Karoline Wisløff Rud, Ruben Isaksen Cheung,
Jørgen Berntsen, Vegard Elshaug Almås

Digital verification of identity

Bachelor's project in Dataingeniør
Supervisor: Deepti Mishra, Rune Hjelsvold
May 2019

Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

buypass

Karoline Wisløff Rud, Ruben Isaksen Cheung,
Jørgen Berntsen, Vegard Elshaug Almås

Digital verification of identity

Bachelor's project in Dataingeniør
Supervisor: Deepti Mishra, Rune Hjelsvold
May 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Digital verification of identity

Author(s)
Ruben Isaksen Cheung
Karoline Wisløff Rud
Jørgen Berntsen
Vegard Elshaug Almås

Bachelor of Science in Engineering - Computer Science
20 ECTS
Department of Computer Science
Norwegian University of Science and Technology,

20.05.2019

Supervisor Deepti Mishra
Rune Hjelsvold

Sammendrag av Bacheloroppgaven

Tittel:	**Digital identitetsverifisering**
Dato:	20.05.2019
Deltakere:	Ruben Isaksen Cheung Karoline Wisløff Rud Jørgen Berntsen Vegard Elshaug Almås
Veiledere:	Deepti Mishra Rune Hjelsvold
Oppdragsgiver:	Buypass AS
Kontaktpersoner:	Jørn Magne Raastad Alexander Dreyer Johnsen
Nøkkelord:	API, Android, ID-verifisering, Buypass, Scrum
Antall sider:	81
Antall vedlegg:	20
Tilgjengelighet:	Åpen

Sammendrag:

Med et økende antall trusler på internett blir sikker autentisering viktigere og viktigere. Buypass er en av de få aktørene i Norge som kan utstede elektronisk ID på nivå Høyt, og denne ID'en muliggjør innlogging hos aktører som Nav, Altinn og Skatteetaten. Dagens utstedelsesprosess er både ressurs- og tidkrevende, og Buypass ønsket å få utviklet en prototype til en Android applikasjon som ville muliggjøre digital identitetsverifisering. Vår løsning involverer en tre-stegs verifiseringsprosess som består av en dokumentverifisering, en ansiktsgjenkenning og en "liveness check". Applikasjonen utforsker også muligheten til å benytte Near Field Communication for verifiseringen av e-pass med mobiltelefon. Den ferdigstilte løsningen inkluderer implementasjonen av et tredjeparts API, samt et backend API skrevet i Node.js

Summary of Graduate Project

Title:	**Digital verification of identity**
Date:	20.05.2019
Authors:	Ruben Isaksen Cheung Karoline Wisløff Rud Jørgen Berntsen Vegard Elshaug Almås
Supervisor:	Deepti Mishra Rune Hjelsvold
Employer:	Buypass AS
Contact Persons:	Jørn Magne Raastad Alexander Dreyer Johnsen
Keywords:	API, Android, ID Proofing, Buypass, Scrum
Pages:	81
Attachments:	20
Availability:	Open

Abstract:

With an increasing number of Internet threats, secure authentication is more important than ever. Buypass is one of the few parties in Norway that can issue electronic IDs at level High, which enables login to actors such as Nav, Altinn and the Tax Administration (Skatteetaten). Today's verification process is both resource and time consuming, and Buypass wanted us to develop a prototype for an Android application that would enable digital verification of identity. Our solution provides a three step verification process consisting of a document image validation, a face comparison and a liveness check. It also explores the possibility of using Near Field Communication to verify e-passports using a smartphone. The end result includes the implementation of a third-party API, as well as a backend API written in Node.js.

Preface

This bachelor thesis is written by Ruben Isaksen Cheung, Karoline Wisløff Rud, Jørgen Berntsen and Vegard Elshaug Almås, students at NTNU in Gjøvik, department of Computer Science.

We would like to thank Buypass AS, and in particular Jørn Magne Raastad and Alexander Dreyer Johnsen for an exciting assignment, and providing feedback and support throughout the project. The assignment has provided experience in a development project in an exciting area within computer science. It has been motivational to work with something that is highly relevant and provident.

We would like to thank our supervisors Deepti Mishra and Rune Hjelsvold. They have provided valuable guidance and expertise throughout the project. We would also like to thank the staff of NTNU in Gjøvik who have provided assistance with problems throughout the project.

We would lastly like to thank Keesing Technologies and their helpdesk for providing and interesting technology and help throughout the project.

Contents

List of Figures

Listings

1 Introduction

1.1 Background

Buypass AS is a Norwegian IT company that specializes in developing solutions for electronic identification, signatures and payment. Their electronic ID's can be used to access services like Altinn, Digipost, Skattemeldingen, medical journals, prescriptions and so on.

The way it is today Buypass cooperates with certain post offices and in order to obtain a Buypass ID the client has to present himself with a valid ID document in order to verify his identity. This process is both expensive and inconvenient for the clients, so Buypass are looking for a way to make the process faster, less expensive and more user friendly.

Today most people in Norway owns a smartphone, and most of these have cameras and NFC-readers[1]. Buypass are looking for an extension of their existing application, providing the functionality to perform digital verification of identity.

1.2 Project Description

Buypass issues ID's with different security levels. In order to get a level 4 (High) electronic ID the client needs to identify himself to a trained post office employee with a passport. Our project revolves around simplifying this process by letting the client identify himself through the use of his phone. For doing remote ID proofing by using a phone there are two main options, one consists of taking a photo of an ID-document and use image analysis to verify it, or it can be verified by using NFC. We want to explore both these options. In order to get a level 4 ID a passport is needed for the verification process, but in order to make the application applicable for issuing IDs on different levels, we also include the option of using a driver's license.

1.3 Keesing Technologies

Keesing Technologies is a Dutch company that specializes in solutions for authenticating ID documents and banknotes. They cooperate with international government agencies, embassies and banks to provide high quality services.

Keesing has a web API, the Keesing Authentiscan API, which has functionality for checking document authenticity by verifying them against a large database of document templates. It is also able to perform biometric facial comparison and a liveness check which tells you whether a person in a video is a real living person or not.

If an ID document cannot be recognized Keesing has trained helpdesk professionals that perform manual authentication of the document. Their API can be used to authenticate both Norwegian passports and driver's licenses.

[1] Near Field Communication

1.4 Boundaries

Buypass collaborates with Keesing Technologies and we will be implementing the Authentiscan API. So far it does not support verification through NFC, but this feature is supposed to be released in a quite near future. For this reason our application mainly focuses on passport and driver's license verification through photos.

1.5 Scope

Our goal is to create a prototype that demonstrates how digital ID proofing can be implemented by using Keesing Technologies' API. For this reason the main focus is on developing functionality that demonstrates our way of performing ID proofing using a smartphone.

Security is a major concern for such an application, but due to the limited time we had to work on the project, we decided that implementing all the security requirements for an ID proofing app falls outside the scope of the project. For the same reason we have considered optimizations for performance and performance testing to be outside the scope of the project.

Buypass explicitly wants to explore working with Keesing Technologies, so we will not be exploring alternative providers of ID checking solutions. Still, because Buypass does not have any previous experience working with Keesing's APIs, an important part of our project is to evaluate to what degree their API fulfills the requirements imposed by our stakeholders. For this reason we also want to do some preliminary testing of the final product to see whether or not this is a solution that could potentially be used for real life ID verification.

Our product is a native Android application. The main reason why we have not developed an iOS application is that at the moment the NFC technology on iOS devices is locked for third party developers, which means we are not able to implement the NFC part of the application for iOS. We could have made a separate iOS application without NFC functionality, but since our focus is to create a prototype to demonstrate how new technology can be used for ID proofing, our priority was to create a full demo application for Android instead.

1.6 Team Members and Roles

Karoline is our chosen group leader, and scrum master, see figure 1. This gave her the final word if there were any disputes within the group. The choice of Karoline as our group leader was made on the background of her earlier experience in conducting a bachelor thesis. Vegard is the deputy head, and will in the case of Karoline's absence function as group leader. Jørgen is documentary responsible within the group, whether this is meeting reports or documentation regarding the development for the product owner and the supervisors.

Due to the small size of the team for this project we have all worked as developers. In order to foresee the necessary progress regarding functionality, documentation and code quality we have divided the tasks into bigger modules where two and two have focused more on one part.

With a small team we all have been involved in every technology aspect of the development process, but we did give each team member one main area of focus. The team members then were responsible for doing extra thorough research on the field. Jørgen had the main responsibility for the server side of the project, Ruben, the main responsibility for the GUI and UX, Karoline, the main responsibility for functionality of the mobile application and Vegard the main responsibility for the security aspect of the project.

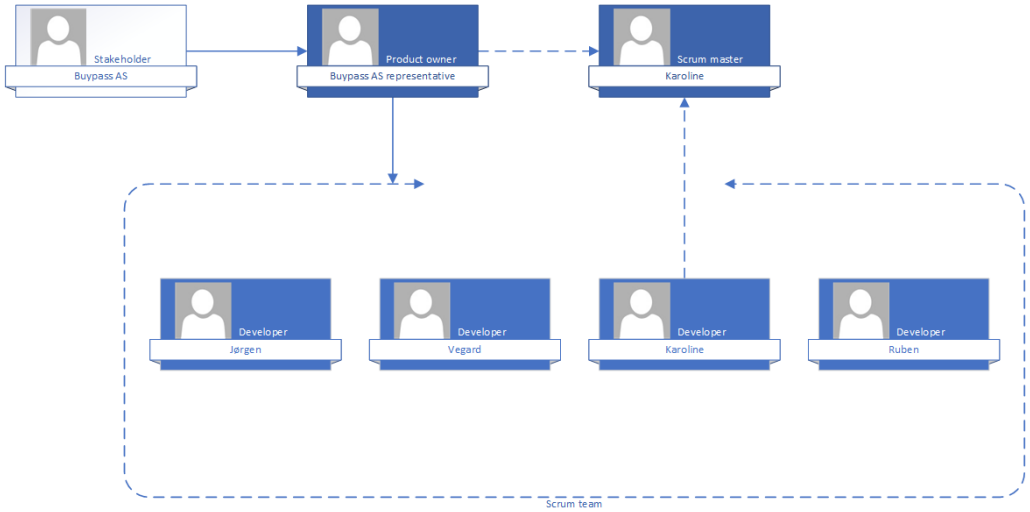


Figure 1: Organization of our developing team

1.7 Target Users

Our target end users is mainly Bypass' old costumers which consists of larger businesses that need authentication smart cards, or private costumers that want an electronic ID. This means anyone of age, and the target audience is large. According to Buypass they have over 2,3 million Norwegian citizens using their products [1].

2 Technologies and Tools

Throughout the project we have used several tools and technologies providing solutions for problems and implementations. Following is a list providing an overview of some of the more specific tools and technologies within our project, see appendix E for a more general and extensive list.

API: An API[1] is a tool for simplifying programming by abstracting the underlying implementation and only exposing objects and actions the developer may need [2].

REST: REST[2] is a software architectural style defining constraints for creating web services [3].

PuTTY: PuTTY is a remote terminal emulator which allows for logging onto our backend VM[3]. Configured via SSH[4] through NTNU's network. SSH is used for remote command-line login and remote command execution [4].

SSL: SSL[5] is a standardized cryptographic protocol and security related technology for securing communication online. See appendix S for more information and configuration details.

Firebase: Google's web and mobile application development platform, providing different functionality such as Google Cloud Functions and Cloud Messaging.

Firestore: Flexible and scalable database for web and mobile applications.

Firebase Cloud Messaging: FCM[6] is a cross-platform messaging solution [5].

2.1 Near Field Communication

Near Field Communication (NFC) is a technology evolved from radio frequency identification (RFID) that enables wireless communication between two devices by bringing them within short distance of each other (4-20 cm) [6]. NFC is used for many purposes, like in contact-less payment, public transportation cards and electronic passports. Most newer Android phones are equipped with an antenna that enables it to read NFC tags.

E-passports: All passports are standardized by ICAO (International Civil Aviation Organization). Since 2016 all passports are required to be Machine Readable (referred to as Machine Readable Travel Documents/MRTD), which means that they need to have two lines at the bottom of the passport which can be read by a machine, and which contain information like name, date of birth, expiry date etc. This is referred to as the Machine

[1] Application Programming Interface
[2] Representational State Transfer
[3] Virtual Machine
[4] Secure Shell
[5] Secure Sockets Layer
[6] Firebase Cloud Messaging

Readable Zone (MRZ). Many passports, including Norwegian passports issued after October, 2005 [7], are so called e-passports. E-passports have a chip (called an RFID chip) that contains all passport information, like personal data (name, age, personal identification number etc), images and biometric info (only in new passports).

Access protocols for e-passports: Since an electronic passport can be read by any device with an NFC reader, certain security measures are required. Otherwise one could imagine personal data being stolen by someone scanning a passport through a person's pocket while waiting in line at the airport or in similar situations. To avoid this the RFID-chip is protected by a Basic Access Control protocol(BAC). BAC involves calculating a key from the passport number, expiration date and date of birth and using this to access the chip. There are many different files stored in the RFID chip. These have different levels of security, so that only authorized readers can access the most sensitive data. In this project we are mainly interested in two files, the DG1 file, which contains all personal data from the MRZ (name, nationality, gender, date of birth, personal identification number and so on), and the DG2 file, which contains passport image data. Both of these files are accessible through BAC.

More sensitive information, like fingerprints and other biometric data is protected by another security protocol, the Extended Access Control (EAC). To pass the EAC the NFC reading device needs to present a CVCA[7]-certificate that proves that it has permission to read sensitive files.

There are also protocols for verifying the authenticity of the RFID chip itself. The Passive Authentication protocol can be used to verify that the content of the RFID chip has not been tampered with, and the Active Authentication protocol can be used to reveal if the chip has been cloned [8].

[7] Country Verifier Certification Authority

3 Requirements

Keesing's API offers three ways to make a verification safe:

1. Checking the authenticity of the chosen ID document.
2. Using facial recognition to verify that the document belongs to the person using the app.
3. Detecting human presence in the verification through a "liveness" check.

In order to go through with these steps our application must be able to

- Take photos and a video.
- Upload these media files to Keesing for verification.
- Retrieve the verification results from Keesing.
- Inform the user when the verification result is ready.
- Convey the verification results to the user.

We will start by giving an overview and description of the most important use cases and then we will dig deeper into the specific requirements for different aspects of the application (camera, database, security etc).

As Keesing is not able to perform ID proofing through NFC[1] yet, the NFC part of our application is not part of the real verification process in our prototype. For this reason we will discuss the use cases and requirements for the NFC application in its own section at the end of the chapter.

3.1 Login

As our application will be integrated into Buypass' existing app, which already have a secure login procedure, creating a safe login is not part of our requirements. However, since a verification process needs to be connected to a specific user, we need to create a "mock" login screen in order to obtain a user name. This screen is only created for demonstration purposes and will not be part of a real integration of the app. For this reason we do not have any specific requirements to the login screen or the login security.

3.2 Use Cases

The following pages contain descriptions of the most important use cases. Figure 2 shows how use cases related to the verification process using the phone camera relates to one another and the different actors.

Name:	Show info
Actor:	User
Goal:	Get more information
Description:	At each verification step, the user can click an information button in order to show more information about that particular step of the process.

[1] Near-field communication

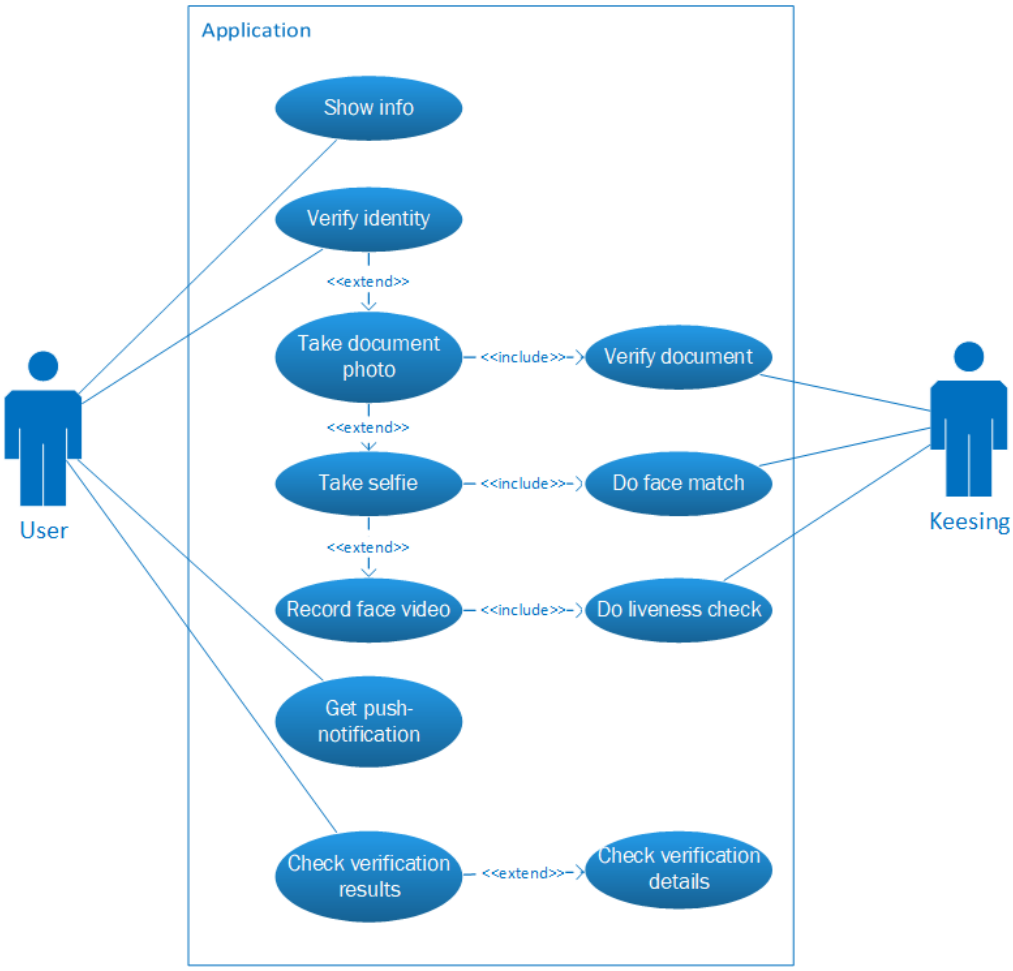


Figure 2: Use cases involving verification using image analysis

Name:	Verify identity
Actor:	User
Goal:	Verify the users identity against an ID document
Description:	The user verifies his identity by going through the three steps: document photo, face photo and liveness check. The verification process starts when the user picks passport or driver's license as his ID document of choice.

Name:	Take document photo
Actor:	User
Goal:	Take a photo of the ID document
Description:	The user takes a photo of the main page of his passport or his driver's license. If he uses a driver's license he takes a photo of the front and back of the document.

Name:	Take selfie
Actor:	User
Goal:	Taking a face photo
Description:	The user takes a photo of his face in order to prove that he is the same person as the person on the ID document.

Name:	Record face video
Actor:	User
Goal:	Record a video proving that the user is a real person
Description:	The user records a two second video where he is instructed to say a sentence or make a few grimaces in order to prove he is a real person.

Name:	Verify document
Actor:	Keesing Technologies
Goal:	Verify the authenticity of an ID document
Description:	Keesing receives a document photo and uses image analysis to perform an automatic verification of the document. If the document cannot be recognized, a manual check is performed at their helpdesk.

Name:	Do face match
Actor:	Keesing Technologies
Goal:	Check if the person on a document photo is the same as the person on a face photo.
Description:	After receiving a document photo, Keesing receives a face photo and uses facial recognition to check that the face on the ID document matches the face in the face photo.

Name:	Do liveness check
Actor:	Keesing Technologies
Goal:	Check that the person on the face video is a real person
Description:	Keesing receives a face video and uses video analysis and blinking detection to verify that the person in the video is a real person.

Name:	Get push notification
Actor:	User
Goal:	Inform the user that the result of a verification process is ready.
Description:	When the result of a verification process is ready, the user should receive a push notification letting him know that he can log in and check the results.

Name:	Check verification results
Actor:	User
Goal:	Check the result of all completed or ongoing verification processes.
Description:	The user logs in to "Min side" in order to see the result of a verification processes. The verifications are sorted by date and time and color coded (green = approved, red = not approved, orange = pending). The user can click on a specific verification to show details.

Name:	Check verification details
Actor:	User
Goal:	See details about why a verification was approved/not approved.
Description:	If the user clicks on a verification in the result list he can see the outcome of each of the verification steps (document photo, face match and liveness check). If either of them failed a message explains why it failed and what to do next.

3.3 Backend

The backend of our project needed to be able to handle requests from both the mobile application and the third party API, and to work as a mediator between the two. The API would further on need to be able to handle photo uploads, video uploads, larger JSON arrays/objects and interaction with the database.

3.3.1 Database Requirements

As we are dealing with quite sensitive data, we do not want to store more data than we need. Our database serves two main purposes:

1. To store the result of a verification process.
2. To store data needed for communication between our server, Keesing's server and the user application.

The first requirement is imposed by Buypass. When a user completes a verification process they want to be able to see the result of the verification and to know details about why it succeeded or not. The verification result we receive from Keesing is a JSON object, see appendix G.

All the fields of the verification result should be stored in our database although the some of the fields (like personal ID number, document number, date of birth etc) are not strictly necessary for our application as the verification process is not dependent on it. If our prototype were to be developed further, it would be useful for Buypass to be able to compare this information from the user's Buypass profile and the National registry, so they requested to store the entire verification result.

The second requirement relates to what information we need to keep stored on a user in order to facilitate communication between app, server and Keesing. Keesing uses a verification ID to identify each verification process. Photos and videos are uploaded individually, and so whenever a file is uploaded from the user, we need to know which verification ID to use. Once the user has started a verification process he cannot start another verification until the first one is either completed or aborted. For this reason we always need to store the verification ID of an ongoing verification process.

Since we use push notifications to inform the user when Keesing is finished processing a verification process, we also need to store an individual identification token for this purpose.

With all this in the database it must support these main queries.

1. Collecting the ID of an ongoing verification process.
2. Check the status of a user's verification processes.
3. Collect a user's token in order to send a push.
4. Find the username associated with a given verification ID.

3.4 Frontend

As our task is to create a native Android application the frontend of our project is set to be a smartphone which will communicate with the backend as mention over. The application will target API level 26 as we want to aim at Google Play requirements for uploading an APK[2].

3.4.1 User Experience

The application is set to be a functionality demonstrating application for Buypass, and the design aspect of the application is important. This involves both UX[3] and GUI[4]. We have created a list of requirements to fulfill in order to reach the desired end result. The requirements are inspired by the articles "Principles of User Interface Design" by Joshua Porter [9] and "7 Key Attributes Of A Quality UI" by Samella Garcia [10], and the Wikipedia article about the principles of UI design [11].

- Simplicity - do not clutter the mobile screen.
- Clarity - make it obvious for the user what to do, and where he or she may get more information.
- Consistency - be consistent in the theme of the app so that there is a running theme throughout the whole application.
- Familiarity - the users familiarize in the way the application is designed.
- Efficiency and responsiveness - the inferface is responsive and it does not leave the user waiting for longer periods unless absolutely necessary.
- Style - the style choices of the application in regards to color palettes, font, button styles and so on should be inspired by Buypass' style of choice.

3.4.2 Camera Requirements

The camera is important to large parts of the application's functionality as face match, liveness check and document check all depend on the camera. The requirements can be separated into functional requirements and requirements related to the user experience.

3.4.2.1 Functional Requirements

Keesing Technologies uses images as the basis for a user verification. The process requires a document photo (front and back if the user is using a driver's license), a face photo (selfie) and a face video. In order to process the images the following requirements must be fulfilled.

- Images must be in JPEG format with no compression.
- Videos must be in MP4 format.
- Images size must be greater than 1.5 Mb.
- Face videos must be at least 2 seconds.

[2] Android application package
[3] User Experience
[4] Graphical User Interface

- Video size cannot exceed 5 MB.
- Image sharpness must be greater than 17 (i.e. blurred images are not accepted).
- In document images, the entire document needs to be inside the image and the document must not be rotated.

The image quality of the document images is particularly important. If the image follows all the previous requirements, Keesing is usually able to perform an automatic document check, which only takes a few seconds. However, if the photo is inadequate (for example if the entire document is not inside the image) the document is usually not recognized and ends up having to be verified manually, which takes a lot longer. To avoid this issue we want to design the camera and camera process in a way that minimizes the number of images of too poor quality for automatic verification.

3.4.2.2 Camera UX Requirements

The camera application should behave in a way that makes it easy for the user to to take valid photos for the different purposes. To accomplish this we have defined the following requirements:

- When taking a document photo, the user should open the rear-facing camera and when taking a selfie or face video it should open the front-facing camera. It should not be possible to change the camera direction.
- The camera screen should show some brief instructions to remind the user of key things to keep in mind.
- The user should be able review and re-take the image/video if he is not happy with it.
- The application should guide the user in a way that makes it easier to take document photos correctly.
- If the captured image/video is not accepted, the user should get appropriate feedback so that he can take a new and better one.

3.5 Security Requirements

Handling personal information is a central part of our application, so handling this information appropriately is of course an important requirement if the app were to go into production. In addition to general security measures, there are certain public legislations that would have to be taken into account. The law on anti money laundering (Hvitevaskingsloven) has certain requirements for issuing electronic IDs [12] and handling of private information must comply to GDPR[5].

However since our app is just a prototype security is not the main focus. We do acknowledge that this is of vital importance, but for our demonstration we have chosen to prioritize focusing on functionality and creating an attractive demo product.

Still, in order to create a good basis for further development, we have evaluated what we consider to be the most obvious security threats and taken action to reduce their risk. In order to enhance the security of our application we have come up with the following security requirements:

- All communication between the server and the application must be encrypted.
- Photo and videos taken have to be stored on the application's internal storage.

[5] GDPR - General Data Protection Regulation

- Files should be deleted as soon as it is no longer needed.
- All sensitive data that is stored should be encrypted.
- If data crosses a trust boundary it should be validated.
- All user input should be sanitized.
- The app should use as few third part API's as possible.
- The server should limit the maximum number of requests within a certain time frame to avoid DOS-attacks.

3.5.1 Misuse Cases

The following section describes some potential misuse cases for our application:

Name:	Steal personal Information
Actor:	Crook
Goal:	To steal personal Information from another user
Description:	By performing spoofing/sniffing attacks, or attack the database the crook can potential gain personal information about a user.
Countermeasures:	Securing our database, encrypting all communication between different components, only storing information that is necessary and deleting information when it is no longer needed.

Name:	Denial of Service attack (Backend)
Actor:	Crook
Goal:	To incapacitate the system so that other users cannot access it
Description:	An attacker can perform a DOS attack by flooding the server with requests resulting the system to be overloaded and thereby unavailable to other users.
Countermeasures:	Limiting the number of requests an IP-address can make within a given time frame.

Name:	Change verification results
Actor:	Crook
Goal:	To change a false verification to be approved
Description:	An attacker enters the database and changes the outcome of a verification process
Countermeasures:	The database can only be accessed through our server. No direct communication between the application and the database.

Name:	Impersonating another user
Actor:	Crook
Goal:	To gain a falsified electronic ID
Description:	A crook steals the identification document of another user and uses this to obtain an electronic ID in their name.
Countermeasures:	The user has to prove that he is the document owner by taking a photo of the document and his face, so that the document can be verified and the face images can be compared.

3.6 Evaluation of the Verification Process

The last misuse case describes how a user with malicious intent could use the application to obtain an electronic ID in another person's name. Preventing this kind of exploits is of

course a vital requirement for our product, which is why we, after recommendations from Buypass, Keesing and BITS[6] decided on a three-step verification process. Nevertheless, our solution depends heavily on the Keesing API, which means that some security issues are outside of our control. If the solutions Keesing provide for document verification, face comparison and liveness check are not reliable, our application can not provide a reliable verification of identity. The experience we gain from developing our application gives us information and insight in Keesing's solutions that could be valuable to our product owner when deciding if (and how) they should develop a digital ID verification application for production. For this reason evaluating how Keesing's solutions work out in the final product is part of the project requirements. We do not intend to do a full analysis of potential security threats in digital ID proofing, but we want to get a general idea of what challenges it may run into.

In order to evaluate this we came up with a few potential threats where a malicious user manages to trick the application by issuing an ID through completely normal use of the application. Implementing countermeasures for these misuse cases is not within the scope of our project, but we want to explore whether they are possible or not.

Name:	Document printing attack
Actor:	Crook
Goal:	To pass the document check using a printed copy of an ID document
Description:	A user obtains an image/photocopy of another person's ID document and passes the document check by taking a photo of that image.

Name:	Trick face comparison
Actor:	Crook
Goal:	Trick the face comparison step by taking a photo of something that is not the person on the document.
Description:	The user passes the face comparison by uploading a fake face photo. This could be a photo of for example document owner's profile picture on social media, or simply a photo of the document face image.

Name:	Trick liveness check
Actor:	Crook
Goal:	Pass the liveness check without filming an actual person
Description:	The user records a video of for example a YouTube clip or similar in order to pass the liveness check.

We also wanted to see if the process was prone to false negatives as this would lead to a frustrating user experience. To evaluate this we simply wanted to explore if verification process is likely to fail when a person uses their own valid ID.

3.7 NFC Requirements

As mentioned Keesing Technologies only provides ID proofing using document images and not NFC at the moment. However, we have discussed this possibility with both Buypass and Keesing Technologies and we know that Keesing will release this feature in the future. Therefore we wanted our application to implement some NFC functionality. The NFC requirements imposed by Keesing are still unclear, so we decided to develop an application that can read the basic information from a passport. As driver's licenses do not contain NFC tags, NFC cannot be used for verification of driver's licenses. The use cases related to NFC functionality are

[6] Bank- og finansnæringens infrastrukturselskap, a company for banking and finance infrastructure

presented in figure 3. We decided on the following requirements, as we consider these likely to be important when verification through NFC becomes available:

- The user should be able to enter expiry date, date of birth and document number as this is necessary in order to perform BAC. The user should be able to choose between entering this information manually and using OCR[7] to read it directly from the passport using the phone's camera.
- The app should be able to read the RFID chip in order to retrieve information like name, date of birth, nationality, gender, personal identification number etc.
- The app should be able to read, decode and show the passport photo from the RFID chip as this will most likely be used for face match when the NFC feature is released.

As we do not have access to the CVCA certificate required for the Extended Access Control protocol (see section 2.1), we will not attempt to read fingerprints, biometrics or other sensitive data.

It would have been interesting to explore the possibilities of doing Active and Passive Authentication for clone detection and for verifying the chip integrity. However, to perform Passive Authentication we would need a Country Signer Certificate Authority (CSCA) certificate which is only available to an officially commissioned document issuer [13]. It seems like AA [8] would be possible to implement, but it proved difficult to find good documentation on how to do this, and because cloning of passport chips probably is not the most pressing security challenge of the application, we decided not to include AA as a requirement.

3.7.1 Use Cases

The main use cases for the NFC application is presented in figure 3 and described below:

Name:	Scan MRZ[9]
Actor:	User
Goal:	Read passport number, expiration date and date of birth.
Description:	The camera opens and the user scans the main page of the passport. OCR is used to read passport number, expiration date and date of birth and these are converted to the appropriate format for computing a BAC key.

Name:	Enter MRZ manually
Actor:	User
Goal:	Get passport number, expiration date and date of birth from the user.
Description:	The user picks the expiration date and date of birth through a date picker dialog and enters the passport number manually.

Name:	Change MRZ
Actor:	User
Goal:	To change expiration date, date of birth and passport number.
Description:	If one of these fields have been entered/scanned incorrectly the user can go back and change the data. The fields is then initialized to their previously entered values.

[7] Optical Character Recognition
[8] Active Authentication

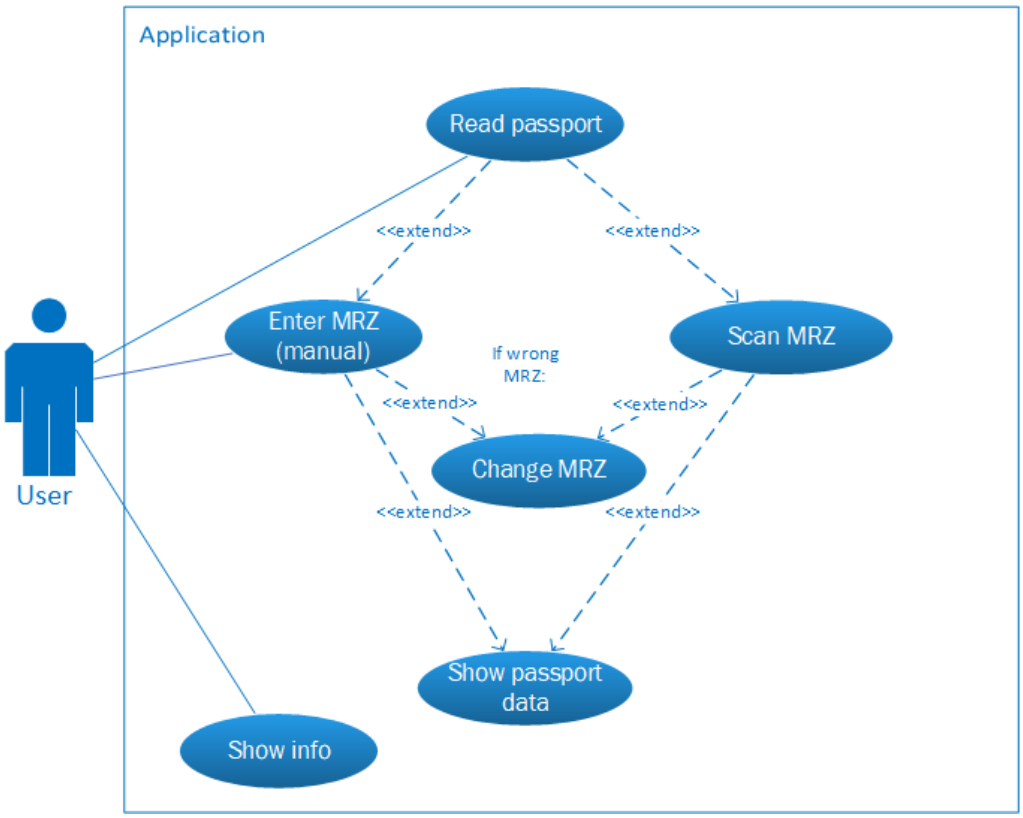


Figure 3: Use cases for reading a passport using NFC.

Name:	Read passport
Actor:	User
Goal:	Read user data and passport image through NFC
Description:	When the user verifies that the MRZ data is correct an animation demonstrates how to read the passport using NFC. When the NFC tag is discovered a BAC key is computed from the MRZ and this is used to unlock the passport chip. The NFC reader can then read data from the passport.

Name:	Show passport data
Actor:	User
Goal:	Display the passport data on the screen.
Description:	When the NFC reader has read the RFID chip the user is redirected to a new page where passport data and passport photo is displayed to the user.

4 Software Development Method

4.1 Scrumban

Our development project extends from February to April, and in a project perspective this could be considered a relatively small time frame.

We chose an agile development method. The main reason for this was that the project requirements were not completely clear from the beginning of the project and because agile methods often work well for small teams and relatively small projects. We considered waterfall, but due to the fact that our project depended on an API that was still being developed and we knew the project requirements were likely to change throughout the project, we decided using a more flexible development method would be beneficial. Additionally we are all inexperienced as developers, so we were quite sure we would encounter problems regarding estimation of time and resources. An agile method allows for more flexibility when planning and estimating each task, which is beneficial when we know our estimates might not be very precise. We will elaborate on this in section 4.1.1, Planning Poker.

The reasons discussed above led us to the choice of Scrum as a development method, which would give us the option to adapt to problems and changes that might occur. Agile development methods are normally used for smaller to medium sized projects with relative small sized teams, and where the client is prepared to participate [14]. After evaluating several agile development methods we came to the conclusion that Scrum was the best suited development method for our project. Scrum uses sprints as defined work periods that focus on different parts of the development process. We chose sprints of two weeks because this would give us time to have something to demonstrate during our meetings with Buypass, and it would also reduce the chance of spending an unnecessary amount of time in meetings.

As we saw our project, Scrum did not accommodate all aspects of our project. It would allow for an agile development, but it is restrictive when it comes to changing tasks within the sprints. We decided to combine it with Kanban, which is also an iterative process and shares similarities with Scrum. This made for an easy integration of Kanban. The perks of Kanban are that it offers greater visual solutions for the defined tasks of the project. Kanban also offers the possibility to change the tasks within the sprints, and in that way provided more flexibility [15].

We also incorporated a Minimum Viable Product approach to our development method. We wanted to focus on creating an early prototype that would give Buypass an impression of what we were developing so that we could receive feedback at an early stage. Between each meeting we would then add what we considered the most important features at that point and get new feedback at the next meeting. In this way we hoped to reduce the risk of not being able to fulfill the most important requirements and of having to do make too large changes based on the client's feedback.

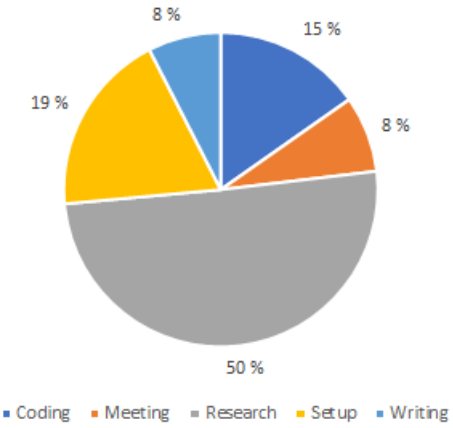


Figure 4: Pie chart representing the time spent in sprint 1

4.1.1 Planning Poker

We used Planning Poker for time estimation of our predefined tasks. This estimation was done in order to get an indication as to how complex our project would be, and what we could expect to get done. Due to the inexperience with development projects the suggestions presented by the different team members could vary from from one day to one week. We discussed the suggestions and came to an agreement. These estimations made for our Gantt diagram, see appendix D. The diagram includes the milestones of the project, as well as what should be conducted during the sprints.

4.1.2 Trello

To keep track of the work tasks during the project we have used Trello. Trello is an online tool which allows us to define our own Kanban board. We made a backlog board containing all the general functionality that needed to be implemented in the project.

The Kanban board included the six steps:

- Backlog
- Doing
- Blocked
- Testing
- Code Review
- Done

The cards containing the tasks were moved throughout the board as they were executed, and allowed for the other team members to see the progression.

4.1.3 Toggl

To keep track of time we have used a time tracking tool, Toggl. Toggl offered the possibility for every member of the group to add a description and a tag, represented by the sectors in figure 4, to each time entry. We predefined the tags, and then each member could elaborate more in the description field of the time entry.

4.1.4 Daily Scrum

We planned on conducting daily scrum meetings every day. The meetings would be structured in a way where each member briefly elaborates on what they did the day before, and what they would be doing the upcoming day. The meetings should help us see the bigger picture throughout the project, and it would be a way of making sure every member had done sufficient work since the last meeting and made sure that the members had a structured plan for what to do the upcoming day. Daily scrums would hopefully make the development less vulnerable to occurrences where one member of the group might be away for some time. These meetings could also offer reflection of problems and further implementations. The members of the group should be able to raise problems in plenum, and get feedback from the other members to help solve them.

4.1.5 Sprint Planning Meeting

Each sprint would start with a Sprint Planning Meeting. During this meeting we define the tasks that need to be performed in order to reach our sprint goal. These tasks would then be added to a new Trello board for the upcoming sprint, and we would loosely define the work tasks.

4.1.6 Sprint Review and Retrospective Meeting

We wanted to have meetings with our employer at the end of each sprint during the development phase of the project. During these meetings we would present the work we had conducted during the sprint, receive feedback and discuss the upcoming sprint.

Within the team we would have concluding Sprint Retrospective Meetings where we could discuss what was good about the finished sprint, what we could have done better and what we would commit to doing in the next sprint. These meetings were supposed to make us see what may have been slowing us down in the development process, and what we have done that we may improve, scrap or keep doing.

4.1.7 Meetings with Supervisors

Throughout the project we would have weekly meetings with our supervisors. In preparation for these meetings we would be sending an email containing information regarding what we had been working on since the previous meeting, if there were any problems or questions and what we would commit to do the upcoming week. These meetings would be good for assuring progression throughout the project, and making sure we were on the right track.

4.1.8 Milestones

Prior to the start first sprint, during the planning stages of the project we set up five milestones for the development work of the project. The milestones were originally set up regarding the sprint endpoints. The estimation of what we would be able to do was made on the background of little to no knowledge of how long the development processes would really be. These milestones were supposed to be reached in order to deliver the desired end result.

The five milestones were originally;

15th of February: The application has a simple GUI, and is able to take a photo which it can send to the server solution.

1st of March: The application has the functionality to send the photo taken to the third party

API for verification, and the user receives feedback regarding the verification.

15th of March: Functionality for NFC reading is implemented, and the info can be sent to the third party API for verification.

29th of March: Functionality for facial comparison and liveness check is implemented.

12th of April: Functionality and GUI completed.

4.2 Version Control

The naming convention for our version control during the development process was originally formatted as follows:

`Major.Minor.Patch`

Major represents a fully release of the application. In our project it has been given the value of one and wont change because the application will only be a prototype during development.

Minor represents the implementation of new functionality.

Patch represents smaller implementation, bug fixes and changes to already implemented functionality.

5 Design

5.1 System Design

The overall structure of the project was designed in a way to keep the mobile application and the backend separated, see figure 5. This made way for using a client-server approach. The server handles all communication with Keesing and Firebase, while the client application interacts with the user and sends requests to the server in order to pass on user information and retrieve verification results.

