1 Modification of Current System

The current MMS lacks a clean, systematic and versatile access control system. Every user should not be able to see and instantiate every form. One possible way to distinguish the access rights to the users in the system is to implement a system based on `access levels` where each form has an associated required access level. This access in the system would then be a hierarchy where all users with the required access level (or higher) would be able to instantiate a particular form. The users who are able to see an instantiated form are the *instantiator*, *signers* and *approvers*. One possible hierarchy of access levels could be:

```
8: admin
7: HS
6: Group Leader
5: Cashier
4: Board
3: Trainer/Coach
2: Group Member
1: Member
0: Non-member
```

However, for an organization with the size like NTNUI, a more flexible and better way to implement access control is via a role-based access control system [31]. Every user is a part of one or several different *access groups*, and every resource in the system or on the site, are available for one or several of these *access groups*. A role-based access control system has become the predominant model for advanced access control for managing complex security administration in large networks [32].

We suggest that this is taken into consideration, and we urge the customer to make the implementation of an access control system a priority to make further development easier.

13.2 Improvements of the FMS

The signatures related to a specific form are saved to the form object as a `foreign key` referencing the correct `UserModel`. A better approach that the team purpose is that the signatures are saved in a separate model with additional meta-data related to the signature. This includes:

- A `foreign key` to the signers `UserModel`.
- A hash string of the object signed, which makes the signature invalid if the content has changed after the signature was made.
- A time stamp of the date and time when the signature was created
- A field that identify the type of the signature (*high* or *low* level)

A separate model for signatures would make the system more robust with strengthened security, mainly because signatures are not only connected to a form, but the actual content of the form through a hash.

13.3 Suggestions for Future Implementation

Signature Level

The FMS facilitate functionality to support two levels of signature: *high* and *low* level. Low-level signature is implemented as password authentication. The functionality for high-level signatures with either BankID or manual paper signature is not implemented, but the system prohibits password authentication and inform the user that a higher form of authentication is required if the form is configured to require a high-level signature.

External Approval/Auditing

Functionality for external approval is implemented in the system, but there is currently no user interface for it. This should be relative easy to implement by combining `info_view` and `sign`.

Forms Validity

The date and time for when a form is instantiated and signed is stored in the database. However, as mentioned in Section 10.3.5, it was discovered that a period-of-validity functionality would be desirable for some forms. This would require adding a `DateTimeField` to the `abstract_form_model` if all the forms should contain this functionality, or to the models of the specific forms that should require it. Logic to evaluate whether or not a form is valid must also be implemented.

Minor Tweaks

During meetings with the customer, the team became aware that the different forms within NTNUI are strongly related to the different sport groups. It may be desirable to relocate the forms tab to the groups page.

As mentioned in Section 10.3.5, it was suggested that an extensive form sorting functionality should be implemented.

13.4 Bugs

UIKit-limitations

Due to UIKit limitations, a form instantiator is only presented with the possibility to send a form to one form signer at a time. It is, however, possible to create a form with multiple form signers if UIKit is disabled for the form-signer selection element.

Multiple Form Signers

It should be noted that if a form with multiple form signers is created, the form signers' input currently manipulates the same form elements. For example, if there are two form signers in a specific `coach_form`, and these signers chooses specific a `compensation`, only the input of the last signer will be stored in the model. To be clear, this bug is only a problem for forms with multiple form signer requiring user input.

Date Input

A user can specify a date older than today in a `DateField` and `DateTimeField`.

Date-picker

Date-picker does work in Chrome and Firefox, but does not work in Safari.

References

- [1] Project Management Institute. *A guide to the project management body of knowledge*. 5th edition, 2013.
- [2] Grude K.V. Haug T. Katagiri M. Andersen, E.S. and J.R. Turner. *Goal directed project management*. 2004.
- [3] About ntnui. <https://ntnui.no/front-page-english/info-about-ntnui/>.
- [4] NTNUI. Become a member - ntnui. <https://ntnui.no/joinus/>, 2018. [Online; accessed 07-Nov-2018].
- [5] Writing your first django app, part 1. <https://docs.djangoproject.com/en/2.1/intro/tutorial01/>.
- [6] Andrew Powell-Morse. *Waterfall model: What is it and when should you use it?* 2016.
- [7] Jon Atle Gulla Letizia Jaccheri, Anniken Holst. *Compendium tdt4240 customer driven project*. 2018.
- [8] Don Wells. *The values of extreme programming*, 2009.
- [9] Sahil Patel. *Continuous integration: How to avoid integration hell*. <https://dzone.com/articles/continuous-integration-how-0>, 9 2014.
- [10] Python Software Foundation. *Python*. <https://python.com>, 2018. [Online; accessed 02-Nov-2018].
- [11] Django. *Django 2.1 release notes*. <https://docs.djangoproject.com/en/2.1/releases/2.1/>, 2018. [Online; accessed 02-Nov-2018].
- [12] Django. *Django*. <https://www.djangoproject.com>, 2018. [Online; accessed 02-Nov-2018].
- [13] Docker Inc. *Why docker? - docker*. <https://www.docker.com/why-docker>, 2018. [Online; accessed 01-Nov-2018].
- [14] GMBH TRAVIS CI. *Travis-ci*. <https://travis-ci.com>, 2018. [Online; accessed 02-Nov-2018].
- [15] PyCQA. *Bandit*. <https://github.com/PyCQA/bandit>, 2018. [Online; accessed 06-Nov-2018].
- [16] Git. <https://git-scm.com>, 2018. [Online; accessed 01-Nov-2018].
- [17] GitHub Inc. *Github*. <https://github.com>, 2018. [Online; accessed 01-Nov-2018].
- [18] Slack Technologies. *Slack*. <https://slack.com>, 2018. [Online; accessed 01-Nov-2018].
- [19] Google LLC. *Using google drive*. <https://www.google.com/drive/using-drive/>, 2018. [Online; accessed 01-Nov-2018].
- [20] Writelatex Limited. *About - overleaf, online latex editor*. <https://www.overleaf.com/about>, 2018. [Online; accessed 02-Nov-2018].
- [21] Draw.io. *Flowchart maker & online diagram software*. <https://www.draw.io>, 2018. [Online; accessed 13-Nov-2018].
- [22] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 3rd edition, 2003.
- [23] *Django introduction*. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>. [Online; accessed 01-Nov-2018].
- [24] *The model-view-controller design pattern*. <https://djangobook.com/model-view-controller-design-pattern/>. [Online; accessed 01-Nov-2018].
- [25] *"design philosophies"*. <https://docs.djangoproject.com/en/2.1/misc/design-philosophies/>. [Online; accessed 30-Oct-2018].