This model aims to minimize the possible leakage of unwanted information as well as relieve the frontend processing by having the backend handle it. The possible error codes, and feedback from the third party API would be translated and processed by our backend, preventing users with malicious intent to get a hold of information of which could lead to possible exploitation of vulnerabilities.

5.2 Backend

In order to accommodate the requirements of our backend we wanted to create a REST[1] API with JSON[2] formatted payload.

We wanted to be separate the functionality of different modules within the backend of the project, see figure 6. This would give us a better understanding and make it easier to comprehend errors occurring during both the development process and at a later possible release point.

The natural approach was to divide the modules where one would be handling verification towards the third party API, one would handle the webhook listening for triggers from

[1] Representational State Transfer
[2] JavaScript Object Notation

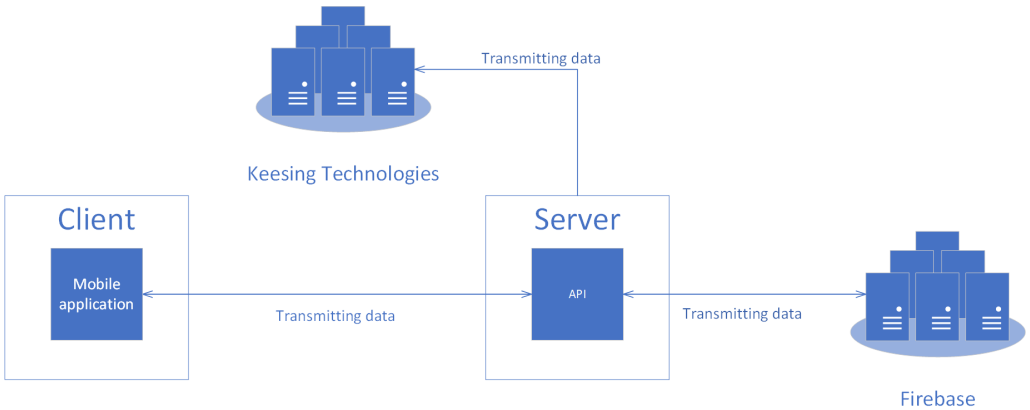


Figure 5: Overall architecture of the project

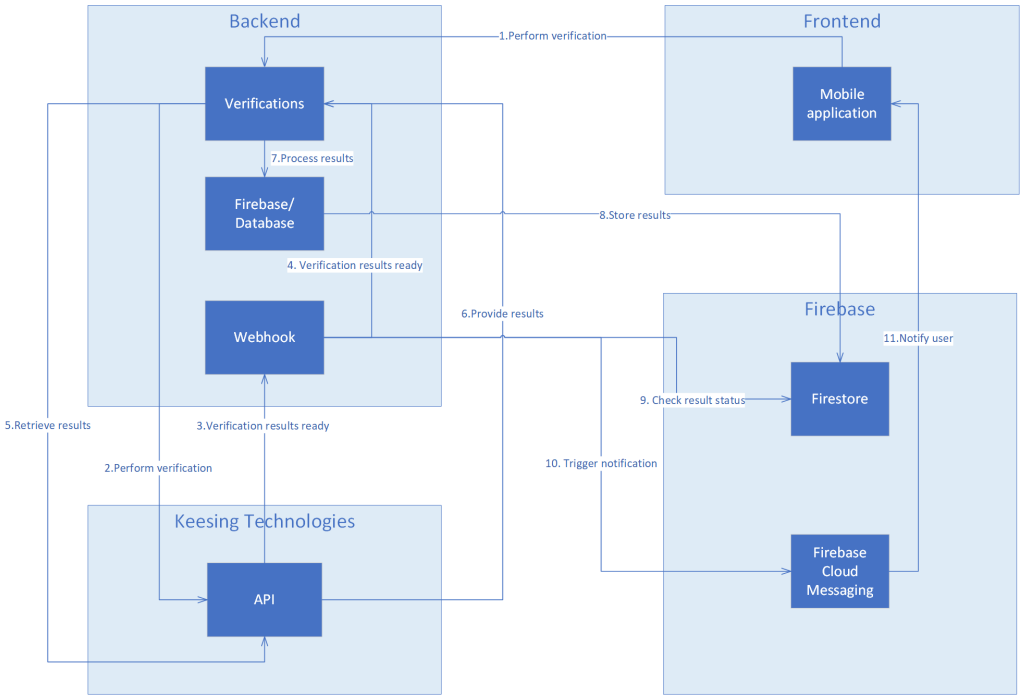


Figure 6: Backend modules communication flow with entities

the third party API and one module which would handle Firebase functionality and communication.

The verification module, as stated above, handles the verification aspect of the application. This involves requests from the mobile application which are processed by the backend module before being sent to the third party API, and the other way around. This module is critical in regards to performing verifications, and contains the main functionality of the application. This module involves everything in regards to Keesing Technologies except the webhook itself.

The webhook module is working as a mediator between the third party API and the other modules. The verification module is notified in order to retrieve the results, and the Firebase module handles database storage and push notifications.

5.3 Frontend Application

As stated previously, our project is mainly a prototype that is meant to explore ID proofing technologies and demonstrate this in a practical setting. The main focus during the development phase of the project in regards to the application has been to implement the necessary functionality.

The Android application part of our project was originally intended to be developed according to MVC[3]. However we ended up using a somewhat more simple approach which is illustrated in figure 7.

The application is divided into different *Activities*, each represented by classes extending

[3]Model View Controller

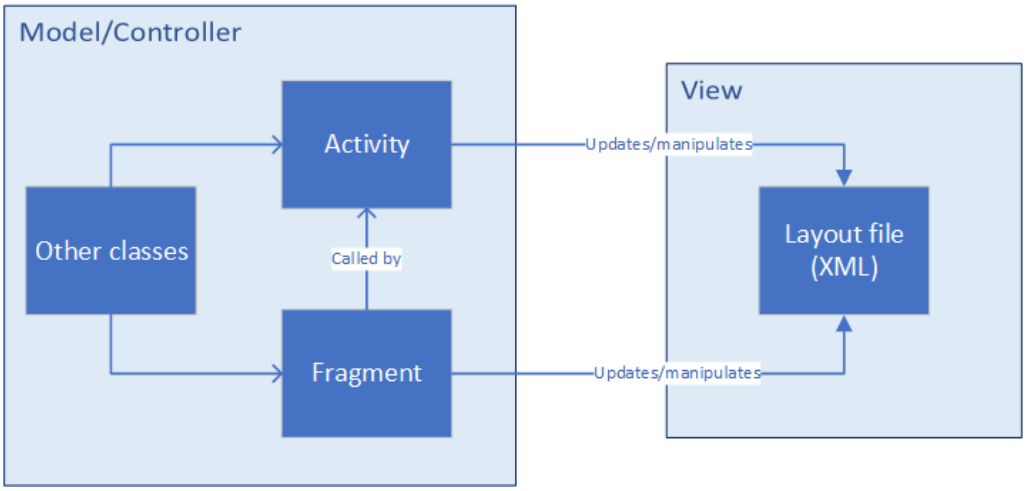


Figure 7: Frontend Architecture

the Android Activity class. Some activity classes also communicate with other helper classes/interfaces and/or fragments. Each activity has one or more layout XML[4] files associated with it. Fragments are also associated with layout files. The layout files contain all view elements and popups used in different activities/fragments. As the figure demonstrates, the Activity classes (and their components) act as a combination of Model and Controller, while the layout files represent the View.

An important principle behind this design is *Separation of Concerns*. We identified what we considered to be the main user activities and created an Activity class with corresponding layout files for each one of these. As most of the actual logic behind our application is handled on the server side, the Activity classes should be quite lightweight and mainly handle interaction with the user, managing view elements and communicating with the server.

Another important principle is avoiding duplication of code. If two user interactions are relatively similar, they should preferably be handled in the same activity. If the same methods are used in multiple modules, it should be written in a helper class that is available to all of these modules.

In general it is recommended that app components depend on each other [16]. Given the natural linear flow between activities in our app we do not really follow this guideline. In order to streamline the verification process and make sure the user completes every step of the process, most of the activities are launched in a pre-defined order (with a few exceptions where the user needs to make a decision, for example between using a passport or a document photo). For example, the user needs to complete the document check step before moving to the face comparison step, and face comparison must be completed before liveness check and so on. For this reason many of our activities depend on the previous activity to be launched and may both receive information and pass information to the previous/next activity. Since most activities can only be launched in the pre-defined order, this is probably not likely to cause any significant problems in our app. Quite on the contrary we avoid some big problems by designing our app this way. For example, if something goes wrong (bad internet

[4]Extensible Markup Language

connection, server is down or similar) when the user tries initializing a verification process, he should get an error message and not be able to try uploading photos before the verification process has been successfully initialized. Otherwise the application would just keep trying to upload photos to a non existing verification process, which would result in new errors. Activities that can be launched in an ”out of order“ fashion will be treated somewhat differently.

5.4 GUI and UI Design

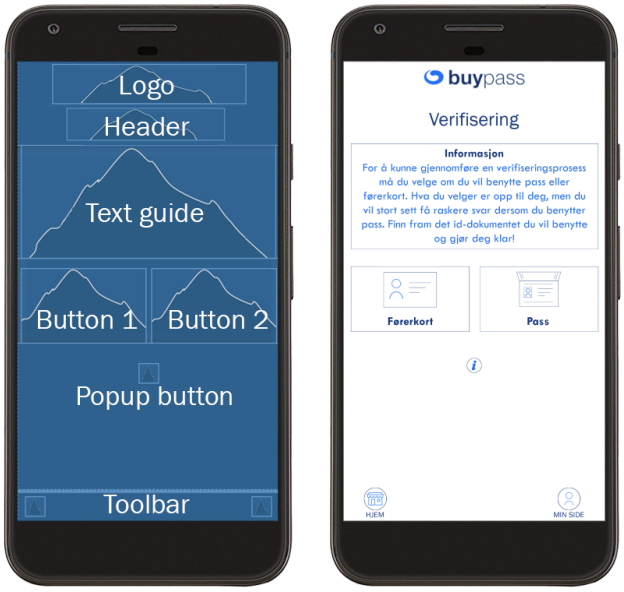


Figure 8: Blueprint of the main template

In order to accommodate the requirements described in section 3.4.1 we have interpreted the design principles and implemented the necessities we felt needed in order to reach the desired end result. This has led to the following design implementations:

- Simplicity
- Clarity
- Consistency
- Familiarity
- Efficiency and responsiveness
- Style

5.4.1 Simplicity

The main theme of the application is shown in figure 8. The approach is to keep the amount of buttons, titles and text to a minimum, but keep the functionality well documented and intuitive. This is accommodated by the implemented design of the application, see section 5.4.6.

The toolbar is implemented on the bottom of the screen to keep its influence with the main functionality presented in the middle of the screen to a minimum.

5.4.2 Clarity

We wanted to provide the user with sufficient information in each step of the verification process, but we did not want to clutter up the screen in regards to the prior requirement. The blueprint of the application offers a information box, but it also provides a information button, see figure 8 (Popup button). This button is rather self explanatory, and displays a popup message providing extended information to the user of the active step in the verification process.

5.4.3 Consistency

Within the application the theme and style stays the same throughout the entire application, both in regards to styles of buttons, colors and how the interaction with the application is.

5.4.4 Familiarity

The application design has its inspiration from the Buypass website which includes the same color palette and the same button designs. In that way a user would be able to tell that this is Buypass' application. The same goes for the interaction with the application, and if the user is a common user to mobile applications he should be able to navigate the application with ease. The buttons, navigation bars, popups and so on should be something that the users have experienced before.

5.4.5 Efficiency and Responsiveness

In order to extend the application to the user, leaving him feeling involved and not left waiting for a long time, we wanted to implement loading screens, animations and messages underway that let's the user know that the process is ongoing and that the application is responding. We are also focusing on the performance of the frontend, in the way of having buttons which are responsive, and data that is already loaded when a user requests it. This is mostly in regards to the user page of the application, where the user is not supposed to experience any delay whether there are many verifications or nothing at all.

5.4.6 Style

The color palette and style choice of the application is made due to inspiration from Buypass' website. Both the colors, figures and buttons are inspired or retrieved from there, see appendix H.

5.5 Camera Design

In order to meet the camera requirements we have defined a common design pattern for all the camera activities. Each camera activity starts with a screen that explains what kind of picture we need, what it will be used for and how to take a valid picture. This screen contains a camera button that opens the camera. The actual camera screen is designed in a way that makes it intuitive for the user to interact correctly with the app. For example: the camera button is placed in the middle of the lower part of the screen (like in most photo applications). The button for proceeding is placed to the right and the re-take button is places to the left, since most people associate buttons on the left side with going back and buttons on the right side with moving to the next screen. The camera design is illustrated in figure 9. The camera screen contains the following elements:

(a) (b) (c)

Figure 9: Camera GUI design

Instruction text: On top of the screen we show the user some short instructions on what to keep in mind when taking the photo/video. This will be specific to the type of image/video that is required (document, face etc). The text disappears when the user clicks on it and reappears if the user clicks the info button.

Information button: If the instruction text is closed an information button appears in the upper right corner, so that the user can re-open the text box.

Camera button: Button for capturing a photo or starting a video recording. Placed at the bottom of the screen.

Countdown timer: Only when recording video. Counts down from 2 seconds every tenth millisecond so that the user knows when the video stops.

Re-take button: Button for re-taking a picture/video. This button is hidden until after an image or a video has been captured. Placed in the lower left corner of the screen.

Upload button: Button for uploading the photo and proceeding to the next step in the process. This button is also hidden until after a photo has been captured. Placed in the lower right corner of the screen.

Preview: When an image is captured it is shown on the screen. When a video has been recorded, a mediaplayer lets the user play the video before submitting it.

Document border: For document photos a rectangle shows where to place the document.

Note that there is no stop button for the video recorder, as the video is automatically stopped when it reaches the desired length. In order to let the user know when the recording the recording ends, a red dot and a timer counts down from 2 seconds. The idea behind the

design is to make it difficult for the user to do things wrong. For example: the upload button is hidden until an image is captured so that it is impossible for the user to proceed before he has taken a picture and the camera is always opened in the direction it is supposed to be used. The reason why the buttons are hidden and not just disabled, is that the screen looks more clean and tidy with less buttons showing at the same time.

As mentioned in the Requirements chapter (section 3.4.2.1) Keesing requires all document photos to show the document in the correct orientation and to show the entire document inside the image. If this requirement is not accommodated for, it will send the photos to helpdesk. This will significantly increase the time it takes to receive the result.

A potential problem could be that it is difficult for us to know how each user would hold the document and the device when taking the photo. Some might hold the phone in portrait mode and rotate the document 90 degrees in order for it to fit the screen, some might hold the camera device in landscape mode and hold the document without rotating it and so on. This would make it difficult for us to know whether we would need to rotate the image before sending it to Keesing or not (and not to mention which way to rotate it).

To overcome this issue we decided that all document photos should be taken with the phone in portrait mode. We created a small rectangle showing the user where to place the document and we placed the instruction text on top of the screen, so that the user is prompted to hold the phone correctly. The area around the rectangle is blurred to make it easier for the user to focus on the document and not background noise. When a photo has been captured only the area inside the rectangle is displayed to the user. Since this area is significantly smaller than the uploaded photo, the idea is that even if a small part of the document were to be outside the guiding rectangle, it will still be in the uploaded photo. Making the user believe that the photo is smaller than it actually is will hopefully make him check the photo more closely before uploading it, so the risk of uploading an insufficient photo will be reduced.

5.6 NFC Design

The NFC part of our application consists of four main steps:

1. Get the expiry date, date of birth and document number (MRZ info).
2. Check that the entered information is correct.
3. Read the passport using NFC.
4. Show the data that is read from the passport.

Figure 10 shows an overview of the flow between the different activities in the NFC application. NFCStartActivity is launched when the user chooses to read his passport using NFC. The user gets the choice between completing step 1 by entering this information manually (ManualNFCActivity) or scanning it using the phone camera (ScanNfcActivity). Once the MRZ info is entered, the user is sent to ReadNFCActivity where the step 3 is conducted. If the user discovers that the registered MRZ is wrong he has the option of changing this by being redirected to ManualNfcActivity. Step 3 consist of unlocking the passport by doing BAC, reading the passport using NFC and parsing the data read from the passport. When the passport has been read, the NFCResult is launched and the passport information is displayed on the screen.

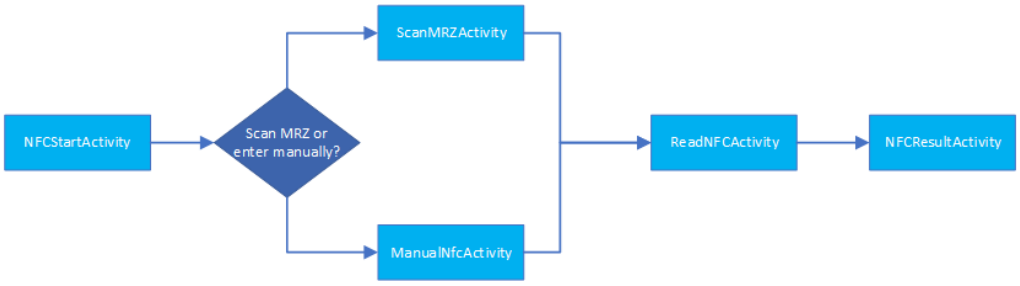


Figure 10: Overview of the activities in the NFC part of the application

5.7 Database

We considered both SQL and NoSQL options for our database. As mentioned, the purpose of our database is to store verification results and facilitate communication between our API, Keesing's API and the mobile application by being able to respond to the queries described in section 3.3.1.

After some consideration we chose to go for Google Firestore, which is a NoSQL database. We chose this partly because the Keesing API is still in Beta. Already in the first few weeks of the project we experienced quite a few changes on their side and we expect more changes may appear in the future (especially when NFC verification is implemented). Given that we might have to change the database schema multiple times throughout the lifetime of the project, the flexibility of a NoSQL database suits us well. In addition, all the results received from Keesing come in the form of a JSON body which is exactly what a NoSQL database stores. A potential limitation of a NoSQL database could be that complex queries are difficult to implement, but as all our queries are relatively simple, we do not consider this to be a problem.

5.7.1 Google Firestore

After deciding on a NoSQL database we decided to use Google Firestore as our database provider. Firestore is a document database that stores data in hierarchical structures.

Firestore provides several advantages. For one thing it is simple to use. Firestore has a graphical interface in the Google Firebase console which makes it easy to create, view and manipulate collections manually. This makes it easy for us to create test cases and to quickly verify that things are stored correctly. The Firestore JavaScript API provides methods to easily connect to the database and update the database from our server code. Another advantage is that Firebase queries are structured in a way that makes it possible to retrieve data at document level, without having to retrieve the entire collection, which gives us effective queries [17]. This is important for example when retrieving a verification instance for a specific user. Firebase also provides convenient mechanisms for database security through Firestore Security Rules [18].

5.7.2 Document Database Design

We decided to create a database consisting of two main collections. The main collection is the *Users* collection. Since we have chosen a document-oriented database the data is organized in documents and collections (see figure 11). *Users* consists of documents identified

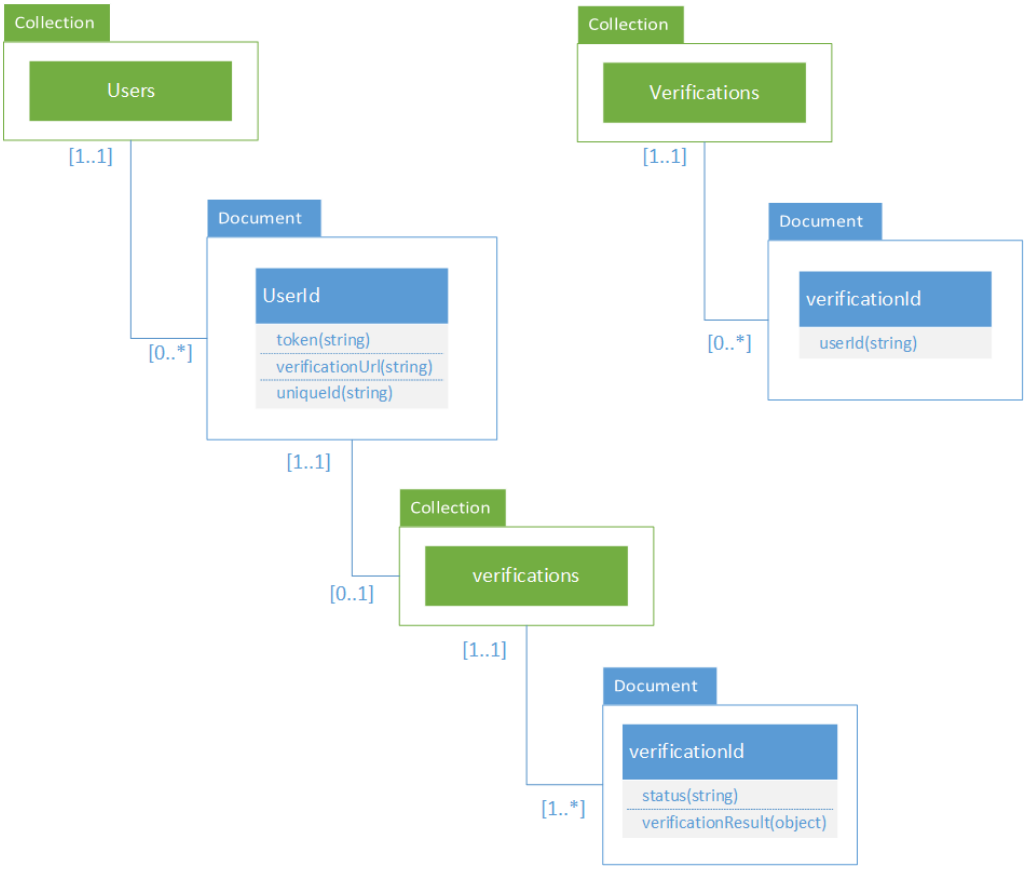


Figure 11: The *Users* and *Verifications* collection and its documents and sub collections

by a user Id. If a user has started a verification process, the user document contains a sub collection *Verifications*, which in turn contains one or more verification documents identified by a verification ID.

Only documents can contain data fields. Each user document has three data fields, a token used for Firebase Cloud Messages (push notifications), a *verificationUrl*, which contains the URL used to update an ongoing verification process and a unique ID to verify the user.

Each verification instance contains a status field which is ”active“ if the verification process has not been completed and ”finished“ if the process is completed. If the user has completed all three steps of the process it also contains a verificationResults object. This object contains fields that describe the outcome of the document check, face comparison and liveness check (see section 3.3.1).

The Users collection is used to answer all queries for updating or checking a client's verification status and retrieve the FCM [5] token for push notifications (query 1-3 in 3.3.1).

The last query, being able to find which username belongs to a certain verification ID, is a bit different from the other queries. We could form a query where we loop through all users in the collection and return the one that has a verification ID that matches the one we want, but this would be quite inefficient if the user collection is large. Instead we chose to create

[5] Firebase Cloud Messaging

a separate collection, *Verifications* (figure 11), which stores verification documents with just one data field, the *userId* of belonging to that verification.

6 Implementation

6.1 System Overview

The application starts with a mock login screen that collects a username. As described earlier the user can choose between verifying his identity using a driver's license or a passport. The process is identical, except that if he chooses driver's license the application asks for a photo of the front and the back of the ID document.

The actual verification process involves five different entities, the user, the client application, the server, the database and Keesing. The first sequence diagram (Figure 12) shows the typical interaction between these entities. The diagram assumes that the user is already logged in and has chosen passport as the preferred ID. Once the user has chosen ID type the application sends a request to Keesing (via the server) to create a new verification instance. The verification ID is then stored in the database, and the user is sent through three verification steps: document photo, face photo and liveness check. Each file is individually uploaded to the server and then Keesing. If the file does not fulfill Keesing's requirements for size or sharpness an error message is returned and the user is prompted to re-take the photo/video. When the user has completed all three steps the application informs the server that all the required files have been uploaded and the server can then send a request to Keesing to start verifying the uploaded files.

The second sequence diagram (figure 13) shows how a verification result is registered. When Keesing has finished one of the verification steps they trigger a webhook on our server with the verification ID and which step was completed. The server will then send a request to Keesing in order to retrieve the verification results. When Keesing has finished all three verification steps (i.e. status = finished) Firebase Cloud Messaging is used to send a push notification to the user to inform that the result is ready. As Keesing only includes the verification ID, we have to look up the user ID in the *Verifications* collection in order to send the notification.

At any point the user can request to see verification results. The results are then fetched from the database and the outcome of each step is displayed on the screen. For a more detailed illustration of the data flow, please refer to the Data Flow Diagram in appendix F. The rest of the chapter will give a more in-depth description of how the frontend and backend solutions have been implemented.

6.2 Backend

For our backend solution we needed something that could fulfill the requirements and design principles described earlier. We needed a technology that had the ability to handle requests from both the mobile application and the third party API, communicate with Firebase, and work as a mediator between all of these. We needed a framework that was well documented and would make it easier to create an API. We chose Node.js' framework Express.js. This is a lightweight framework that offers methods for implementation of robust REST API's. This solution would to a great extent provide all the functionality we described in the requirements

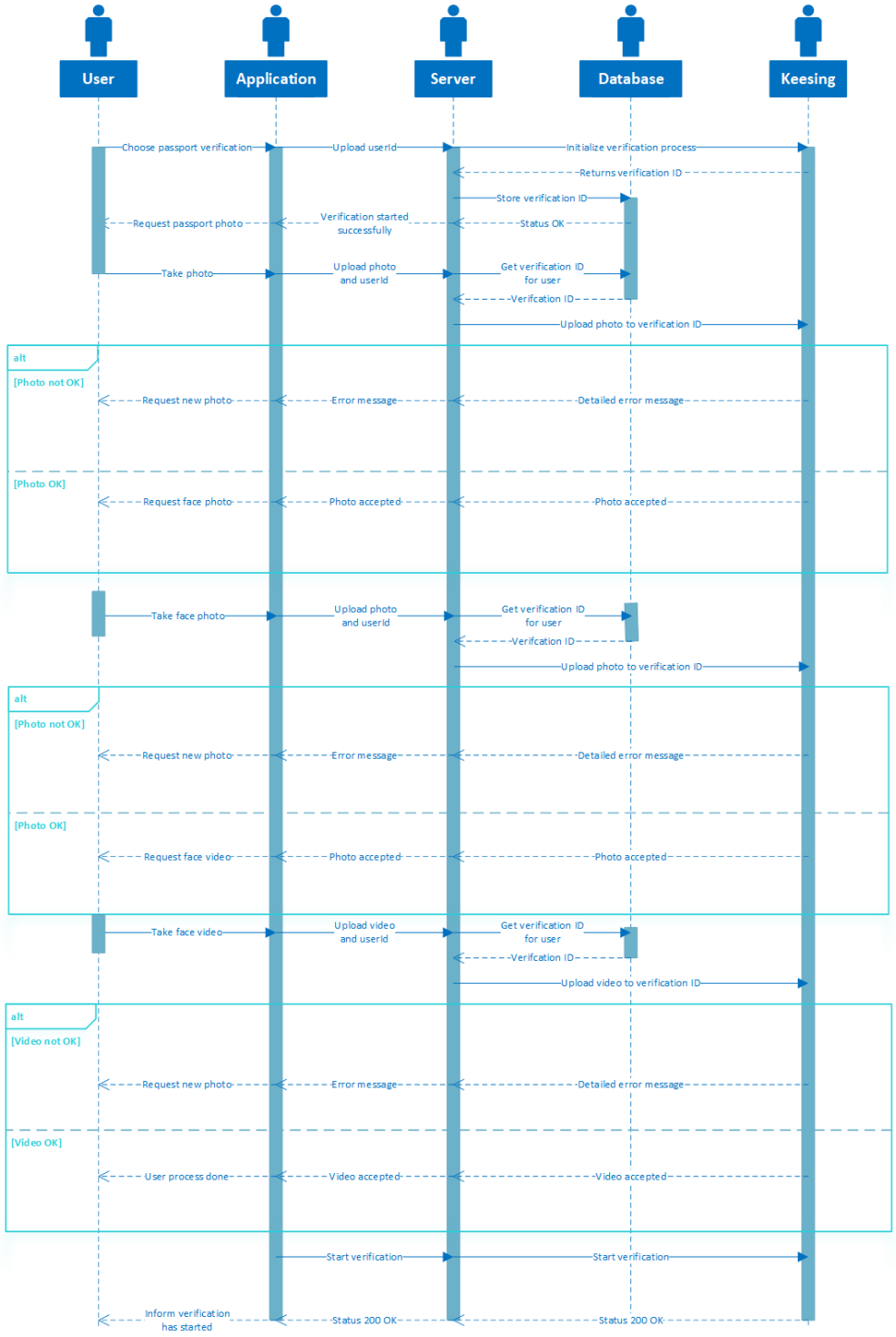


Figure 12: Sequence diagram showing a typical passport verification process

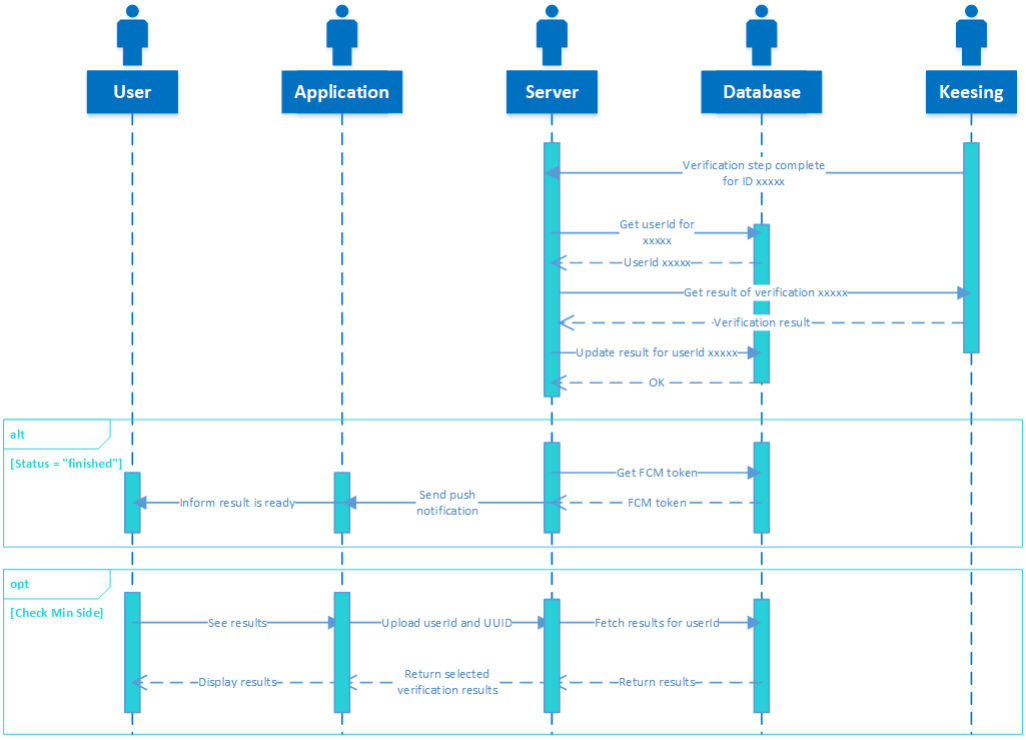


Figure 13: Sequence diagram showing how verification results are updated and queried.

chapter, section 5.2.

Express allows for easy creation of API's as stated above, and this goes to show in the listing 6.1 which is a replication of our server file. This file handles all the routing and the set up of the API in general.

```
const express = require("express");
const server = express();
const rateLimit = require("express-rate-limit");

const port = 8080;
server.use(express.json());

// Environment variables
require("dotenv").config();

/**
 * Rate limit for requests to API
 * 30 requests per minute
 */
server.enable("trust proxy");
const limiter = rateLimit({
  windowMs: 60 * 1000,
  max: 30
});
server.use(limiter);

// Routes
const verifications = require("./routes/verifications/");
const webhook = require("./routes/webHook");
const firebase = require("./routes/firebase");

server.use("/verifications", verifications);
server.use("/webhook", webhook);
server.use("/firebase", firebase);

server.get("/", (req, res) => {
    res.send("API for bachelor thesis, NTNU Gjoevik");
});

server.listen(port, () => console.log(`Server listening on port ${port}!`));
```

Listing 6.1: Server file, server.js

6.2.1 API

The API calls and methods are described in figure 14, and further elaborated in appendix K. All of the API calls are processed through our domain; `https://www.verifyid.ml`. The API is handling all the requests from the mobile application, translating and processing them before making requests to the third party API.

JavaScript executes commands concurrently, meaning that the compiler will not wait for a called function to complete before trying to execute the next command. As a consequence of this we needed to make sure the code was executed in the right order, for example that the function retrieving a token for the third party API would run before the functionality for uploading a picture. We accomplished the desired functionality by using Promises [19]. A promise in JavaScript will allow for locking down a process until it is either fulfilled or rejected. In practice this would mean that either a token or an error would be generated before the process would be able to proceed. The ”getToken“ function in listing 6.2 shows an example of how promises have been implemented. On line nine it starts by declaring that it will return a Promise, and then on line 28 it checks whether or not the token is undefined. If the token turns out to be undefined it will return an error saying it could not retrieve a token, but if the token is defined the promise will return the token for use in other functions.

URL	Description	Method	Parameter(s)
Verifyid.ml/		GET	
/verifications/	Call for instantiating a verification process	POST	refNo - string, unique user reference)
/verification/fileupload	Call for uploading image/video file	POST	FORM file – file upload type – string, type of upload userId – string, unique user reference
/verifications/startverification	Call for starting a verification process	POST	userId – string, unique user reference
/verifications/getpdfreport	Not functioning	POST	userId – string, unique user reference
/firebase/uploaduser	Call for storing user and token in database	POST	user – string, unique user reference token – FCM token UUID – UUID for identifying in regards to retrieval of verification data
/firebase/getallverifications	Call for retrieving all verifications	POST	userId – string, unique user reference
/webhook/	Call for triggering the webhook	POST	event – string, “verification.finished” / “verification.report.created” / “verification.liveness.finished” / “verification.facematch.finished”

Figure 14: API calls, methods and parameters

```
const request = require("request");

/**
 * Function for retrieving a token to be used in further processes regarding the
    Keesing API
 * @return token, which is used to do more processes.
 */
module.exports = {
  getToken: function() {
    return new Promise(function(fulfill, reject) {
      request.post(
        {
          url: "URL",
          form: {
            grant_type: process.env.GRANT_TYPE,
            scope: process.env.SCOPE,
            client_id: process.env.CLIENT_ID,
            client_secret: process.env.CLIENT_SECRET
          }
        },
        (error, response, body) => {
          console.log("TOKEN");
          console.log("StatusCode:", response.statusCode);
          console.log("StatusMessage:", response.statusMessage);
          //console.log("Headers:", JSON.stringify(response.headers));
          //console.log("Body:", body);
          let jsonToken = JSON.parse(body);
          let token = jsonToken.access_token;
          if (token === undefined) {
            reject("Could not retrieve token. ");
          } else {
            fulfill(token);
          }
        }
      );
    });
  }
};
```

Listing 6.2: Node.js function getToken()

6.2.2 Communication with Keesing

In order to communicate with Keesing we need to have a valid access token. This is retrieved with the getToken() function shown in listing 6.2. This token must be included in all requests to Keesing. From Keesing's perspective the verification process consists of three steps:

1. Initialize verification
2. Upload files
3. Start verification

The first step is handled in *.../verifications* and involves sending a request to Keesing to create a verification instance and then storing the verification ID in the database.

The second step involves uploading photos and videos that are posted from the application to *.../verifications/fileupload* to Keesing. The application uploads the files along with a user ID only, so before passing the file to Keesing the verification ID must be looked up in the database and included in the request.

The last step lets Keesing know that the necessary files have been uploaded and that they can start processing the uploaded data. This happens when the application calls *.../verifications/startverification* after successfully going through the document check, face comparison

and liveness check.

6.2.3 Webhook

As the time from when a verification process is started until the results are ready can be anywhere between a few seconds and several days, we have implemented a webhook which can be triggered when a result is ready. The route created for the webhook is `https://verifyid.ml/webhook`. The webhook is triggered every time a verification step is completed (i.e. once for document check, once for face match and once for liveness check)[1]. The webhook has predefined cases for what events it may receive from the third party API. If it receives something unexpected it will respond with error code 400, bad request.

The example below shows an example of webhook a request body from Keesing.

```
{
    "event": "verification.facematch.finished",
    "data": {
        "id": "xxxxxxxxxxxxxxxxxxx",
        "verification_url": "/verifications/xxxxxxxxxxxxxxxxxxx"
    }
}
```

As you can see it does not contain the actual results, so these are retrieved through a POST request to Keesing using the verification ID (handled in verificationDone.js). The results are then stored in the database. As all steps might not be completed at the same time, each verification contains a status field that we update from “active” to “finished” as soon as all checks have been conducted.

6.2.4 Push Notifications

When the status field has been set to “finished” we need to inform the user that the verification results are ready by sending them a push notification. To implement this we use Firebase Cloud Messaging (FCM), which was quite straightforward to implement given that we had already created a Firebase project in order to use the Firestore database.

FCM uses a token to identify which device to send the notification to. This token is collected from the client application and stored in the database every time the user logs in, so that in case the user were to switch phones, it will be sent to the last device the user logged into.

The push notification is sent to via the Firebase Admin SDK which handles the transmission of it. How push notifications are handled within the client application will be described later in this chapter (section 6.3.6).

6.3 Frontend

Our task is to build a native Android application and therefore we wanted to use Android Studio as it was well documented and it is the official integrated development environment (IDE) for Google’s Android operating system (see section 2). We use so called *activity* classes for the application logic and user interaction, and *layout* files to define the view elements.

[1] The webhook is also triggered when a PDF report of the results is ready, but since the PDF report cannot be retrieved from the current version of Keesing’s API, we do not handle this event apart from logging that the webhook was triggered.

6.3.1 Application Structure

As previously described, the Android application is primarily built from Activity classes and layout files, with some supporting fragments and classes. The most important activities are described below:

Camera2Activity: Activity that launches a custom made camera to take document or face photos.

LicenseCameraActivity: Starts up the photo verification process if the user chooses driver's license as ID, asks for front and back photo of the document. If the verification process is started successfully the user can launch the camera.

LoginActivity: Mock login screen for entering username and password. Uploads username and FCM token to the server.

ManualNfcActivity: Activity for manually entering MRZ data before reading passport NFC tags.

NFCResultActivity: Displays information that is read from a passport NFC tag.

NFCStartActivity: Explains the process for reading passport tags and lets the user choose between entering MRZ data manually using OCR.

PassportActivity: Starts up photo verification of passport. Asks for document photo. If a verification is successfully started the user is sent to the camera.

PassportOrLicenseActivity: Explains the difference between using passport or license as the ID of choice and lets the user decide.

ReadNFCActivity: Reads a passport RFID chip using NFC.

ScanMRZActivity: Reads expiry date, document number and date of birth from the MRZ using OCR.

SelfieActivity: Explains how to take a valid face photo.

UploadActivity: Uploads images/videos to our server. Uses the **ServiceGenerator** class and the **FileUploadService** interface in order to upload the file. Displays an error message if something goes wrong. See appendix I for more information on the implementation.

VerificationCompleteActivity: Sends a request to the server to inform that all files have been uploading and the verification can be processed by Keesing.

VideoActivity: Information about the liveness check.

VideoRecordActivity: Starts up the video recording application by using the two fragments **CameraSetUpVideoFragment** and **CameraRecordVideoFragment** and the **AutoFitTextureView** class. See section 6.3.7.

WelcomeActivity: The first activity that is launched when the app opens. Uses a slider to explain the full verification process. See section 6.3.3.

In addition to the activities described above we use a few other classes/fragments to support the activities:

MinSideFragment: Displays the results of all verification processes a user has completed.

MyFirebaseMessagingService: Class that manages how push notifications should be displayed and handled.

GlobalClass: Stores the user ID of a logged in user. Contains various helper methods used throughout the application.

HomeButtonFragment: Handles what happens if the user tries to go to the home page while inside a verification process 6.3.2.

SectionStatePageAdapter: Manages the state of MinSideFragment (see section 6.3.4).

ServiceGenerator and FileUploadService Used in order to upload multipart form data in UploadActivity.

As described in the Design chapter most activities are launched sequentially in a predefined order. This is illustrated in figure 15. As you can see, there is a natural flow from WelcomeActivity to VerificationCompleteActivity, although there may be different path depending on the user's choice of ID. As the figure shows, activities like UploadActivity and Camera2Activity are launched from several different activities. This will be further described later in this chapter. MinSideFragment can be called from almost anywhere in the application, so this is is modelled outside the typical activity flow, together with other classes/fragments that are used in many different activities. The implementation of MinSideFragment will be discussed later in this chapter.

An overview of the different application screens is provided through screenshots of the verification process in appendix T.