- [26] Philippe Kruchten. Architectural blueprints—the “4+1” view model of software architecture. <https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>, 1995. [Online; accessed 13-Nov-2018].
- [27] Django. Security in django — django documentation — django. <https://docs.djangoproject.com/en/2.1/topics/security/>, 2018. [Online; accessed 01-Nov-2018].
- [28] Microsoft. The stride threat model. =[https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878\(v=cs.20\)](https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)), 2009.
- [29] Jakob Nielsen. Estimating the number of subjects needed for a thinking aloud test. *International journal of human-computer studies*, 41(3):385–397, 1994.
- [30] Bruce. W Tuckman. Developmental sequence in small groups. 1965.
- [31] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
- [32] National Institute of Standards and Technology. Role based access control — csrc. <https://csrc.nist.gov/projects/role-based-access-control>, 2018. [Online; accessed 05-Nov-2018].

A Evolution in Requirements

A.1 Evolution in user stories

Final version of user stories is shown in Table 5. Table 26 shows the evolution of user stories. It is the last version that is used in the final user stories. The initial user stories were created the 6th of September 2018. Table 25 shows when changes in user stories were made.

Date	What happened
6 th of September 2018	Initial stories
18 th of September 2018	Modifying US1-v1 to US1-v2
6 th of November 2018	<ul style="list-style-type: none"> • Modifying US1-v2 to US1-v3 • Merging US7-v1 and US8 to US7-v2
6 th of November 2018	Removing US8

Table 25: The dates for the changes in user stories

ID	User stories
US1-v1	As a form-signer, I should be able to read and fill out a form in both Norwegian and English
US1-v2	As a form-signer, I should be able to read and fill out a form in Norwegian
US1-v3	As a form-signer, I should be able to fill out a form in Norwegian
US2	As a form-owner I can require that a form needs external approval to be completed
US3-v1	As a user with the appropriate privileges, I should be able to see available forms
US3-v2	As a user with the appropriate privileges, I should be able to see both signed and unsigned forms
US4	As a form-owner with form instantiation privileges I should be able to instantiate a form
US5	As a form-owner I should be able to create a new form
US6	As a form-owner, I should be able to make a form available for signing for relevant form-signers.
US7-v1	As a form-owner I should be able to see the current status of my forms
US7-v2	As a form-owner I should be able to see all my instantiated forms and their status
US8	As a form-owner I should be able to see all my instantiated forms
US9	As a form-signer I should be able to view and fill out content in a form
US10	As a form-signer I should be notified when a form is available to be signed
US11	As a form-owner I should be notified upon form completion
US12	As a form-signer I should be able to sign a form using password when required
US13	As a form-signer I should be able to sign a form using BankID when required
US14	As a form-signer my access-rights to the system should change upon the completion of specific forms.

Table 26: Evolution of user stories

A.2 Evolution in functional requirements

The fourth and final version of functional requirements is shown in table 6. The first version of the functional requirements is shown in table 27. After talks with our supervisor, the functional requirements were changed, see table 29. Table 28 show removed requirements from the second version to the third version. The third

version is shown in table 31, and table 30 shows changed made during the third version of the functional requirements.

Functional requirement (Version 1)
System
All forms are saved in the database
All instantiated forms are saved in the database
Send email requesting signing to form-signers on form instantiation
Send email to form-owner on form completion
Communicate with 3rd party BankID entity
Change access rights on form completion
Send email to form-signer about updated account on form completion
User
Display forms available for instantiation
Instantiate an already defined form
Add user or groups to sign form on instantiation
Create form - add fields and text - add validation-rules
Input to define whether a form needs external approval
Form-owner
Display instantiated forms
Display number of people that have signed each instantiated form
Display number of people that have not yet signed each instantiated form
Send mail as reminder to sign form
Form-singer
Fill out and show forms both in Norwegian and English
Show appropriate error message if fields are filled out incorrectly.
Display password field to sign
Show appropriate error message if password is wrong
Password should not be viewable at point of entry
Display bankID to sign form

Table 27: Version 1 of the functional requirements. Created 6th of September 2018

Date	What happened
18 th of September 2018	Removed from version 2 to version 3

Table 28: The dates for the changes in functional requirement version 2

A.3 Evolution in non-functional requirements

Final version of non-functional requirements is shown in Table 7. Table 33 shows the evolution of non-functional requirements. It is the last version that is used in the final non-functional requirements. The initial non-functional requirements were created the 6th of September 2018. Table 32 shows when changes in non-functional requirements were made.

ID	Functional requirements version 2
FR12	The system should contain a general “trustee statement”-form (Tillitsvalgts erklæring) in the database.
FR3	A user should be able to view all their available forms given their access rights
-	The system should give form-owners the ability to request signing of a trustee statement by selected form-signer.
FR11	Upon requesting the signing of a trustee statement, the system should automatically notify the selected users by email.
FR7	The system should automatically send an email to form-signer(s) requesting signing of a trustee statement form.
FR10	Upon receiving a sign-request, the system should present the trustee statement to form-signer along with the ability to sign the form.
FR4	The system should contain functionality to sign forms using username and password.
-	The system should contain functionality to sign forms using BankID provided by Posten.
-	The system should give users the ability to sign a “trustee statement”-form by BankID provided from Posten
-	The system should give users without BankID the ability to sign a “trustee statement”-form with user password
FR2	The system should save a signed “trustee statement”-form in the database
-	Upon signing the trustee statement, the system should automatically notify the the form-owner(s) by email
FR9	The system should automatically change form-signers access rights when a “trustee statement”-form is signed. (Access rights must be defined)
FR5	The system should allow a form-owner to specify which user receives a signing request.

Table 29: Functional requirements version 2. Created 13th of September 2018

What happened	Date
Modified requirement from version 2 to version 3	18 th of September 2018
Added requirement	18 th of September 2018
Removed requirement	9 th of October 2018
Added requirement	23 rd of October 2018

Table 30: The dates for the changes in functional Requirements version 3

B Testing

B.1 User Test Tasks

The table was the basis for the tasks presented to the participants during user testing. Only the ”Task” column was presented to them. The text was translated to Norwegian before the test.

ID	Functional requirements version 3
FR1	The system should contain a modular abstract form that can be inherited by other forms
FR2	The system should save signed forms in the database
FR3	A user should be able to view all their available forms with the required access rights
FR4	The system should contain functionality to sign forms using password.
FR5	The system should allow a form-owner to specify which user receives a signing request.
FR6	The form-owner should be able to view all the forms they have instantiated and the status of these forms
FR7	Upon signing a form, the system should notify the the form-owner(s) as defined by the form's notification policy
FR8	The forms can be customized for higher order authentication
FR9	After all form signers have signed the form, the access rights of relevant users should be changed automatically if this is specified by the form during form-creation
FR10	Upon receiving a sign-request, the system should present the form to form-signer along with the ability to sign the form.
FR11	The system should be able to notify actors on form instantiation if specified in the form's notification policy
FR12	The system should contain a general "Team contract"-form (Teamkontrakt) in the database.
FR13	Presidents/Leaders in a group should be able to instantiate forms to be signed by members in that group.