6.3.2 Efficiency and Responsiveness

- **In-between verifications:** In all steps of the verification process there are parts in-between them where a loading screen will appear in each upload. The screen is there for displaying a loading bar and at the same time letting the user know he has to wait a short amount of time before proceeding to the next verification step. If the upload is unsuccessful it will notify him to retake/recapture the photo/video. If not, the user always has the opportunity to exit the whole verification process, as mention in section 6.3.4 when pressing the go back -or home button. The application is responsive and always lets the user know what happens in that particular activity. When approaching the end of the verification we inform the user when the process is almost at its end. At the end we also inform the user by congratulating him for completing the verification process.
- **Verification result:** While running the app, the list of verification result(s) is updated repeatedly in the background. This gives the user the most updated result every time he accesses the user page. When inside the user page it does not do background updates instead we give the user the opportunity to refresh the list manually by swiping down.
- **Activity stack:** The way we have structured the app, we go from one activity to another and send information to Keesing along the verification steps. If the user uploads multiple document/face photos during a single verification process, he will get an error, so the user is not allowed to go back without interrupting the current process. To avoid filling up the activity stack if the user wants to go back and start a new process,

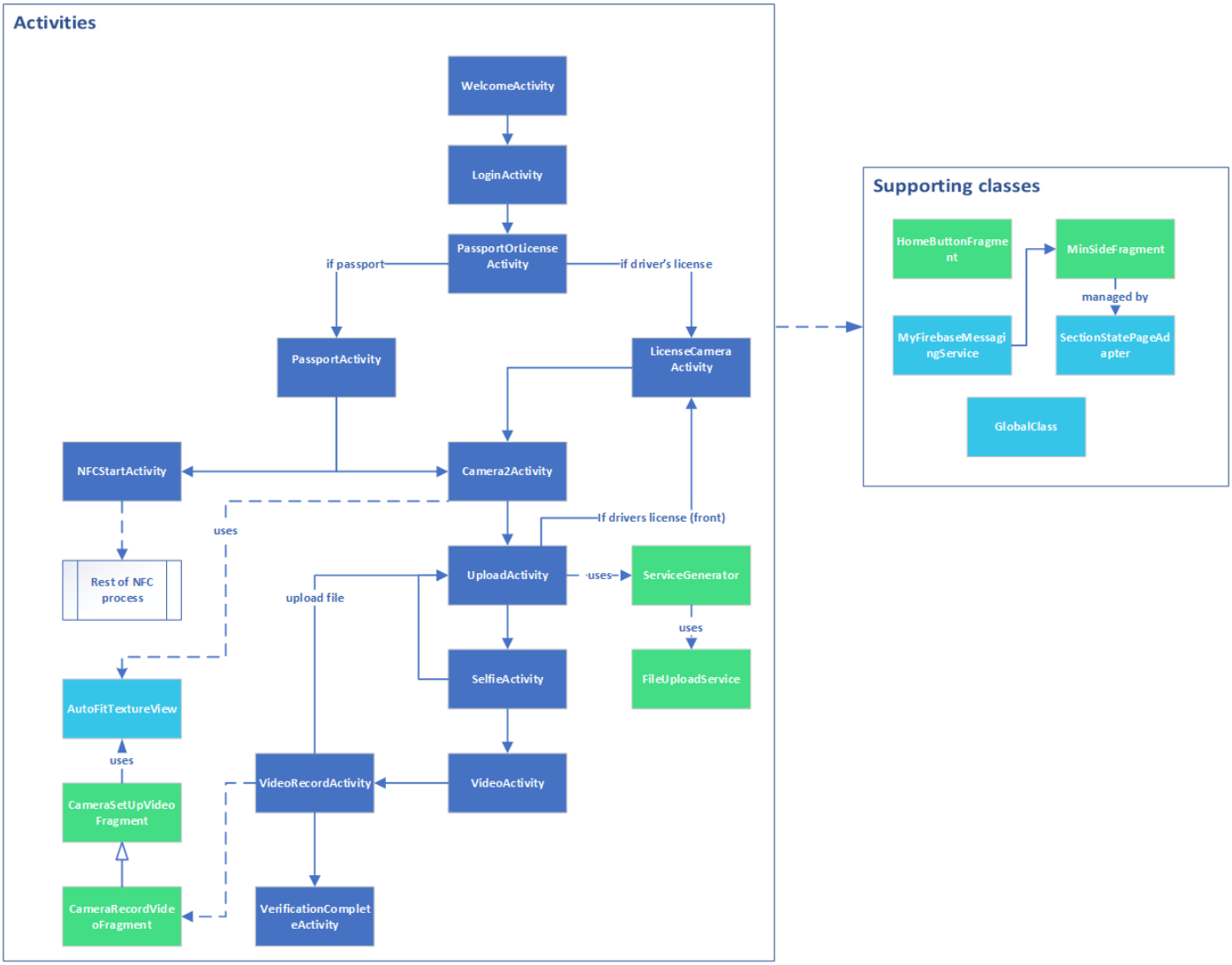


Figure 15: Overview of the transition between activities and their supporting classes. The activity transitions are represented by full arrows, while dashed arrows indicates that a class uses another class. The classes labeled *Supporting classes* are classes that are connected to multiple classes. The NFC activities are not included as these are shown in figure 10

.

we have added a "noHistory" tag to some activities, meaning that when you leave an activity it will be removed from the stack. When you press the home -or back button all the other activities on the stack are removed. This is done in order to unload the memory and keep the application running smoothly. If at any point the user wants to go back while a verification is ongoing he gets a clear message that this will interrupt the process and he would have to start a new verification process.

6.3.3 Slider and Popups

Figure 16: A selection of the introduction slider for the application

We chose to implement a introduction slider as the user starts the application, see figure 16. This slider acts as a tutorial for the user, describing the steps of the verification process in general. The purpose of the tutorial is to makes sure that the user is prepared for the necessary steps of the verification process, and it provides a pointer as to what is necessary in order to perform the verification, for example prepare the passport.

In every step of the verification process there is an ”i“ icon provided, and this represents an information button. The user can press the button and a popup message will appear providing info about the current step of the process, see figure 17. In some steps the popups are shown automatically to ensure that the end users receive important information.

6.3.4 Toolbar

The toolbar was implemented to make it more simple for the user by always providing a default menu at the bottom of the screen. We used Androids own library, ”Toolbar“, for the implementation. The toolbar is a stand alone layout file and is included within every activity. The two buttons on the toolbar offers the functionality of navigating back to the main screen of the application, and to show the user page containing the verification results. These two pages are controlled by fragments. Fragments have it’s own lifecycle within an activity, and this implementation allows for multiple fragments in a single activity. For handling the setup

Figure 17: A selection of popups

of these fragments we use a pagerAdapter[2]. In the layout we have a ViewPager that provides the different views. The pagerAdapter handles the switch between fragments on the ViewPager. Since this has to be setup in every activity, we have tried to reduce the amount of duplicated code by making the functions global.

6.3.5 User Page

(a) (b)

Figure 18: User Page and Verification Report

[2]FragmentStatePagerAdapter

The user page provides information about all the verification processes a user has performed, see figure 18 (a). The list is sorted by the date and time the process was completed. We have not implemented any way for the user to sort this in any other way because we are under the impression that the latest attempt of a verification process is the one that is the most relevant to the user. If the last attempt of a verification failed, then the user may be interested in understanding what went wrong, and to not the repeat the same mistake in the next attempt if the user is to blame for the decline.

The list of verifications is color coded according to the verification outcome. Red indicates that the verification was declined, yellow indicates that the verification is pending and green indicates an approved verification.

The verification process involves several steps, and therefore it may be interesting for the user to know which step of the process that failed. If the user wants more information regarding the verification, he or she may simply press the one verification from the list they want to see more about, and they will then get a popup message containing information about the verification, see figure 18 (b). The colors of the verification and the status is set depending on the status of the different steps within the verification process. If the ID document, the face match and the liveness check is approved then the verification is approved and the color is set to green. If one of the steps mentioned above fails or is declined then the color will be set to red because the verification process requires all the steps to be approved. Sometimes there may take some time for a verification to be done, whether it is a unknown ID document or the facial comparison is done manually. If that is the case for one of the steps or more, and none of the other steps that has been performed did fail, then the color of the will be set to yellow, representing a pending verification.

In order to inform the user of the declined steps, we are providing the user with some info regarding the error messages we receive from Keesing's API. We are processing the error messages, and also providing a default message if there is an unknown error introduced. The error messages does provide errors such as if there was no face detected or the video of the liveness check was blurry. This provides the user with information for consideration when trying to complete a verification process later on. It is important to not give the user too much information regarding what went wrong, because of potential users with malicious intent wanting to infiltrate or break the system.

6.3.6 Push Notifications

Push notifications are received and handled in the *MyFirebaseMessagingService* class. Here a "NotificationBuilder" is used to set the header, message and icon in the notification and to decide what happens if the user clicks on it. What happens when the user clicks on the notification depends on whether the user is within the application or not. If the user is within the application he is redirected directly to the user page, and if the application is closed, the user is sent to the log in screen of the application.

6.3.7 Camera2 Implementation

In order to implement our camera design we used the Android Camera2 API, which gives you more or less complete control of the camera. As mentioned, our goal was to customize the camera in a way that made it intuitive and easy for the user to take the photos/videos correctly and to meet the specific requirements for the camera functionality. To see how the photo and video application ended up looking, please see the screenshots in appendix T.

To get an overview of how the API worked we got inspiration from the Android Developer documentation [20] and several tutorias, especially Mateusz Dzuibek's tutorial *The least you can do with Camera2* [21] and AndroidWave's *Video Recording with Camera2 API in Android* [22].

6.3.7.1 Photo Application

The photo application is implemented in *Camera2Activity.java*. The implementation uses a range of classes from the Camera2 API [20]. The most important ones are described below.

CameraManager: System service for detecting connected cameras, getting camera characteristics and connecting to a camera device.

CameraCharacteristics: Describes properties of a connected camera device (for example output sizes and which way the lens is facing).

CaptureRequest.Builder: Uses one of the templates defined in the Android CameraDevice class for creating a CameraCaptureSession. We use the template TEMPLATE_PREVIEW for the regular camera preview and TEMPLATE_RECORD when recording video.

CameraCaptureSession: A session used to capturing images from the camera. It is created by the CaptureRequest.Builder, which provides a target surface and a camera device. Has the method setRepeatingRequests() which continuously captures images that are used to show the camera preview and the method capture() for capturing the actual image.

AutofitTextureView: Not part of the camera2 API, but a class that extends TextureView[23]. It is used to automatically fit a TextureView to the size of the selected camera so that the TextureView can be used to show the camera preview.

The Figure 19 shows the most important function calls in this activity and the typical flow when taking a photo. In order to avoid problems where the app exits before closing the camera, we use a semaphore that is acquired while the camera is opened or closed and released after it has opened/closed (and if there is an error or the camera is disconnected).

onCreate(): Here we set up the layout. To know how to set the layout, the activity is started with a StringExtra[3] describing the type of image that shall be captured ("front" or "back" for document photos and "face" for face photo). We also define a *stateCallback-function* that will be used when opening the camera and add onClickListeners for the different buttons.

onResume(): Every time the activity is resumed this function handles resuming the camera function by setting up and opening the camera.

setUpCamera: Uses the CameraManager to list the available camera devices on the phone and to get CameraCharacteristics for each one. Loops through and picks the front camera for face photos and the back camera for document photos. Then loops through the selected camera's possible output sizes and picks the one with a width:height ratio as close as possible to the selected ratio 3:4. The selected size is used in configureTransform() to set the size of the AutofitTextureView.

[3] Extras are used to pass variables from one intent to another.

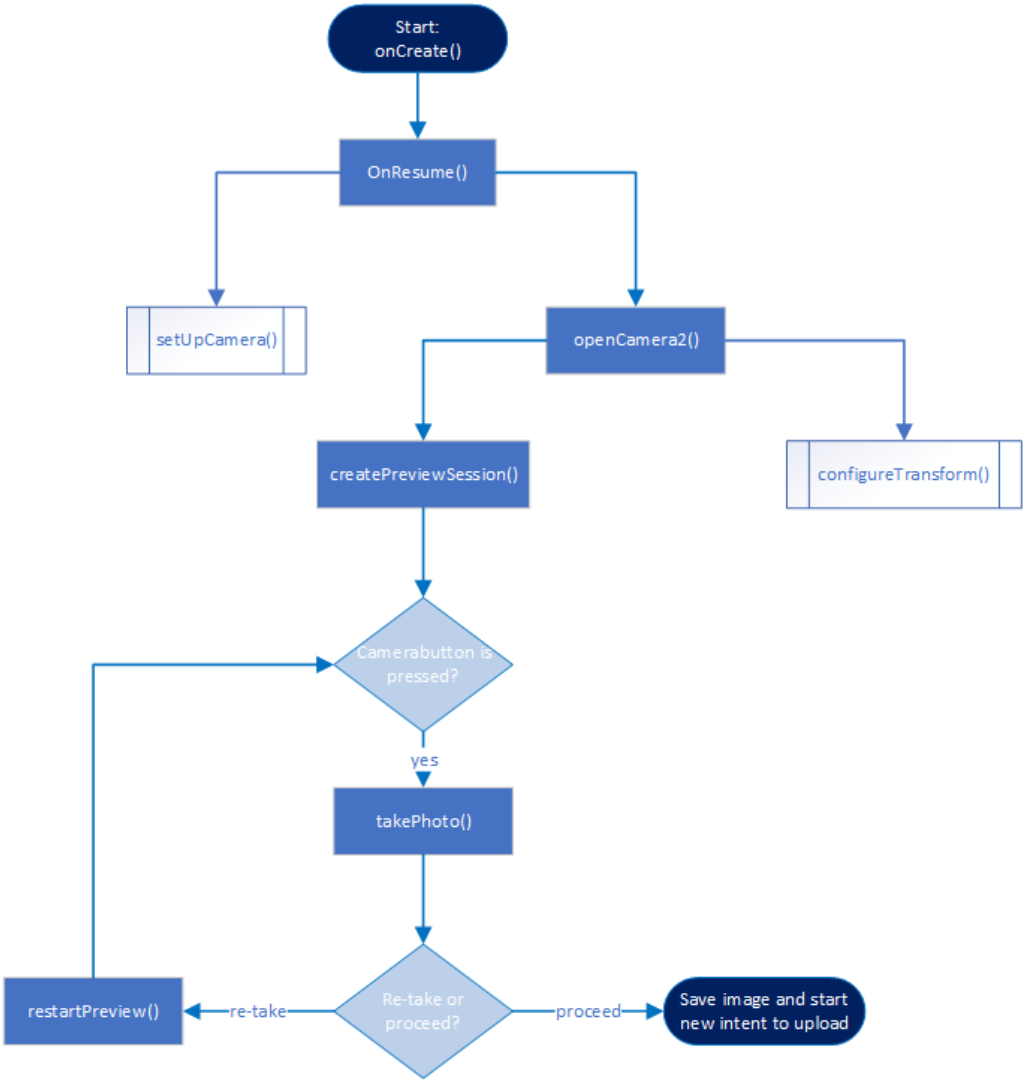


Figure 19: Overview of the photo application

openCamera2(): When openCamera is called a camera has been selected, the output size has been determined and an AutofitTextureView has been defined. The AutofitTextureView is configured in configureTransform(). When the camera is correctly opened the previously defined stateCallback-function is called. In the stateCallback a CaptureRequestBuilder is created and the AutofitTextureView is added as the target surface. The CaptureRequestBuilder is then used to create a CameraCaptureSession. In the CameraCaptureSession

takePhoto(): Called when the user clicks the camera button. The function updates the view so that the camera button is hidden while the accept button and retake button appear, the area around the marked rectangle is blacked out (for document photos). The captured image is displayed by calling stopRepeatingRequests(). If the image is accepted, the captured image is saved internally in the application and the uri [4] is passed to a new intent which uploads the photo to the server.

restartPreview(): If the user is unhappy with the image restartPreview() is used to change the view in order to restart the camera preview so the user can take a new photo.

6.3.7.2 Finding the Right Camera Size

One of the biggest challenges when building the camera application was configuring the AutoFitTextureView to get the correct size. Both available camera sizes and screen sizes vary between different types of phones, so we need to create a TextureView that matches an available camera size, has the desired width:height ratio and fits on the screen of the phone where it is being shown. We set 3:4 as a preferred width:height ratio, loop through all available sizes for the SurfaceTexture[5] and pick the best match. This is used to set the aspect ratio of the AutoFitTextureView.

6.3.7.3 Applying Camera Filters

As mentioned in 5.5 we wanted to create some filters to make it easier for the user to take good document photos (as shown in figure 9). In order to create the rectangle to guide the user we created a PNG image in PhotoShop and used an ImageView to add this to our view. We then created a two other images, one that blurred the area around the rectangle and one that just covered it in black. The size of these filters are configured in an onSurfaceTextureSizeChanged-listener so that we make sure that the filters fit each other and that the sizes of the filters are determined *after* the camera preview size has been determined.

6.3.7.4 User Instructions

To give the user instructions we show use a TextView in the upper part of the screen. This disappears when the user clicks it and can reappear by clicking an info button. To make the app look a little more alive we use an animation to make the textbox slide up when it disappears and slide down from the top when the user clicks the info button. The content of the textbox changes after capturing an image to remind the user to check the quality of the image before proceeding.

[4] Uniform Resource Identifier
[5] SurfaceTexture is an object for capturing images/videos, so these sizes represent the possible camera sizes.

6.3.7.5 Video Application

The video application uses many of the same android.camera2 classes as the photo application and the setup for the camera is quite similar. However, recording a video is a little more complicated, so the implementation is a little different. Both setting up the video recorder and managing the user interface requires quite a bit of code. For this reason we chose to initialize the activity in VideoRecordActivity and then use two fragments to manage the camera. We use the abstract class CameraSetUpVideoFragment to configure, open and close the camera, start a preview, record video etc. Then we have CameraRecordVideoFragment, which extends CameraSetUpVideoActivity and manages views and user interaction.

Setting up the camera preview is quite similar to the photo application. We pick the front camera as our camera device and we use an AutofitTextureView to show the camera preview. The difference between the regular photo app and the video app is that in addition to choosing an output size that fits our preferred ratio, we need to make sure the mediarecorder supports this output size. For this reason we first select the video size in chooseVideoSize(...) and then we use the ratio we find here to select an optimal preview size.

6.3.7.6 MediaRecorder

The MediaRecorder class [24] is central to the video application. We configure the MediaRecorder in setUpMediaRecorder. Here we choose mp4 as the output format and we choose a video quality that allows us to take sufficiently long videos without exceeding the 5 MB limit imposed by Keesing. We also need to set the orientation of the MediaRecorder so that it matches the orientation of the camera lens. Since the livenesss check does not require sound, we chose not to record sound in order to reduce file size. Like photos the video file is saved internally in the application. In order not to worry about deleting the video if the user chooses to re-take it, the output file is always set to the same file name. In that way the new video will just overwrite the old one if the user chooses to record a new video.

When the user starts the recording we need to close the preview session and set up a new CameraCaptureSession using the CameraDevice.TEMPLATE_RECORD template this time. We create a new preview surface from our AutofitTextureView in order to show the video while recording and we get a surface from the MediaRecorder as our recording surface. Both of these are added as targets for the CameraCaptureSession. Once the CameraCaptureSession has been configured a callback function starts the MediaRecorder.

6.3.7.7 Implementing Video Requirements

As mentioned in 3.4.2.1 videos must be at least 2 seconds and the size can be no more than 5 MB. In order to make sure these requirements are always met we have chosen to limit the user's control over the video. When the user presses record we use a CountDownTimer [25] to make the recording run for 2.1 seconds before automatically stopping. All buttons are hidden or disabled during this period, so that the user is prevented from stopping the recording too early. In order for the user to know that the recording is running a timer shows a timer counting down from 2 seconds next to the recording button. We do not specifically control the video sizes, but we use the following measures to keep the video size well within the limits:

- The quality is fixed at 480P
- We do not record audio

- Duration is fixed at 2.1 seconds.

6.4 NFC

Our NFC functionality is based in the JMRTD library [26]. JMRTD is a Java implementation of the MRTD[6] standards specified by ICAO[7] and it provides methods for reading, decoding and validating electronic passports.

The JMRTD library is a quite extensive one, so to figure out how to implement our own NFC passport reader we spent a long time reading the documentation and looking at different code examples on GitHub. We are especially thankful for Anton Tananaev's e-Passport NFC Reader [27] which was a great inspiration to our implementation.

As mentioned in section 5.6, the ReadNFCActivity.java handles most of the logic behind the reading of the passport, while the other NFC activities are mainly used as supporting activities to get the information needed for the BAC-protocol[8] and to display the result.

6.4.1 Getting MRZ Information for the BAC Key

As mentioned the document number, expiration data and date of birth is necessary in order to unlock the passport chip by using the BAC-protocol. In the first version of our application we did this by letting the user enter the expiration date, date of birth and document number manually. In order for the BAC-protocol to work the dates both need to be entered on a yymmdd-format. We soon realized that this would be very easy for the user to get wrong, so we decided to implement a date picker dialog and then parse the selected date to our desired format. Entering the document number is pretty straightforward, but to provide some extra control, we control that it consists of exactly 8 characters and only numbers are allowed.

This solution worked fine, but we wanted to improve the user experience further, so we decided to create a solution using the camera and OCR[9] to read the information directly from the passport MRZ.

We considered a few different options, and ended up using Anyline SDK's solution for scanning passport MRZ [28]. The reasons why we chose Anyline were mainly that they have very well document APIs that were easy to integrate into our application and because they provide a fast and accurate character recognition specifically adapted to scanning passports. We signed up for a trial period for using their APIs so that we could demonstrate to Buypass that using OCR makes the app a lot more user friendly.

In order to make sure the information was entered correctly, we ask the user to double check it before reading the passport. If they discover an error they are redirected to ManualNfcActivity. The reason why they are redirected here and not to ScanMRZActivity is that if the OCR scanner was to read the MRZ wrong once, chances are that would probably happen again (as the algorithm does not change between each try), so it would be safer to re-enter the information manually.

6.4.2 Implementing the NFC Reader

The actual reading of the passport is handled in ReadNFCActivity. To make sure the user understands how to read the passport chip, we have created an animation that shows the

[6] Machine Readable Travel Document
[7] International Civil Aviation Organization
[8] Basic Access Control
[9] Optical Character Recognition

user how to do this. We insert BouncyCastle as our security provider, since JMRTD depends on BouncyCastle for safe data transmission. Figure 20 shows an overview of how an NFC tag is discovered, read and passed to the next intent. The main tasks of each function is described below:

onResume(): In order to read NFC tags the phone needs to have a default NfcAdapter, if this is not available, the user gets an error message. We also define a ”techlist“ that lets us choose which types of NFC tags we are interested in. Since we only want to read passport chips our techlist consist of only android.nfc.tech IsoDep, as isoDep is the technology in passport tags. If an NfcAdapter is available a PendingIntent is created for the adapter. The new intent is created with a Single Top Activity flag so that if the activity is already running on top of the stack, it is not re-launched, but onNewIntent is called instead.

onNewIntent(intent): If the tag is of the correct type (isodep) readNFC is called with the tag and a BAC key constructed from the MRZ data as parameters.

readNFC(tag, BACkey): Performs BAC and reads the RFID chip. More about this in section 6.4.3. If the tag is read successfully NFCResultActivity is launched. Otherwise the user gets an error message and is asked to try again.

6.4.3 Accessing and Reading the RFID Chip

As figure 21 demonstrates, when readNFC is called the discovered tag is stored in an Isodep object. This is used to create a CardService[10] and from the CardService we are able to open a **PassportService**. PassportService is a central class in the JMRTD library. It lets us use our previously created BAC key to unlock the RFID chip using a call to PassportService.doBAC().

If BAC succeeds, we use the PassportService to read an input stream containing the DG1 and DG2 file (see chapter 2.1) into a special JMRTD data structure called LDS[11]. From this file structure we can access the actual files. We are then able to read all MRZ data from DG1File and we use the subroutine *getImageBitmap* to get a bitmap from DG2File. DG2File contains a list of FaceInfo[12] objects which are used to store image data. In *getImageBitmap* we use the FaceInfo method *getFaceImageInfos* to get the image objects. In some cases there might be more than one image, but we just want the first FaceImageInfo-object as this is the passport photo. To read the image we create an input stream and then parse the input stream into a bitmap. A selection of MRZ data and the bitmap is then passed to the next activity, which displays everything to the user. An example can be found in figure 46 in appendix T.

6.5 Database

As described in section 5.1, System Design, our database is separated from both the server and the application. We decided to let all communication with the database go through our server (no direct communication between the application and the database). This makes it easier to secure our database since database access is denied to everyone except our server. API calls the application makes to update or retrieve data from Firestore are routed to .../firebase, see

[10] Service class in JMRTD
[11] Logical Data Structure
[12] FaceInfo is a JMRTD object

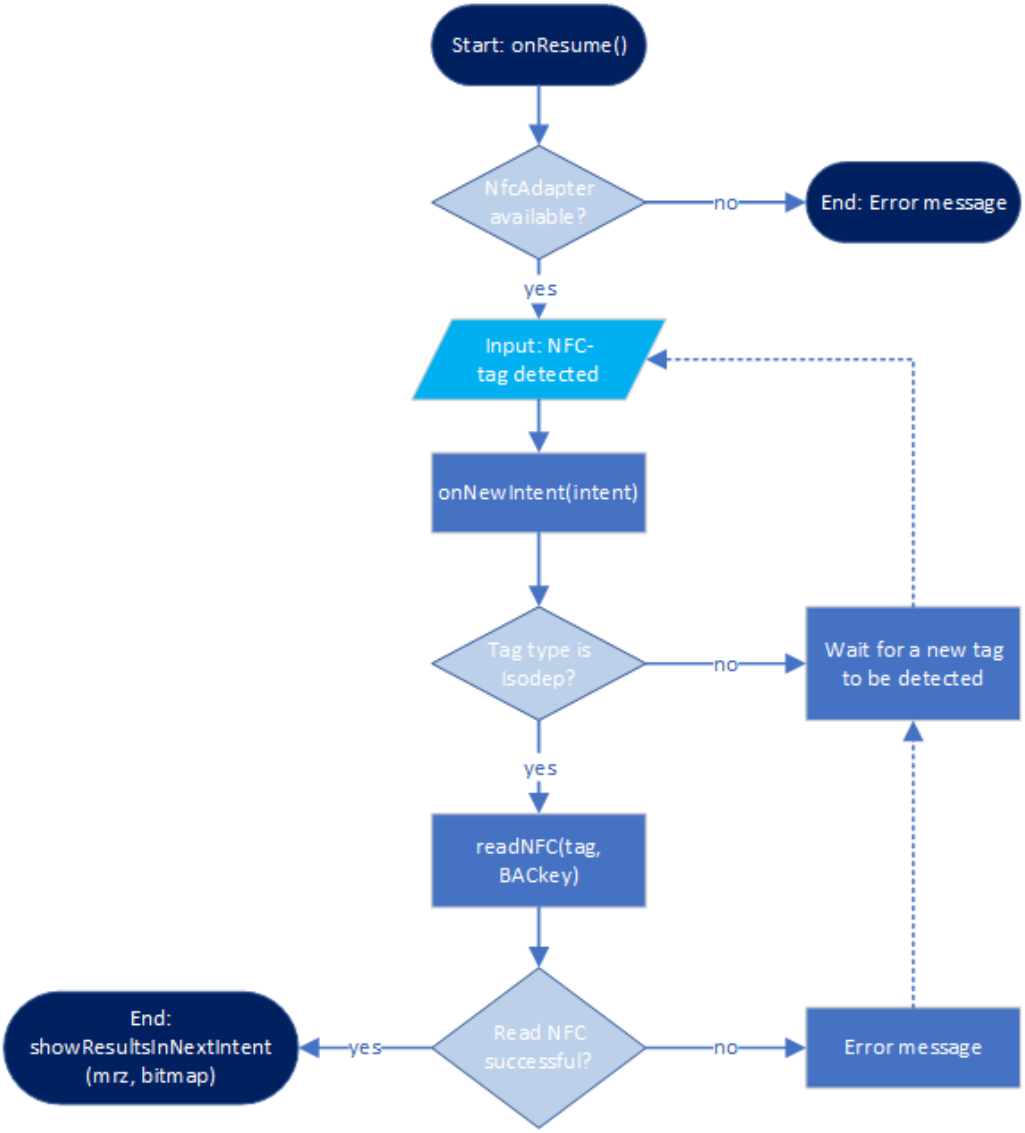


Figure 20: The figure shows the process of detecting and reading a passport RFID chip

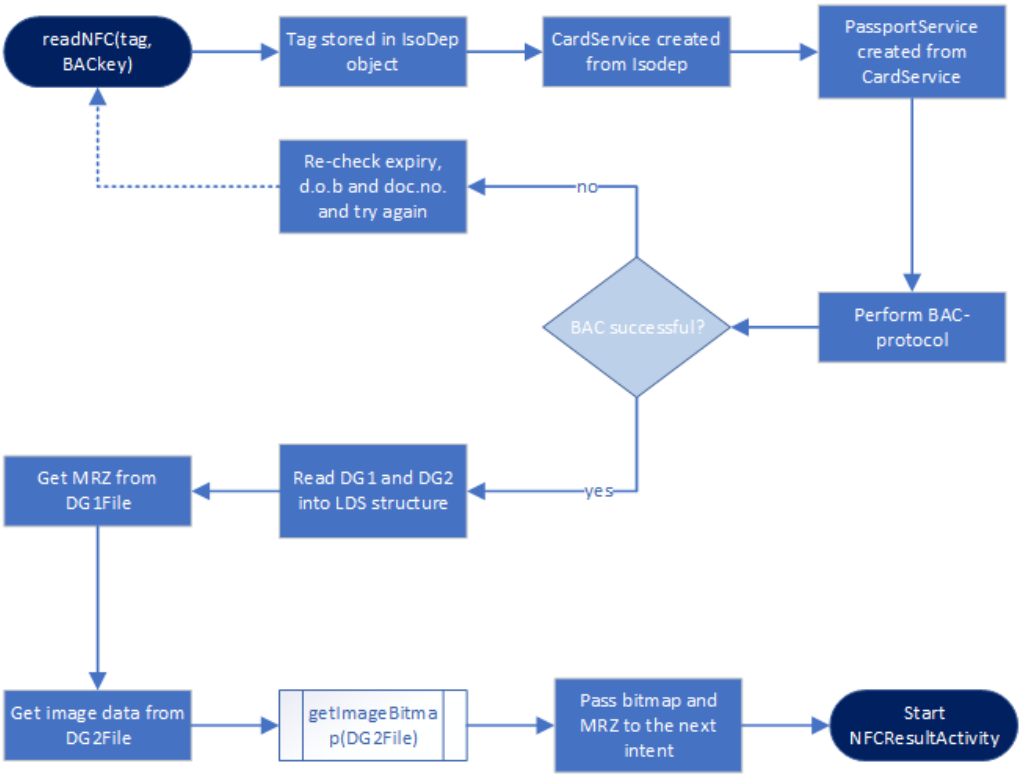


Figure 21: Reading the NFC tag in a passport

figure 14 or appendix K, which handles the requests by calling on methods from *databaseFunctions.js*. As database queries are executed asynchronously, we use Javascript Promises [29] to handle the query result.

Firestore provides its own methods for writing database queries. To collect data we use the method *get()*, which can be called on both documents and collections. An advantage of the Firestore queries is that they are ”shallow“ by default, meaning that one can retrieve data at document level without collecting the entire collection or the document's sub collections. This is beneficial for example when we need to retrieve the verification URL of an ongoing verification from a user document without collecting the entire *verifications* sub collection.

To update the database we use either the set() or update() method. The difference is illustrated in the two code examples below. UpdateStatus and storeVerificationToDb update the ”status“ and ”verificationResults“ field of a verification document of a selected user. The reason why we use update in updateStatus, is that we already know the status field already exists since it is updated once a verification process is started. When updating the verificationResults field on the other hand, we do not know if this already exists, because this field is not created until we receive a result from Keesing. That is why we use the set() method, which creates a document containing a verificationResults field if it does not already exist. Since we know that the document already contains a status field, we use the {merge: true} option to merge the updated document with the old one instead of overwriting it.

```
updateStatus: function(userId, verificationId) {
    return new Promise((fulfill, reject) => {
      database
        .doc(`users/${userId}/verifications/${verificationId}`)
        .update({
          status: "finished"
        })
        .catch(error => {
          reject(error);
        });
      fulfill("Field updated");
    });
  }


  storeVerificationToDb: function(userId, verificationId, result) {
    return new Promise((fulfill, reject) => {
      database
        .doc(`users/${userId}/verifications/${verificationId}`)
        .set({ verificationResults: result }, { merge: true })
        .then(() => fulfill("Verification results stored"))
        .catch(error => {
          reject(error);
        });
    });
  }
```

6.6 Security

The following section describes how we have implemented the security requirements described in 3.5. The application is built with the Principle of Least Privilege in mind, meaning that the user can only access information that is absolutely necessary. We have implemented this by using a client-server model that makes sure there is no direct communication between the user, Keesing and the database. For this reason most of the security measures are implemented on the server side, but we have also taken a few considerations in the frontend

application.

6.6.1 Frontend

In our application frontend security has two main purposes, validating and sanitizing user input and making sure the user can not access information he should not have access to.

Input validation and sanitation: We get textual user input two places in the application, when the user logs in and when he enters MRZ information manually (see section 5.6). When the user enters the MRZ information manually he is asked to enter the document number, expiration date and date of birth. When entering the document number the application rejects anything that is not an eight digit number and when entering the two dates, the user has to pick from a date picker dialog, so the user input here will always be valid.

MRZ scan validation: When the MRZ is scanned optically we validate the result by using *MrzConfig.setStrictMode(true)*, which is a method provided by Anyline for validating MRZ input. It is built on an algorithm developed by ICAO [30] for calculating a ”check digit“ based on the MRZ data to verify that the MRZ has been interpreted correctly. If the MRZ is not valid, no result will be returned.

File uploads: We ensure that no unwanted files can be uploaded by only letting the user upload photos and videos taken within our photo application and not letting them upload existing files.

Principle of Least Privilege: The user can only access information that is necessary in order to correctly go through the verification process and later check the verification outcome. We do inform the user if a verification step fails, but we do not give any specific information about exactly why, say, a document check failed.

6.6.2 Backend

The backend security implementation involves mitigating server failure, input validation and securing the data transmissions.

6.6.2.1 Validating Requests

As our server acts as a mediator between the user, the database and Keesing, it is responsible for checking all user requests before passing them on. In order to validate the request body we use the Express Validator middleware. We use provided methods for checking that any uploaded user name is less than 40 characters, and to trim whitespace at the beginning and the end. We also escape <, >, &, ', " and / by replacing them with their corresponding HTML entities. We then check that the request body contains the expected fields.

In the webhook we validate that the JSON objects we receive have the correct format. Express validator lets us define our own custom check function, and we use regular expressions to verify that the ID and verification URL that matches our pattern are accepted. The verification IDs issued by Keesing are series of lowercase hexadecimal digits in groups of 8, 4, 4, 4, and 12 digits and separated by hyphens (36 characters in total), so we could make specific regular expressions to make sure the verification ID has the right format, but after recommendations from Keesing, we created a quite general format that accepts all 30-50 character strings containing digits, lowercase letters and hyphens. The reasoning behind this

is that since Keesing's API is still in Beta and we have experienced quite a bit of changes throughout the project, we want our server code to be resistant to minor changes in the verification ID format. Also, it would be difficult to do any damage with the uploaded ID when the string length is restricted to 50 and only alphanumerical characters and hyphens are allowed. Invalid requests receive a 422 invalid request error message.

6.6.2.2 Preventing DOS-attacks

We have used two complementary strategies for preventing DOS attacks.

1. Limiting file upload size. As only photos and videos taken inside the application (which are limited to 2 seconds) are uploaded from the application, performing DOS-attacks by uploading large files is already quite difficult, but as an additional security measure we have configured the server not to accept requests with a larger request body than 5 MB. This limit is chosen because it is the maximum file size that Keesing accepts. For more information regarding this implementation see appendix S.
2. Limiting requests per minute. We are using the Express Rate Limit middleware to limit the maximum number of requests from the same IP address to 30 per minute.

6.6.2.3 Secure Data Transmission

Our API will be handling sensitive personal information with both photos and JSON objects. In order to secure these sensitive data, and to accommodate Keesing Technologies' requirements for a HTTPS secured webhook we have obtained a SSL-certificate for our domain.

Buypass is the only CA[13] in Norway, and as our employer it was natural to look into what they could provide regarding SSL-certificates. The certificate ensures that the communication between the mobile application, Keesing Technologies' API and the server is encrypted [31]. See appendix S for detailed information about configuration.

6.6.2.4 Database Security

As described in chapter 3.5 there are two major security concerns for the database. One is that an attacker might try to change the result of a verification process and the other one is that someone might try to access data they should not have access to.

In order to protect the integrity of our database, all user input that is sent to the database is validated and sanitized. The only user provided data that is stored in the database is the user ID and the FCM token. The actual verification result can only be updated when the webhook is triggered with a verification URL. When this happens our server makes a call to the Keesing API to collect the verification result for the requested verification and uses the response to update the database. Of course, if an attacker knows a verification ID he could potentially create a request that triggers the webhook. Still, all he would accomplish would be that our server sends a request to Keesing to update the database with their response.

To prevent unauthorized users from accessing data they should not have access to, each user is associated with a Universal Unique Identifier (UUID). Whenever a user logs into the application a UUID is generated and sent to the server which stores it in the database. When the server receives a request to get the verification results the UUID must be added to the request. We then check that the UUID stored on that user matches the UUID in the request. Ideally we would have checked the UUID for every call to our API, so that no one could start a verification process or upload files to another user's account. Unfortunately we did not think

[13] Certificate Authority

about this security hole until quite late in the project, so we only had time to implement it where we considered it to be most important.

6.6.3 Evaluation of the Verification Process

As described in chapter 3.6 evaluating the quality of the final verification process after implementing Keesing's solutions for document checking, face comparison and liveness check is an additional goal for our project. The main purpose of this was to see whether a malicious user, through normal use of the application, could obtain an e-ID in another person's name. Throughout the development we tested our application with countless different images/videos, so without doing any structured testing, we were already quite familiar with the different checks by the time our prototype was ready. We used the insight we had gained throughout the development process to come up with a few potential misuse cases that we wanted to look at more closely (described in Requirements).

First we wanted to see if we could pass the document check using an image of a passport, so printed out an picture of a valid passport and used this for the document check. It turned out it was just as easy to pass the document with a photo of another ID photo as with a photo of the actual document.

Then we wanted to see if someone could pass the face comparison for another person's document. From experience with Keesing's APIs we know that their algorithm seems quite good at detecting image faces for the face comparison, even if there is quite a bit of background noise, so for the face photo we simply took another picture of the ID document. Again, we passed the face comparison most times we tried this.

Lastly, we tried fooling the liveness check by filming previously recorded video clips. This did not succeed in any of the attempts we made.

When testing regular use of the application with a valid ID we discovered that as long as the document photo was of good quality and showed the entire document, the first two verification steps were accurate. We saw practically no valid documents failing the document check, although some document photos did end up at helpdesk. The liveness check on the other hand turned out to be quite prone to false negatives as some testers had difficulties passing the liveness check without significant effort. This was an important reason why we chose to include a popup instructing the user to say a sentence or make grimaces during the liveness check.

7 Testing and User Feedback

Testing is of great importance when developing a mobile application. End users should not experience any problematic bugs or flaws within the application at the point of release.

Buypass wanted a prototype application providing a demonstration of the possibilities with digital ID proofing, our main area of focus during the development process has been implementing the necessary functionality. The testing was done mostly manually as we went on. This was due to the complications implementing the third party API functionality, and the response codes as well as unstable availability.

7.1 Static Code Analysis

Testing the code statically for bugs, linting errors, vulnerabilities and code smells is important in regards to both readability and possible functionality malfunctions.

7.1.1 Frontend

By using SonarQube throughout the project we have been able to minimize the amount of bugs and vulnerabilities. This has provided us the opportunity to eliminate both bugs and vulnerabilities within our frontend code, and to obtain both a security, maintainability and reliability rating of an A. The security rating indicates the weaknesses a hacker might exploit, the maintainability provides information as to how the code will be harder to update than it should, and the reliability indicates weaknesses as to how the code will act in difference to what is expected.

7.1.2 Backend

As for static code analysis on the backend portion of our code we have used ESLint. We have configured ESLint to disable error messages when an unexpected console log appears in the code. These console logs are done for tracking the processes on the server, and have been used in order to debug, as it makes it possible to detect where the errors occur. The decision to disable the ”no-console“ rule have been made in order to easier process the errors that represented a possible vulnerability in the program. A solution to the unexpected console log errors would be to remove all the logs within the code, but as this is a prototype and the code will be handed over to Buypass we saw it best to leave it as is.

We have implemented all the recommended rules for Node.js [32], but the two rules for ”no-process-env“ and ”no-sync“ are implemented as warnings. We have also turned on warnings for ”no-useless-escape“ which complains about a unnecessary escape character in our RegEx[1] for JSON schema validation. The schema validation is dependent on these escape characters, and therefore we chose to leave it on as a warning. The ”no-process-env“ rule disallows the use of process.env which we have used in order to have the client credentials for Keesing Technologies' API as environment variables, and not directly in the code. The ”no-sync“ rule disallows the use of synchronous methods, but we are dependent on it in regards to the file upload aspect of the backend. ESLint has allowed us to eliminate several

[1] Regular Expression

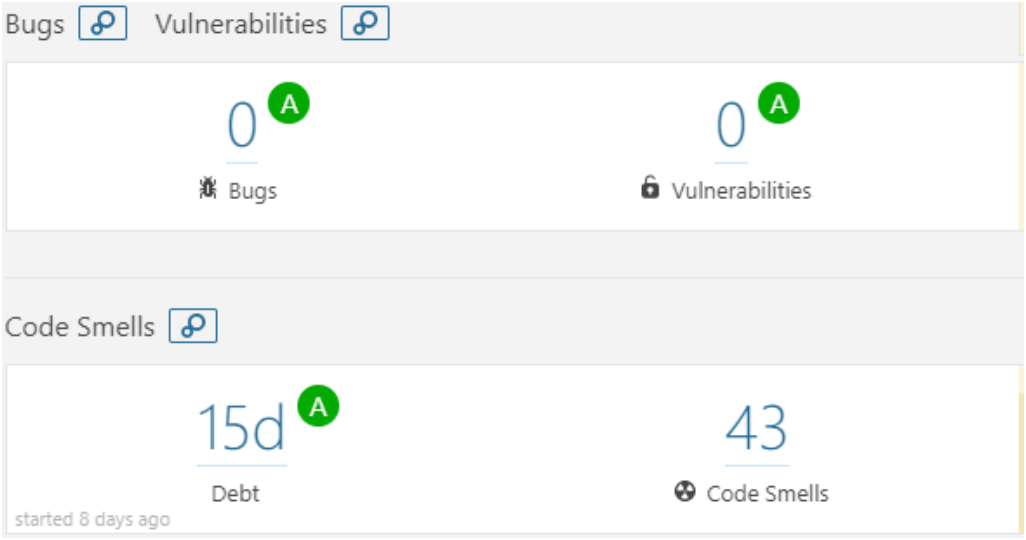


Figure 22: SonarQube results of the frontend code

vulnerabilities in the code, and we are only left with some warnings regarding the rules mentioned above. For more information regarding the configuration of ESLint see appendix R.

7.2 Manual Testing

In order to continuously test the functionality we implemented during the development process we have been writing manual test cases, see appendix J. The file containing the test cases has been on the group's Google drive, and that has provided us with the ability to perform tests, write new and to see the state of the different tests. The test cases also contained a field in which a mitigation has been suggested.

The manual tests on the backend of our application have been performed by using Postman, and the mock API of the third party API up to a certain point, when we were told it was insufficient. The manual tests has allowed us to keep testing parallel to the implementation of functionality. We have written simple tests within Postman which are being passed or failed whether we receive the expected response or not, see appendix K. These tests are triggered when we perform the desired API call, and the outcome is determined by the JSON request bodies.

The manual tests performed on the frontend of our application has been done through testing the implemented functionality and appearance with emulators and test phones. Both the emulators and the test phones have been used in testing the appearance of the application. The test phones has also been used for testing the functionality involving file uploads. This has allowed us to easily take blurred photos, face images not containing any face, liveness check videos not containing faces or being blurred easily.

When testing the entire verification process, as well as logging in and viewing the results through the push notifications the manual testing has been helpful because it gives us an idea of an interaction the end users may have had with the application. We found this more important than the automated tests as we could uncover bugs and flaws in the way the

automated tests would, but we would also test the interaction with the application.

7.3 User Testing and Feedback

The project has evolved around making a prototype application for Buypass, as stated in the introduction, see chapter 1. It has therefore been in our interest to create an end result that is satisfactory to them. The continuous source of feedback throughout the project has therefore been Buypass.

The process for feedback has involved implementation of functionality and improving the GUI of the application with each sprint, and the sprints have been concluded with the status meetings with Buypass. During the meetings we have provided demonstrations, and we have received feedback on the application. After the meetings, the feedback has been taken into consideration and we have discussed how to reach the desired results.

During the development phase of the project we have had continuously implementation with manual testing as mentioned in section 7.2 above. The manual testing involved the test cases to be executed by the other team members, ensuring that the implemented functionality would work and act as intended. The disputes that occurred regarding the GUI and the implementation of functionality was addressed in the group, and we came to an agreement together. This ensured that we implemented what seemed like the best solution for us.

Towards the end of the project the time schedule caught up with us, and we were not able to conduct user testing performed by outsiders of the project. User testing would have provided information in regards to the UX within different end user groups, such as visually impaired for both colors, font size and style. The user testing would also provide information as to how easily a verification could be conducted by a user unknown to the system.

8 Development Process

Although we had a clear plan for our development method and process, we soon realized that we had to adapt our plan according to what happened in real life. As expected many tasks were more (or less) difficult than we expected and we frequently had to re-evaluate or modify our goals and requirements. We also realized that sometimes we had to evaluate the actual development method and make adjustments due to occurrences such as group members being absent, assignments in other subjects that required intensive work over a period of time and so on. In this chapter we aim to give an overview of how the actual development process went down.

8.1 Scrumban as Development Method

Throughout the project we stuck to our plan of organizing the development process using a combination of Scrum and Kanban. The sprints ended with a meeting with Buypass where we gave a demonstration of our current functionality and received feedback. After this meeting we discussed the past sprint in our retrospective meeting. At the beginning of each sprint we planned the next one based on feedback from Buypass and what we considered the top priority tasks according to our MVP approach. We were usually able to keep a routine of starting a sprint every other Monday and ending it Friday two weeks later. However, at some points we had to adjust this due to external factors. This could be due to Buypass having to advance or postpone a meeting, due to multiple group members being indisposed at the last/first day of a sprint etc. Generally we dealt with this by extending one sprint by a day or two and shorten the next sprint by the same, or vice versa.

8.1.1 Group Meetings

We went through with sprint planning, retrospective and daily scrum meetings as described in the Software Development Method chapter. During the daily scrum meetings we kept each other updated on each other's work, so the retrospective meetings mainly focused on the actual development process and not so much the product itself (this was more the focus of the sprint planning meetings).

In the early phase of the project we had a tendency to work a bit unorganized. Thanks to our Kanban board and our daily scrums we usually had a general idea what everyone was working on, but if someone got stuck on a task or did not have a clear idea how to deal with a challenge, we did not really have any routines to deal with this. Often these problems were difficult to pick up on during the sprints as we could all be a little to focused on our own jobs. During the retrospective meetings we were able to discuss these problems and figure out how to deal with them. See appendix L for the meeting logs from the three first sprints.

In the beginning, the tasks on the Kanban board were often quite large and the task description a bit vague. The idea behind having a Kanban board was that when someone finished a task, they could pick a new one from the To Do section, but when the tasks were this way they often seemed a bit overwhelming and difficult for one person to start working on. After discussing this during our retrospective meetings we started breaking down the tasks

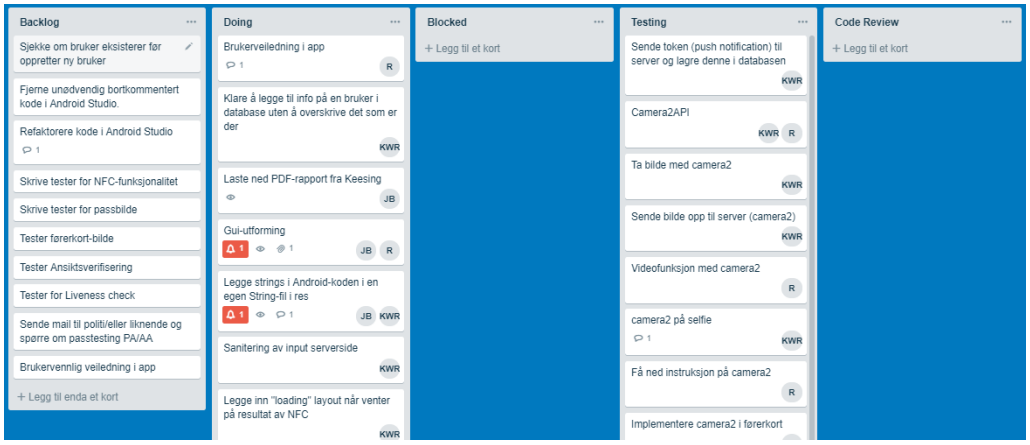


Figure 23: Trello board during sprint 4

into smaller, more specific pieces. For an example of one of our Trello boards, see figure 23. This gave everyone more of an overview of what needed to be done to complete a task and it encouraged us to be proactive and take initiative instead of waiting for the others to tell us what to do.

Another challenge we encountered was that since we usually worked on distinct part of the application and everyone sort of had their own area of competence, we became very vulnerable to people being away from work for different reasons. An example was when we developed the NFC reader. This was mainly researched and implemented by one of the group members and the rest of the group had done little work with this technology. There were two major problems with this. 1) If the person was not present, the rest of the group was unable to continue the work, and 2) If that person did not know how to solve a problem, no one else were really able to help. In order to overcome such issues we decided that for each of the main development challenges (for example camera application, database implementation, server calls to the Keesing API, uploading files etc) at least two people should work on and be familiar with the code, so that if one person was away, the other one would know what needed to be done. We also made sure to write some notes to document our work in a shared document.

During the sprint planning meetings we had more of a product focus. We usually started by discussing the feedback from Buypass and decided on what we wanted to implement by the next meeting with them. Based on the experience from the last sprint, we then discussed which one of the items from the product backlog that would make the greatest improvement of the product. In general our first priority was to implemented the changes our client suggested. This was partly because our main objective is to create a product that satisfies their requirements and because we thought it would give a good impression of us as a team to be able to present a product that better adapted to the client already at the next meeting. We also appreciate that our partners in Buypass are experienced project managers, which means their feedback is likely to be valuable to our development process.

Additionally we had weekly meetings with our supervisors from NTNU. At these meetings we primarily discussed the development process itself and made sure we were on the right track regarding our priorities. See appendix O for a selection of meeting logs.

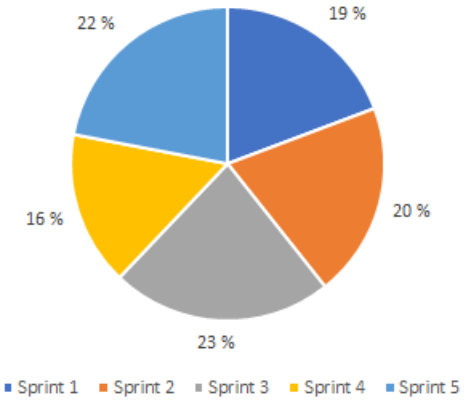


Figure 24: The total time conducted divided between the five sprints

8.1.2 Pair Programming

In the beginning of the project our plan was to use pair programming. in that way it would always be two people with the knowledge and it would not matter if one of us was absent. It could also increase the quality of our work since everything is double checked. Although this sounds good in theory, we quickly learned that in some cases this was a bit counterproductive. If a task is not particularly challenging, using pair programming felt like a bit of a waste of time. On challenging application modules, like the camera implementation and the communication with Keesing, pair programming was very useful. In the development of these modules we frequently got stuck due to quite small programming errors, and these were a lot easier to avoid or discover when two people were working together.

8.2 Sprints and Milestones

Due to complications during the development process we saw the need to adjust both the milestones and the contents of each sprint in regards to the progression. The agile form of development allowed for this, and it helped both us and Buypass come to a better satisfaction and agreement as to how the application should both act and look. During the project we have reached several milestones representing implementation of functionality, demonstrations and completion of the project, and the milestones of the project are listed below:

Milestone 1 - 8th of February: The application did have a simple GUI with the ability to take photo.

Milestone 2 - 14th of February: The communication between the application and the server was established, and we were able to send simple JSON objects between the two.

Milestone 3 - 7th of March: We were able to upload a photo to the third party API for a verification.

Milestone 4 - 11th of March: The application's implemented NFC reader was able to extract information from a passport.

Milestone 5 - 15th of March: We were able to conduct a verification of an ID document, performing both the face comparison and liveness check in addition.

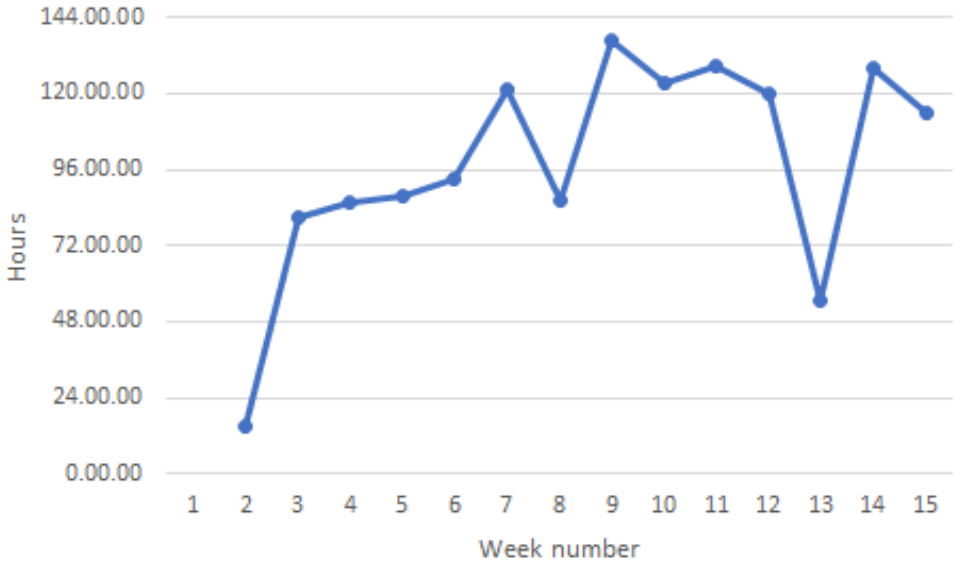


Figure 25: The total time week to week throughout the entire project. Sprint one of the development process started in week six

Milestone 6 - 20th of March: The webhook was triggered when the verification results were done, and we were able to retrieve the results and storing it in the database.

Milestone 7 - 27th of March: The push notifications was working properly. The GUI was updated in regards to feedback from previous meetings. Bugs and vulnerabilities were eliminated from the application.

Milestone 8 - 10th of April: Recorded demonstration of the application, demonstrating all the implemented functionality.

Milestone 9 - 12th of April: Elimination of bugs and vulnerabilities in the frontend code of the application.

During the five sprints of the project we conducted 1105 hours of work, see figure 24 for distribution between the sprints. This leaves us with an average of 27,625 hours of work each conducted to the project in the development phase. Including the time conducted to research and planning in the project planning phase of the project we have conducted 1374 hours of work, see figure 25 for the total time from week to week during the project and appendix Q for the Toggl time reports. The distribution of the hours conducted during the sprints, as well as more in-depth description of the five sprints can be found in appendix M.

8.3 Minimum Viable Product Approach

Throughout the project we have used a Minimum Viable Product (MVP) mindset when deciding which features to develop. Our MVP strategy involved creating an overall plan for when we hoped to reach different milestones, but the day-to-day and week-to-week planning largely focused on the question “What is the most important thing in order for the application

to work?".

In the early phase of the project the objective was quite simple: creating an application that could register a username, take photos and videos and upload these to the server, and a backend that could start a verification at Keesing's and then upload necessary files in the correct format. The way we saw it, creating an application that could take the user through the verification process would represent a minimum of what our application should be able to do. When creating the first prototype we were mainly focused on the concrete steps that needed to be implemented, and we paid less attention to things like GUI, security and database management. For one thing this enabled us to get early feedback from Buypass. Another thing it gave us a sense of security to know that the most essential part of the application was in place so that we were free to choose which aspects of the application to focus on in further development. As we knew the basic functionality was there we were able to spend quite a lot of time designing the GUI, developing the NFC reader and implementing a solution to handle verification results.

8.4 Interaction with Stakeholders

8.4.1 Initial Meeting with Buypass

Our first meeting with Buypass regarding the project occurred on the 18th of January, for meeting log see appendix P. Prior to this meeting our initiate starting point had been the project assignment, see appendix A, researching the different technologies and tools mentioned. During the first meeting we constructed a more updated scope for the assignment. The redefined scope involved the development of a native Android application with a backend solution of our choice where we decided on a RESTful API, as mentioned in chapter 1.

8.4.2 Initial Meeting with Keesing Technologies

During the planning phase of our project we did not have access to much information about Keesing Technologies apart from what we could find on their website and what we had learned from our initial meeting with Buypass. We were supposed to have a meeting with a representative from Keesing quite early, but due to this representative being indisposed, this meeting was postponed several weeks. In the meantime, in order to get started, we received a detailed integration manual for their API and a link to a web based API documentation. We were initially a little confused by this since the two documentations seemed surprisingly different given that they were supposed to describe the same API, and since the integration manual was a lot more extensive and explained the process and server calls a lot more in detail, we assumed this was the one we should use.

Up until the meeting with Keesing (on the 22nd February, see appendix P for meeting logs) we worked on implementing a document check using the API described in the integration manual and we succeeded at creating a server call that could upload a document photo and get this verified by Keesing. During this meeting we were informed that Keesing actually offered two separate APIs and that we should use the API in the web based documentation[33] and not the integration manual.

After receiving this information we needed to make quite a few changes to what we had decided so far. For one thing, the first documentation had stated that we needed a static IP address for our server, but when using the second API, this was no longer required. For this

reason we re-evaluated our server solution (this will be discussed later in the report). We also had to re-write all calls to Keesing, as the verification process was conducted quite differently using the new API. Luckily most of our work on the client application was quite independent of what we had implemented on the server side, so here it did not set us back that much.

During the initial meeting we also got new information regarding the NFC functionality. We were informed that this would be in the testing phase at the end of February, but it was uncertain if or when this would be available to us.

8.4.3 Continuous Feedback

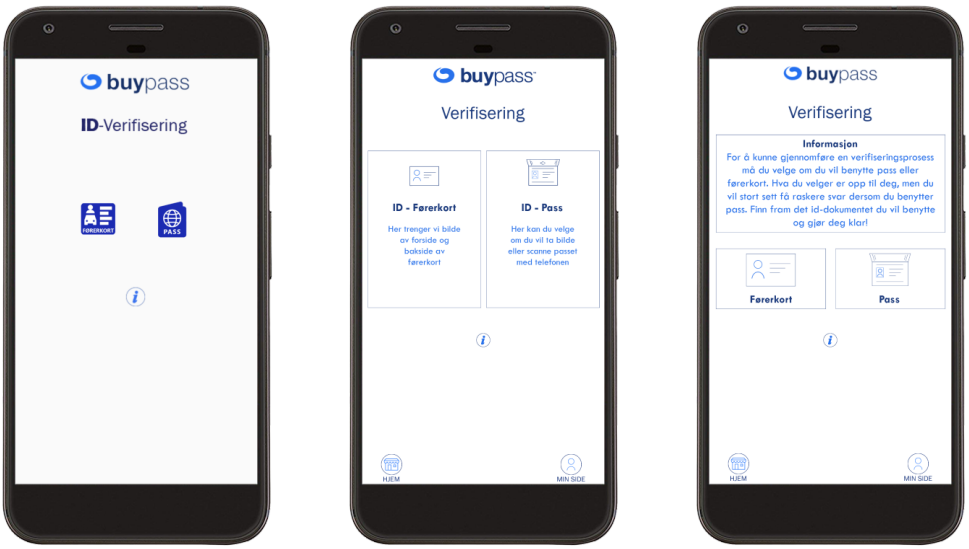


Figure 26: The different version changes from the first version to the left to the last on the right

As mentioned in chapter 7, section 7.3, Buypass provided continuous feedback throughout the project by demonstrations, meetings and email exchanges. This feedback was taken into consideration and implemented for the next demonstration that would occur, see appendix P for meeting logs.

During our meeting on the 1st of March we received feedback for providing more verbose explanations throughout the verification process. This feedback was taken into consideration and implemented for the upcoming demonstration on the 20th of March. During this demonstration we presented the verbose explanation implemented by pupops and more informative buttons. After the demonstration we received feedback on how the verbose explanations and the application could seem a bit too technical and formal. They wanted us to create more of a informal dialogue with the users, and this was implemented for the next demonstration. See figure 26 for some of the changes in the GUI throughout the development process.

8.4.4 Challenges

During our implementation of the Keesing API we learned that the API documentation we used was still under development, so we ran into a few challenges along the way.

File uploads The API documentation stated that photos and videos must be individually uploaded as form data and a mock server could be used to check that the format was

correct. We tried several different approaches and formats but were unable to fulfill the upload. Buypass put us in touch with one of their in-house developers, and we initiated conversations with Keeing Technologies' helpdesk. We were not able to uncover the possible error within our code, not by the help of Keesing or the in house developer. After a couple of days of conversation with Keesing Technologies' helpdesk we were able to perform an upload. The upload provided now the wrong status code in regards to the documentation, and after this point we had continuous conversation with their helpdesk uncovering flaws with status and error codes.

"Grumpy Servers" Around mid March we encountered an unexplanatory error with our previous implemented functionality, see the listing below.

```
SyntaxError: Unexpected token < in JSON at position 0
```

We were unable to locate the file containing the error, and the code had been working prior to the sudden error. After an hour the functionality would then work again without any changes to the code. A couple of days later we were still experiencing the same error, and we approached Keesing's helpdesk. The issue was addressed and we were told that their servers had been down at the time, and this provided an explanation to the error. We continued experiencing unstable servers during the early stages of the day throughout the project, and we adapted by trying not to be dependent on their servers during these hours. This workaround has mostly included frontend work such as GUI improvements, NFC functionality or implementation of the Camera2 API. By conversation with Keesing's helpdesk and planned demonstrations the servers were not scheduled for any down time during our requested time.

Beta testing At one point during the development process, in advance to the meeting with Buypass on the 20th of March we disclosed the information during an email exchange with Keesing's helpdesk that their API was in the beta stages of testing. Up to this point both we and Buypass were under the impression that the NFC functionality was the only part of their API that was in the implementation and testing stages. This information was a bit of a relief to us as it gave an explanation to some of the problems we had experienced up until that point.

8.4.5 Concluding Demonstration

On the 10th of April we had a concluding demonstration for Buypass at their premises in Gjøvik. This involved a recorded demonstration of the implemented functionality within our application. The demonstration would mark the end of the application implementation, and we would here on out focus on increasing the readability, maintainability and the security aspect of our code in regards to bugs and vulnerabilities.

8.5 Version Control

During our development process we have deviated from the intended naming convention of versions, where as the last digit, the intended patch number, has been implemented more as a functionality implementation number. We usually worked in branches named after the functionality that was being developed in that branch (for example "camera" or "toolbar") and creating a new patch name for each added functionality simply got a little confusing, given that we were often working on multiple features in parallel.

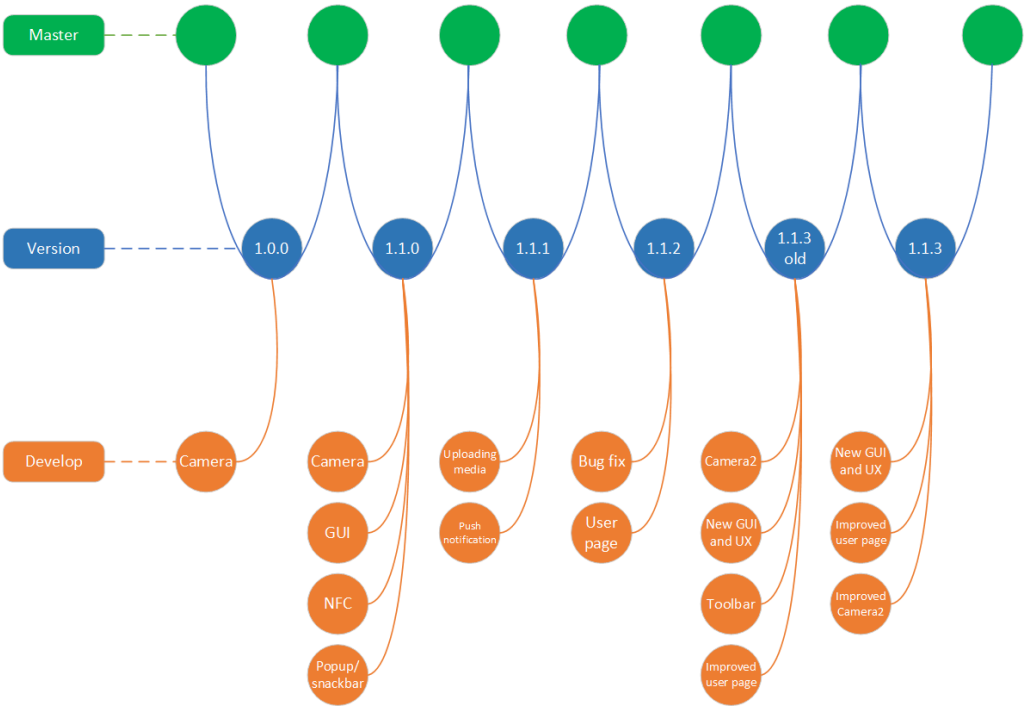


Figure 27: Workflow

Our version control is illustrated in figure 27. The figure represents how the master is the head of the branch and contains the latest working version of the application. Some versions consists of more implementation than others. This has to do with our agile development style, and the constant improvement or implementation to some areas of the application. See appendix N for more extensive information regarding the different versions.

8.6 Developing the Camera Application

The camera functionality is a central aspect of our application and for this reason we have given a lot of thought to which method to use for this. Given our minimum viable product mindset our first goal was to get our application up and running. Early in the process it was important to us to start experimenting with the Keesing API and in order to do this we needed to quickly implement a working camera application. We started out by implementing the camera application that is built in the phone. This implementation was fairly easy and it allowed us to start testing the Keesing API early on.

Still there were some significant down sides to this method, the most important one being that we had little control over the camera. Whenever the camera was launched we switched from our application to the camera application, so we had little control over how this would behave. The camera app would both look and behave differently on different types of phones. Not all phones allowed the user to retake a photo that was unsuccessful and only some devices let us decide whether to open the front or back camera. Moreover some phones allowed the user to edit the photo after taking it, which for obvious reasons should not be allowed.

Another great drawback of the built in camera application was that we were not able to customize our camera app to our specific needs. Many of the camera requirements were im-

possible to meet unless we implemented our own application and given that some of these were quite essential to both the functionality and user experience, we decided this was necessary.

Android has two options for implementing a custom camera, the camera API, which is not too hard to implement, but which was deprecated in API level 21 [34] and the much more complicated, but also more powerful Camera2 API. As our camera targets higher API levels we choose to use the Camera2 API.

It is fair to say that implementing the Camera2 API required significantly more work than our first solution and we did not start working on this until quite late in the process. The main reason why we did not implement this from the start was partly our MVP approach to the project and partly that early on we were all quite inexperienced in Java/Android programming, so understanding the rather complex Camera2 API seemed a bit out of reach. However throughout the project we learned a lot and after most of the basic functionality of our app was working we decided to give it another shot. Luckily through our development process we had learned a lot about Android programming, and when making a second attempt at implementing a custom camera, the Camera2 API did not seem as overwhelming as before.

9 Discussion

9.1 Development Method and Process

Looking back we think Scrum as a development method worked quite well in our case. We knew from the very beginning that the project requirements were far from set in stone, and as it turned out, many of the requirements changed or evolved throughout the project. Initially the project requirements were quite simple, as reflected in the original milestones in section 4.1.8 and in our project plan (appendix C). As Scrum does not require too detailed long term planning, it allowed us to start developing the application and experimenting with the Keesing's APIs at an early stage. This proved to be important, as we ran into quite a few time consuming challenges along the way. Another important point was that many of the product requirements did not become clear until after developing the initial features. With regards to this the meetings with the product owner, sprint retrospective meetings and sprint planning meetings were valuable. Here we got to discuss the features we had implemented and reflect on what needed to be implemented in order to further improve the product. The use of push notifications is a good example. Originally this was not part of the product requirements since we initially had little information about how Keesing would communicate the results of a verification process. After developing the initial prototype of the application we started testing the practical use of their APIs. We soon learned that quite often a document photo would end up at helpdesk, which means a result was not immediately available. This made us realize the need for a way to alert the user when a result is ready, so we added push notifications to our requirements. Such changes were usually not problematic since we re-evaluated the product requirements every sprint.

A challenge with using Scrum and Kanban the way we did is that it requires the team members to take a lot of initiative. As described in section 8.1.1 we initially struggled creating tasks that were small and well contained enough. Although we had daily scrums where we allocated tasks to each team member, there were still quite a few occasions where people felt unsure regarding what to work on. We tried handling this by improving the Kanban board and by talking about it in our daily scrums, but it still led to frustration and poor productivity at times. This problem was mostly evident in the first half of the project. As described earlier an initial problem was that we all had very distinct areas of competence. Some team members only had experience with certain modules on the server side, while some had only worked on the Android application, so if someone got stuck on a problem it was hard to find alternative tasks to work on in the meantime. After a while, when we made sure everyone had a more broad area of competence it got a lot better. Still, as our project had limited time and resources, making the most out of what we had was essential. By having a better course of action for what to do when these situations arose we probably could have increased the total productivity.

Another challenge was that as a consequence of our initiative-based way of allocating tasks, people tended to pick tasks involving new development from the *To do* column over "boring" tasks like refactoring, testing and code review. Given that developing new functionality was indeed the main focus of the development process this was not entirely a bad thing,

but it did have the consequence that these tasks were neglected or postponed at times. In general this was not a huge problem, but we did have some bugs, cases of unnecessary code duplication and so on that might have been avoided if testing, refactoring and code review had higher priority at an early stage.

9.1.1 Minimum Viable Product Approach

As mentioned earlier we have kept a Minimum Viable Product (MVP) approach to the development process. This product minded development has ensured the implementation of the necessary functionality, in order to provide a satisfactory prototype for Buypass. Given that we are all quite inexperienced as software developers, it was difficult to estimate the amount of time and resources required for the different tasks, so while planning the project it was hard for us to say how much functionality we could realistically expect to develop. We feel that the MVP mindset helped us identify the most important tasks and keep the most important employer requirements on top of our priority list. For example, it would not really have mattered if we had a great camera application if we were unable to get the files verified by Keesing. We also think it reduced the risk of spending too much time fine tuning a feature and not having time to implement other important features.

Still there were some down sides to this approach. For one thing it did mean that some features, security in particular, was somewhat neglected until quite late in the process as it was not essential to create a demo app. Another disadvantage was that it led to re-doing work in some areas because the initial implementation needed improvement.

The camera implementation is a good example of how the MVP approach was incorporated. As described in chapter 8.6 we started out by implementing a simple camera solution before deciding to start over and implement the more complicated Camera2 API halfway through the project. In retrospect this did mean we sort of did twice the work, and in theory implementing the Camera2 API right away would have saved us hours of work. However, in real life this might not have been the case. Early on we had a lot of trouble understanding how the Camera2 API worked, so if we would have committed to creating a custom camera application from the very beginning, it might have gone a long time before we had been able to start developing other modules that were dependent on the camera functionality. Figuring out how to communicate and upload files to Keesing turned out to be quite time consuming, so we believe it was the right decision to start working on this early on. We also discovered that after working on the project for a few months we had learned a lot. When we decided to start working on the custom camera, we realized that the Camera2 API did not seem as difficult as it was at the earlier stage, so we were probably able to create the camera faster than we would have earlier in the project. Hence we might not have saved all that much time/resources by implementing the Camera2 API right away.

9.1.2 Testing and Development

Looking back at the testing aspect of the project we probably would have benefited from automated API tests. This would have allowed easier validation of the functionality in our API when changes to the source code occurred. We spent quite a lot of time performing manual testing in Postman, so automated API tests would probably have saved us quite some time. We tried implementing automated tests in the Android application using the Espresso library. The problem was that large parts of the application require the user to take a photo or read an NFC tag, and these processes are difficult to simulate through automated testing.

We did write a few tests in order to check that all user interface elements appeared the way they were supposed to, but since we had to test most of this functionality manually, these tests did not really provide much valuable information.

9.1.3 Implementing an API in Development

As described in chapter 8, the Development Process chapter, there were quite a few cases of miscommunication in the beginning of the project that led to a quite significant delay in the development process. This was largely due to some unfortunate circumstances, and we were not able to have a clarifying meeting with Keesing until several weeks into the project. We do take some self-criticism in this situation, as we could have made absolutely sure we implemented the correct API from the beginning. We may not have been able to get an earlier meeting, but we could perhaps have worked harder to contact Keesing directly (and not just via Buypass) in order to get answers to our most pressing questions. Still we think we handled the situation quite well. Instead of focusing on the setbacks we saw the conducted work as part of the process of getting to know Keesing and their methods, and we immediately started implementing the correct API. Our idea of developing the client application and the server in parallel worked well in this case. When implementing the new API, we had to change large parts of the server code, but since the way the application code was independent to the communication with Keesing, we did not have to make any changes here. This experience also made us pay more attention to keeping a close dialog with both Keesing and Buypass in order to avoid similar issues in the future.

Throughout the project we have been focused on always being proactive and try not to let problems out of our control affect our productivity. In the beginning of the project, bottlenecks like miscommunication and lack of documentation slowed us down quite a lot. We frequently (while waiting for a response from Keesing) spent hours trying to solve a problem that could easily be solved once we got their response. As mentioned in the previous chapter the problems were sometimes related to changes or errors on Keesing's API, which meant that we occasionally spent hours or even days trying to fix code that was not actually malfunctioning.

However after a while we learned that in most cases, the best thing we could do when encountering a strange or unexpected error was to send an email about the issue right away and then proceed to work on modules that were not dependent on Keesing's API. Neither the camera application nor the NFC reader depend on any calls to Keesing, and since they both required quite a bit of work, we were able to keep our productivity up by shifting our focus to these modules. In this way we always had work to do, and we avoided spending an unnecessary amount of time fixing issues that might not even be related to our code at all.

9.2 User Testing

In retrospect we realize that lack of user testing is a major weakness in our project. As described in chapter 8 we have mainly focused on creating a product that satisfies the product owner and in the middle of developing the application based on their feedback, we sort of ”forgot“ the importance of feedback from target end users. User friendliness is an important requirement for our project and there is no doubt that it would be useful to get feedback from people with no prior knowledge of the application. Since everyone in the group knows the application inside out it is difficult for us to evaluate how the user experience of a typical

end user would be.

This would be of particular relevance to the camera application. As described in section 3.4.2.1 document photos that are of sufficient quality could be verified automatically, and if the quality is insufficient, they are redirected to Keesing's helpdesk and the response time is a lot longer. Given that the main purpose of the camera application design is to make it easy for the user to upload valid photos, it would be very useful to get a group of users to test the application using a valid ID. Ideally they would all pass all three steps, but if it turned out a large portion of the users were unable to take sufficient photos to get their document approved right away, we would have to reconsider the user instructions and camera design.

We did originally plan on conducting user testing once we had the main functionality in place and then make adjustments based on the user feedback. Unfortunately as we spent a lot of time fine tuning the user interface and functionality according to feedback from Buypass, by the time we felt we had developed a prototype fit for user testing, we realized we had underestimated the amount of planning and work required to conduct valuable user testing. At that point in the process we would have had little time to actually make adjustments based on the test results. We also knew that we had quite a bit of work left for fixing bugs and vulnerabilities, and cleaning up our code. It was important to us that we could deliver code that is readable, easily maintainable and well documented, so we decided our time would be better spent getting the project ready for Buypass to take over than conducting more or less improvised user testing.

In retrospect it is hard to say whether or not we made the right call at the end of the project, but we do believe we should have planned and set aside more time for user testing earlier on in the process. We have indeed received very useful feedback from our meetings with Buypass, but real user feedback would have provided another dimension to it.

9.3 Backend Solution

We originally considered several different approaches for our backend solution. Originally our plan was to have a "serverless" solution using Google Cloud Functions. Using Cloud Functions would have meant minimal set up and configuration of our server, automatic scaling and basically no need for server maintenance. We had already decided on using Firebase for our database and for push notifications, so using Cloud Functions would have meant this could be implemented quite seamlessly. Unfortunately, as mentioned in chapter 8, Development Process, some miscommunication/misunderstandings led us to believe that we needed a static IP address to implement the Keesing API, and since this was not possible when using Cloud Functions we decided to create a Node.js server on a VM[1].

During our initial meeting with the Norwegian Keesing representative (see 8.4.2) we discovered that we were to implement a different API that did not require a static IP address, we considered going back to our original plan of using Cloud Functions. This would mean re-doing all the work we had done in the meantime, but if it would mean easier integration with Firebase, easier implementation of a RESTful API etc, we might still have saved time in the long run. Still, we decided to keep our backend solution. One reason was that we had already done a fair amount of work that would probably take quite some time to re-do in Cloud Functions and we were already worried that we had gotten behind schedule due to implementing a different API than the one we ended up using. But more importantly,

[1]Virtual Machine

although Cloud Functions are written in JavaScript/TypeScript the syntax is a bit peculiar, so transferring it to a backend solution not using Cloud Functions would mean quite a lot of code would need to be changed. We knew that if Buypass were to further develop our solution, they would run the backend service on their own servers, so we figured having a Node.js server on a VM would mean easier integration into Buypass' existing solutions.

9.4 Application Design

As none of the group members have any previous experience with app development, developing our product has been a ”learn as you go“ process. We have learned a lot about Android development throughout the project and there is no doubt that if we were to start over, we probably would have done things a bit differently. This is perhaps most apparent in the design pattern of the application. As described in chapter 5 our application is mainly built up by Activity classes with their XML layout files and a few supporting fragments and other helper classes. The idea was that the client application logic would be lightweight and separating everything into Activities would make both implementing and reading our code quite straightforward. It would also allow us to start developing right away and to get familiar with Android Studio and Android programming.

With hindsight we do see that there may have been more organized and perhaps better ways to design the app. For one thing it may have been beneficial to reduce the number of Activities in the application. We have mentioned responsiveness as an important UX requirement, and in some cases using fragments instead of activities could perhaps made our app more responsive. This is why we use a fragment, and not an activity for the User Page (”Min Side“). This screen can be reached from anywhere inside the app, so instead of creating a new activity every time, we just re-use a fragment. This approach could also be done is some other activities. Examples could be UploadActivity and Camera2Activity. These activities are reused multiple times throughout the verification process and instead of having to create a new activity every time we take a photo or upload a file, we could have created fragments that could have been re-used. This could have given us some potential benefits:

- It would be easier to pass information between screens. Whenever the either of these activities are launched we need to pass a string extra letting the activity know which type of photo (document or face) to take or which type of file to upload and UploadActivity needs an extras containing the file uri. If using fragments instead, we could just retrieve this information from the calling activity.
- We would not have to create a new activity every time.
- Each verification step could have a neat and logical structure of an Activity class that has one fragment for user information, one for taking a photo/video and one for uploading the file attached to it.
- We could attach the toolbar to the Activity class and not have to repeat it in every one of the fragments (less duplication of code).

One could probably also argue that we could have made our code more modular by separating the background logic from the Activity classes. Android Developers recommend that Activities are only used for handling the user interface and interacting with the operating system [16] and mostly this is exactly what our activity classes do. However some of them also have a bit more background logic, like communicating with the server (LoginActivity,

PassportActivity, LicenseCameraActivity, UploadActivity, VerificationCompleteActivity etc) or analyzing and manipulating data before presenting it to the user (MinSideFragment). It may have been beneficial to extract this logic into a separate module in order to make the code more maintainable and not to mention easier to test. Still, most of our activities/fragments, with the possible exception of MinSideFragment, have such simple background logic, that we do not consider it too much of a problem to keep this inside the activity/fragment. Another advantage is that having the server calls inside the activity/fragment classes simplifies the synchronization of concurrent work as all code that should be executed after receiving a response can be called in the onResponse()-callback.

9.5 System Security

As mentioned in the Requirements chapter, chapter 3, we did not aim towards implementing all security requirements that would have been necessary for a real world ID proofing application. Still, it was important to us that we would avoid the most obvious threats. Given our MVP[2] mindset we started working on system security quite late in the development process, as functionality took priority over security in our project requirements.

In retrospect we see that although this may have let us start developing functionality at an early stage, it may also have increased the total workload of our project as we had to rewrite quite a bit of existing code in order to improve application and server security. For some security requirements, like validating/sanitizing input and limiting the number of requests per minute, this worked quite well. Through the developing the main functionality we realized where this was needed, and once we decided on allowed values and how to implement it, adding these security measures to the app went quite fast.

For more complex security requirements on the other hand, this approach did not work as well. Up until quite late in the process our database security consisted of Firebase Security Rules, which does not allow anyone except our server to change or access data in the database. However, a little too late we realized that anyone could send a POST request with the userId of any user to *www.verifyid.ml/firebase/getallverifications* and get a list of all their verification results, including personal information like document number, personal identification number etc. To overcome this security hole we created more or less an ad hoc solution where a UUID[3] was used to verify the identity of the user making the request (as described in chapter 6.6.2.4). However we do realize that this is not exactly an optimal solution. Authenticating the client requests would probably have been solved in a better either by using session cookies that are generated on the server side after the user has successfully logged in (we assume Buypass' login is safe) or by a token based authentication.

The main reason why we did not implement this was that early on in process we did not see the need to authenticate users since log in in would be handled by Buypass. Later we realized that even though the user will already be logged in when entering our application, we still need to authenticate requests they make to the server. Unfortunately this realization happened at a time in the process where we did not have time to implement this feature.

[2] Minimal Viable Product
[3] Universal Unique Identifier

9.6 Evaluation of the Verification Process

During the entire lifetime of the project we have given quite a lot of thought to whether or not we consider the solution we have implemented to be a reliable and safe way to perform digital ID proofing. As described in section 6.6.3 we have uncovered quite a few weaknesses with the actual verification process. Since both the document check and face comparison can be fooled quite easily, anyone with access to a copy of another person's ID document could easily impersonate that person and get an ID in their name. All they have to do is take a document photo of the copy, take a face photo of the document image and then just use their own face for the liveness check.

Of course, in a real world setting, stealing someone's identity is not quite that easy. When the app is integrated into Buypass' existing app, the document name would be cross checked against the registered user name, so an impostor would first have to log into the victim's Buypass account and then manage to issue the ID to himself instead of the other person. We assume that Buypass' login is safe, so passing this obstacle would be quite a challenge to most attackers. Still, the whole motivation for creating an ID verification application is that a regular login is not safe enough to order an ID with security level High, so if our application does not include an additional security step it would serve no purpose.

The way we see it, our implementation of digital ID proofing is a good starting point for further development, but not yet a complete solution. One could argue that it is difficult (although definitely not impossible) to obtain a high quality copy of someone's passport or driver's license, and in that way our application does make it significantly more difficult to impersonate someone else. Nevertheless we have considered some suggestions to things that could have been done differently.

Once Keesing releases functionality for verification through reading the RFID chip with the phone's NFC reader, it would be possible to implement what we would consider a much more reliable document check than by using a document photo. NFC would eliminate the possibility of going through the verification process without access to a real ID document and it could open up to a range of possible additional security measures.