Table 31: Version 3 of functional requirements. Created 18th of September 2018

What happened	Date
Initial requirements	6 th of September 2018
Removed requirements	13 th of September 2018
Added requirements	13 th of September 2018
Modified requirements	18 th of September 2018
Added requirements	23 rd of October 2018

Table 32: The dates for the changes in non-functional requirements

ID	Non-functional requirements
NFR1	The system must be written as a Django-app. (Django 2.1)
NFR2-v1	The code should have few, but large, integration tests and pass every test.
NFR2-v2	The code should have one or two integration tests and pass every test.
NFR3-v1	The code should be thoroughly user tested.
NFR3-v2	The code should be user tested.
NFR4-v1	The code should be well documented, both in code and at GitHub wiki.
NFR4.v2	The code should be documented, both in code and at GitHub wiki.
	There should be none high-vulnerability, high-likely risks regarding the security of the system
NFR5-v1	The system should be available in both Norwegian and English
NFR5-v2	The system should be available in Norwegian
NFR6	UIKit should be the main css-library for styling.
NFR7	All forms should use a Cross-Site Request Forgery (CSRF) token.

Table 33: Evolution of non-functional requirements

Preparation (done by us)	Task (said out loud to the participant and presented beside them in written form)	Goal (what we expect the participant to achieve)
Send out a form to Leslie Knope. Log in as Leslie Knope.	B1 You are Leslie Knope and have received an email that you have a form from your group leader that you can fill out. Find more information about the form. B2 You've read the contract and you are ready to fill it out. Find out how.	Navigate to "aktive skjema" and open the form.
	Fill out information: Posisjon: Trener Kompensasjon: Treningskort Startdato: 01.11.2018	Press "Neste" and fill out necessary fields.
	B3 Now that you're happy with what you filled out, you are ready to sign. Please sign the form. You can choose any of the saved passwords.	Sign the form by pressing "Neste" and confirm password.
	B4 You are unsure whether the data you put into the form is correct. Check if the data is correct.	Find the form in archive and scroll down to input fields.
Log in as Ron Swanson. Delete the form created in the previous tasks.	L1 You are now Ron Swanson - the leader of Park and Recreations group. You are to send out a Midlertidig Ansettelse form to Leslie Knope.	Navigate to "skjema" then "send skjema" . Send out Midlertidig Ansettelse to Leslie Knope.
	L2 Some time has passed and you're wondering what's the status of the form. Please check the status.	Confirm that it says "ikke signert" next to Leslie Knope in "utsendte skjema"
Sign the form as Leslie Knope and log in again as Ron Swanson.	L3 Leslie Knope should have signed the form by now. Check if that's true.	Navigate to "arkiv utsendte skjema" and confirm that Leslie Knope has signed.
	L4 You're wondering whether the form was filled out correctly. Check that "kompensasjon" is "trener".	Open the form and scroll down to the input fields. Understand the information presented.

B.2 User Test Introduction

This is the basis for the script that we used to introduce participants to the user test.

After welcoming the participants, they will explained the form and the purpose of the test. The purpose is to discover possible deficiencies in the prototype, so that it can be improved. We are not testing the ability of particular participants to work with the system as it is. Any possible failures will be attributed to the implementation in the purpose of future improvement. This test is voluntary so the participants can stop the test if they feel the need to do so. The system has not been user tested in such a manner before, so it is presented in its current work-in-progress state. We need the participants to say out loud their thought process during testing, so that we can more easily analyze possible difficulties. The participants should solve the task without our help, unless they get completely stuck or are about to give up. The general idea of the prototype and the tasks will be quickly explained before the test starts. We will tell the participant to say out loud when they think they are finished with a task. The participant will asked if they have any concerns and the test will begin.

We will present the tasks out loud to the participants and also they will be presented in written form on a piece of paper next to them. The timer should be started right after the task is said out loud and stopped after the participant says they are finished with a task. After all the tasks are solved we will ask the participant to freely express their thoughts about the prototype in general. We can ask them:

Did you find the system intuitive?

Was there anything that confused you?

Do you think some things could have been done better?

After we are finished with a participant, the prototype should be reset.

B.3 User Test In-depth Analysis

The most prominent observation was that the participants felt more connected to their respective sport groups than to NTNUI. This became apparent immediately at the start of every test run in 4 out of 6 cases. The participants were navigating to the group for which they were required to send a form, even though there was a button in the same screen which leads users to form functionality. This mindset can be taken into account through adding the same button to the group navigation bar.

A significant flaw in the test layout was that the system was tested with only one single form instance present at a time. This influenced the test validity, because the deployed system might show even hundreds of form instances at a time. The test was carried out with only one form instance at a time. The effect of this could be observed in participants' behaviour when met with the leader perspective. The participants were sometimes navigating randomly between multiple views until they found one which had a form instance in it.

Most participants did not have any extensive opinions about the system. An exception was one test subject, a group president, who looked beyond the current state of the product during the follow-up discussion and gave us many recommendations. One issue this test subject brought up was that the form lists should have extensive functionality regarding form sorting and restricting the query set. Considering that the customer asked us to implement minimal front end functionality, this is a valuable input to future work. Another thing was that a form should include a period during which it is valid. This would allow for a separation of valid and expired forms to increase visibility. This insight was enlightening due to team's lack of contact with NTNUI members who handle a large number of forms.

Some participants were shortly confused by the moving of form instances from the active view (figure 18) to the completed view (figure 20) after they are signed. A suggestion is to have both the signed and non-signed

forms in the same view with non-signed at the top, combined with the sorting/restricting functionality described above. The division based on incoming/outgoing forms might need reworking, because some of the participants expected to *receive* a signed form, thus checking the incoming list for signed forms.

The signing status of a form is currently shown next to the signers name. It might be more convenient to have a separate column for the status or use bold text or colours/icons. Multiple participants found the browser-specific date picker to be cumbersome. This is something that the team had little control over, and it might be good to find a better solution in the future. To give the users feedback about the different modules they might find themselves in, the buttons in the top navigation bar (top of figure 16) should be visibly active. This might not be relevant if the navigation bar is reworked at a later date.

C Views

C.1 Final product

This section gives an overview of the different views for the end user.