- If the application could obtain a CVCA certificate to do Extended Access Control and the phone has a fingerprint reader, one could add fingerprint verification to the process.
- Passive Authentication and Active Authentication could be used to check if the RFID chip has been tampered with or cloned.
- One can extract high resolution face images from the passport for the face verification step.

We also think there might be alternative ways to do the face comparison and liveness check. The way it is now, the liveness check does not perform any kind of face recognition, so the person in the video recording could easily be a completely different person than the one in the document image. We also know that the face match can be fooled by a taking a photo of another photo. However, if face recognition could be implemented as part of the liveness check, we could possibly achieve a safer and more user friendly process. For one thing the verification process would be one step shorter, which means less work for the user. Secondly it would be more difficult to trick the face comparison step. For an impostor it would probably be easy to find a face photo of a potential victim as for most people this could be achieved through a simple Google search. Finding a good quality face video on the other hand, would

probably pose more of a challenge and as the liveness check seems quite good at detecting whether or not the video shows a real person, this could be a method that would be more resilient to fraud.

10 Conclusion

When developing our product the main objective has been to create a prototype of an Android application that can showcase how digital ID proofing could be implemented using the APIs provided by Keesing Technologies. The current method for ID verification is expensive, inconvenient and time consuming, and we wanted to demonstrate an alternative solution. Through our project we have fulfilled this task to a certain extent, by implementing the functionality Keesing Technologies' API has provided and tied it all together within our Android application.

As the time frame of the project was quite limited we had to make some tough decisions when deciding which aspects we would focus on. We implemented a three step solution consisting of a document check, face comparison and liveness check. Each step is meant to target a distinct security aspect; document validity, user identity and determining user presence. All these steps depend on the camera application. The implementation of the Camera2 API has been of big importance as it has allowed us to customize the camera according to our specific requirements. This makes for an easier, more secure verification process, and a more user friendly experience.

Although NFC verification is not yet supported by Keesing's API, this technology is very likely to be used for passport verification in the future. We took the NFC functionality as far as we could by extracting the information that could be extracted with our security clearance. This will hopefully make future development easier, and it demonstrates the possibilities of NFC.

We also made it a priority to create an attractive and user friendly product. We wanted to motivate our stakeholders to keep developing the product after our project is over, and we believe creating a nice looking application that fits their typical application design is a good selling point. This was an important motivation for us when we decided to spend quite a lot of time developing a custom camera application tailored for our needs and paying extra attention to details that might affect the user experience in a positive way.

Another objective has been to get experience working with the ID proofing technologies Keesing provides and gain insights to their strengths and weaknesses on behalf of Buypass. There is little doubt that the application we have developed is not yet a complete, safe way to perform digital verification of identity. The way we see it, this is partly due to limitations in our implementation and partly due to limitations in the currently provided functionality. Both the application and the verification process itself have some significant flaws, but we hope that our work can provide a good starting point for further development.

We do see great potential in digital ID proofing as a concept, especially if incorporating the use of NFC, and we do believe that a refined, thoroughly tested ID proofing application could eventually be a very good alternative to the manual process that is used today.

After working on this project for about five months we have learned a lot about software development and extended our skills within programming and team work. This has been the first long term project any of us have participated in and we feel that it has given us a taste of

software development in a real world context, as opposed to a normal school project. As our report demonstrates we have run into quite a few bumps in the road, but we believe this has been an important learning experience when it comes to dealing with unexpected challenges. The importance of keeping a close dialog with stakeholders have possibly been one of the most valuable lessons from this project. When running into problems, we have learned to be proactive and at any given day focus on problem solving and the factors we can control. We think these are important skills to bring into our professional lives.

Another great learning experience has been the large range of technologies we have been able to explore during this project. The end product itself uses many new, interesting technologies which have been exciting to work with. Additionally we have learned a lot from having to research and evaluate different technologies and tools when deciding which ones to include in the product. Overall we have truly enjoyed working on this project and we believe it has been a very important experience in order to prepare us for a career in software engineering.

Bibliography

[1] Buypass. Amount of users using buypass id. https://www.buypass.no/selskapet. [Online; accessed 14-May-2019].

[2] Wikipedia contributors. Application programming interface — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=895664282, 2019. [Online; accessed 9-May-2019].

[3] Wikipedia contributors. Representational state transfer — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=896116624, 2019. [Online; accessed 9-May-2019].

[4] Wikipedia contributors. Secure shell — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Secure_Shell&oldid=890240515, 2019. [Online; accessed 26-April-2019].

[5] Firebase. Firebase cloud messaging. https://firebase.google.com/docs/cloud-messaging. [Online; accessed 9-May-2019].

[6] Nærfeltskommunikasjon. https://no.wikipedia.org/wiki/N\T1\aerfeltskommunikasjon. [Online; accessed 25-april-2019].

[7] Fra 1. april kommer du ikke inn i USA uten riktig pass. https://www.dagbladet.no/tema/fra-1-april-kommer-du-ikke-inn-i-usa-uten-riktig-pass/60436340. [Online: accessed 08-March-2019].

[8] Overview security mechanisms in ePassports. https://readid.com/blog/Overview-security-mechanisms-in-ePassports. [Online; accessed 09-may-2019].

[9] Joshua Porter. Principles of user interface design. http://bokardo.com/principles-of-user-interface-design/. [Online; accessed 3-May-2019].

[10] Samella Garcia. 7 key attributes of a quality ui. https://www.webdesignerdepot.com/2016/12/7-key-attributes-of-a-quality-ui/. [Online; accessed 2-May-2019].

[11] Wikipedia contributors. Principles of user interface design — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Principles_of_user_interface_design&oldid=887071432, 2019. [Online; accessed 6-May-2019].

[12] Lov om tiltak mot hvitvasking og terrorfinansiering (hvitvaskingsloven). https://lovdata.no/dokument/NL/lov/2018-06-01-23/. [Online; accessed 06-May-2019].

[13] Security Mechanisms in electronic ID documents. https://www.bsi.bund.de/EN/Topics/ElectrIDDocuments/SecurityMechanisms/securEAC/eac_node.html. [Online; accessed 09-may-2019].

[14] Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010.

[15] Kanban vs. scrum: What are the differences? https://leankit.com/learn/kanban/kanban-vs-scrum/.

[16] Android Developers. Guide to app architecture. https://developer.android.com/jetpack/docs/guide. [Online; accessed 02-May-2019].

[17] Store and sync app data at global scale. https://firebase.google.com/products/firestore/. [Online; accessed 20-February-2019].

[18] Get Started with Cloud Firestore Security Rules. https://firebase.google.com/docs/firestore/security/get-started. [Online; accessed 20-February-2019].

[19] javascript.info. Promise. https://javascript.info/promise-basics. [Online; accessed 1-May-2019].

[20] android.hardware.camera2. https://developer.android.com/reference/android/hardware/camera2/package-summary.html. [Online; accessed 14-March-2019].

[21] Mateusz Dziubek. The least you can do with Camera2. https://android.jlelse.eu/the-least-you-can-do-with-camera2-api-2971c8c81b8b. [Online; accessed 16-March-2019].

[22] Video Recording with Camera2 API in Android. https://androidwave.com/video-recording-with-camera2-api-android/. [Online; accessed 26-march-2019].

[23] AutofitTextureView. https://github.com/googlesamples/android-Camera2Basic/blob/master/Application/src/main/java/com/example/android/camera2basic/AutoFitTextureView.java. [Online; accessed 20-March-2019].

[24] MediaRecorder. https://developer.android.com/reference/android/media/MediaRecorder. [Online; accessed 18-March-2019].

[25] CountDownTimer. https://developer.android.com/reference/android/os/CountDownTimer. [Online; accessed 30-April-2019].

[26] JMRTD API Doc. https://www.javadoc.io/doc/org.jmrtd/jmrtd/0.7.11. [Online: accessed 08-March-2019].

[27] e-Passport NFC Reader. https://github.com/tananaev/passport-reader. [Online; accessed 07-march-2019].

[28] Anyline SDK. https://anyline.com/.

[29] JavaScript Promises: an Introduction. https://developers.google.com/web/fundamentals/primers/promises. [Online; accessed 03-May-2019].

[30] Machine Readable Travel Documents, Part 3: Specifications Common to all MRTDs. https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf. [Online; accessed 12-March-2019].

[31] Buypass AS. Buypass Go SSL. https://www.buypass.no/ssl/products/acme. [Online; accessed 26-April-2019].

[32] JS Foundation. Rules:node.js and common.js. https://eslint.org/docs/rules/. [Online; accessed 3-May-2019].

[33] Keesing Technologies. Verification services api (v1.0). https://verificationsrv.docs.apiary.io/#. [Online; accessed 1-May-2019].

[34] Camera. https://developer.android.com/reference/android/hardware/Camera. [Online; accessed 11-March-2019].

[35] json.org. Introducing json. https://www.json.org/. [Online; accessed 9-May-2019].

[36] Wikipedia contributors. Plug-in (computing) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Plug-in_(computing)&oldid=892555943, 2019. [Online; accessed 30-April-2019].

[37] Wikipedia contributors. Java (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=895521846, 2019. [Online; accessed 6-May-2019].

[38] Wikipedia. Ssh file transfer protocol — wikipedia,. https://no.wikipedia.org/w/index.php?title=SSH_File_Transfer_Protocol&oldid=15265562, 2015. [Online; accessed 24-April-2019].

[39] Node.js Foundation. About node.js. https://nodejs.org/en/about/. [Online; accessed 30-April-2019].

[40] Node.js Foundation. Express. https://expressjs.com/. [Online; accessed 25-April-2019].

[41] JS Foundation. Eslint. https://www.npmjs.com/package/eslint. [Online; accessed 3-May-2019].

[42] Android Developer. Volley overview. https://developer.android.com/training/volley. [Online; accessed 30-April-2019].

[43] Square. Retrofit 2. https://square.github.io/retrofit/. [Online; accessed 30-April-2019].

[44] Future Studio. Retrofit 2 tutorial. https://futurestud.io/tutorials/retrofit-2-how-to-upload-files-to-server. [Online; accessed 30-April-2019].

[45] Square. Okhttp. https://square.github.io/okhttp/. [Online; accessed 30-April-2019].

[46] NGINX INC. What is nginx. https://www.nginx.com/resources/glossary/nginx/. [Online; accessed 24-April-2019].

A Project assignment

Oppdragsgiver Buypass AS

Kontaktperson Jørn Magne Raastad eller Alexander Dreyer Johnsen

Adresse Nydalsveien 30A, 0484 Oslo

Telefon 926 33 060 / 977 65 888

Epost jorn.raastad@buypass.no eller Alexander.Johnsen@buypass.no

ID-verifisering på nett

Buypass leverer blant annet digitale ID-er på nivå Høyt, som muliggjør at en person kan identifisere seg gjennom digitale tjenester, både privat og i jobbsammenheng (for eksempel ID-porten). For å få opprettet og utdelt en digital ID på nivå Høyt må brukerens fysiske identitet verifiseres. I dag gjøres dette gjennom en manuell prosess, hvor en må møte opp personlig f.eks. på et postkontor, fylle ut papirer og identifisere seg ved bruk av for eksempel pass.

Dagens prosess er manuell, tidkrevende, kostbar og vanskelig å integrere med digitale løsninger. Buypass ønsker å teste ut en ny digital metode for å verifisere og utstede en digital ID på nivå Høyt.

Oppgaven

Utvikle en digital løsning for å verifisere en brukers identitet. Løsningen bør være mobil web-basert og basere seg på å kombinere eksisterende byggeklosser for å bygge en slik løsning – for eksempel ved å lese digitalt innhold i moderne pass ved hjelp av NFC på mobiltelefon, bruke kameraet på mobilen til å ta bilde av et ID-dokument, sammenligne bilde av ansikt tatt med mobilkamera med bilde på ID-dokument, online videosamtale med kundeservice osv. Brukervennlighet og sikkerhet må være i fokus. Oppgaven vil gå opp et nytt og uutforsket territorium innen ID verifisering og bruk av nye tjenester i Norge.

Ønsker til endelig løsning

- Kjører i en nettleser på telefon, desktop og nettbrett, men med hovedfokus på mobil
- Tilbyr brukerregistrering med oppslag mot folkeregisteret
- Opplasting av bildekopi av pass med verifisering
- Mulighet for videosamtale mellom bruker og Buypass for endelig verifisering

Programmering

Oppgaven vil bl.a. bestå i:

- Integrasjon mot APIer fra Buypass og våre partnere
- Web-app utvikling, foretrukket rammeverk er React
- Grensesnitt mellom tjeneste, server og database
- Verifisere, teste og utvikle et fungerende grensesnitt
- Teste og utvikle mot ny teknologi
- Utfordre et eksisterende system og tankegang med nye tanker og teknologi

Studentgruppen vil gjennom prosjektet få erfaring med:

- Utvikling av web-app med Reactjs
- UX
- Digitale IDer og sikkerhet knyttet til slike IDer
- Ny teknologi innen ID-verifisering

Oppgaven passer best for en gruppe på 3 til 4 personer, hvor minst én har interesse for og litt bakgrunn fra UX/tjenestedesign.

Om selskapet

Buypass AS er en ledende leverandør av brukervennlige og sikre løsninger for elektronisk identifikasjon, elektronisk signatur og betaling. Vi tilbyr en komplett portefølje av tjenester for identifisering og kryptering av informasjon gjennom hele den elektroniske verdikjeden. Buypass er også Norges eneste utsteder av internasjonalt godkjente SSL-sertifikater.
Selskapets største kunde og samarbeidspartner er Norsk Tipping, der Buypass leverer tjenesten for identifisering og betaling på alle digitale flater (mobiltelefon, nettbrett, web).
Buypass ble etablert i 2001, og har siden starten satset tungt på å bygge opp et unikt kompetansemiljø for utvikling av brukervennlige sikkerhetsløsninger som tilfredsstiller markedet og myndighetenes strengeste sikkerhetskrav. Buypass leverer løsninger til bruk både i forbrukermarkedet, private bedrifter og offentlige virksomheter.
Selskapet har ca. 80 fast ansatte, samt et varierende antall innleide/engasjerte fordelt på våre kontorer i Oslo og på Gjøvik. Kontoret på Gjøvik er lokalisert i gangavstand fra NTNU campus, og vi rekrutterer jevnlig fra NTNU Gjøvik både lokalt og til kontoret i Oslo.

B Project agreement

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Buypass AS

(oppdragsgiver), og

Jørgen Berntsen, Vegard Elshaug Almås, Karoline Wisløff Rud, Ruben Isaksen Cheung

(student(er))

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

1. Studenten(e) skal gjennomføre prosjektet i perioden fra Jan 2019 til Jun 2019 .

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

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra NTNU på Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

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

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

Merknad: Kildekode skal unntas offentliggjøring, men tilgjengeliggjøres for faglig veiledning og sensur.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Deepti Mishra, Rune Hjelsvold

Oppdragsgivers kontaktperson (navn): Jørn Raastad

Student(er) (signatur): _____ dato 15/1-19

Karoline W. Rud _____ dato 15/1-19

Jørgen Bentzen _____ dato 15/1-19

Vegard F. Thuen _____ dato 16/1-19

Oppdragsgiver (signatur): Jørn Raastad Digitalt signert av JØRN MAGNE RAASTAD DN: c=NO, o=BUYPASS AS-983163327, cn=JØRN MAGNE RAASTAD, serialNumber=9578-4050-129031142 Dato: 2019.01.17 14:35:00 +01'00' dato _____

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): Marius Pedersen Digitally signed by Marius Pedersen Date: 2019.01.28 13:06:11 +01'00' dato _____

C Project Plan

Prosjektplan

Karoline Wisløff Rud karoliwr@stud.ntnu.no
Ruben Isaksen Cheung rubenic@stud.ntnu.no
Vegard Elshaug Almås vegardea@stud.ntnu.no
Jørgen Berntsen jorgbern@stud.ntnu.no

24. april 2019

Innhold

1 Mål og rammer

1.1 Bakgrunn

Vår oppdragsgiver, Buypass AS, er en ledende leverandør av brukervennlige og sikre løsninger for elektronisk identifikasjon, elektronisk signatur og betaling på mobiltelefon, nettbrett og web. Løsningene tas i bruk både av forbrukere, private bedrifter og offentlige virksomheter. Digital ID er en viktig del av dette. De digitale ID-ene kommer på ulike nivåer. Slik det er i dag innebærer registrering for, og utdeling av digital ID på nivå Høyt at brukeren må verifisere sin identitet ved å fysisk møte opp på et postkontor for å fylle ut papirer og identifisere seg med gyldig dokumentasjon. Dette er både tidkrevende og kostbart, så Buypass ønsker nå å utvikle en ny digital metode for å verifisere fysisk identitet, slik at digital ID kan utstedes uten personlig oppmøte.

1.2 Prosjektmål

1.2.1 Effektmål

Applikasjonen skal benyttes innenfor et område som i dag kan sees på å være gammeldags etter dagens standarder. Det blir applikasjonen sin jobb å avskaffe den tid- og ressurskrevende prosessen for verifisering ved bestilling av elektronisk ID på nivå Høyt. Per dags dato er det ansatte på postkontor sin jobb å verifisere personer som bestiller, men de har gjerne lite erfaring med dette og feil kan forekomme.

Prosessen forutsetter at den ansatte ved postkontoret klarer å skille et falskt ID-dokument fra et ekte, og verifiserer at personen med tilhørende id-dokument er personen den utgir seg for å være. Postkontor har gjerne korte åpningstider, og er ikke like tett lokalisert over hele Norge. Det kan derfor være både tid- og ressurskrevende for ansatte, eller enkeltpersoner å få utstedt disse elektroniske ID-ene. I helsesektoren benyttes det i stor grad ID-kort, og de har egne ansatte dedikert til håndtering av ID-kort, og disse vil kunne løses ut ved å benytte denne løsningen.

Vi har ikke tilgang på data over hvor mye ressurser som går med på dagens løsning, men vi regner med at det skal være mulig å kunne redusere disse tallene betraktelig.

1.2.2 Resultatmål

Målet for prosjektet er å lage en fungerende native Android applikasjon for mobil og nettbrett. Denne skal kunne sjekke om et godkjent ID-dokument (pass eller førerkort) er gyldig ved hjelp av enten NFC-leser (kun pass), eller ved bilde av ID-dokumentet. Deretter skal brukeren kunne ta et bilde av seg selv med kameraet på mobilen/nettbrettet, og applikasjonen skal kunne verifisere at dette er den samme personen som på ID-dokumentet.

1.3 Rammer

Vi skal lage en native app for Android-telefoner. Vi vil ta i bruk API fra Keesing AuthentiScan for gyldighetssjekk av ID-dokumenter og sammenlikning av bilde på ID-dokument med bilde tatt med kamera.

2 Omfang

2.1 Fagområde

2.1.1 NFC og lesing av passinformasjon

NFC (Near Field Communication) er en teknologi som gjør at to elektroniske enheter kan utveksle informasjon dersom de bringes innen fire centimeter fra hverandre[19]. Systemet brukes blant annet for kontaktløs betaling, i nøkkelkort og i elektroniske pass. Alle norske pass utstedt etter oktober 2005[12] er e-pass med RFID-brikke (Radio-Frequency Identification) som kan leses ved hjelp av NFC[14]. BAC-protokoll (Basic Access Control) beskytter RFID-brikken slik at passet ikke kan leses for eksempel gjennom lommen til en person. Denne fungerer slik at MRZ (Machine Readable Zone) først må leses. En individuell tilgangsnøkkel kalkuleres ut ifra det som leses her, og leseren bruker denne nøkkelen for å få tilgang til å lese av RFID-brikken[4].

2.1.2 UX

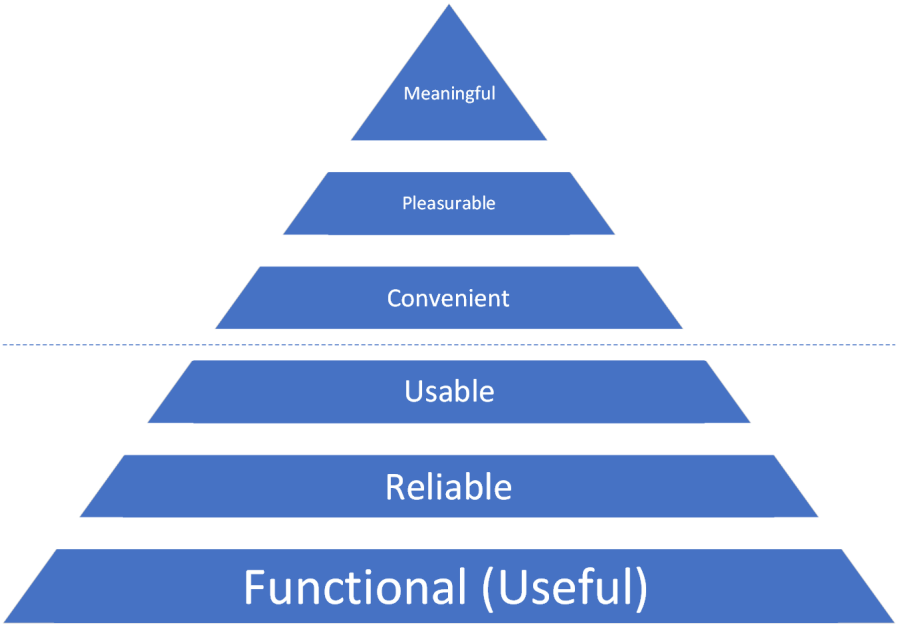
UX-design (User experience design) går ut på å styrke brukernes tilfredsstillelse ved å fokusere på å forbedre brukervennlighet, tilgjengelighet og glede ved interaksjon med produktet[18]. Med utgangspunkt i UX-pyramiden, se figur 1, vil vi fokusere på de nederste byggestenene i pyramiden først. Dette sikrer en funksjonell applikasjon, og er nødvendig før man kan fokusere på å finpusse den for å kunne tilby den beste brukeropplevelsen.

2.1.3 Keesing AuthentiScan

Keesing Technology er et nederlansk selskap som har verdens største samling av ID-dokumenter fra over 200 land (pass, ID-dokumenter, førerkort, visum og liknende)[8]. Selskapet samarbeider med blant annet internasjonale myndigheter, ambassader og konsulater for å holde høy kvalitet på tjenestene sine. Selskapet har utviklet produktet AuthentiScan, som er en automatiskert programvare for å autentisere ID-dokumenter[9]. Pass kan autentiseres enten ved hjelp av scanning av den elektroniske informasjonen i passet (NFC) eller ved å ta bilde av passet og autentisere det mot Keesings database[10].

2.1.4 Node.js

For utvikling av server-siden ved applikasjonen vår vil vi benytte Node.js. Node er et open-source cross-plattform system for kjøring av server- og nettverksap-



Figur 1: UX-pyramiden laget i Microsoft Visio

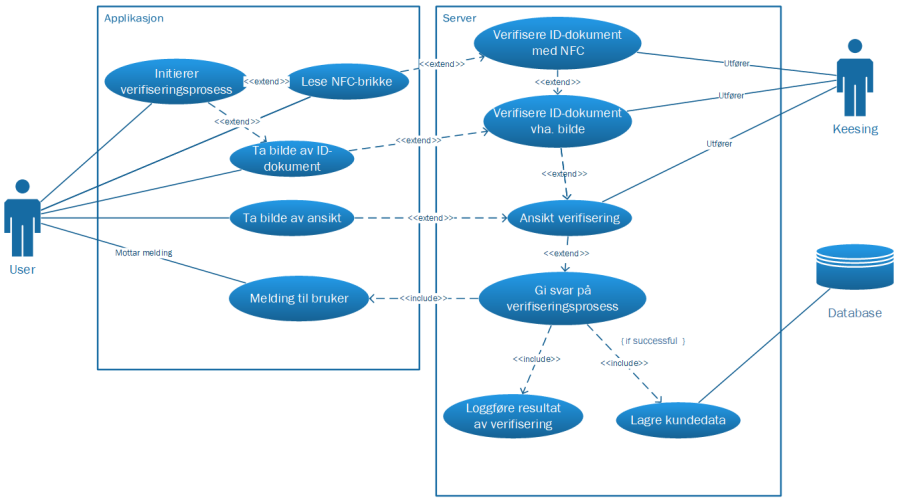
plikasjoner, og det benyttes for å eksekvere JavaScript utenfor nettleseren[20]. Vi vil benytte Express.js som er et rammeverk basert på Node, og gjør det lettere å utvikle serversiden ved applikasjonen[15].

2.1.5 GDPR

Som det kommer frem i oppgavebeskrivelsen, vil denne oppgaven innebære at det skal håndteres sensitive data. Pass og førerkort skal behandles ved scan eller bilde, og vil bli sendt til Keesing Technologies for verifisering. Av denne grunnen blir det viktig å ta hensyn til GDRP (General Data Protection Regulation), på norsk kalt personvernfordringen. Det vil være en høy prioritet å sørge for at personopplysninger ikke kommer på avveie ved lagring eller behandling, men også å sørge for at dersom personopplysninger lagres gjøres dette i henhold til GDPR[1, 2].

2.2 Avgrensing

Oppgaven vår er i samhandling med arbeidsgiver begrenset til utvikling av en native applikasjon for mobil og tablet med Android operativsystem. Oppdragsgiveren er hovedsakelig ute etter en prototype på applikasjonen, så vi vil foreløping ikke lage noen iOS versjon. Dersom Android-versjonen blir vellykket, vil det selvfølgelig være ønskelig å utvikle en iOS versjon, men dette vil ikke inngå i



Figur 2: Use Case-diagram

vårt prosjekt. Dersom det blir tid og mulighet kan en eventuell utvidelse være å lage en web-basert versjon av applikasjonen som kan benyttes på mobil, tablet og pc. Dog vil dette være en stor utvidelse og sannsynligvis være vanskelig å rekke i tide. Hovedfokuset vil være å implementere all funksjonalitet vi ser for oss nå i startfasen, og dersom noe funksjonalitet nedprioriteres underveis kan den hentes frem igjen dersom vi ser vi har tid til å implementere det.

2.3 Oppgavebeskrivelse

Vår oppgave er å lage en native mobilapplikasjon for Android som gjør det mulig å verifisere brukeres identitet slik at digital ID på nivå Høyt kan utstedes. Dette skal gjøres ved at brukeren leser av innholdet i et gyldig identifikasjonsbevis ved hjelp av NFC eller kameraet på mobilen, og disse opplysningene sjekkes mot Keesing Technologies sitt API. Deretter vil brukeren ta et bilde av seg selv dersom ID-dokumentet godkjennes. Dette bildet sammenliknes deretter med bildet eller ansiktsbiometrien i ID-dokumentet for å verifisere brukerens identitet, se figur 3.

2.4 Use Cases

Aktuelle Use Cases er beskrevet nedenfor. En oversikt over Use Cases og aktører er fremstilt i figur 2, og typisk interaksjon med applikasjonen vises i sekvensdiagrammet i figur 3.

Navn:	Initiere verifiseringsprosess
Aktør:	Bruker
Mål:	Sette igang verifiseringsprosessen
Beskrivelse:	Brukeren ønsker å registrere seg for å motta ID på nivå Høyt og starter derfor verifiseringsprosessen.

Navn:	Ta bilde
Aktør:	Bruker
Mål:	Ta bilde av ansikt eller ID-dokument
Beskrivelse:	Brukeren tar et bilde av ID dokumentet som skal autentiseres eller ansiktet sitt for verifisering. Dersom brukeren skal laste opp bilde av ID-dokument vil det være mulig å ta bilde, og å laste opp fra kamerarullen. Bilde for ansiktgjenkjenning vil kun være mulig ved å ta bilde direkte i applikasjonen.

Navn:	Autentisere ID-dokument vha. bilde
Aktør:	Keesing
Mål:	Å autentisere et dokument vha. Keesing API
Beskrivelse:	Bilde av ID-kort sendes til Keesing Authentiscan. Dette sjekkes mot deres database og godkjennes dersom det er gyldig.

Navn:	Lese NFC-brikke i pass
Aktør:	Bruker
Mål:	Hente ut informasjon som ligger i RFID-brikken
Beskrivelse:	Først må informasjon fra MRZ leses, denne gir tilgang til RFID-brikken som leses ved hjelp av NFC-leser på telefonen.

Navn:	Autentisere pass ved hjelp av NFC
Aktør:	Keesing
Mål:	Sjekke om pass er gyldig vha. informasjonen som er lest vha NFC.
Beskrivelse:	Informasjonen som leses ved hjelp av NFC sendes til Keesing Authentiscan. Her sjekkes informasjonen og vi får vite om passet er gyldig eller ikke.

Navn:	Ansiktsverifisering
Aktør:	Keesing
Mål:	Verifisere at personen som bruker applikasjonen er den samme som personen som eier passet.
Beskrivelse:	Brukeren tar et bilde med kamera på mobilen (selfie). Deretter brukes Keesings API for ansiktsgjenkjenning til å sammenlikne dette bilde med bildet på ID-dokumentet.

Navn:	Melding vedrørende verifisering
Aktør:	Keesing
Mål:	Opplyse brukeren om at autentiseringen var vellykket.
Beskrivelse:	Responsen på verifisering fra Keesing inneholder en statuskode. Denne statuskoden skal formidles til brukeren på en brukervennlig måte som opplyser om hvordan verifiseringen har gått, og gir informasjon om hvordan verifiseringen eventuelt har feilet.

Navn:	Loggføre verifiseringsprosess
Aktør:	Server
Mål:	Å loggføre verifiseringsprosess med statuskoder
Beskrivelse:	Resultatet av verifiseringsprosessen loggføres, slik at det er mulig å undersøke loggen for å se hvorfor en verifisering har vært mislykket eller lignende.

Navn:	Lagre kundedata
Aktør:	Database
Mål:	Lagre relevante kundedata.
Beskrivelse:	Når verifisering er vellykket vil relevante kundedata lagres i databasen. I vår demonstrasjonsapplikasjon vil dette være en database vi setter opp på vår server, men dersom applikasjonen kommer ut på markedet vil dette være Buypass sin egen database.

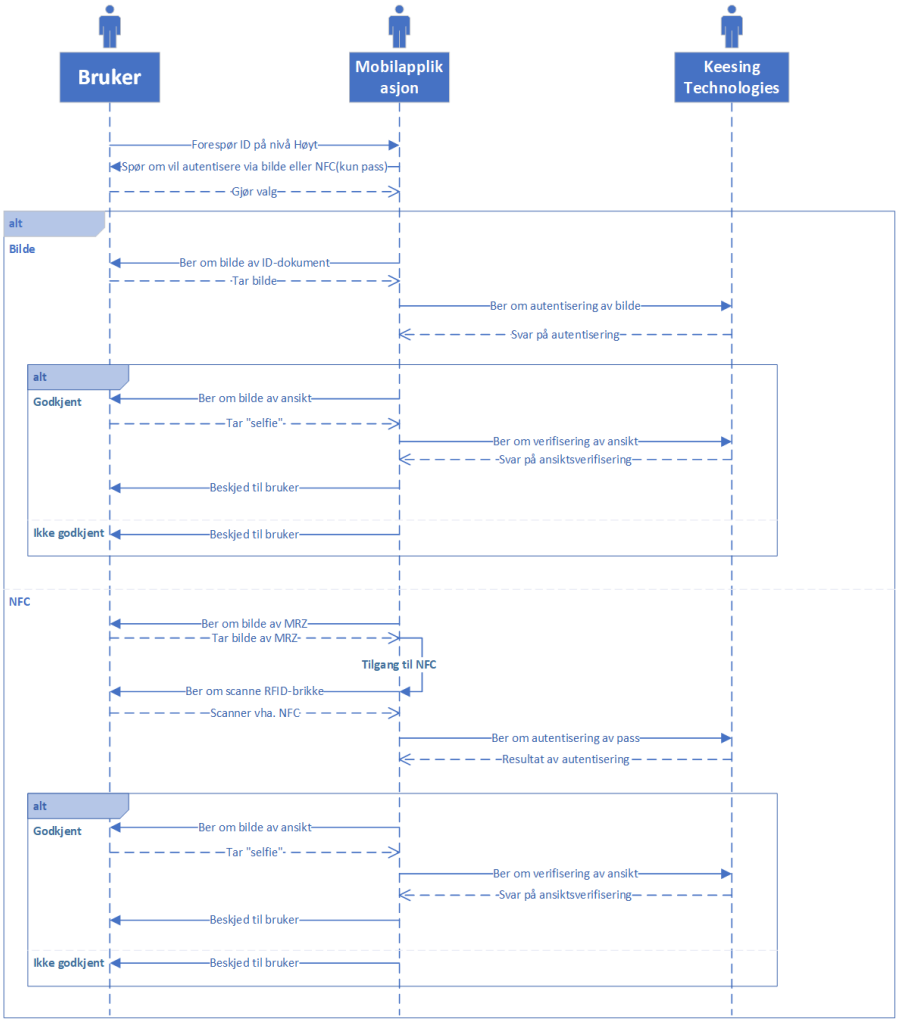
2.5 Arkitektur

Figur 4 viser oversikt over hovedkomponentene i applikasjonen vår. Vi har to hovedmoduler; klient og tjener. Klientmodulen interagerer med brukeren og sender deretter informasjon videre til tjeneren. Her konverteres data til et format som kan kommunisere med Keesing sitt API. Sjekking av informasjon fra ID-dokument, ansiktsgjenkjenning osv. skjer på Keesing sin side. Resultatet av dette sendes så tilbake til serveren, som videresender dette til klientapplikasjonen. Vi har valgt MVC (Model View Controller) som arkitektur i klientapplikasjonen. Logikken i programmet vil ligge i Model, så det er gjennom denne modulen klientapplikasjonen kommuniserer med serveren. Brukergrensesnittet styres i View og vi har en Controller-modul som håndterer input fra brukeren i form av tastetrykk, kamerabilder og NFC. View vil altså vise knapper og informasjon til brukeren, Controller vil håndtere hva som skjer når brukeren interagerer med applikasjonen. Dette vil igjen trigge at noe skjer i Model, som igjen oppdaterer View basert på dette.

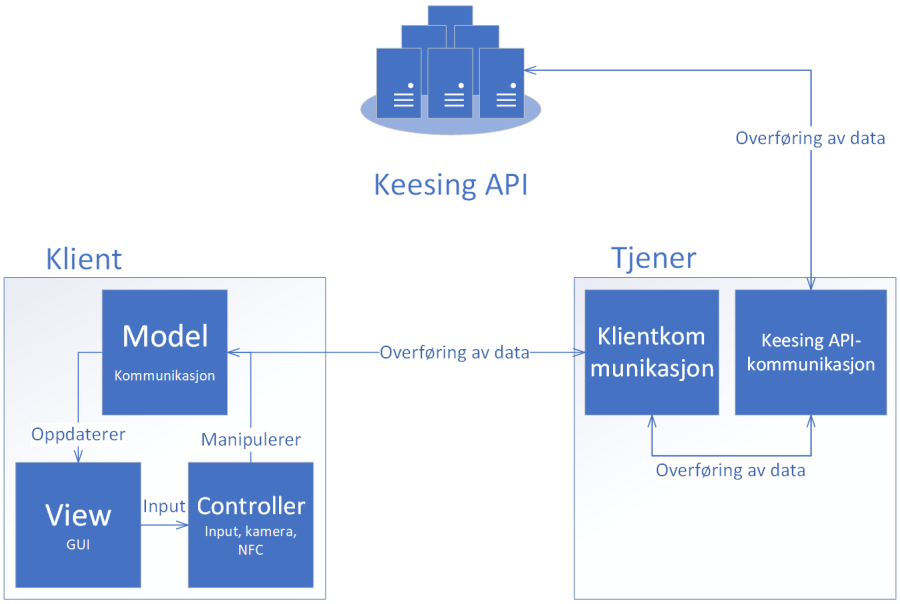
3 Prosjektorganisering

3.1 Ansvarsforhold og roller

Karoline er valgt som gruppeleder, og vil ha det siste ordet ved uenigheter innad i gruppen. Dette valget er tatt på bakgrunn av hennes sterke faglige kompetan-



Figur 3: Sekvensdiagram



Figur 4: Overordnet komponentarkitektur laget i Visio

se, samt erfaring fra tidligere bacheloroppgave. Vegard er valgt som nestleder, og vil dermed fungere som leder ved de anledninger Karoline er fraværende. Jørgen er utpekt dokumentansvarlig. Han vil ha ansvar for å skrive referater fra gruppemøter, og å dokumentere arbeidsprosessen for veilederne og arbeidsgiver.

På grunn av størrelsen på gruppen i prosjektet vil alle være utviklere når den tiden kommer. Det er ikke slik at hver enkelt skal kode en modul i applikasjonen alene, men hver person vil gjerne få hovedansvaret for hver sin modul. Det blir da denne personen sitt ansvar å se til at nødvendig fremdrift, tilstrekkelig dokumentasjon og kvalitet på kildekode overholdes etter de standarder som er bestemt innad i gruppen.

For å sikre kompetanse på alle områder og for å kvalitetssikre de ulike aspektene ved oppgaven har vi definert noen spesielle temaer hver enkelt skal sette seg ekstra godt inn i. Dette innebærer at personen skal gjøre grundig research på dette emnet, slik at vi kan drive internundervisning innad i gruppen. Personen har også ansvar for å følge opp, og sikre at denne jobben blir gjennomført på en god måte.

Ruben har hovedansvaret for UX. Dette er basert på hans interesse og bakgrunn fra grafisk design, samt valgfaget Computer Graphics som han hadde under sitt utvekslingsopphold. Karoline har hovedansvaret for implementasjon av Keesing Technologies sitt API. Jørgen har hovedansvaret for oppsett av server, og kommunikasjonen mellom klienten og serveren. Vegard har hovedansvaret for sikkerhet. Dette er basert på hans erfaring fra de to valgfagene: nettverkssikker-

het og programvaresikkerhet.

3.2 Rutiner og regler i gruppa

Gruppen har bestemt at det skal jobbes rutinemessig med bacheloroppgaven. Det er blitt fastslått at det skal jobbes hver ukedag, mandag til fredag, med kjernetid fra klokken kvart over ni til fire. Denne rutinen skal overholdes så fremt det ikke går på bekostning av forelesninger og seanser i andre emner. Denne avgjørelsen er tatt på bakgrunn av at det i arbeidslivet vil være nødvendig å innfinne seg i de rutiner arbeidsplassen har, og vil forhindre sporadisk jobbing. Det skal oppnås et minimum med 30 arbeidstimer i uken per person til prosjektet, så fremt dette ikke hindres av stor arbeidsmengde i andre emner, reise, sykdom, eller andre personlige årsaker.

Man er pliktig i å møte på forhåndsbestemte møter, men dersom man av årsaker, nevnt i avsnittet over, ikke har mulighet, skal dette meldes ifra om i så god tid som mulig.

Hvis utdelt arbeidsmengde blir for mye og man ikke klarer å forholde seg til avtalte tidsfrister, må vedkommende melde dette i fra til enten prosjektleder eller gruppemedlemmer. Gruppen og vedkommende blir enige om endringer som må gjøres for at arbeidet skal komme i mål. Om ikke vedkommende tar kontakt kan det forekomme tilsnakk fra enten gruppeleder eller gruppemedlemmer om at vedkommende må øke produktiviteten. Dersom det ikke påfaller noen endringer skal det rapporteres til veilederne, og vi vil ta det videre derfra, men det kan i verste fall ende med utkastelse fra gruppen.

Arbeidstimer skal dokumenteres ved hjelp av det nettbaserte verktøyet Toggl[1]. Dette gjøres for å kunne dokumentere de arbeidstimene hvert medlem har gjennom prosjektet, og det skal skrives en beskrivelse av hva tiden brukes på. På denne måten oppnår vi også effekten av å kunne effektivisere de områdene vi ser vi benytter for mye tid på.

Vi vil benytte Trello[2] for å kunne planlegge og dokumentere arbeidsflyten. Dette er med på å sikre fremdriften, for ved å benytte Trello mot Gantt-skjemaet, se vedlegg A, kan vi se hva som skal gjøres, og skulle vært gjort til forskjellige tider.

4 Planlegging, oppfølging og rapportering

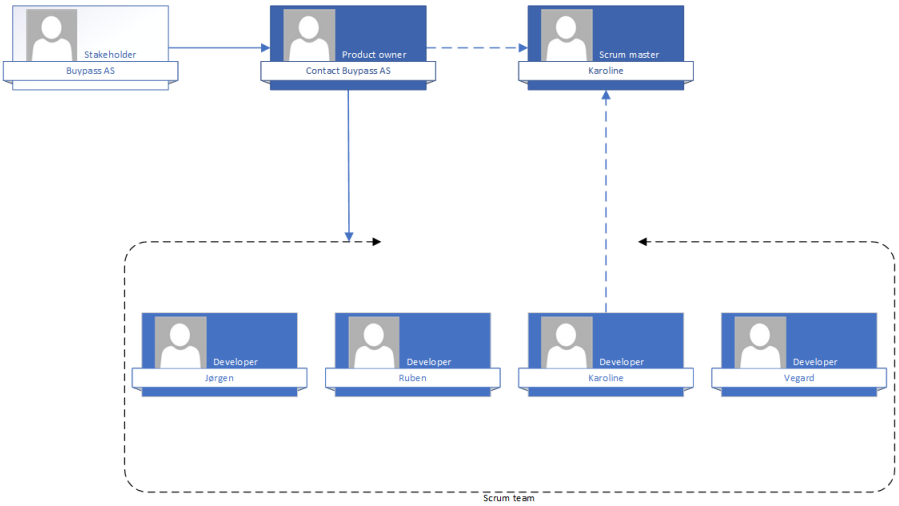
4.1 Hovedinndeling av prosjektet

4.2 Valg av SU-modell

Vi har valgt å bruke Scrum som SU-modell (Systemutviklingsmodell). Ingen i gruppen har noe særlig erfaring med systemutviklingsprosjekter fra før, så vi er

[1] Toggl er et arbeidsverktøy for og lettere dokumentere arbeidstimer ved prosjektarbeid, se `https://toggl.com/where-did-time-go/`

[2] Trello er et arbeidsverktøy for å kunne planlegge arbeidsoppgaver, og for å dokumentere arbeidsflyt, se `https://trello.com/`



Figur 5: Organisasjonskart laget i Microsoft Visio

forberedt på å støte på en del utfordringer med tanke på estimering av tid og ressurser. Samtidig vil det være gunstig å få arbeidsgiveren sin tilbakemelding underveis på hva som er bra, og hva som bør endres. Vi tenker derfor at plandrevne metoder, som f.eks fossefall kan være lite egnet fordi disse gir lite rom for fleksibilitet[13] og at en smidig utviklingsmodell passer bedre.
Smidige metoder passer gjerne godt for små/middels store prosjekter med relativt små team der kunden er innstilt på å være delaktig i utviklingsprosessen[13]. Dette stemmer godt overens med vårt prosjekt.
Etter å ha vurdert ulike smidige utviklingsmodeller falt valget vårt på Scrum. Begrunnelsen for dette er at Scrum er en relativt ukomplisert modell som legger godt til rette for fleksibilitet, samarbeid, involvering av kunden og inkrementell utvikling[13]. Daglige Scrum-møter vil dessuten sørge for at alle i gruppen til enhver tid er oppdatert på hva de andre i gruppen holder på med, slik at vi er bedre rustet til å håndtere eventuelt fravær/sykdom og til å kunne hjelpe hverandre med arbeidsoppgaver dersom det er behov for dette.
Vi har lagt opp til to uker lange sprinter med Sprint Planning Meeting og Sprint Retrospective Meeting henholdsvis før og etter hver sprint. Sprintlengden ble valgt fordi vi kom frem til at en uke vil bli såpass kort tid at det vil gå med unødvendig mye tid til møter, planlegging osv, og det vil bli vanskelig å ha klart et nytt inkrement ved enden av hver sprint. Med tanke på at vi alle er ganske uerfarne ønsker vi hyppig kontakt med kunden for å være sikre på at vi er på rett spor. Vi føler derfor at sprinter på tre til fire uker blir for lange. Møtene med Buypass vil derfor sammenfalle med Sprint Retrospective Meeting og skje annenhver fredag. Under disse møtene vil vi oppdatere Buypass på hva vi har gjort i den siste sprinten og demonstrere ny funksjonalitet. Buypass vil ikke være tilstede under Sprint Planning Meeting, men vi vil bruke møtet med

dem til å få innspill til hva vi skal jobbe med i neste sprint.

Ettersom valget falt på Scrum fant vi også ut at denne SU-modellen kan forbedres. Ved og i tillegg benytte oss av Kanban[6] vil vi kunne kombinere disse utviklingsmodellene for å effektivisere vår arbeidsflyt. Kanban er også en iterativ prosess som har likheter med Scrum. Dette gjør at Kanban enkelt kan tilpasses til vår allerede eksisterende SU-modell, Scrum. Kanban tilbyr bedre visuelle løsninger slik at arbeidsoppgaver kan enkelt representeres ved hjelp av utviklingsverktøy som Toggl og Trello. Vi loggfører arbeidsoppgaver med Toggl, og benytter oss av et Kanban/Scrumboard representert i Trello. Ved Toggl vil alle medlemmer av gruppen legge inn individuelt hvor lenge man har brukt på de forskjellige arbeidsoppgavene man er satt opp på og markerer det med en "tag" som bestemmer hvilke kategori det hører til. Slik kan vi summere opp og vise til statistikk for hva vi har brukt tid på. Trello er et verktøy som viser fram arbeidsoppgaver som skal gjøres (backlog), som er under arbeid, og hva som er ferdig. Det byr dermed på muligheter som gjør det enklere å visualisere ulike arbeidsoppgaver.

4.3 Valg av metode og tilnærming

Vi vil ha fokus på inkrementell utvikling. Vi ønsker på et tidlig tidspunkt å kunne ha en enkel prototype av applikasjonen med et enkelt brukergrensesnitt og noe grunnleggende funksjonalitet på plass, slik at vi gradvis kan legge til funksjonalitet og forbedre design/UX. "Minimal Viable Product" står sentralt i tankegangen vår. Dette innebærer at vi ønsker å få på plass et fungerende produkt med den mest grunnleggende funksjonaliteten tidlig, slik at man kan få fortløpende tilbakemelding fra kunden og dermed videreutvikle produktet[16].

Utviklingen vil også i stor grad være gjenbruksorientert. Keesing Technologies sin løsning AuthentiScan har allerede godt fungerende løsninger for verifisering av ID-dokumenter og ansiktsgjenkjenning[7]. Ettersom vi har tilgang til deres API'er vil vår hovedoppgave være å sy sammen en løsning tilpasset de kravene Buypass stiller til produktet.

4.4 Plan for statusmøter og beslutningspunkter i perioden

Gjennom prosjektperioden vil vi få tett oppfølging av veilederne våre, Rune Hjelsvold og Deepti Mishra, heretter omtalt som veilederne. Det er planlagt ukentlige møter, mandager klokken elleve. I forkant av møtene skal veilederne motta en liten rapport som besvarer tre hovedpunkter: hva som har blitt gjort siden forrige møte; hva det er gruppen ønsker å diskutere, gjerne med støtte i dokumentasjon; hva som er planen for den kommende uken.

Det første møtet med Buypass AS forekom fredag 18. januar, og det ble på dette møtet avklart at vi skal ha møter fredag hver andre uke, fra og med 1. februar klokken elleve. Disse møtene vil forekomme over Skype, men lenger ut i prosjektet hvor det kan bli hensiktsmessig med fysiske demoer vil det bli lagt opp til møter deretter. Møtene med arbeidsgiver vil forekomme ved slutten av hver sprint, og vi vil på denne måten kunne gi arbeidsgiver oppdateringer

på fremdriften, diskutere veien videre og demonstrere det vi har utviklet under den siste sprinten. Disse møtene vil også åpne for justeringer av funksjonalitet, milepæler og det konkrete målet for applikasjonen. Dette er viktig ved valget av en smidig utviklingsmodell, for man vet ikke om man oppnår det man planlegger i startfasen av prosjektet.

5 Organisering av kvalitetssikring

5.1 Dokumentasjon, standardbruk og kildekode

5.1.1 Programmeringsspråk

Vår oppdragsgiver har gitt oss muligheten til å velge om vi ønsker å skrive applikasjonen vår i Java eller Kotlin. Android-applikasjoner har tradisjonelt blitt skrevet i Java, men den siste tiden har Kotlin blitt mer og mer populært. Kotlin har en del fordeler Java ikke har, som for eksempel lambda-uttrykk og inline-funksjoner, samt at de unngår null-referanser, noe som er et stort problem i Java[11]. Til tross for dette har vi likevel bestemt oss for å skrive applikasjonen vår i Java. Dette skyldes hovedsakelig to ting. For det første har vi ganske begrenset tid å bruke på å lære oss et nytt programmeringsspråk og ettersom flere av oss har tidligere erfaring med Java vil vi mest sannsynlig bruke en god del mindre tid på å lære syntaks og kodestruktur. For det andre finnes det langt flere kodeeksempler i Java, slik at det er lettere å feilsøke koden. Buypass har også informert oss om at de har mer kompetanse i Java enn Kotlin, så det blir mest sannsynlig lettere for oss å få veiledning fra deres side dersom vi skriver i Java.
For å sikre at koden er leselig og for å redusere sjansen for feil, vil vi følge Google Java Style Guide[5]. Til å sikre at denne standarden følges vil vi bruke programmet CheckStyle.

5.2 Konfigurasjonsstyring

5.2.1 Bitbucket og Trello

Skylagring av kode vil bli gjennomført ved bruk av Bitbucket. Dette vil bidra til tilgjengelig kode på en private repository blant oss på gruppen. Ved å benytte oss av dette vil vi videre opprette branches ut fra master, hvor sist fungerende versjon av den ferdigstilte applikasjonskoden ligger.

Vi følger en liste over generell arbeidsflyt ved koding, og det er viktig at denne prosessen følges for å sikre god kodekvalitet og en ryddig arbeidsprosess.

1. Velg et kort fra backlogen i Trello og merk det med utvikleren sitt navn
2. Opprett en tilhørende branch og bruk den lokalt
3. Skriv tester
4. Skriv kode

5. Flytt kortet til Testing
6. Test funksjonalitet gjennom tester og manuelt (hvis mulig)
7. Hvis OK, opprett en pull request, koble denne til Trellokortet og flytt kortet til kodegjennomgang.

Ved valg av en oppgave fra backlogen skal man først opprette en branch fra Trello ved å velge Bitbucket - Create Branch. Dette tillates ved hjelp av en tilleggsfunksjon i Trello. Navnet på branchen skal være det samme som navnet på kortet i Trello. Etter at branchen er opprettet skal en flytte kortet til under arbeid. På denne måten kan man se hvem som jobber med hva til en hver tid, og det gjør det enkelt for de andre gruppemedlemmene og kunne se hva man holder på med, eller bistå med hjelp ved problemer.

Dersom man mener en modul er ferdig, og testene ikke feiler skal en pull request opprettes. Etter at denne er opprettet flyttes kortet i Trello til kodegjennomgang, og kobler det til pull requesten. En annen på gruppa skal da se på pull requesten og gjennomgå koden for å se om den er god nok, og deretter godkjenne og merge med master. Lukk source branchen, og flytt kortet i Trello til ferdig. Dersom den som gjør kodegjennomgang mener at koden ikke innfrir krav for funksjonalitet eller kvalitet skal man flytte kortet i Trello tilbake til backlogen, og gi tilbakemelding i form av kommentar på kortet i Trello. Da vil prosessen starte på nytt, og innehaveren av kildekoden vil ha ansvar for å rette opp i de feil som ble påpekt under kodegjennomgangen.

5.2.2 Versjonskontroll

Det er viktig at gruppen er samstemte rundt formalitetene i utviklingsfasen. Dette handler i stor grad om hvordan vi benytter Git[3]. Arbeidsflyten for versjonskontroll er illustrert ved figur[6], og nummereringen av versjoner vil forekomme på følgende format;

Major . Minor . Patch version . Build number

Major representerer en full lansering av applikasjonen, og dette vil i løpet av vårt prosjekt ikke skje. Derfor vil dette tallet være null gjennom hele vårt prosjekt, og vi vil evaluere om det er hensiktsmessig å inkludere det. Minor representerer en ny funksjonalitet, eller en stor forandring i hvordan applikasjonen oppfører seg. Patch versjon representerer fiksing av bugs i applikasjonen. Build number representerer fiksing av bugs spesifikke for en bruker/tester i prosjektperioden. Disse endringene kan inkluderes ved lansering av patch versjon, for å hindre at det innføres for mange små versjoner av applikasjonen.

Slik som figur 6 illustrerer skal det foretas en fork av master dersom en ny versjon skal iverksettes. Denne navngis etter formatet listet over, og er basert på hva slags implementasjon som skal foretas. Den nye versjonen vil inneholde flere grener hvor gruppemedlemmene kan jobbe individuelt, eller sammen, uten forstyrrelser fra de andre medlemmene sin kode. Den nye versjonen skal ikke

[3]Git er et open source-distribuert versjonskontrollverktøy, se `https://git-scm.com/`

merges (flettes) med master dersom den ikke er feilfri, i form av at funksjonalitet som er implementert skal fungere. Det er viktig at alle grener som er opprettet i versjonsgrenen fungerer sammen, og hele applikasjonen skal testes grundig før den merges med master.

Master representerer den sist oppdaterte, fungerende versjonen av applikasjonen. Den oppdaterte versjonen vil være klar til å bli demonstrert for kunden. Endringer eller nye funksjonaliteter kunden ber om at legges til vil forårsake at en ny versjon av applikasjonen vil bli opprettet.

5.2.3 Testing

I applikasjonen vår vil det være behov for manuelle tester, ettersom det skal tas bilder, scannes pass osv. Vi vil likevel ta i bruk testrammeverket Espresso for automatisert testing av brukergrensesnitt. Dette er et rammeverk for Android Studio som tilbyr API'er for å skrive tester som simulerer interaksjon med applikasjonen[3]. For å lete etter bugs og potensielle problemer i koden vil vi benytte statisk kodeanalyse ved verktøyet SonarQube. Dette er et verktøy som kan rapportere om duplisering av kode, kompleks kode, hvor stor dekningsgrad testene har, osv[17]. Vi vil bruke Test-Driven Development som et ledd i kvalitetsikringen av koden. Hver gang vi legger til ny funksjonalitet skal vi derfor gjennom følgende steg[21]:

1. Skrive test
2. Kjøre test (denne skal feile)
3. Skrive kode
4. Teste
5. Refaktorere

5.3 Risikoanalyse

Vi har utarbeidet en risikoanalyse som skal stå i fokus hvis noe uheldigvis går galt under prosjektperioden. Denne analysen er illustrert i figur 8. Vi har valgt å demonstrere tre ulike fargekoder som består av rødt, gult, og grønt felt. Fargene representerer hvilke grad av risiko koden er. En akseptabel risiko er vist som grønn farge. Dette er risikoer det vil være unødvendig å forberede seg på, enten på grunn av den lave sannsynligheten eller den lave risikoen. Gul farge er en middels risiko og tiltak vurderes individuelt. Rød farge har størst verdi og tiltak er nødvendig. Se figur 7.

5.3.1 Mulige utvidelser av oppgaven

Fordi det er mye usikkerhet knyttet til omfanget av oppgaven vil vi legge en plan for mulige utvidelser dersom oppgaven vår viser seg å være for lite kompleks. Vi er på det nåværende tidspunkt litt usikre på hvor mye av funksjonaliteten vi

trenger som faktisk er tilgjengelig gjennom Keesing sitt API (vi venter fortsatt på fullstendig dokumentasjon). Vi er derfor litt bekymret for at dersom Keesing kan brukes til mer eller mindre all funksjonalitet, vil vår oppgave bli litt for liten i forhold til hva som er forventet av en bacheloroppgave.
Vi vil diskutere muligheten for mulige utvidelser av oppgaven med Buypass under møtet 1. februar, men per nå har vi følgende forslag til utvidelser:

- Å lage en webapplikasjon i tillegg til mobilapplikasjon
- Å lage en innloggingsportal
- Å initiere videosamtale med kundeservice hos Buypass

6 Plan for gjennomføring

Som nevnt tidligere vil vi jobbe i sprinter på to uker. Det er ganske mye usikkerhet knyttet til hvor lang tid vi kommer til å bruke på de ulike modulene, så vi har ingen detaljert tidsplan for når de ulike delene skal implementeres. Ren planlegging og fordeling av oppgaver vil skje på Sprint Planning Meeting i forkant av hver sprint. Vi har likevel laget en grov plan for når vi planlegger å jobbe med de ulike arbeidsoppgavene og når vi har som mål å nå de ulike milæpælene vi har definert, se vedlegg A.

6.1 Work Breakdown Structure

Vi har laget et WBS-diagram (Work Breakdown Structure), se figur 9. Dette gjenspeiler Gantt-diagrammet, se vedlegg A, og viser hvordan komponentene er avhengig av hverandre for ferdigstillelse. De nederste operasjonene/modulene må på plass før en prosess/modul på et nivå over kan gjøres, og det viser på denne måten hvilke komponenter man er nødt til å prioritere. Diagrammet er laget på et overordnet nivå, og det kan forekomme feil som en følge at vi nå er i planleggingsfasen og eventuelle utfordringer og forandringer kan oppstå underveis i prosjektet.

6.2 Milepæler og beslutningspunkter

Som det allerede fremkommer i prosjektplanen benytter vi oss av en smidig utviklingsmodell som gjør det vanskelig å fastsette datoer for milepæler. Selv om gruppen innehar lite erfaring med større utviklingsprosjekter har vi likevel valgt å sette opp mål for hver av sprintene, og vi mener disse må oppnås for å nå målet om en ferdigstilt applikasjon. Vi har valgt å sette opp mange milepæler, for på denne måten kan vi evaluere ved hver endte sprint om målet er innen rekkevidde basert på disse. Det vil være beleilig og kunne komme med, og få, tilbakemeldinger til/fra arbeidsgiver tidlig, for klarhet rundt eventuelle justeringer av den ferdige applikasjonen. Det er ikke viktig at tidsfristene rundt milepælene overholdes til punkt og prikke, og de er mer ment som retningslinjer

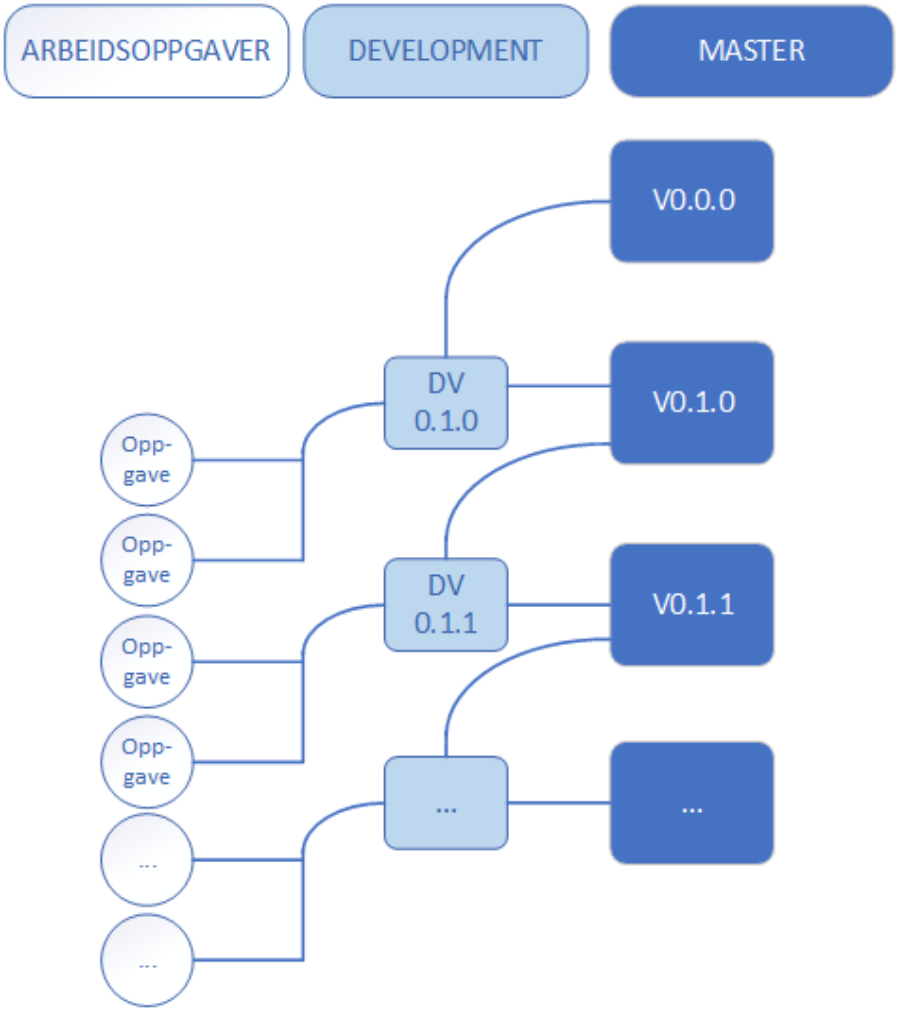
for den smidige utviklingen. Det er vanskelig å estimere hva som er oppnåelig innenfor en to ukers sprint, og vi vil ha overestimert eller underestimert på visse punkter.

- **Milepæl 1**: 15. februar - Applikasjonen har et enkelt grensesnitt, og kan ta bilde som kan sendes til server.
- **Milepæl 2**: 1. mars - Applikasjonen kan sende bildet man har tatt til Keesing for verifisering, og brukeren får beskjed om dokumentet er godkjent eller ikke.
- **Milepæl 3**: 15. mars - NFC-lesing av dokument er implementert, og man sende det til Keesing for verifisering. Mottar beskjed om dokumentet er gyldig eller ikke, dersom det er et ID-dokument.
- **Milepæl 4**: 29. mars - Brukeren kan ta bilde av ansiktet og sende til Keesing så de kan kjøre en ansiktsverifisering imot de dokumenter de har motatt, og se om det er samme person eller ikke.
- **Milepæl 5**: 12. april - Funksjonalitet i applikasjon skal være ferdigstilt, men finpuss og testing kan gjenstå.
- **Milepæl 6**: 12. mai - Rapport og applikasjon er ferdig og leveres.

6.3 Tids- og ressursplan

Se vedlegg A for Gantt-diagram.

- **Sprint 1**: 1. - 15. feb - Research Server/Node.js/Express.js.
  Etabler server med enkel funksjonalitet som spesidfiseres på møtet med Buypass 1. februar, eller ved sprint planning meeting mandag 4. februar.
  Research Android Studio/Java, og lag et enkelt grensesnitt med implementasjon av kamera og ta bilde.
  Research Keesing API.
- **Sprint 2**: 18.feb - 1. mar - Research NFC og implementasjon av det.
- **Sprint 3**: 4. - 15. mars - Research ansiktsgjenkjenning hos Keesing og implementer dette.
- **Sprint 4**: 18. - 29. mars - Implementasjon av verifisering ved ansiktsgjenkjenning.
- **Sprint 5**: 1. - 12. april - Implementasjon av gjenværende funksjonalitet, finpuss av applikasjon og testing.
- **Sprint 6**: 23. april - 12. mai - Finpuss av applikasjon og rapportskriving.



Figur 6: Versjonskontroll ved utvikling

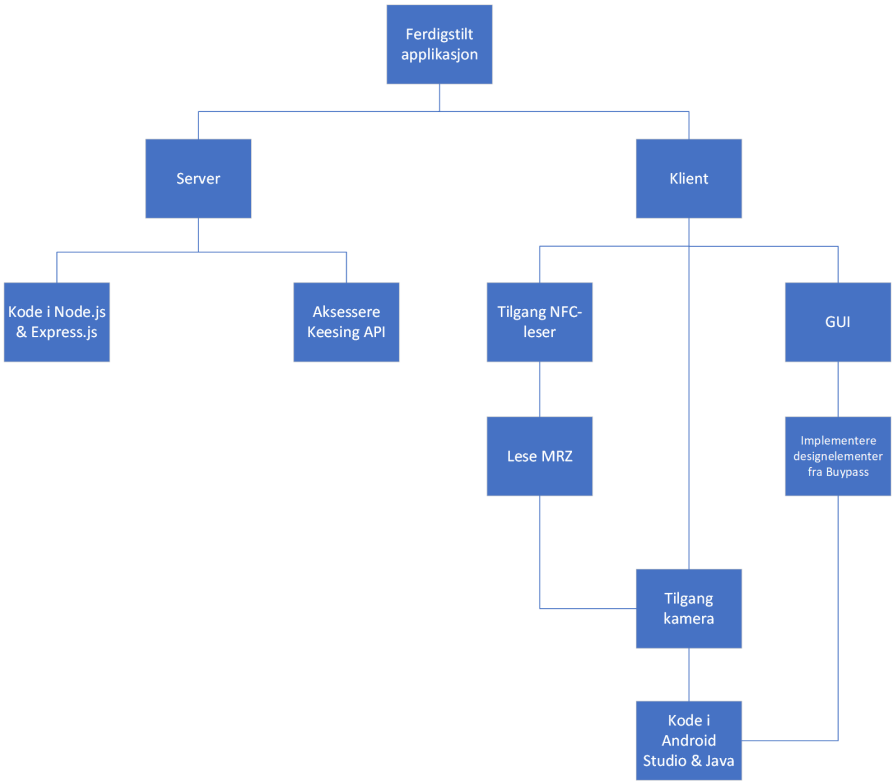
	LAV	MEDIUM	HØY
LITE SANNSYNLIG	1	2	3
SANNSYNLIG	2	4	6
HØYT SANNSYNLIG	3	6	9

Akseptabel risiko Vurderes individuelt Tiltak er nødvendig

Figur 7: Riskikoanalyse

Nr.	Beskrivelse	Sannsynlighet	Konsekvens	Risikoverdi	Tiltak
6	Implementasjon av teknologi (NFC, Keesing API, Kamera)	3	3	9	Starte med dette tidlig, søke kompetanse hos fagpersoner hos Buypass eller NTNU, starte med andre oppgaver hvis teknologi ikke er ennå tilgjengelig
9	Begrenset/Lite omfattende prosjekt	2	4	8	Kontakte Buypass om mulige utvidelser: web-applikasjon, initiere videosamtale med kundeservice.
3	Forsinkelser	3	2	6	Loggføre arbeid, benytte Toggl, Trello
5	Sprinter blir forlenget	3	2	6	Fleksible arbeidsoppgaver under hver sprint
2	Kontaktsvikt	1	3	3	Utføre prosjekt i utgangspunkt fra sist møte, benytte veileder, ta egne valg
4	Tap av arbeid	1	3	3	Ha backup av data (lokalt, skylagring)
7	Offentliggjøring av Kildekode som ikke skal offentliggjøres	1	3	3	Holde alt i private repository
8	Manglende kompetanse	1	3	3	Sette av god tid til research og god kommunikasjon med oppdragsgiver og veileder
1	Sykdom	1	2	2	

Figur 8: Riskikoanalyse



Figur 9: WBS-figur laget i visio

Referanser

[1] Lover og regler. https://www.datatilsynet.no/regelverk-og-verktoy/lover-og-regler/. [Online; accessed 22-January-2019].

[2] Virksomhetens plikter. https://www.datatilsynet.no/rettigheter-og-plikter/virksomhetenes-plikter/. [Online; accessed 22-January-2019].

[3] Test ui for a single app. https://developer.android.com/training/testing/ui-testing/espresso-testing. [Online: accessed 30-January-2019].

[4] Federal Office for Information Security. Security mechanisms in electronic id documents. https://www.bsi.bund.de/EN/Topics/ElectrIDDocuments/SecurityMechanisms/securBAC/bac_node.html, 2019. [Online; accessed 17-January-2019].

[5] Google java style guide. http://checkstyle.sourceforge.net/reports/google-java-style-20170228.html?fbclid=IwAR2bawhOFqZw8ygnJIUxc9G40oAYOhI8dTINtUpCg6xHB58f_0L1N2PHvBA#s3.3-import-statements. [Online; accessed 25-January-2019].

[6] Kanban vs. scrum: What are the differences? https://leankit.com/learn/kanban/kanban-vs-scrum/.

[7] Infographic: Automated id verifications with keesing authentiscan. https://keesingdocumentchecker.com/infographic-automated-id-verifications-with-keesing-authentiscan/. [Online; accessed 21-January-2019].

[8] Keesing technologies - about us. https://www.keesingtechnologies.com/about-us/. [Online; accessed 22-January-2019].

[9] Keesing technologies - keesing authentiscan standard. https://www.keesingtechnologies.com/automated-id-checking/standard/. [Online; accessed 22-January-2019].

[10] Keesing authentiscan - integration manual version 1.42. [dato: 14-05-2018].

[11] Comparison to java programming. https://kotlinlang.org/docs/reference/comparison-to-java.html/. [Online; accessed 22-January-2019].

[12] Odd R Lange. Fra 1. april kommer du ikke inn i usa uten riktig pass. *Dagbladet*, Mar 2016.

[13] Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010.

[14] Odd R Valmot. Passet kjenner deg. *Teknisk Ukeblad*, Feb 2011.

[15] Wikipedia contributors. Express.js — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Express.js&oldid=875960505, 2018. [Online; accessed 22-January-2019].

[16] Wikipedia contributors. Minimum viable product — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Minimum_viable_product&oldid=870722777, 2018. [Online; accessed 21-January-2019].

[17] Wikipedia contributors. Sonarqube — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=SonarQube&oldid=875137323, 2018. [Online; accessed 30-January-2019].

[18] Wikipedia contributors. User experience design — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=User_experience_design&oldid=874311100, 2018. [Online; accessed 16-January-2019].

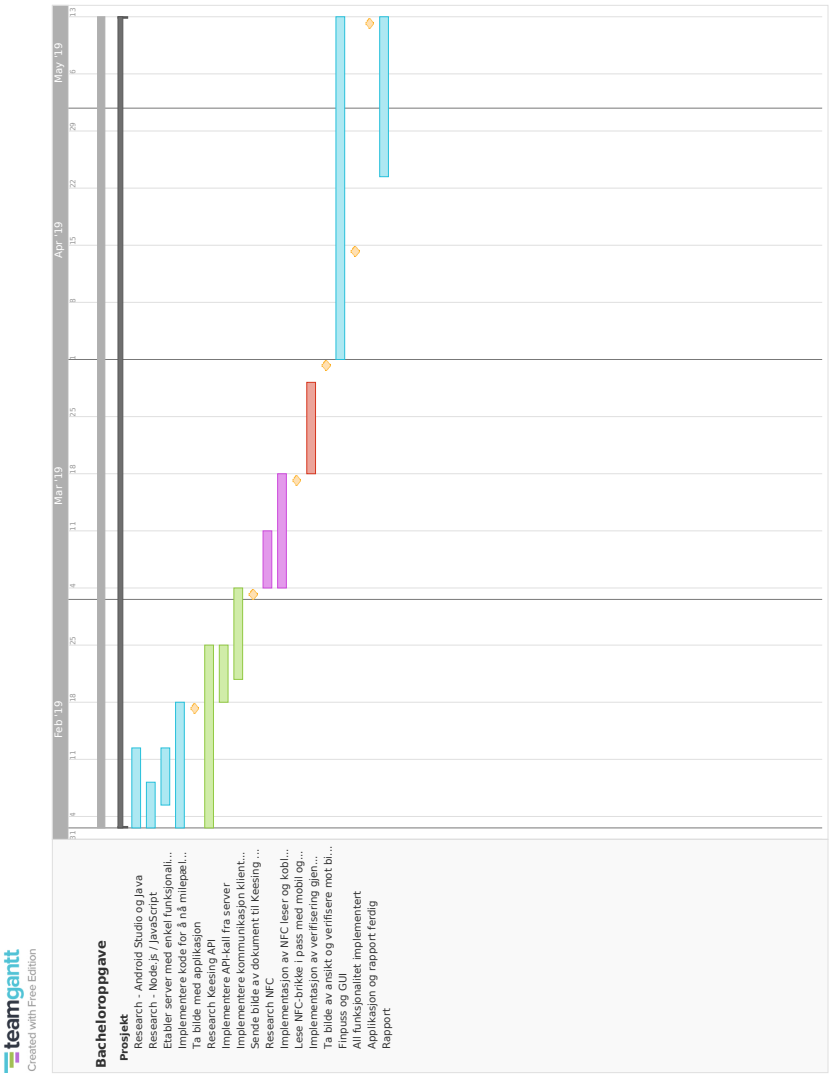
[19] Wikipedia contributors. Near-field communication — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Near-field_communication&oldid=877746177, 2019. [Online; accessed 17-January-2019].

[20] Wikipedia contributors. Node.js — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Node.js&oldid=879075496, 2019. [Online; accessed 22-January-2019].

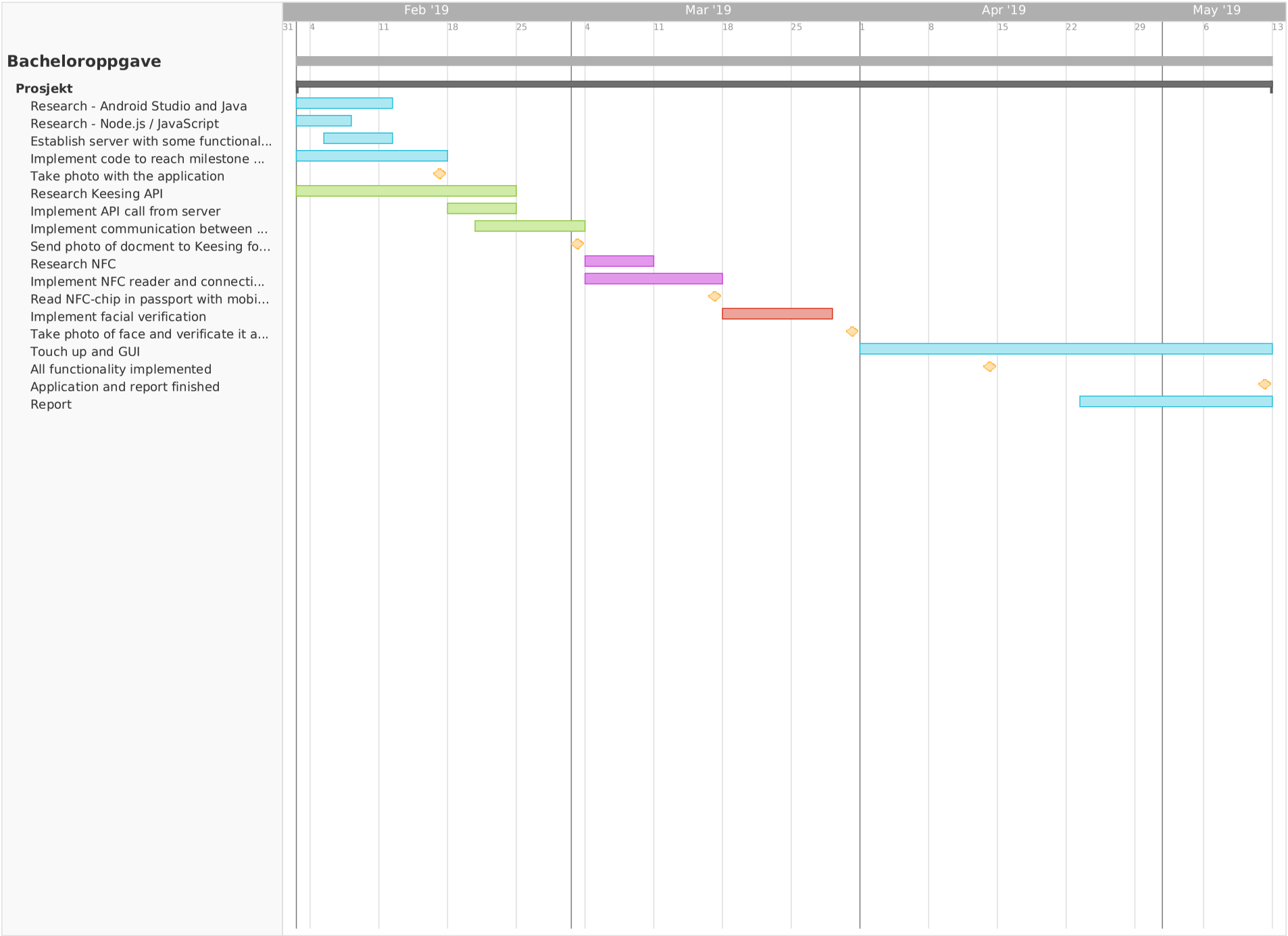
[21] Wikipedia contributors. Test-driven development — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Test-driven_development&oldid=879976508, 2019. [Online; accessed 25-January-2019].

Appendices

A Gantt-diagram



D Gantt Diagram



E Technologies and tools

Overleaf: Online LaTeXeditor, which allows for sharing. Our preferred choice for writing our bachelor thesis.

Google Drive: Online file- and storage sharing solution. Allows for sharing of documents within a group.

Bitbucket: Web based version control repository for source code and development projects.

Git: Version control system for handling code uploads to repositories.

Trello: Online visual tool for task organization.

Toggl: An online time tracking tool which allows time entry tags and descriptions.

Cisco AnyConnect Secure Mobility Client: VPN[1] for allowing remote use of NTNU's network.

JSON: JSON[2] is a lightweight data-interchange format [35].

Android Studio: The official IDE[3] for Android, and is designed specifically for Android development.

Plugins: A software component that adds a specific feature to an existing computer program [36].

- **SonarLint:** Extension for improving code quality. It acts like a spell checker, notifying us while coding in regards of code complexity, bugs and coding standards.

Java: Java is a class based, object oriented programming language, and is designed to have as few implementation dependencies as possible [37].

Visual Studio Code: Code editor used for our backend development.

Plugins: **sftp:** SFTP[4] is a network protocol for safely transfering files over a unsecure network. It offers the functionality such as copying, editing, renaming and deletion of files [38].

- **Prettier:** Code formatter for Node.js amongst other programming languages. Built in presets for formatting of JavaScript code, making it more readable and making sure coding standards are followed.

Node.js: An asynchronous event driven JavaScript runtime coding environment. It is designed to build scalable network applications, and may handle several connection concurrently and when there is no work it will sleep [39].

npm: npm[5] is the default package manager for Node.js. It allows for downloading packages

[1] Virtual Private Network
[2] JavaScript Object Notation
[3] Integrated development environment
[4] SSH File Transfer Protocol
[5] Node.js package manager

which may help the development process.

Express.js: A lightweight framework for Node.js which allows for quick and easy creation of a robust API, and web applications in general [40].

ESLint: A tool for detecting code patterns and vulnerabilities in JavaScript code [41].

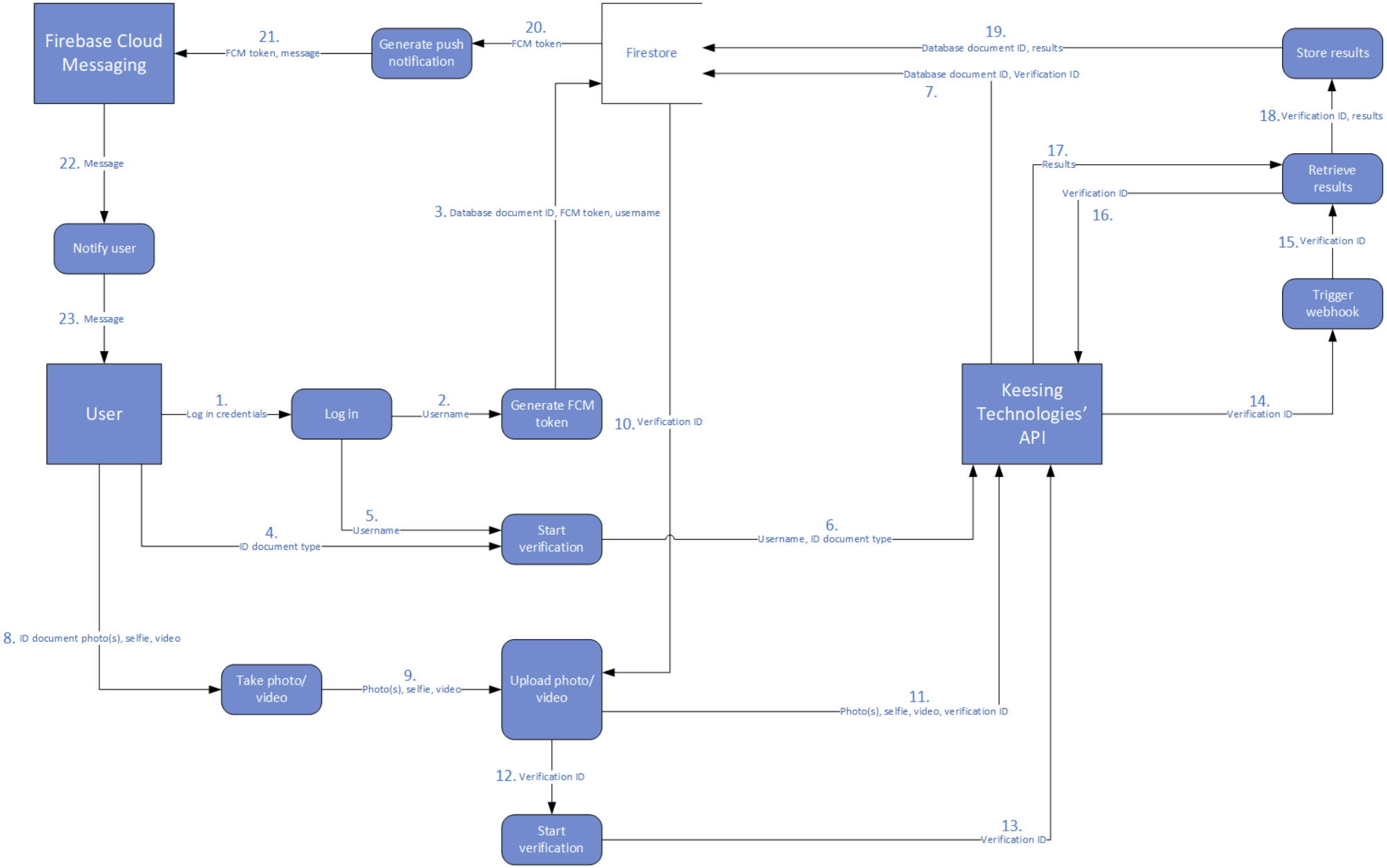
Postman: API tool for sending requests and receiving responses. It allows for JSON formatted payload amongst others, and provides for tests of API's.

SonarQube: Acts as an online server providing information regarding code quality. It is able to detect bugs, code smells, security vulnerabilities and duplication of code.

Adobe Photoshop: A graphic design software (raster graphics editor) used for creating the GUI of the application.