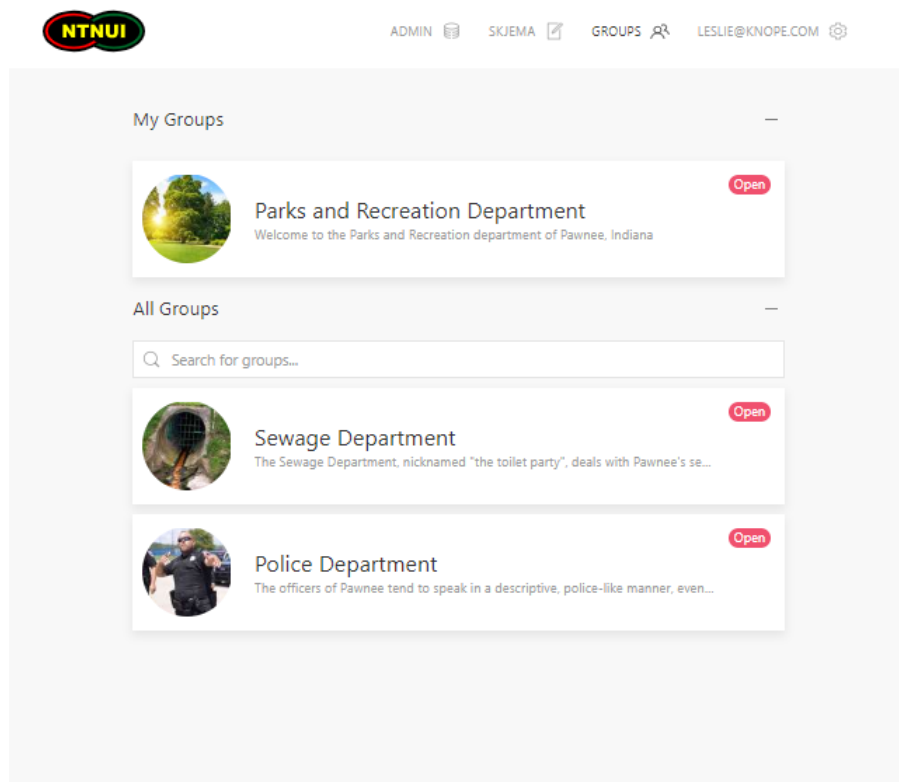


Figure 16: The main page of the system that was implemented by the customer.

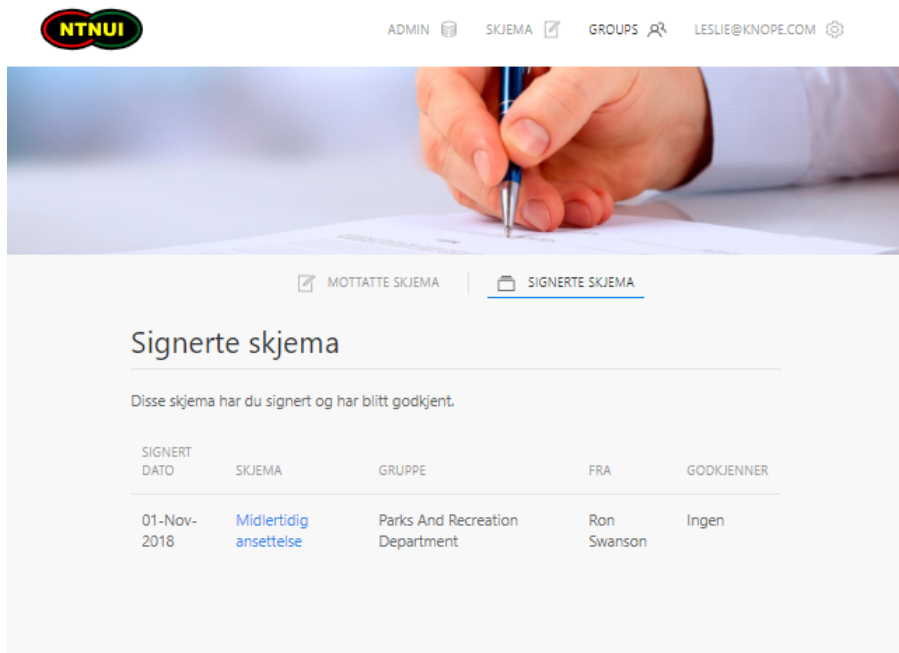


Figure 17: View of the regular user perspective containing received forms that have been signed.



Figure 18: Leader perspective view containing incoming forms that need to be signed by the current user. Analogous to the regular user view in Figure 15.



Figure 19: Leader perspective view containing outgoing forms that need to be signed by the receiver.



Figure 20: Leader perspective view containing incoming forms that were signed by the current user.

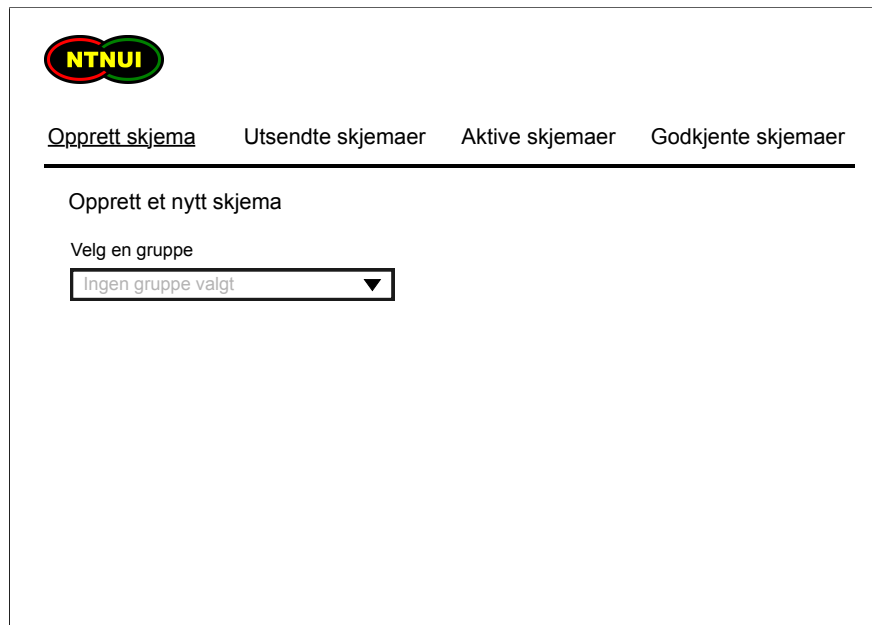


Figure 21: Leader perspective view containing outgoing forms that were signed by the receiver.

C.2 Paper prototype

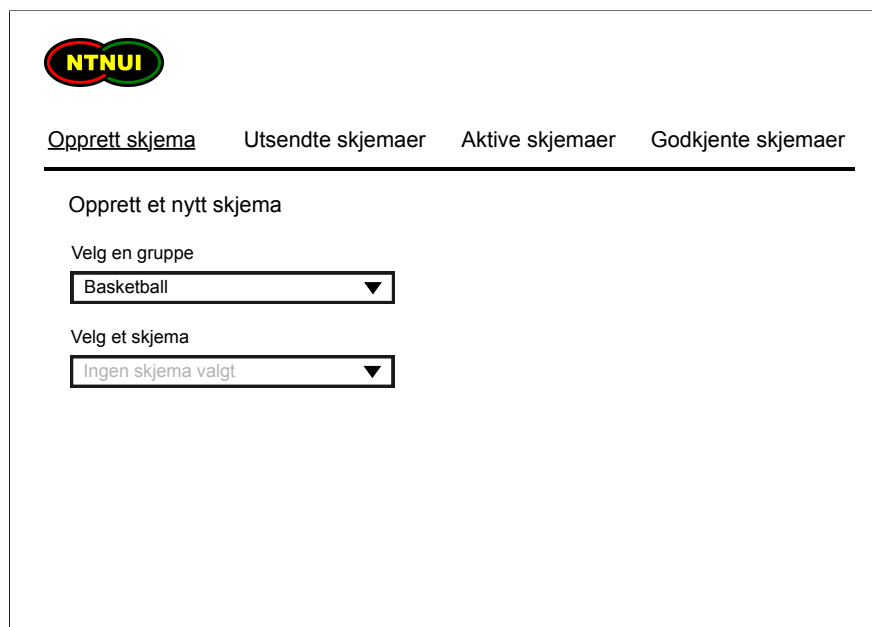
In the planning phase of the project the prototypes were created to guide the developers and give the developers an impression of what the system would be capable of. In the prototypes we used the terms "Approved forms" and "Active forms". "Approved forms" are forms that the user have signed and have been approved. "Active forms" are forms that needs the user's signature.