Microsoft Visio: Microsoft program for creating diagrams and drawings

F Data Flow Diagram

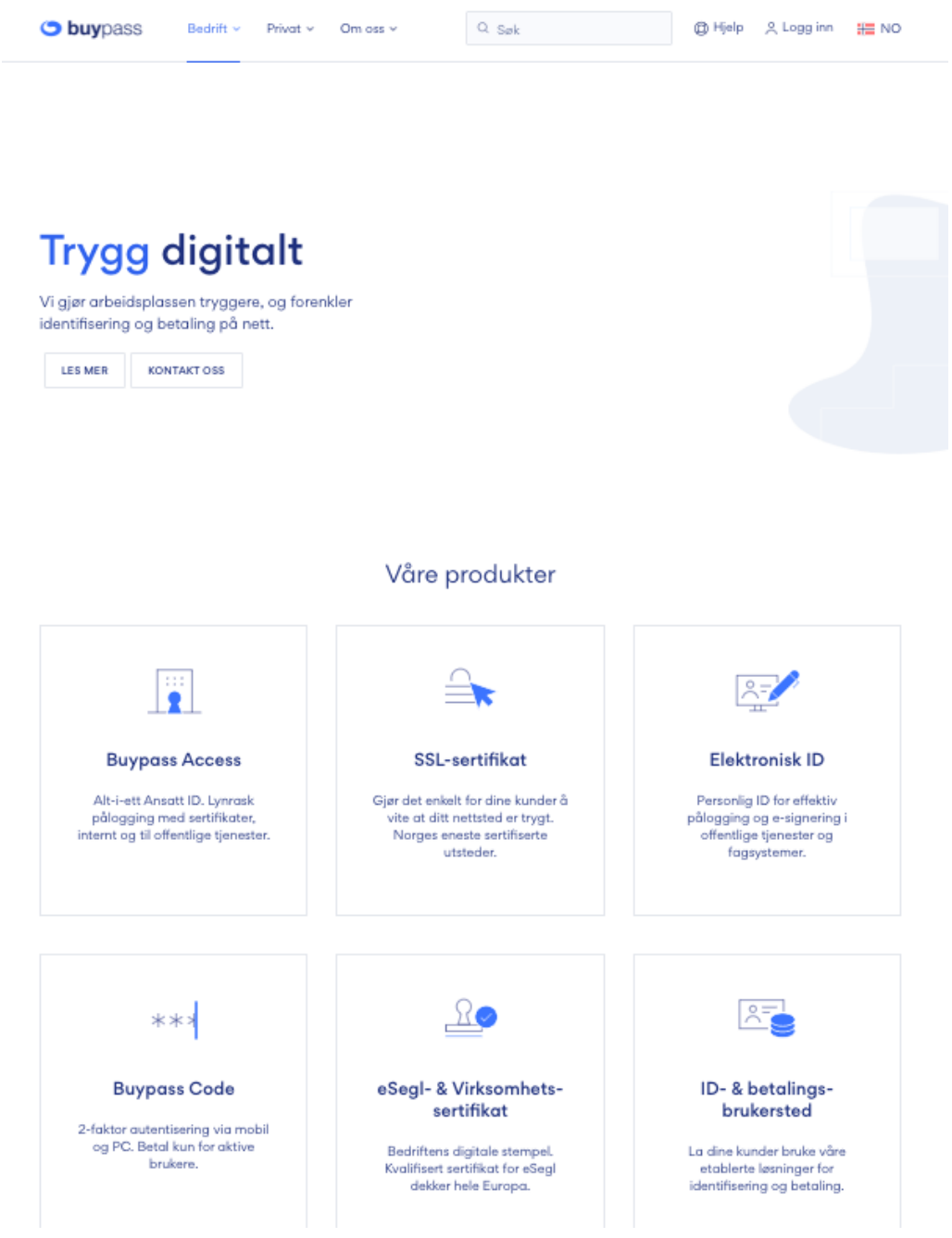

Firebase Cloud Messaging
21. FCM token, message
Generate push notification
20. FCM token
Firestore
19. Database document ID, results
Store results
Database document ID, Verification ID
7.
18. Verification ID, results
17. Results
Retrieve results
Verification ID
16.
22. Message
3. Database document ID, FCM token, username
Notify user
15. Verification ID
Trigger webhook
23. Message
User
1. Log in credentials
Log in
2. Username
Generate FCM token
10. Verification ID
Keesing Technologies' API
14. Verification ID
5. Username
4. ID document type
Start verification
6. Username, ID document type
8. ID document photo(s), selfie, video
Take photo/ video
9. Photo(s), selfie, video
Upload photo/ video
11. Photo(s), selfie, video, verification ID
12. Verification ID
Start verification
13. Verification ID

G JSON verification result


```
{
  "id": "7b473e7c034d40dbae19d275c871aedb",
  "reference_number": "7b473e7c034d40dbae19d275c871aedb",
  "state": "finished",
  "created_at": "2015-01-07T14:03:43Z",
  "error": "null",
  "results": {
    "status": "ok",
    "description": "Verification finished successfully",
    "remarks": null,
    "report_url": null,
    "checks": [
      {
        "name": "mrz_reading",
        "description": "MRZ reading",
        "status": "ok",
        "message": "MRZ read successfully"
      }
    ],
    "document": {
      "doc_type": "PASSPORT",
      "document_number": "999999990",
      "given_names": "OLA",
      "surname": "NORDMANN",
      "gender": "male",
      "date_of_birth": "1965-03-10",
      "nationality": "NOR",
      "personal_number": "11111111111",
      "valid_through": "2024-03-09",
      "issuing_country": "Norway"
    }
  },
  "face_comparison": {
    "match": true,
    "error": null
  },
  "liveness_check": {
    "alive": true,
    "error": null
  }
}
```


Listing G.1: Verification result JSON object

H Buypass' website


buypass
Bedrift
Privat
Om oss
Søk
Hjelp
Logg inn
NO
Trygg digitalt
Vi gjør arbeidsplassen tryggere, og forenkler identifisering og betaling på nett.
LES MER
KONTAKT OSS
Våre produkter
Buypass Access
Alt-i-ett Ansatt ID. Lynrask pålogging med sertifikater, internt og til offentlige tjenester.
SSL-sertifikat
Gjør det enkelt for dine kunder å vite at ditt nettsted er trygt. Norges eneste sertifiserte utsteder.
Elektronisk ID
Personlig ID for effektiv pålogging og e-signering i offentlige tjenester og fagsystemer.
Buypass Code
2-faktor autentisering via mobil og PC. Betal kun for aktive brukere.
eSegl- & Virksomhets-sertifikat
Bedriftens digitale stempel. Kvalifisert sertifikat for eSegl dekker hele Europa.
ID- & betalings-brukersted
La dine kunder bruke våre etablerte løsninger for identifisering og betaling.

I Uploading Files from the Application

I.1 Volley

For uploading regular JSON objects we have used a HTTP library called Volley. This enables the possibility for our android application to do network requests. It supports raw texts, images and JSON [42].

I.2 Retrofit 2

Retrofit 2 [43] is a HTTP client tool for uploading media files. It uses two different classes which is provided by the OkHttp (section I.2.1) library:

- RequestBody - used for sending a text or value.
- MultipartBody.Part - used for sending a file which includes the binary file data, name on the file and the content type.

Using Retrofit 2 it needs to declare the interface which is in the form of an multipart post request. This will allow to encapsulate multiple files and strings into one single post request. Further on, it has to generate a service client for the upcoming upload. This is done by creating a ServiceGenerator that will connect to the given URL address resulting in a client enabled to execute network requests on that specific address [44].

I.2.1 OkHttp

OkHttp is an open source project included in the Retrofit 2 client tool as mentioned in section I.2. OkHttp is an HTTP client which allows for multiple HTTP requests [45].

J Test cases

Test case - Login	Expected outcome	Success (Y/N)	Problem	Mitigation
Username/passwd	Successful login	Y		
Username/ no passwd	Unable to attempt login	Y		
No username/ passwd	Unable to attempt login	Y		
Neither username or passwd	Unable to attempt login	Y		
Test case - Slider tutorial	**Expected outcome**	**Success (Y/N)**	**Problem**	**Mitigation**
Go through tutorial	Swipe through, or press "neste", press "ferdig"	Y		
Press "hopp over"	Skip the tutorial, come to login	Y		
Test case - Verifications upload	**Expected outcome**	**Success (Y/N)**	**Problem**	**Mitigation**
Retrieve token with correct credentials (server)	Retrieve token for other processes, 200 (OK)	Y		
Retrieve token with incorrect credentials	Unable, 400 (Bad request)	Y		
Retrieve entry point (server)	Entry point for verification access, 200	Y		
Initiate verification instance (server)	Status code 201 (Created)	Y		
Upload photo of document	Successful upload, proceed to next step, 200	Y		
Upload too small JPEG file	Unsuccessful upload, retake photo, 440 (Invalid file)	Y		
Upload blurred photo of document	Unsuccessful upload, retake photo, 443 (Invalid file)	Y		
Upload too short video	Unable, restrictions set to length and size, 440	Y		
Upload too big video file	Unable, restrictions set to length and size, 440	Y		
Upload file, wrong file type	Unable, 400	Y		
Upload file, no file attached (server)	Unable, 400 (Resource not found)	Y		
Upload file type two times restricted to one (server)	Unable, error code 400 (Bad request)	Y		
Upload only one file type "face" (server)	Unable to perform face comparison	Y		
Start verification process (server)	Status code 200 (OK)	Y		
Start verification procees, wrong verification ID (server)	Unable, error code 404 (Not found)	Y		
Start verification process, finished verification (server)	Unable, error code 400	Y		
Start verification process, no files uploaded (server)	Unable, error code 400	Y		
Test case - Min side/ verification results	**Expected outcome**	**Success (Y/N)**	**Problem**	**Mitigation**
Get result of finished verification process (server)	Successful	Y		
Get result of non existing verification (server)	Unable, error code 404	Y		
Get result of unfinished verification (server)	Unable, error code 400	Y		
List all verifications performed by user which have some	List of the verifications performed	Y		
List verifications so that the listview is scrollable	Scroll down to see more verifications	Y		
Refresh the list of verifications by dragging list down	Refresh the list (if not visual in app check server log)	Y		
Press a verification in list to see detailed information	Popup providing detailed information	Y		
User page when there is no verifications for the user	Provide message stating that there is no verifications yet	Y		
Retrieve results when webhook is triggered (server)	The webhook triggers functionality for retrieving the results of the verification, and stores it in the database	Y		
Push notification, document to helpdesk	Push notification not sent as the verification is still ongoing	Y		
Push notification, liveness not performed	Push notification not sent as the verification is still ongoing	Y		
Push notification, face comparison not performed	Push notification not sent as the verification is still ongoing	Y		
Push notification, either one of the above not performed but the other two or one	Push notification not sent as the verification is still ongoing	Y		
User press push notification when not in the application	User is taken to log in screen	Y		
User press push notification whithin the application	User is taken to the user page	Y		
Test case - Application flow (General flow with buttons)	**Expected outcome**	**Success (Y/N)**	**Problem**	**Mitigation**
User chooses passport for verification, press home before photos	Screen saying that this will cancel the verification process (Yes/No)	Y		
Chooses to cancel home	Resumes the verification process	Y		
Chooses to be taken to home screen	Is taken to the first screen, and is now able to choose either passport of licsense again	Y		
The above applies to all stages of the verification processes----------	----------	----------	----------	----------
User presses button to take a photo within the app	Our camera opens up and a popup message up top comes to sighth, providing instruction for the photo	Y		
The user presses the instructions popup	The instruction popup closes	Y		
The user presses the info icon to get the information again	The instruction popup opens	Y		
The user takes a photo	The photo is now able to be reviewed by the user	Y		
The user presses button to retake photo	The photo goes away, and the user is able to take a photo again	Y		
The user approves the photo and presses the button to go on	The photo is being uploaded and the user is taken to the next step	Y		
The above applies to the video functionality as well----------	----------	----------	----------	----------
The user presses the user page button in order to see the verifications	The user page will appear	Y		
The user presses the same button again	The user page wil close	Y		
The user presses Home while in the user page	The user is taken home if he accepts and stays if he presses no	Y		

K API structure

URL	Description	Method	Parameter(s)	Return value(s)	Postman test
Verifyid.ml/		GET		«API for bachelor thesis, NTNU Gjøvik»	tests["Body matches string"] = responseBody.has("API for bachelor thesis, NTNU Gjøvik");
/verifications/	Call for instantiating a verification process	POST	refNo - string, unique user reference)	statusMessage	var data = JSON.parse(responseBody); tests["Instantiate verification process"] = data.statusMessage == "Verifiseringsprosessen har startet";
/verification/fileupload	Call for uploading image/video file	POST	FORM file – file upload type – string, type of upload userId – string, unique user reference	statusMessage	
/verifications/startverification	Call for starting a verification process	POST	userId – string, unique user reference	statusCode	var data = JSON.parse(responseBody); tests["Start verification process if not started, response 200"] = data.statusCode == 200; tests["Start verification already started, response 500"] = data.statusCode == 500;
/verifications/getpdfreport	Not functioning	POST	userId – string, unique user reference		
/firebase/uploaduser	Call for storing user and token in database	POST	user – string, unique user reference token – FCM token UUID – UUID for identifying in regards to retrieval of verification data	422, «Brukernavnet er ikke gyldig» / message	var data = JSON.parse(responseBody); tests["message ok if username valid and token provided"] = data.message == "OK"; tests["token or username not provided"] = data.statusMessage == "Noe gikk galt" && responseCode.code === 422; tests["token or username not valid"] = data.statusMessage == "Noe gikk galt" && responseCode.code === 422;

/firebase/getallverifications	Call for retrieving all verifications	POST	userId – string, unique user reference	Verifications array, empty or not	var data = JSON.parse(responseBody);
					
					tests["User with not verifications"] = data.verifications.isEmpty();
					
					tests["No username provided"] = data.verifications.isEmpty();
					
					tests["User with several or one verifications"] = !data.verifications.isEmpty();

/webhook/	Call for triggering the webhook	POST	event – string, “verification.finished” / “verification.report.created” / “verification.liveness.finished” / “verification.facematch.finished”		var data = JSON.parse(responseBody); tests["Invalid webhook event"] = data.message == "Invalid request"; tests["Push notification already sent"] = data.statusMessage == "Push notification already sent";

L Retrospective Meeting logs

L.1 Sprint 1 Retrospective Meeting

L.1.1 What worked well?

We have learned a lot, and all the mistakes/problems throughout the sprint will help us in the report. We actually reached our sprint goal, even though we ran into several problems. The last week we were good at putting down the hours needed to reach our goal. We made the decision to create a server on our own due to lack of communication in the last week. But Buypass were satisfied with our solution.

L.1.2 What could be improved?

Need to evaluate all different sides of a solution, so that there is no implementation of a solution that will not be used, like the serverless Firebase implementation again. The last week there were more defined work tasks, and this gave a better workflow. At the same time we need to be better at conducting work within all the different aspects of the project. This is important so that we are not dependent on one person for a specific task or problem. Get better at writing and reflecting around the choices we make.

L.1.3 What will we commit to doing in the next sprint?

Sprint 2 goal, implement a lot more functionality in the apiary documentation

L.2 Sprint 2 Retrospective Meeting

L.2.1 What worked well?

L.2.1.1 Workflow

The first week of the sprint we focused on working in parallel where Vegard worked on Firestore, Jørgen worked on the server solution of parsing incoming multipart/form-data and Karoline and Ruben worked on uploading the multipart from the application. This has enabled us to work on several different problems at the same time, and by rotating this throughout the project we will ensure that we are not dependent on one person for a specific problem.

L.2.1.2 Demo

The demonstration went well, and the application works well with the backend and the logging to the database. The flow of the application is fine, but there are some improvements.

L.2.2 What could be improved?

L.2.2.1 Workflow

During the last days of the sprint we saw that the time limit was catching up on us. This made us prioritize the lacking functionality on the server regarding the file upload, and as this makes the chance to find the solution four times bigger, it also restricts the progress on other aspects. This is something that we need to consider, but as for this time it happened it

was okay because there were no other priorities in the application.

We need to be better at writing javadoc for our functions, and in general.

L.2.2.2 UX

Take the user more through the process before hand, as well as during the verification process. Have a little tutorial before hand maybe, and include both figures and much more text in order to clarify what the user is going to do, and what the purpose of it is. We can also eliminate the main screen, not really relevant.

L.2.2.3 API

We we're not able to reach our goal of uploading the photo to Keesing, but we are close. We are apparently lacking the proper file extension in the upload, and we're looking into it.

L.2.3 What will we commit to doing in the next sprint?

Get the file upload to work, and also implement a NFC reader for passports.

L.3 Sprint 3 Retrospective Meeting

L.3.1 What worked well?

Everyone has always tasks to do, we have been good at delegating tasks. There has been a high level of productivity, and we're getting better and better at the parallel working. The time we have dedicated to working on the project is being used in a productive way

We reached the sprint goal, and we're on route for the end goal.

L.3.2 What could be improved?

Smaller merges rather than big ones that take up more time.

Get better at documenting choices we make, and why we do it. Also documenting problems that occur.

L.3.3 What will we commit to doing in the next sprint?

Implementing push notifications, "user page" for the user and complete the GUI for demonstration purpose. Maybe extend the NFC functionality. Maybe take a look at Camera2 API.

M Sprints

Sprint 1 - 4th to 15th of February The sprint goal was to start on the Android application, and implementing the camera functionality. We wanted to make our own camera, implementing the Camera2 API. We also wanted to establish connection with Keesing Technologies, the third party company which would handle the verification aspect of the application. We also wanted to create our server, and to establish connection between the mobile application and the server.

The sprint planning meeting was conducted on the 4th of February, but prior to this we had a meeting with Buypass on Friday the 1st of February. During this meeting we discussed a work around for the disclosed information prior to the meeting. The third party company handling the verification aspect of the application did not have the functionality for NFC yet. We were told that the functionality would be in place by the end of February, and we scheduled a meeting with Buypass and the Norwegian representative of Keesing for the upcoming week. Unfortunately this meeting was postponed into the next sprint due to illness.

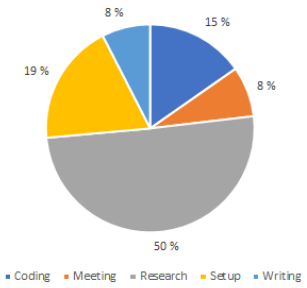
We implemented the camera within the application relatively fast in a way of starting an intent which launched the existing camera application of the mobiles. We went on to implement the camera2 API but we came to the conclusion after working with it for some time that we were not going to be spending more time on it at this stage of the development process, and that we would come back to it at a later point. This would allow us to focus on implementing the functionality regarding the verification process, and we felt that this was the right choice as the application now was able to take a photo.

During our sprint retrospective meeting we concluded that we were happy with the result of the sprint. We learned a lot during the first sprint, and it was a realization for us in the way that not everything will run smoothly in development work. We made decisions on our own regarding the server solution, and also the camera implementation at that time. We chose to go with an easier implementation, and we would come back to it at a later time when we had more functionality implemented.

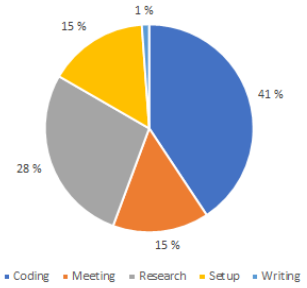
We saw some things that we could improve on, such as dedicating more time to research before settling on a solution. This improvement was made due to the choice we made of using Firebase as our server solution, before we decided to make our own server with Node.js and Express.js. We also saw during the last week of the sprint that when we defined the tasks better for each of us the work flow got better.

During this sprint we conducted 211 hours, 50 minutes and 54 seconds. We have separated the time conducted into categories provided in figure 28.

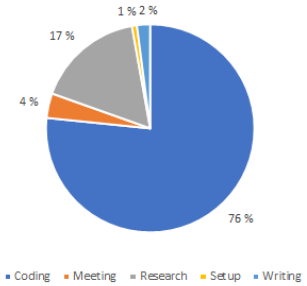
Sprint 2 - 18th of February to 1st of March The sprint goal for sprint 2 was to verify a passport at Keesing through the use of our application, and there after store the verification in the database. We also wanted to incorporate a NFC reader for this sprint.



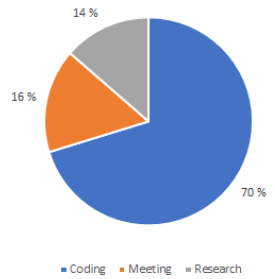
(a) Sprint 1



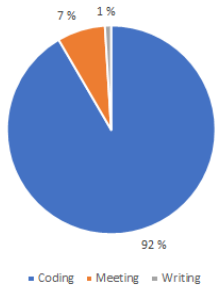
(b) Sprint 2



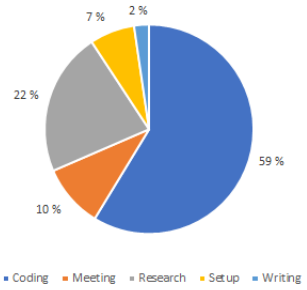
(c) Sprint 3



(d) Sprint 4



(e) Sprint 5



(f) Overall project

Figure 28: Pie charts representing time consumed to different activities within each sprint, and of the overall project

During this sprint we had issues regarding file uploads to Keesing Technologies' API, and did note reach our sprint goal due to this issue. We worked on the problem a lot, and Keesing was contacted regarding the issue.

We made an decision to work on the NFC reader of the application, and by doing that we isolated the problem of the file upload and made the development progress.

We had a status meeting with Buypass by the end of the sprint, and got some feedback on the GUI of the application. This feedback mainly focused on the user experience of the application, and how they wanted us to take the user on a "journey" through the application. They wanted us to make more use of verbal explanations, and also to make it less formal. Take the user more through the process before hand, as well as during the verification process. Have a little tutorial before hand maybe, and include both figures and much more text in order to clarify what the user is going to do, and what the purpose of it is.

In the sprint retrospective meeting we we are pretty happy with the result of the development process, even though we did not reach our sprint goal. The sprint goal was hard to reach due to the problems that occurred during the sprint, but we were happy with the work done in the sprint. We had a high level of productivity, and the parallel working in the group were doing better and better. The parallel working is conducted as a precaution for development work. It allows two people to work on the same field of development/technology, and at the same time it makes it less probable to be affected by personal problems.

During this sprint we conducted 220 hours, 23 minutes and 2 seconds. The time is illustrated in figure 28.

Sprint 3 - 4th to 15th of March The main goal of sprint 3 was to be able to go through a whole verification process, uploading photos and to get the results from Keesing. This was lacking a bit from the previous sprint in form of the file uploads. We we are also hoping to get the NFC reader implemented for the sprint.

We concluded that when we would run into more problems as we did in the previous sprint we could perhaps work with something else the next day, as it did become demotivating to work for days on something we were unable to implement.

During the sprint retrospective meeting we all agreed that we we are happy with how we are always doing something. The members of the group is always conducting some work related to the project. We saw during this sprint the advantages of having smaller merges rather than too big ones, as it got a bit problematic solving some merge conflicts during this sprint. This is something we proceeded with for the next sprints. We also agreed on that we would be better at documenting choices we made, even though we did not write much about them. Just include a small note about why we did what we did.

During sprint 3 we conducted 250 hours, 31 minutes and 52 seconds. This is illustrated in figure 28.

Sprint 4 - 18th to 29th of March The goal of sprint 4 was to have a proper demonstration of the application for Buypass, with just some GUI touch up remaining. We also had to just make some small changes to the code regarding our webhook to make it work, and

look into downloading the PDF report from Keesing. We would also focus on writing tests and test cases if the time aspect would allow it. As we talked about in the first sprint, we would now go back on the camera, and implement the Camera2 API. This would allow for us to make a customizable camera, making it more user friendly.

The sprint goals where met to a certain extent, where we had a demonstration with the Camera2 API implemented and the webhook working fine. We got feedback on the application, and took this feedback with us for the next sprint. This feedback was regarding the push notifications and how to make them more stable, small bug fixes and some more work on the GUI to make it even more user friendly.

During sprint 4 we conducted 174 hours, 51 minutes and 8 seconds. This is illustrated in figure 28.

Sprint 5 - 1st to 12th of April The goal of sprint 5 was to take the feedback we received during our presentation in the last sprint, and apply the changes needed to the application. We also wanted to minimize the amount of bugs in the application, and to make it as user friendly as we could. The end goal of the sprint was a demonstration for Buypass on Wednesday the 10th, where they would do a recording of the application, showing what it could do. This would also be the meeting with Buypass for this sprint, and the last two days were conducted to bug fixing and testing of the application.

During sprint 5 we conducted 241 hours, 54 minutes and 17 seconds. This is illustrated in figure 28.

N Version control

Version 1.0.0 Enabled our first working camera application. A button has been initialized and set to start the camera intent. This invoked and launched the existing camera application. In this version we enabled the functionality to take a picture.

Version 1.1.0 This version contained a lot of new functionality. A GUI within the requirements to style was implemented, and enabled the user to navigate through the application. Both driver's license and passport verifications was implemented. The application was able to take a picture, and to upload the it to the integration manual API. The implemented NFC reader was working. We implemented popups which provided information and guidance to the user upon every functionality choice. Snackbar functionality was implemented, providing information in regards to successful, or not, uploads.

Version 1.1.1 The app was now communicating with our RESTful API which was made with Node.js. This version offered the functionality for images and recordings to be uploaded to our server. The server would then send the request to Keesing's API. The server would receive the results from the API and send it to the application. The server would further on communicate with the Firestore database and send a push notification to the application. The result could be reviewed in the application.

Version 1.1.2 A new simple user page was constructed which displayed the full verification result from Keesing. Version 1.1.2 was the first fully functional application ready for demonstration purposes.

Version 1.1.3-old A new GUI design and improved UX. Automatic popups were disabled, and a information button was introduced that would show the popups. A toolbar was implemented with two buttons, a home button and a user page button. We improved our camera functionality by implementing the Camera2 API. This version has also restricted the SDK[1] level to a minimum of 26. A new and improved user page GUI was also introduced.

Version 1.1.3 The last and most updated version. This version contains a new GUI. It focused more on UX, and provides a information box in every screen. The user page was improved by making a list out of all the verification results the user may have and a popup showing the report when pressing a specific verification instance. Within the camera we introduced a rectangular border where the document should be placed, making it harder for the user to provide an insufficient photo.

[1] Software Development Kit

O Meeting logs supervisors

Veiledning 15/1

Hvordan er det med offentliggjøring av kildekode og eventuell ferdig nettside/applikasjon, kan dette holdes unna offentliggjøring men være åpent for sensur?
Ikke noe problem, men sensur må se hvordan kildekoden er i form av god kode-praksis

Kan vi ha ukentlige møter, eller om ikke mulig, annenhver uke? Gjerne til en fast dag og tidspunkt så fremt dette lar seg gjøre?

What type of process?
API as a architectural drawing
Do we need to let them read the thesis
We dont want to public any secrets
Be careful with the data
Store it locally, safe

Discuss english and norwegian

Organize the group properly, maybe a tech lead or security leader

Conflicts - make rules for the bad days
Inform Deepti and Rune early on, tell them if warnings are given to a person for not showing up.

Reflect and analyticals over source code, that needs to be good, but it's more important to write a good thesis

Pair programming may be good for periods

Be active, and present as much as possible for meetings with Buypass

Define the main goal for reflecting in may
Project goals
Result goals - what did we do to help them achieve this

Plan meetings with Buypass to show them what is done within sprints, but if there is not much done, then there may not be a need to meet as much
More frequent meetings in the beginning

Mandag klokka 11, Ukentlige møter med Deepti og Rune
Tre punkter:

1. Hva har vi gjort siden siste møle?
2. Hva er det vi ønsker å diskutere, referere til dokumenter og andre ting?
3. Hva er planen for den kommende uken?

Spesifikt si hva som skal gjennomgås, kapittel det og det i det og det dokumentet slik at vi får bedre feedback

Møte med rune 21. januar 19

Tilstede var Rune, Karoline, Ruben og Jørgen

Evaluere Kotlin mot Java, men foreløpig Java i tet.

Rolleinndeling

Tips:

- Prøv ikke å fordel for absolutt, dette er sårbart dersom en arbeidsoppgave er mindre kompleks enn en annen. Mindre utnyttelse av ressurser, men det er mer sikkert og man er ikke sårbar dersom en person er borte eller mindre effektiv.
- Robusthet versus effektivitet, men vær fleksible og kjør gjerne parprogrammering eller inndeling i par noen ganger.
- Må gjerne inndele i «eksperter» som kan coache hverandre, og muligens holde workshops innimellom.

Fremdrift

Det mest grunnleggende må på plass først, hovedfunksjonaliteten til applikasjonen.

Risikohåndtering, fokuser på de største utfordringene. Ta det med størst usikkerhet først, start med keesing og implementasjon av API først for eksempel.

Scrum

Ved hver sprint, skal det kjøres et scrum planning meeting, da gjennomgår vi hva som har blitt gjort og hvor effektivt dette har vært, retrospective meeting. Samtidig må vi se på hva som er planlagt å gjøre fremover, og om dette er mulig eller må justeres.

Gjennomgå punkter:
Hva skjer de neste ukene?

- Noe som gjør at man må jobbe mindre?
- Hva er det viktigste å få på plass?
- Se på hvor mange timer vi har og hvor mye vi kan få til på disse timene?
- Estimere og dokumentere underveis for å muligens endre funksjonalitet ved å se om man klarer å nå de milepælene som er sagt
- Benytte det retrospektive hvor og se hvor bra man fungerer som team, og hva man har fått gjort i løpet av den forrige sprinten
    - Jobber man bra sammen?
    - Er man på riktig vei?
    - Må vi endre på gruppesammensetning dersom vi har delt inn?

Milepæler

Milepæler som må vise til hvordan man ligger an i prosjektet, ikke for ofte og ikke for sjelden. Gjerne mer sjelden dersom det er mer usikkerhet. I starten vedrørende Keesing bør vi ha flere, og det er vanskelig å estimere ettersom det er usikkerhet knyttet til det. Det er mye mulig det er enkelt å håndtere API'et og det løser seg fort, men det kan også hende vi må se etter løsninger som må implementeres og det tar mer tid.

Demo muligens hver andre sprint for å få tilbakemelding, milepæler etter to-tre sprinter. Trengs systematikk for å vite at man ikke kommer ut helt feil.

Testing
Sjekke for gode rammeverk for android testing, finnes mye i web-basert programmering.

Lurt å ha en testplan, med tanke på hva slags testing.

Vi må benytte enhetstesting, og samtidig se på testing for hele prosjektet. Selv om en modul kjører og fungerer som den skal alene så vil den muligens ikke fungere ved implementasjon i hele applikasjonen.

Sjekk etter OWASP-testing.

Se etter automatisering av rammeverk for testing, som går gjennom workflowen til applikasjonen.

Legg til rette for en god teststrategi tidlig

Code review bør gå på tvers av inndeling dersom vi deler inn absolutt.

Må ha et øye for testing gjennom kodefasen, og code review skal gjøres unna. Ikke la denne hope seg opp slik at man plutselig har flere timer eller dager med code review.

Generelt
For tilbakemelding på bacheloroppgave, skriv hva som er nytt og endret, samt hva vi vil ha tilbakemelding på i starten av dokumentet. Kom heller med litt og litt nytt innimellom, og ikke masse av gangen. Da blir det lettere å skulle gi tilbakemelding dersom det er snakk om 30 minutter istedenfor tre timer.

Møte mandag 28/1

Prosjektplan
Arkitekturmodel
Vedlig generell. Hva er serveren sin funksjonalitet, kun kommunikasjon, eller er det mer?

Use case
Fikse pilene, og gjøre det i Visio. Ikke la applikasjon være en egen en, muligens ha selvaktivering?

Sekvensdiagram
Fikse så brukeren initierer prosesser

Ansiktsgjenkjenning
Trenger dette å inkluderes, siden det håndteres av Keesing?

Utvidelse
Oppgaven kan virke liten, og hva er det vi da kan utvide med?

Kan man utvide til å få lansert applikasjonen på web også?

Det må skrives om at det er en risiko for at det er for lite å gjøre, og dette må diskuteres. Hva gjør vi dersom det er for mye eller for lite arbeid?

Ordvalg
Dokumentasjonsansvarlig, ikke sekretær.

Testing
Prøv å integrere automatisert testing, og om vi ikke finner noen verktøyer så må vi skrive egne scripts.

Automatiser alt som kan automatiseres, og dersom det trengs input noen steder så gjør man det manuelt om man ikke får automatisert.

Start med å skrive test caser, og deretter tester, men da må også kravspesifisering være noenlunde i orden.

Kravspesifiering
Presenter det som at det kan komme flere krav underveis, dette er viktig for en smidig utviklingsmodell.

Møte 4/2 Deepti og rune

Diskuter i rapporten hva slags database vi vil bruke, det er en fin prosess for rapporten.
Sette oss inn i databasene, og SQL, se på hva vi mener er det riktige og bruke, og deretter få feedback fra Deepti og Rune.

Vi finner det vanskelig å implementere et “eget” kamera, og kan allerede . Håper å få det til
Hva er fordelene og ulempene med begge kameraene. Hvorfor vil vi helle har et kamera i applikasjonen istedenfor å bruke det vi allerede har? Argumenter for dette i rapporten også.

Få mest mulig erfaring med API’et, da blir det lettere å få stilt spørsmål.
Få hentet ut informasjon på visse punkter, så vi vet hva buypass vil ha.

Bryt ned sprint backlog, det gjør det lettere å jobbe parallelt. Bryt ned det som tar mer enn to dager, lettere å holde oversikt. Det gir også mer mestringsfølelse ved at man faktisk får krysset av noe i Trello.

Team - ta avgjørelser og jobb deretter. Mulig vi velger feil, men vi er produktive og jobber selv om det ikke er det beste å jobbe med. Gjør det lettere å jobbe videre.
Vi har ikke hatt noen diskusjoner ennå, men det er viktig å huske på at det kan bli problemer.
Dokumenter diskusjoner, og få disse punktene ned for å benytte i prosjektrapporten.
Hvordan tar vi avgjørelser, på bakgrunn av hva Buypass vil ha senere eller om det går på hva som er best for oss å benytte.

Prøv å være generelle, dersom vi vil benytte en database og dersom den skal byttes så påvirker det ikke så mye kode. Lag uavhengig kode, fordi det gjør det lettere å endre visse kodesnutter senere.

Branches - huske å merge master inn i versjoner dersom det kommer bugs så de kommer inn i de forskjellige versjonene.

Møtereferat 18. februar

Setup a database for client data, and logging the server traffic.

We think the camera quality is good enough for now. We should systematically test different camera situations, different light, and other thing that may interfere with the quality.

Need to plan more ahead, send documents to Buypass more in advance to make sure that they are reminded of the meetings, and they are prepared for the meeting.

NoSQL - for now at least
SQL - maybe later
Start looking at MongoDB querying and data modeling. Shared data, you can use FK or do a hierical tree.
Makes it easier for the later when we get the requirements, because the research is done already.

Keep the workflow, and open up for the possibilities that you may go back on a decision.
Always going back on decisions - not good.

Møte med Rune 25/2

Noter rundt jobben ved å flytte koden fra en plattform til en annen. Få det på plass i rapporten.

Problematikk i det å gå tilbake rundt kode. Ser man et problem når man gjennomgår kode, så lag en issue. Gjør commits rundt issues, derfor kan man se når man endret en issue og lettere se endringer hvor det gikk galt.

Møtereferat 4/3

Hvorfor nådde vi ikke målet? Ta det med videre, og noter hva som kan gjøres annerledes. Notere ned rundt hva som er gjort og ikke, hva som er lært og ikke. Viktigheten ligger i jobben som har blitt gjort, har vi tatt initiativ. Det er mange ganger i arbeidslivet at man møter mangelfull eller eksperimentell funksjonalitet i teknologier. Fortsette å skrive ned valg vi tar, ved å ta valg om å gå for NFC, at vi isolerer problematikk for å ikke være avhengig av den.

Få skrevet ned litt mer når det gjelder retrospective. Det er viktig i sammenheng med at vi skal diskutere resultatene ved et senere tidspunkt gjennom diskusjon og konklusjon. Gå tilbake på valg i retrospective, er de fortsatt effektive og de beste valgene?
Bør vi jobbe mer i par, individuelt? Tenke gjennom alt sånn.

Hvordan ble timeplanen i praksis i forhold til hvordan vi satte opp først og underveis.

Begynne å gå gjennom rådata, referater, timeplaner osv. Tematiser det og prosessen rundt hva som har skjedd på det området.

I konklusjonen, hva er naturlig å gjøre videre? Hva ville vi gjort/bør Buypass gjøre?

Retrospective, hva gjør vi videre basert på en diskusjon som er basert på resultatene.Gå dypere inn i resultatene for å hele tiden ha prosessforbedring. Se litt mer på det overordnede rundt hvordan det skal lages, ikke hvordan det lages.

Møtereferat Mandag 18/3

Kontakte Buypass etterpå

Dokumentere for hva som er automatisert og hva som er manuelt.
Unit og application tests, tester litt forskjellige ting.

Dokumentere hva slags påvirkning vi har mot Keesing API.

Kanskje kjøre noen flere forsøk ved verifisering.

Se på det med sikkerhet knyttet til server og database.

Møtereferat 25/3

Diskuter rundt de to mulighetene vi hadde med implementasjon av kamera, intent launching camera eller camera2 API. VIktig med tanke på reflektering i rapporten. Ville vi gjort det annerledes, dersom vi hadde visst at vi skulle fått det til som vi gjør nå. Det viktige er diskusjoner, så ta med dette videre.

Diskuter rundt hvordan vi har jobbet i forhold til det at Keesing har vært problematisk, og hvordan vi har gjort dette. Vi har tross alt vært beta-testere, og det har vært problematisk til tider. Hva gjorde vi for å komme oss rundt problemene, og hvordan har vi sørget for å minimere at vi har klart å komme oss rundt det. Vi har løst dette ganske godt.

Kanskje vi kan se på hvordan buypass sin login er, og dermed implementere en. Kan vi se hvordan buypass vil implementere vår funksjonalitet og dermed få merget dette på best mulig vis.

Diskuter rundt arbeidsmetoden, ved “parprogrammering”.

P Meeting logs Buypass

Møtereferat med Buypass AS 18/1

Alle gruppemedlemmer var tilstede på skype-møte med Jørn fredag 18/1 klokken 10.

Det ble avklart at hovedprioriteten ved oppgaven er å få på plass en native Android app, med scaling for både mobil og tablet.
Denne applikasjonen skal ta for seg ansiktsverifisering ved bestilling av enten elektronisk id eller mobil id. Dette er for å slippe den tid- og ressurskrevende delen hvor man må møte opp på postkontoret for å gjennomgå et personlig oppmøte med identifikasjon, det er da opp til den postansatte å verifisere at personen er den den utgir seg for å være. Dette ender ofte med feil ettersom de postansatte ikke har nok erfaring med denne prosessen eller nok kjennskap til id-kort som pass og førerkort. Dersom personen får hentet ut sitt id-kort vil man etter noen dager motta en aktiveringskode fra Buypass for id-kortet.
I helsesektoren benyttes det i stor grad id-kort, og de har egne ansatte dedikert til id-kort, og det vil være mye penger å spare på denne løsninger. Det samme gjelder for leger som må reise langt for å få utstedt disse id-kortene.

Det er et nytt EU-regelverk i gang, men det vil ta litt tid før dette skal revideres i Norge, men vi kan anta at reglene blir noenlunde det samme som for EU.

IDmee er en norsk avlegger av DnB som tilbyr omtrent den løsningen vi ønsker å lage. Det er mulig å gjøre research på deres, samt andre løsninger ute i Europa.

Leverandøren av API'et vi skal benytte er lokalisert i Belgia eller Nederland. De sitter på mye kompetanse innen id-kort, og har alle godkjente id-kort i sin database. Dersom det kommer et id-kort som skal verifiseres som ikke godkjennes automatisk går dette videre til et ekspertpanel som skal godkjenne eller underkjenne dette. Dersom sjekken på id-kortet går automatisk vil det være snakk om sekunder, og ved manuell godkjenning er det snakk om minutter.

Keesing Technologies var i ferd med å implementere liveness detection i november, og denne må vurderes opp mot andre slike, som IDmee sin løsning.
De har løst dette med matching av bilder med id-kort som ligger i backend.
Jørn tror det med å ta bilde av seg selv for godkjenning ligger i løsningen deres.
Test-api'et ligger mest sannsynlig tilgjengelig, og Jørn skal sende det.

Det blir skype-møter med Buypass hver andre fredag klokken ti til elleve, og det starter fra og med 1. februar.

Det vil være et begrenset antall personer vi kan kjøre lokal testing på, og det vil i stor grad gjelde for de involverte i prosjektet, muligens de rundt oss involvert i prosjektet dersom de er inneforstått med det.

Ta utgangspunkt i at teknologien for ansiktsgjenkjenning og diverse teknologier levert av Keesing Technologies er gode, og vi skal heller fokusere på implementasjonen av disse. Det

må begrunnes dersom vi skal benytte annen teknologi, og valg må tas på vegne av dokumentasjon og refleksjon.

For å utvikle server-siden av applikasjonen er Node.js fint å bruke, og for å utvikle Android-applikasjonen ser Buypass gjerne at vi benytter Kotlin eller Java, og dersom det er snakk om web er det fint å benytte React.js.

Vi får tilsendt diverse designelementer av Buypass når det kommer til logoer og andre ikoner.

Vi kan benytte bitbucket eller github for kildekode, så fremt dette gjøres i et privat repository. Google disk kan benyttes, men vi må se det an når det kommer til API og koden mot API'et.

Sikkerhetsmessig skal vi ta utgangspunkt i OWASP.

For veien videre til neste møte må vi gjøre research på tilsendt dokumentasjon og reelle teknologier.
Det bør lages en arkitekturskisse, over komponenter/moduler og hvordan disse snakker sammen. Det bør også inndeles hvem som jobber med hvilke komponenter, lage en overordnet plan. Denne planen bør utformes med hensyn på arbeidsmengde og tidsbruk.

Møtereferat med Buypass 1/2

Det er noe uklarhet vedrørende NFC/RFID og når denne funksjonaliteten kommer på plass. Funksjonaliteten skal være på plass mot slutten av februar, men det er usikkert når API'et kan håndtere det. Det er uansett ønskelig at vi skal implementere en NFC-leser som kan hente ut informasjonen som ligger lagret i pass. Det er ikke sikkert denne informasjonen blir benyttet, men det gir muligheten for videre utvidelse dersom API'et kommer på plass.