C.2.1 Form-owner perspective



The image shows a paper prototype of a web interface. At the top left is the NTNUI logo. Below it is a horizontal navigation bar with four items: "Opprett skjema" (underlined), "Utsendte skjemaer", "Aktive skjemaer", and "Godkjente skjemaer". A horizontal line is positioned below the navigation bar. Below the line, the text "Opprett et nytt skjema" is displayed. Underneath, the label "Velg en gruppe" is followed by a dropdown menu with the text "Ingen gruppe valgt" and a downward-pointing arrow.

Figure 22: Paper prototype view that shows the first step in the process of sending a form.



The image shows a paper prototype of a web interface, similar to Figure 22. It features the NTNUI logo and the same navigation bar with "Opprett skjema" underlined. Below the navigation bar, the text "Opprett et nytt skjema" is displayed. Underneath, the label "Velg en gruppe" is followed by a dropdown menu with the text "Basketball" and a downward-pointing arrow. Below this, the label "Velg et skjema" is followed by a dropdown menu with the text "Ingen skjema valgt" and a downward-pointing arrow.

Figure 23: Paper prototype view that shows the second step in the process of sending a form.

NTNUI

[Opprett skjema](#) [Utsendte skjemaer](#) [Aktive skjemaer](#) [Godkjente skjemaer](#)

Opprett et nytt skjema

Velg en gruppe

Velg et skjema

Velg mottaker av skjema

Figure 24: Paper prototype view that shows the third step in the process of sending a form.

NTNUI

[Opprett skjema](#) [Utsendte skjemaer](#) [Aktive skjemaer](#) [Godkjente skjemaer](#)

Opprett et nytt skjema

Velg en gruppe

Velg et skjema

Velg mottaker av skjema

Send skjema

Figure 25: Paper prototype view that shows the fourth step in the process of sending a form.



Figure 26: Paper prototype view that shows the fifth step in the process of sending a form.

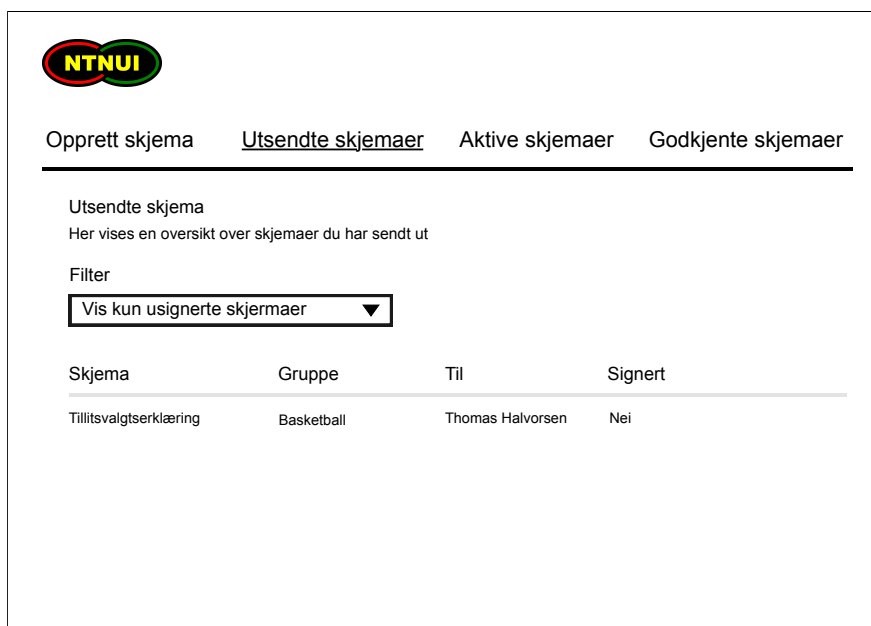


Figure 27: Paper prototype view that shows the forms sent and filter functionality.

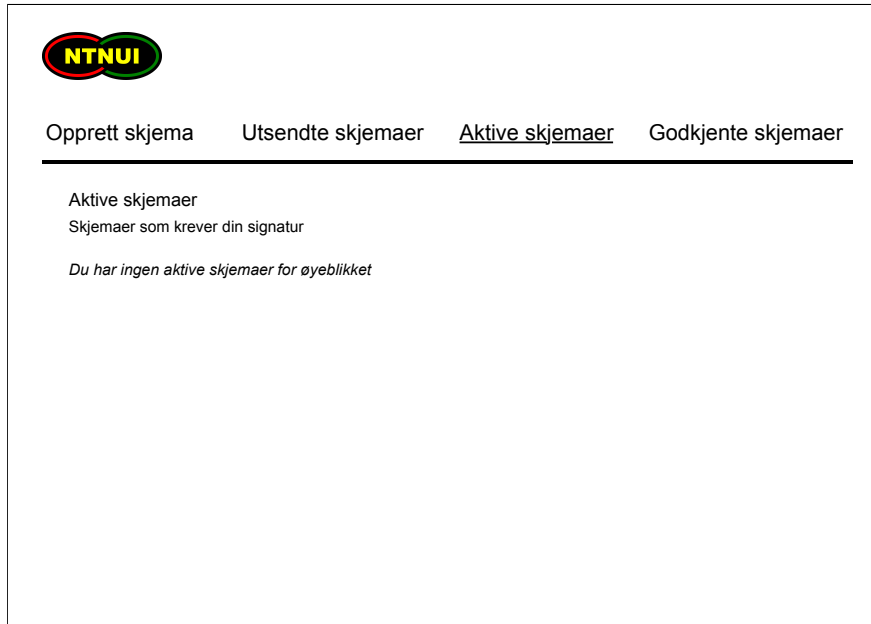


Figure 28: Paper prototype view that shows the active forms.

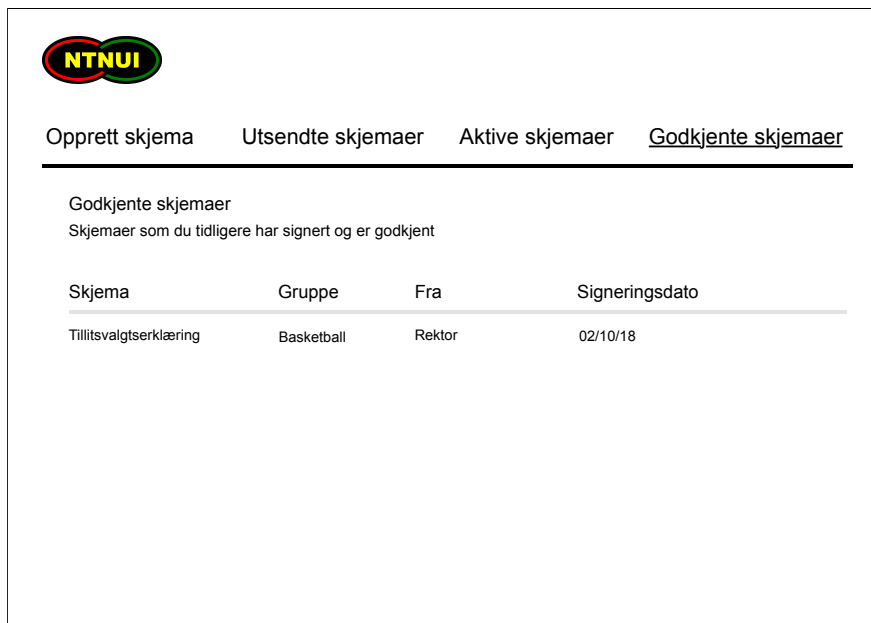
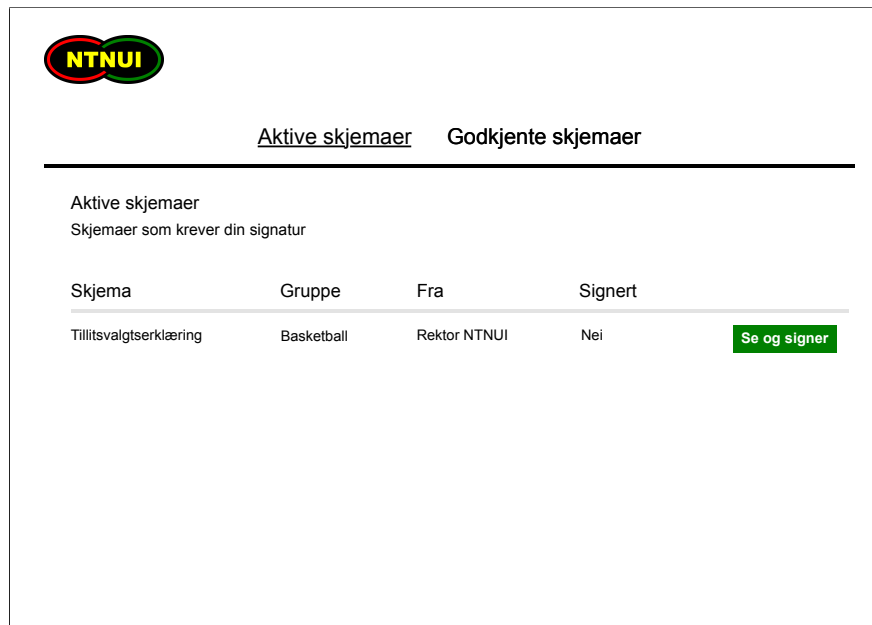


Figure 29: Paper prototype view that shows the approved forms.

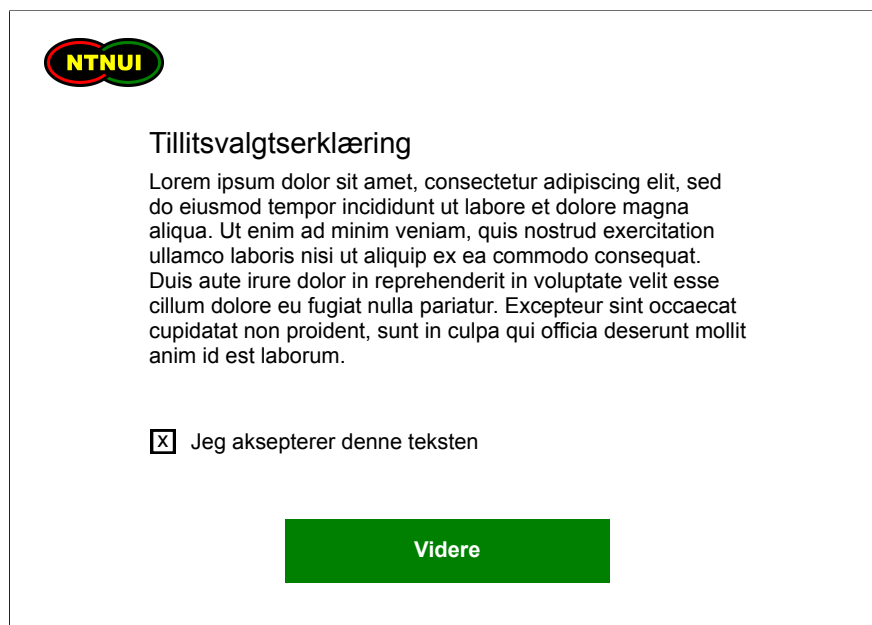
C.2.2 Form-signer perspective



The image shows a paper prototype of a web interface for signing forms. At the top left is the NTNU logo. Below it, there are two tabs: "Aktive skjemaer" (Active forms) and "Godkjente skjemaer" (Approved forms). Under the "Aktive skjemaer" tab, the text "Aktive skjemaer" and "Skjemaer som krever din signatur" (Forms that require your signature) is displayed. Below this is a table with four columns: "Skjema", "Gruppe", "Fra", and "Signert". The first row of the table contains the text "Tillitsvalgtserklæring", "Basketball", "Rektor NTNU", and "Nei". To the right of the table is a green button labeled "Se og signer" (View and sign).

Skjema	Gruppe	Fra	Signert
Tillitsvalgtserklæring	Basketball	Rektor NTNU	Nei

Figure 30: Paper prototype view that shows the active forms.



The image shows a paper prototype of a form titled "Tillitsvalgtserklæring". Below the title is a paragraph of placeholder text: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum." Below the text is a checkbox with the text "Jeg aksepterer denne teksten" (I accept this text). At the bottom of the form is a green button labeled "Videre" (Next).


Figure 31: Paper prototype view that shows the first step in the process of filling a form.

**Skriv inn ditt NTNUI-passord
for å signere dette dokumentet**

Passord

Signer med passord

Figure 32: Paper prototype view that shows the signing step in the process of filling a form.




Aktive skjemaer Godkjente skjemaer

Aktive skjemaer
Skjemaer som krever din signatur

Du har ingen aktive skjemaer for øyeblikket

Figure 33: Paper prototype view that shows the active forms. The list is empty.



[Aktive skjemaer](#) [Godkjente skjemaer](#)

Godkjente skjemaer
Skjemaer som du tidligere har signert og er godkjent

Skjema	Gruppe	Fra	Signeringsdato
Tillitsvalgtserklæring	Basketball	Rektor	16/10/18

Figure 34: Paper prototype view that shows the approved forms.

D Reports

This section covers the status reports that were sent to supervisor(status report),meetings with the customer, and team contract.

D.1 Customer feedback

Author: Anders Kirkeby, representative of the NTNUI IT Committee

With around 12.000 members, NTNUI is considered the largest sports association in Norway, organising more unique sports than the Norwegian Sports Federation (NIF). Being the university’s sports association the organisation is run entirely by students, and has been since its inception over 100 years ago.

In 2017 the NTNUI also held a spot as one of the customers of the customer driven project. The resulting CRM system has since been worked on by the volunteers at NTNUI Sprint (NTNUI’s IT Committee). The feedback from other student sports association has been great and other organisations (like NIF) is now turning towards NTNUI for feedback on modernising sports organisations in Norway.

To help alleviate some of the paperwork that unavoidably comes with a hundred year old organisation, the CRM system needed to be extended with a form module where users can fill out, sign and manage paperwork that would otherwise be handled physically. The student group were tasked with creating this module by extending the existing CRM-solution. Integrating with the pre-existing system set precedence for which technologies the students would need to build the system in. Frequent meetings were therefore held in the beginning of the project to alleviate friction and potential unclear requirements. Due to the communication pipeline between the student group and the IT-committee we agreed to omit using ”user stories” and instead focus more on ”functional- and non-functional requirements”.

As the project developed we realised the scope would need to be reduced in order to ensure the robustness of the system. Our team worked along with the student group to find common grounds and remove any low

priority requirements. The group held a lot of good viewpoints on the functionality of the system, which in turn helped us discover previously unseen shortcomings in the old system. Not blindly accepting all provided requirements but giving constructive criticism where relevant has undoubtedly increased the overall value of the end product.

The student group has delivered on all key aspects of the product order, in terms of both functional- and non-functional requirements. They have developed a product of high quality that can easily be incorporated into the existing solution, and accommodates our demand for modularity. Overall we are extremely satisfied with the final product and thankful for the collaboration with both IDI and the student group.

D.2 Summary customer meetings

TDT4290: First customer meeting resume 28.08.2018

Group 6

August 30, 2018

1 Problem

NTNUI's management system today mainly consists of non-digital solutions when it comes to form management and control over various members rights. NTNUI wants to improve the member management system to increase efficiency and to show a higher level of professionalism. As of today, the distribution of rights each member has is done manually. There is also a lack of control and oversight over the distribution of these rights. When NTNUI members take on different responsibilities, they have to sign a declaration of these responsibilities. As of today, this is also done manually. This manual labour is both resource- and time-consuming.

2 Expected functionality

Expected functionality in (approx.) prioritized order:

1. Possibility to sign predefined ('hard coded') forms with and without changes in access rights
 - Automatically send a "Tillitsvalgterklæring" to new board members that they are required to sign.
2. Authorization of board changes by the group admins. This authorization should be signed.
3. External auditing (Admin/HS(?) approval) for board changes in the different NTNUI-groups.
4. *Bonus*: Possibility to fill out and signing other predefined forms, like travel expenses form, disbursement form, internal auditing form (one for accounting, and one for GDPR)
5. *Bonus*: Possibility to create, fill out and sign generic forms.

3 Requirements by NTNUI

The solution must be integrated with existing member system. The existing system is developed as a Django application, so our solution must also be developed as a Django application with limited external libraries. Due to expected merging of solutions with the existing system and further post project development, the code must be extensively documented both in-code and using Github Wiki. This includes assumptions and choices made when developing. The code should also be modular. The solution must be thoroughly tested using user testing to ensure usability by people from non-technical backgrounds. It should be mobile friendly and be implemented using both Norwegian and English. Throughout the development stage of the solution the code will be merged with the NTNUI code base. The code should be of high quality and security is important.

4 Expectations to NTNUI

NTNUI made several promises regarding access and resources provided, this has led to the following expectations:

- Test server running on AWS
- Establishing common work platforms: Github, Slack, Google team drive (already fulfilled)
- Provide test subjects for user testing
- Provide high availability on Slack and in person when needed
- Provide a forked repository from main code base on Github
- Provide an authentication service using BankID (Still looking at possible providers)

Summary of Meeting with Customer - 9th of october 2018

1. We agreed upon an implementation of the Abstract Form with the following functionalities:
 - a. Who can instantiate a form
 - i. We agreed that the instantiator is a user with role president within a group
 - b. Who can sign form
 - i. Users in a group
 - c. Who can view form
 - i. Signer and instantiation as well as the approver
 - d. Who should be notified using send mail (Django)
2. Notification should be generated via email for form signer on form instantiated and form owner when form is signed by form signer.
3. Modularity of forms is required for future development.
4. Form owner can be a user with a role "President" and part of administration with option for instantiating and viewing forms for that group
5. Implementation of access right is out of scope, however role-based access rights is suggested for future work.
6. We agreed that the mock-data we have received should be sufficient now given that access-control is out of the scope
7. We agreed that the customer should respect the deadlines agreed upon, and we Group 6 should be clearer and give tasks written and with a due-date, preferably on Asana
8. NTNUI will provide people for a user-test, most likely in the beginning of November
9. NTNUI said that all functional requirements with a priority 'low' will be considered as Future Work.
10. Terminology:
 - a. Group = A physical collection of people, could be a sports group (Including all members of that sport) or could be an administrative group such as Hovedstyret
 - b. Form-owner = form-instantiator = User with role "President"
 - c. Form-signer = a user in the group of the form-owner that has been requested to sign a form

D.3 Important status reports

Status report - 23. October

Since last status report

- Good progress on the development
 - Many of the functional requirements in our scope is now implemented
- Slow progress on the report
 - After the group leader meeting last week, we discovered we were far behind on the report writing. We need to step up. We have made some changes in the group to increase the report writing performance. Last Thursday more team members were assigned to primarily focus on the report.

Customer meeting

- The customer confirmed which functional requirements are done. All functional requirements are considered done.
- The customer was happy with the demo of the product and the code review and had feedback on our project. They had a minor change they wanted us to do:
 - When you sign a document and click sign the user is sent to "Arkiv" and not "Aktive skjema" - This is completed
- During the demo the customer gave some tips on how to improve our code, if we have time, we may improve our code by following these tips
- After our project is finished the customer wanted a list of what they can do next. This we'll be written about extensively covered in our report.
- The customer will provide at least six users for user testing next week. The testers will be non-technical as well as technical.
- They also gave best practices for how to document the code and write documentation on GithubWiki.

The next weeks

- The implementation of the project will be finished by next week, and then everyone will work on the report. The group members will work with report sections that are the most relevant to them, for example people who have written integration testing will write the testing part of the report.
- User testing will be done 1st of November
 - If there are minor feedback that is possible to change fast, it may be implemented.
 - Larger changes that are difficult to implement will be considered future work

Status report - Week 44

This week we have mainly been working with report writing and user testing.

On Thursday we held a user test for six users selected by NTNUI. We got a lot of good feedback from the users, both in terms of what was good about the system and what was bad of the system. The feedback we collected is now being analysed and the results will be put into the report together with relevant information about the user test. As a result of the feedback, some small changes will be done to the code, but all of the bigger suggestions for improvements we got from the user testing will be considered future work since we do not have time to implement it.

When it comes to the report, we have made great progress and many of the suggestions you mentioned in the meeting last Tuesday are now integrated into the report.

D.4 Team contract

Team contract KPRO

Group 6:
Jonathan Linnestad
Anders Salvesen
Erik Liodden
Siren Finvik Johansen
Jan Burak
Kristian Thoresen
Maria Iqbal
August 2018

1 General

- Democratic decision, by tie Siren decides
- Ask for help, there are no stupid questions. If you are stuck ask.
- If you say you will complete a task within a certain time and you get delayed, inform the team.
- Cake punishment if you are substantially late to work hours or if late to customer/supervisor meetings

2 Routines

- Work together Tuesday 12:15-16:00 and Thursday 10:15-16:00
- When needed and team agrees we have room on Tuesdays until 20:00 and Thursday until 18:00
- Be on time. If you cannot attend or are delayed, inform the other group members.
- Hours spent must be registered after each session.

3 Meetings

- Mail with time and location of meeting in good time before said meeting
- Meet on time and read agenda for meeting before meeting

4 Coding guidelines

- Everyone should sufficiently comment their code.
- Follow code style from NTNUI
- Do not commit IDE or environment specific files
- Code review, at least one person from team needs to approve code
- Linting, decide on common linting style
- Keep functions simple and atomic within what is possible
- Write variable and function names that are as self-describing as possible
- Test what makes sense to test

5 Definition of done

- When is a task done. Tested, code reviewed, and user story is done
- When is a User Story, when NTNUI accepts it as done, and user testing has been conducted where needed

6 Crisis management

- If a group member gets sick inform supervisor
- Inform NTNUI immediately about the issue

7 Roles

- Team Leader: Siren
- Product Owner/Customer: NTNUI
- Test manager: Jonathan
- Quality manager: Jan

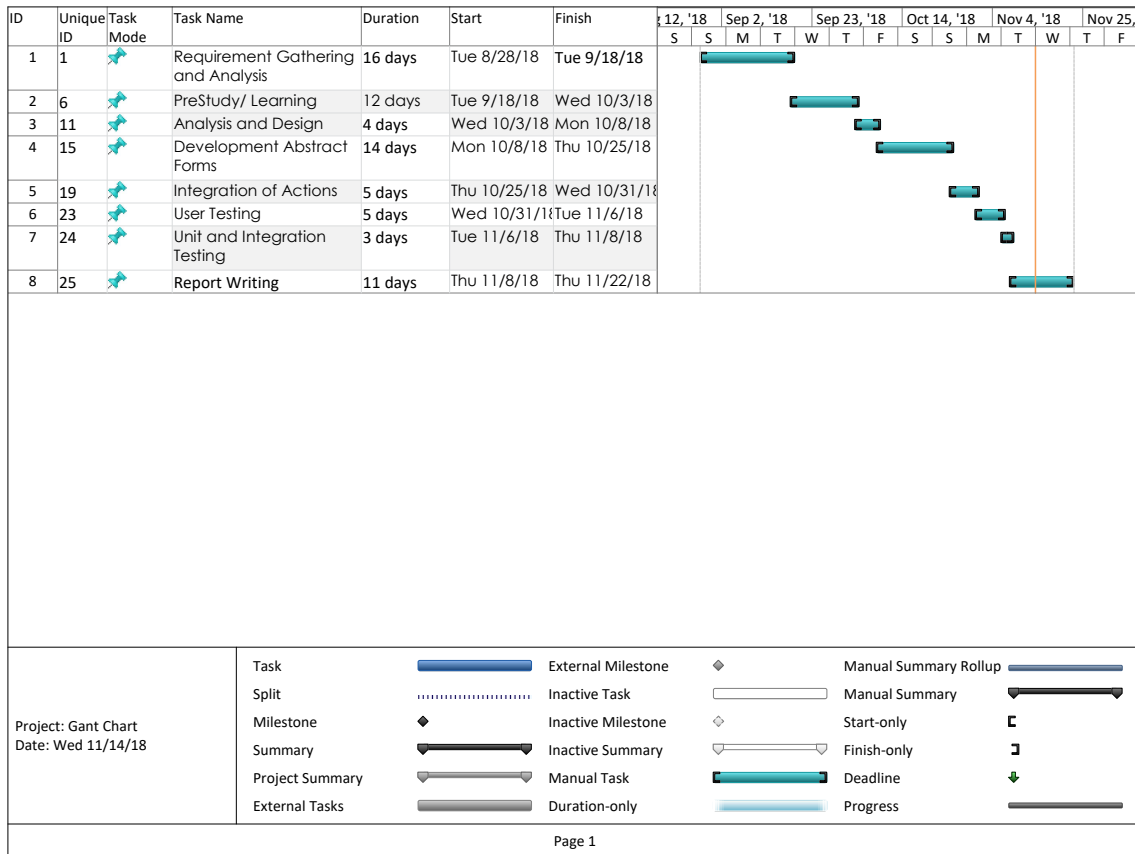


Figure 35: Summary of Gantt Chart

D.5 Gantt Chart

An outline of the Gantt Chart is shown in figure 35. The complete chart can be found here: http://folk.ntnu.no/eriklio/gantt_chart.mpp.

E Other

E.1 Working Hours

Figure 36 shows the hours each team member spent on the project. Week 47 is an estimate of how much time the team will spend preparing the presentation and video that is to be delivered. Figure 37 shows the total time spent on each phase, and is the absolute data that the pie chart in Figure 2 is based on.

Navn	Uke 35	Uke 36	Uke 37	Uke 38	Uke 39	Uke 40	Uke 41	Uke 42	Uke 43	Uke 44	Uke 45	Uke 46	Uke 47	Total
Jonathan Linnestad	15	11	13	10	10	13	10	13	10	19	25	25	25	199
Erik Lodden	15	11	6	11	14	23	15	14	15	15	35	35	19	228
Anders Salvesen	15	11	13	11	9	19	15	14	19	15	30	38	19	228
Jan Burak	15	16	6	20	19	24	18	23	19	15	34	25	19	253
Kristian Bjørn Thoresen	15	11	5	13	9	21	10	15	18	19	30	35	19	220
Siren Finvik Johansen	15	6	16	13	15	3	4	21	20	23	45	41	19	241
Maria Iqbal	0	11	10	14	6	18	6	13	5	18	19	20	16	156
													sum:	1525

Figure 36: Working Hours per team member per week

	Estimated Working Hours
Report Writing	460
Analysis and Design	170
Requirement Gathering	190
Development	410
Testing	75
Learning Django	220
Sum:	1525

Figure 37: Estimated Hours the team spent on the different phases.