Forslag for videre utvidelser er webapplikasjon som nå er nevnt av veiledere og arbeidsgiver, derfor vil dette være en utvidelse om vi ser vi har tid til dette. Samtidig ble det nevnt en videosamtale med kundeservice i Buypass, og her er de i samtaler med en annen aktør som skal håndtere videooverføringen.
VI ble anbefalt å benytte Firebase, AWS eller Azure som rammeverk for server, og vi har valgt å benytte Firebase ettersom det var den de anbefalte mest. Det skal også lages en log/database på serveren som skal lagre vellykkede tilfeller av verifisering og der det feilet. Dette er for at Buypass skal kunne se på kundedata i en databasetabell, og se hendelsesforløp og eventuelle feilkoder i loggen.

Vi har fått tilgang til nøklene for å benytte API'et nå, men det kan hende at registrering av IP-adresse er nødvendig. Det er avtalt at et møtet med Keesing sin representant og Alexander fra Buypass skal forekomme neste uke enten tirsdag, onsdag eller torsdag.

Vi vil motta telefoner for testing av applikasjonen, og dette skal Buypass håndtere så fort som mulig. Buypass vil sette oss i kontakt med deres UX-ansvarlig, og da vil vi få mer input på hvordan applikasjonen skal se ut og motta forskjellige designelementer.

Det ble enighet om at vi skal presentere alt vi har gjort på neste møte, og at vi gjerne forteller for mye enn for lite. Dersom vi har noe å vise til kan vi gjerne presentere en demo, som gir en god indikasjon på hvordan vi ligger an. Dersom vi føler vi når noen milepæler underveis kan vi gjerne dele disse via mail.

Møtereferat 22. februar

Tilstede på videomøtet fredag 22.02 var Keesing sin representant i Norge Jonas, Alexander og Jørn fra Buypass og alle gruppemedlemmene.

Autorisering
Tok opp problematikken rundt det å få hentet ut en access token. command “curl https://linkt-to-token -d ‘grant_type:client_credentials&scope=public_api…..’”, men vi har ikke fått til å gjøre dette om til et POST-kall. Dette problemet kan videresendes til Buypass for å få hjelp av noen “inhouse” der, evt til Keesing om vi oppdager at problemet ligger der.

Dokumentasjon
Ta utgangspunkt i https://verificationsrv.docs.apiary.io/# ettersom denne går på det å iverksette en prosess, mens integrasjonsmanualen går mer på hvordan det administrative rundt ansikts- og id-verifisering håndteres, og funksjonalitet rundt dette. Se bort ifra integrasjonsmanualen nå. Dette gjør at det ikke lenger er nødvendig med statisk IP, og vi må revurdere om vi skal benytte oss av en VM med server, eller om vi skal gå tilbake til løsningen vi ville ha med Firebase.

Ansiktsgjenkjenning
Krever to bilder, men dette ene bildet skal være fra id-dokumentet som er lastet opp.

Svartid
Pass går gjennom sjekken med en gang omtrent 99% av gangene ifølge Jonas, og dersom det går til Keesing Helpdesk skal det gå fort der også.
De nyeste norske førerkortene vil gå til helpdesk ettersom de ikke er ferdig opplært ennå.
Ellers skal førerkortene stort sett gå gjennom. Dette gjør at vi må benytte oss av push varsler til telefonene, og dette ble vi anbefalt å bruke Firebase til.

NFC
NFC er ikke på plass ennå på grunn av sykdom, men de er i gang med å finne nye utviklere som skal gjøre dette ferdig. For å kunne hente ut all informasjonen fra pass må man først hente ut nøkkelen man kan låse opp resten av informasjonen.
Det ligger mye informasjon på nivå 1, og dette vil muligens holde for oss.

Liveness check
Jonas trodde denne gjorde en ansiktsgjenkjenning, men han var ikke sikker på dette.

Møtereferat med Buypass 1/3

Helpdesk er kontaktet vedrørende API-kallet for opplasting av filer. Håper på svar, og fiksing av problemet fortest mulig.

Se etter eksempler i liknende applikasjoner. Benytt verbose forklaringer. og ta brukeren gjennom hele prosessen. Kan få satt opp et møte med UX-designerne til Buypass ved et møte i Oslo på et senere tidspunkt.

Database, benytt id fra Keesing som nøkkelen til dokumenter i verifications collectionen.
Lagre den pågående nøkkelen, for å kunne sjekke status på denne lettere..
Lagre rapport som kommer fra Keesing, alt skal med. Brukes til statistikk og analyse.

Jørn tok oss gjennom slides fra et seminar torsdag 28.2 for å vise hvor relevant det vi jobber med er, og for å motivere.

Dersom det stopper seg opp lenger hos Keesing eller løsningen på problemet ikke kommer med en gang så kjører vi på med NFC, og implementasjonen av dette.

Møte hos Buypass 20/3

Keesing
API'et er som vi fikk opplyst forrige uke, i beta-testing. Det gir mer svar på mangelfull dokumentasjon og at det er såpass ustabilt. Vi kan kjøre to tilnærmede like verifiseringsprosesser hvor det blir to helt ulike resultater.

Vi henvendte oss også til Keesing på grunn av de ustabile resultatene på ansiktsgjenkjenning og liveness, men fikk da til svar at ansiktsgjenkjenning noen ganger ikke vil utføres, og noen ganger vil. Angående liveness så sa de at dette ikke er aktivert for kontoen vi benytter, men vi har ved noen anledninger sett at dette har blitt godkjent, eller underkjent. Det har også vært tilfeller hvor den ikke kjøres, men vi har fått error-koder som at den ikke finner noe ansikt, og det tyder på at det er aktivert, og at den prøver å gjennomføre det.

GUI
Tilbakemelding på GUI.
Vi har tatt utgangspunkt i fargepalett og designelementer fra nettsiden deres. Vi har tatt til ettertanke det å veilede brukeren mer gjennom prosessen, i form av popups og slidere.

PDF
All funksjonalitet utenom PDF-rapport er implementert, men det gjenstår finpussing og mer testing.
PDF-rapport får vi ikke hentet ut, selv om webhooken trigges, for vi får bare {"error":"not_found","error_description":"Record not found"}.
Vi sendte mail angående dette til Keesing på mandag, men har ennå ikke fått noe svar fra dem.
Det kan dog virke som at de ikke genererer noen PDF ennå, og at denne funksjonaliteten ikke er implementert fra deres side.

NFC
Vil dere vi skal se videre på NFC-sjekking, i form av å kunne finne ut om MRZ er blitt tuklet med, gyldighet av pass. Eller er dette noe dere ser på som irrelevant i og med at Keesing etterhvert skal håndtere det uansett.

Webhook
Det kan virke som at keesing trigger webhooken med en gang vi etterspør verifiseringsresultatene, og at den bare trigger uansett. For selv om den trigger PDF så er ikke dette noe som fungerer.

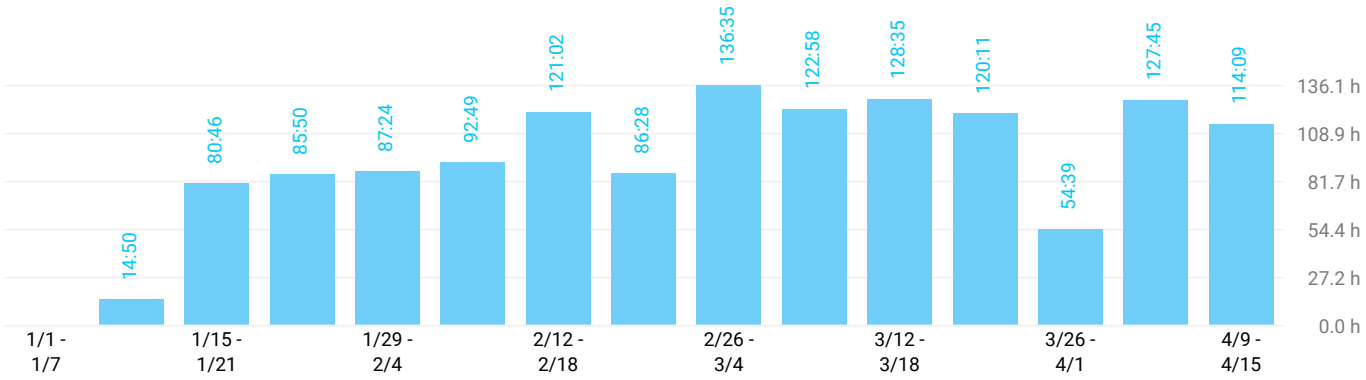
Mer folkelig, og hyggeligere. Man snakker til en kunde, og man vil etablere en dialog med kunden.
Camera2 API

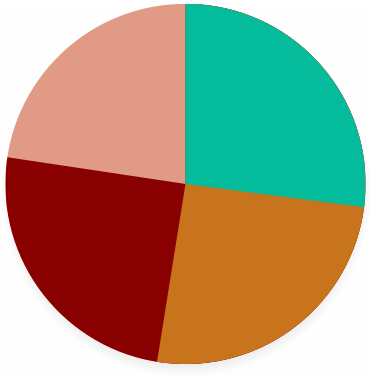
Q Toggl time reports

Summary Report

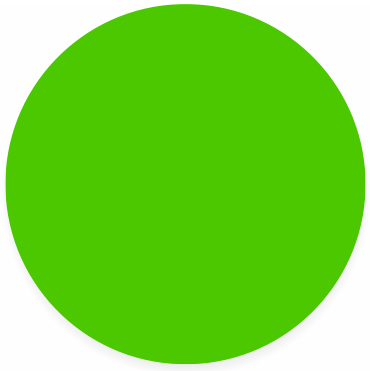
January 01, 2019 – April 12, 2019

TOTAL HOURS: 1374:05:25





USER	DURATION
KW Karoline W Rud	371:59:37
RU Rubche	349:48:49
JR Jrberntsen	341:17:45
AM Amleveg	310:59:14

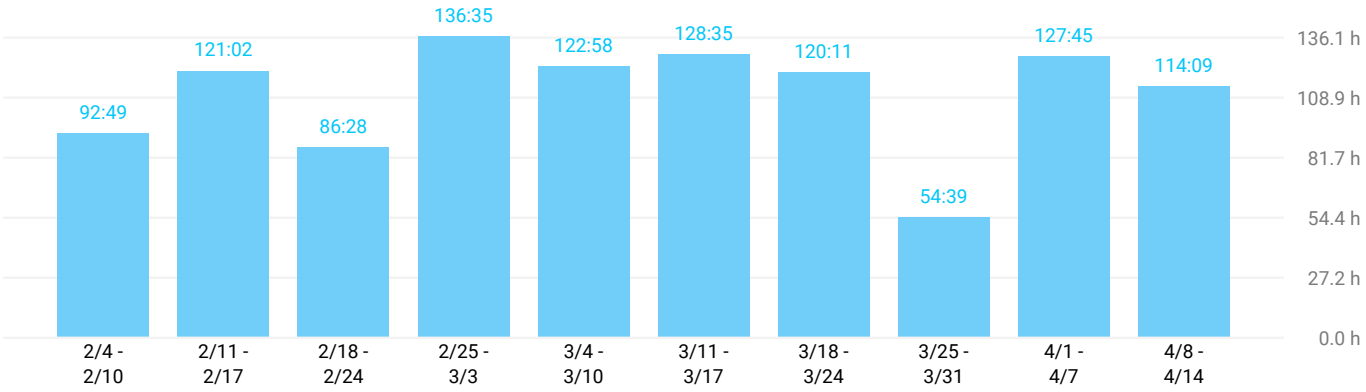


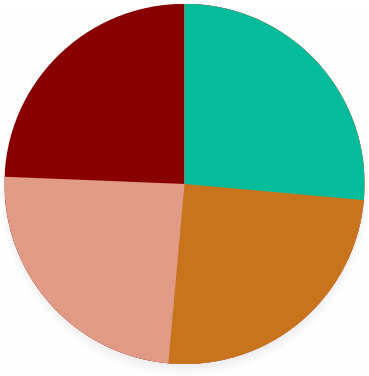
PROJECT	DURATION
Bacheloroppgave	1374:05:25

Summary Report

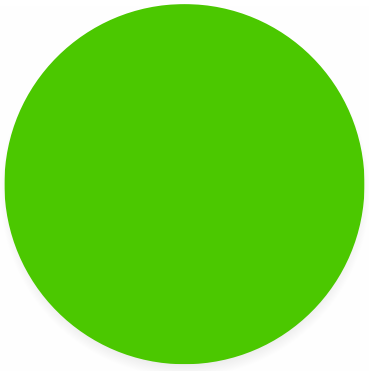
February 04, 2019 – April 12, 2019

TOTAL HOURS: 1105:14:18





USER	DURATION
KW Karoline W Rud	291:45:11
RU Rubche	276:02:30
AM Amleveg	268:49:34
JR Jrberntsen	268:37:03



PROJECT	DURATION
Bacheloroppgave	1105:14:18

R ESLint configuration

```
{
    "env": {
        "node": true,
        "es6": true
    },
    "extends": "eslint:recommended",
    "globals": {
        "Atomics": "readonly",
        "SharedArrayBuffer": "readonly"
    },
    "parserOptions": {
        "ecmaVersion": 2018
    },
    "rules": {
        "no-console": "off",
        "callback-return": 2,
        "global-require": 2,
        "handle-callback-err": 2,
        "no-buffer-constructor": 2,
        "no-mixed-requires": 2,
        "no-new-require": 2,
        "no-path-concat": 2,
        "no-process-exit": 2,
        "no-restricted-modules": 2,
        "no-sync": "off",
        "no-process-env": "off"
    }
}
```

Listing R.1: ESLint configuation file

The ESLint configuration file, see listing R.1, contains the rules defined for our setup of ESLint. It includes the most important rules for Node.js [32]. The rules are defined within the rules object of the JSON file. ”off“ or 0 represents that the rule is disabled, 1 represents that the rule is on as a warning and 2 represents that the rule is on as an error.

S SSL and NGINX configuration

First in order to obtain a SSL certificate for our domain we had to install Certbot.

```
$ sudo add-apt-repository ppa:certbot/certbot
```

This would allow us to install Certbot's NGINX package by running the following command:

```
$ sudo apt-get install python-certbot-nginx
```

NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more [46]. We have used it as a reverse proxy for our server solution. It allowed us to redirect all the inbound and outgoing traffic to our domain through to the the server running on the VM's localhost at port 8080.

```
server {
    server_name verifyid.ml www.verifyid.ml;
    location / {
        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
     }
```

Listing S.1: NGINX configuration file

In order to obtain a SSL certificate for our domain we needed to confirm the NGINX configuration by editing the "default" file within "/etc/nginx/sites-available/" directory as in listing S.1. We would use NGINX as a reverse proxy. In practice this would route all the incoming traffic to our domains `https://verifyid.ml` and `https://www.verifyid.ml` to our backend VM on localhost:8080.

We restricted the maximum size for file uploads in NGINX to 5 MB. This was accommodated by adding a restrictive line of code to the http object within the "nginx.config" file:

```
http {
    client_max_body_size 5M;
    ...
}
```

After the Nginx configuration was done we had to allow NGINX through the firewall, simply by the following command:

```
$ sudo ufw allow 'Nginx Full'
```

We then had to obtain a certificate which would grant the communication encryption we needed. By using certbot and Buypass' Go SSL [31] we obtained a certificate by running the following command:

```
$ certbot certonly --nginx -d verifyid.ml -d www.verifyid.ml --server
    'https://api.buypass.com/acme/directory'
```

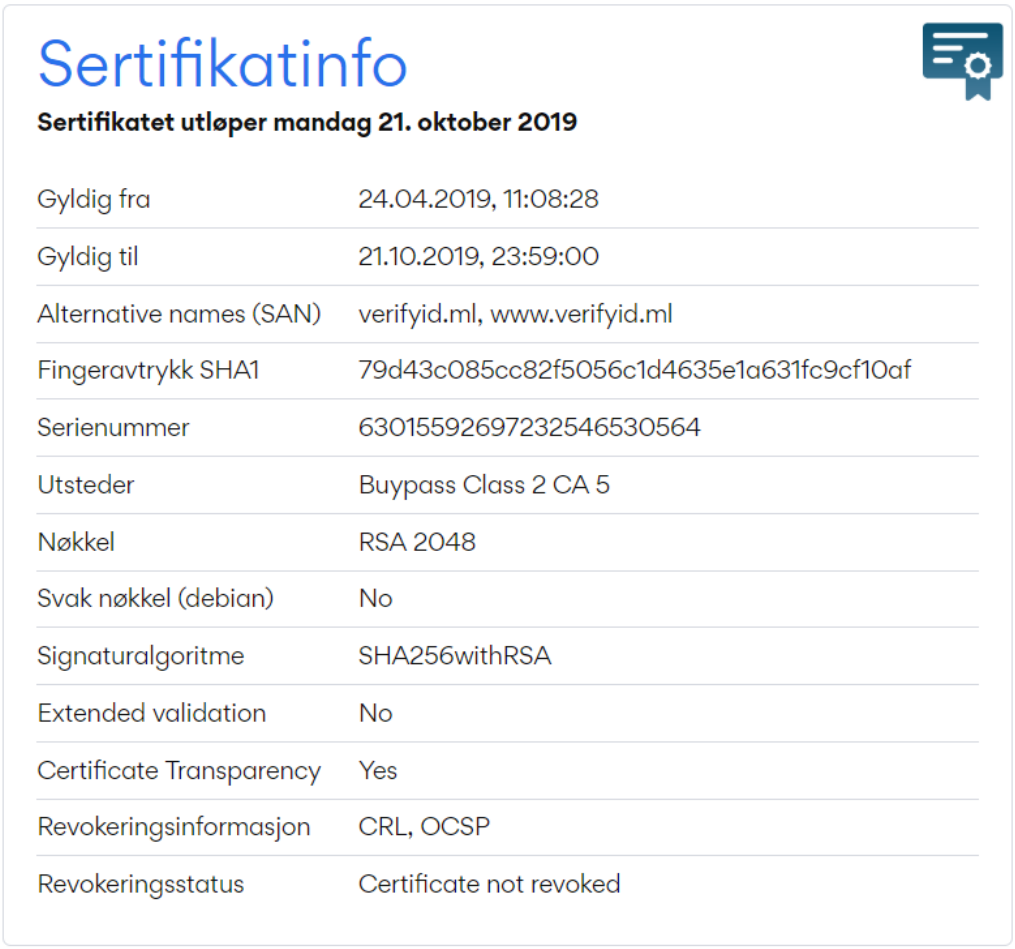
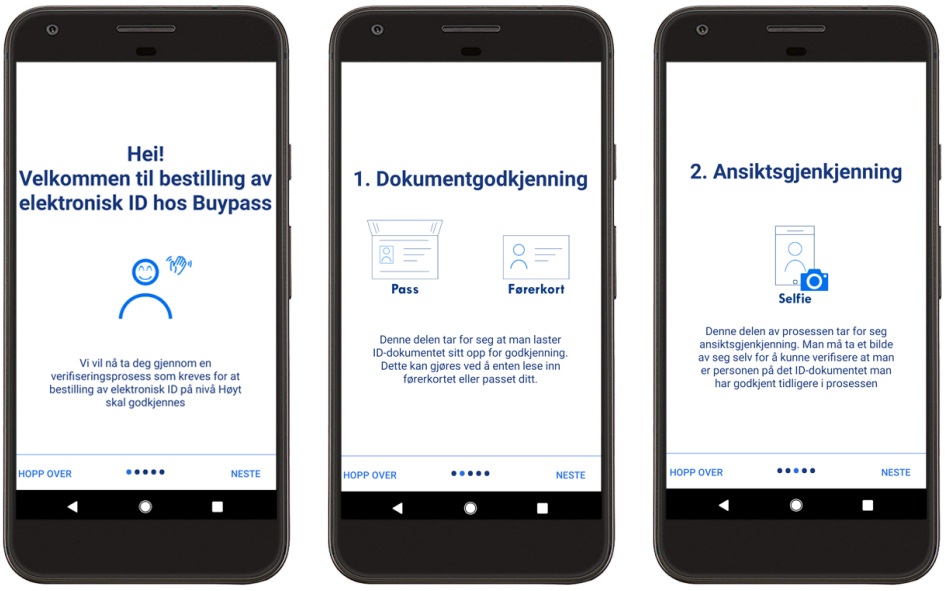


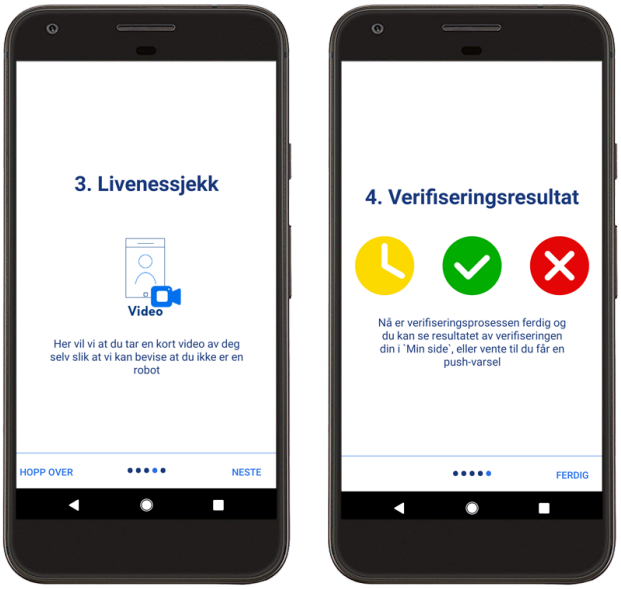
Figure 29: Go SSL certificate information

For info about the SSL-certificate of our domain see figure 29. The date of issue is the 24th of April due to the fact that we needed to change our domain name as the previous one, https://www.verifyid.gq was suspended.

T Application screens

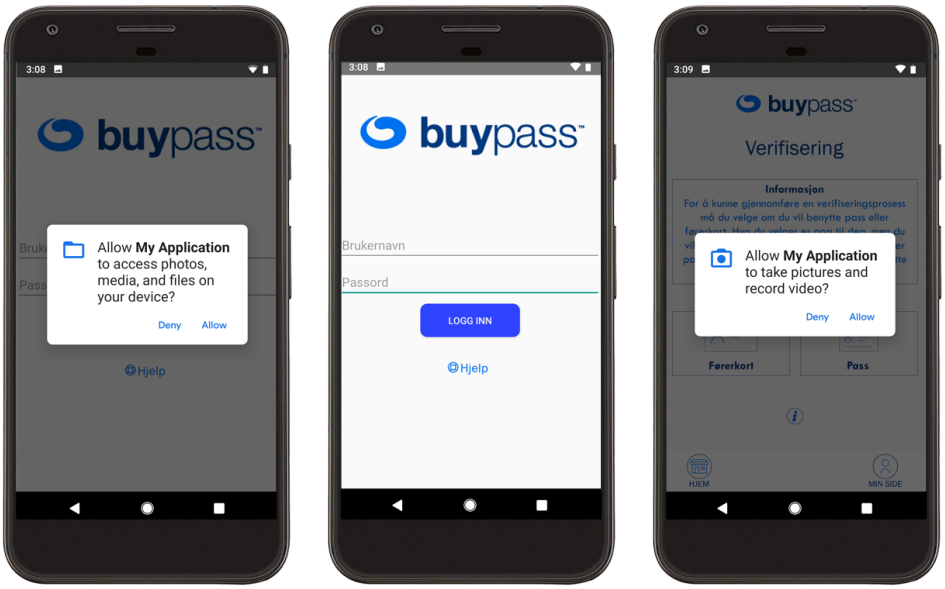


(a) (b) (c)



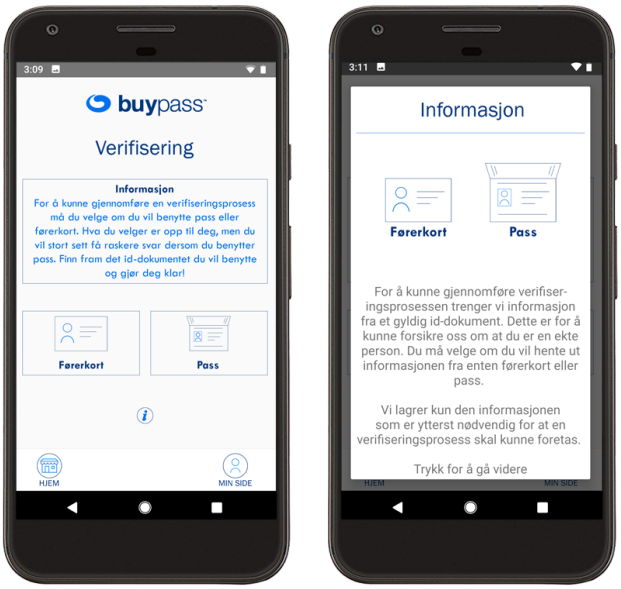
(d) (e)

Figure 30: Describes the introduction slider providing the user with general information of the verification process and the necessary steps.



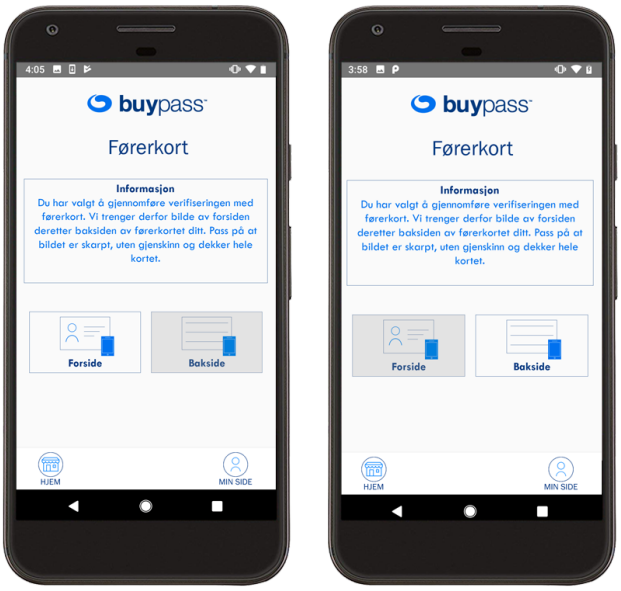
(a) (b) (c)

Figure 31: shows when launching the application it asks for permission using the camera and access to media files as this has to be allowed for using our app. After allowing this, the user may log in.



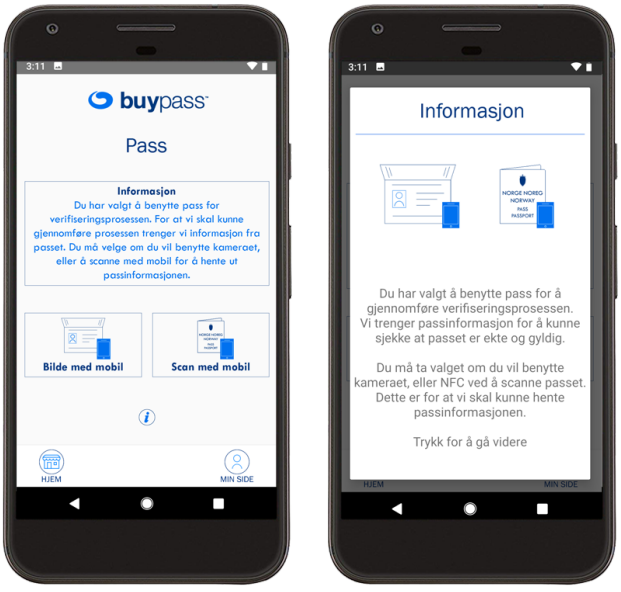
(a) (b)

Figure 32: (a) is the main screen of the application, and this is where the user starts the verification process, deciding between a two sided or one sided ID document. (b) represents the information popup the user may access if he or she finds the information provided in the main screen to be insufficient or vague.



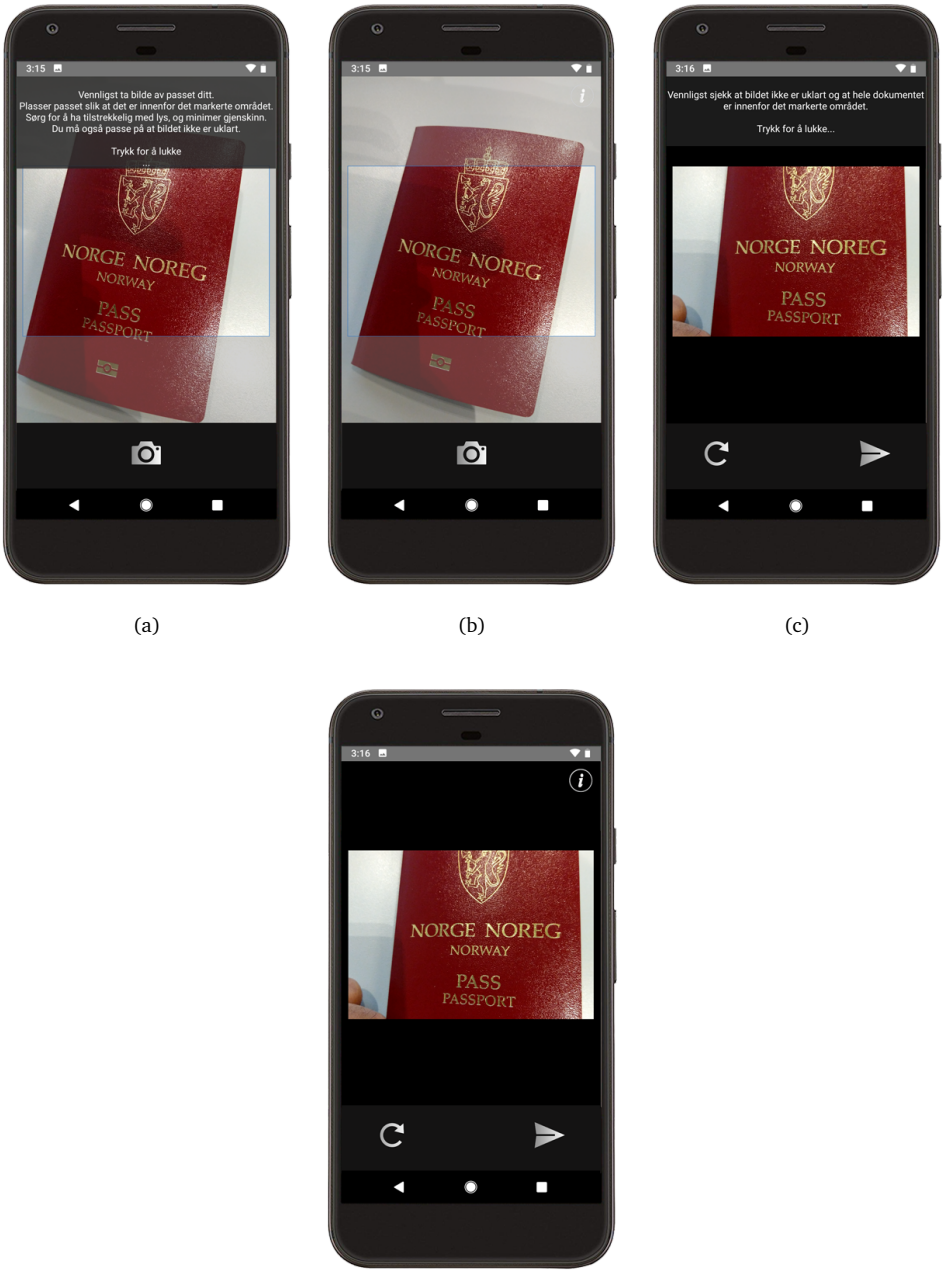
(a) (b)

Figure 33: represents the two steps in order to perform a verification with a two sided ID document. In between (a) and (b) a photo has been taken of one of the two sides of the ID document. In (a) the button for taking a photo of the back of the ID document is gray and disabled in order to show the user that the front photo needs to be taken first.



(a) (b)

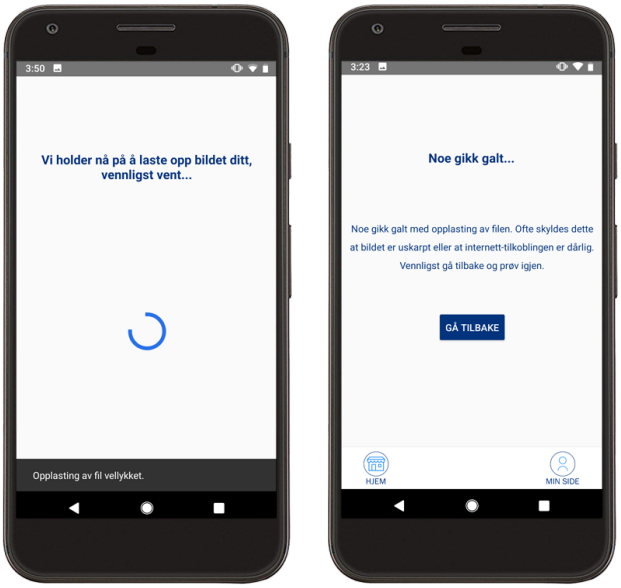
Figure 34: (a) shows the GUI if the user is to perform a verification with a passport. This choice further offers the possibility to perform a verification through NFC or photos. (NFC verification is not yet implemented, but offers the functionality to extract passport information that could be used at a later time). Figure (b) represents an informational popup regarding passport verification.



(a) (b) (c)

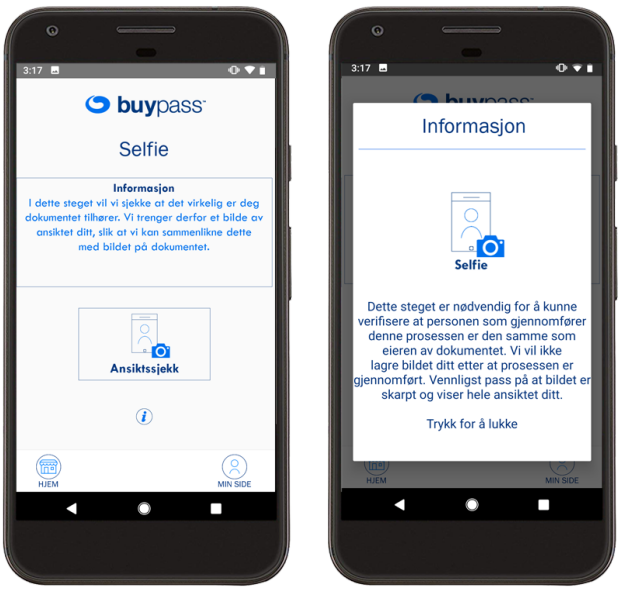
(d)

Figure 35: shows how the GUI of the camera is implemented for document photos. (a) shows how an information window provides the user with information regarding the photo. Both (a) and (b) provides a document border to where the document should be within. (c) allows the user to review the photo, and provides information as to some regular flaws with the photos.



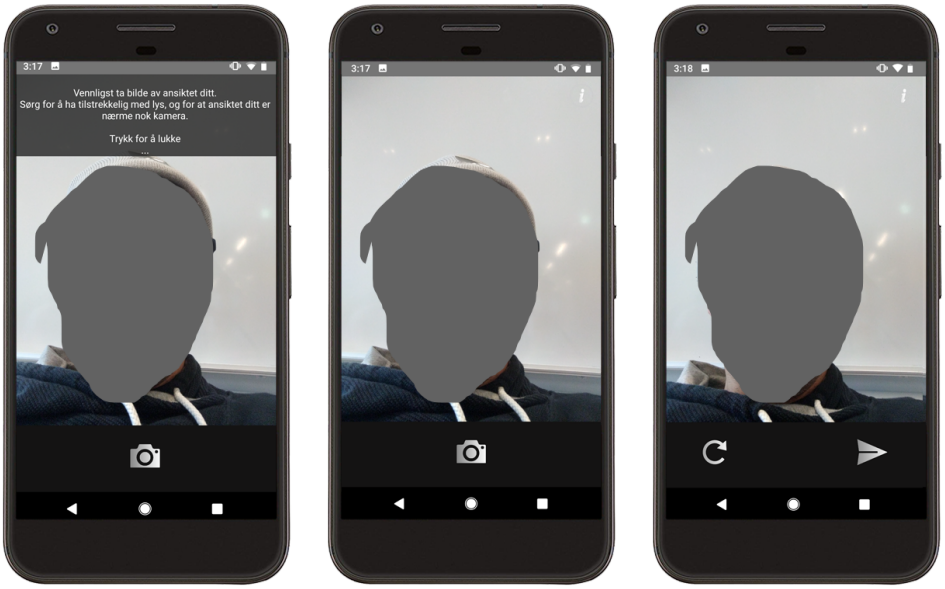
(a) (b)

Figure 36: represents an upload, either video or photo. (a) provides the user with a message saying that the upload is underway, and also a loading circle. There is also a snackbar at the bottom of the screen stating that the upload was successful and this screen would soon change. (b) represents an upload that has failed, providing the user with some of the usual reasons for these fails.



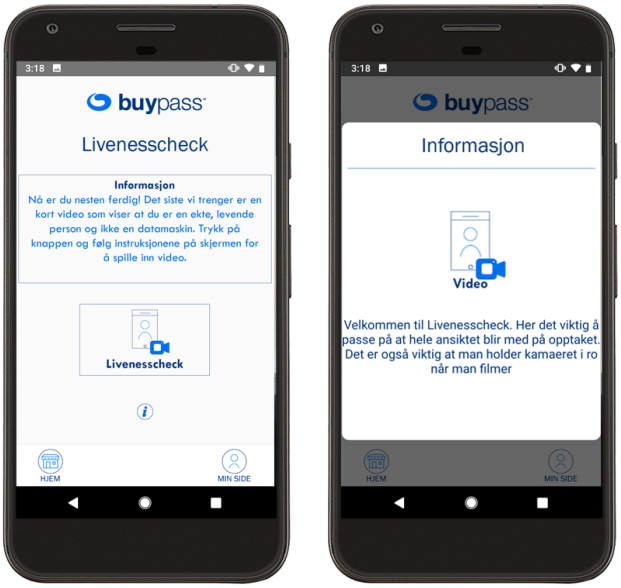
(a) (b)

Figure 37: represents the facial comparison step of the verification process, and provides information on the step.



(a) (b) (c)

Figure 38: shows the implemented camera for the face comparison step. It allows the user to take a photo and review it, and provides information to the user.



(a) (b)

Figure 39: shows the verification step of the process, and provides information on the step

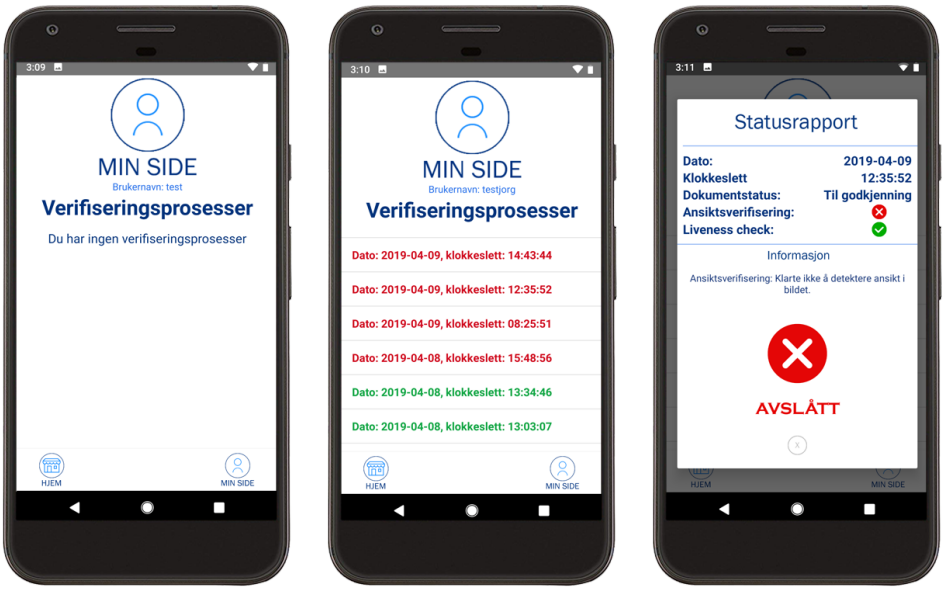
(a) (b) (c)

(d) (e)

Figure 40: represents the camera provided for the liveness check. (a) shows the starting point, providing the user with information and allows for starting a recording. (b) provides the same with the exception of the information popup. (c) represents an ongoing recording, with the remaining duration on the bottom of the screen to the left of the record button. (d) allows the user to review the video, and (e) provides the possibility to re-take or upload the video.

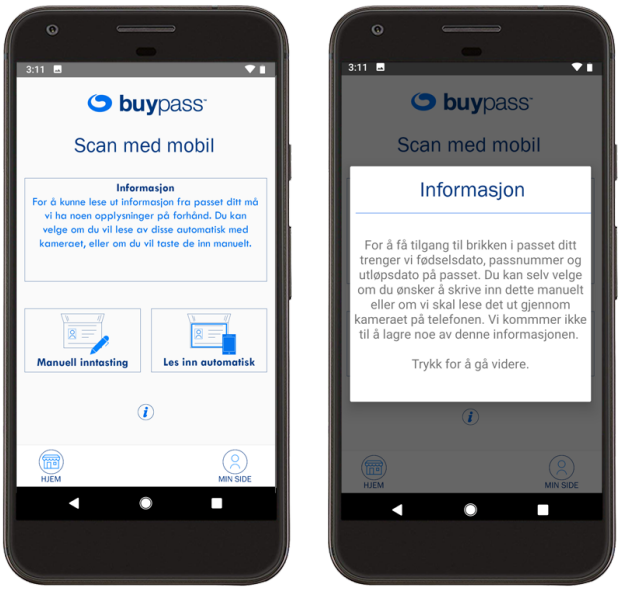
(a)

Figure 41: show the screen that greet the user as he or she has completed the verification process. It provides information as to where the verification results can be found, if it is done.



(a) (b) (c)

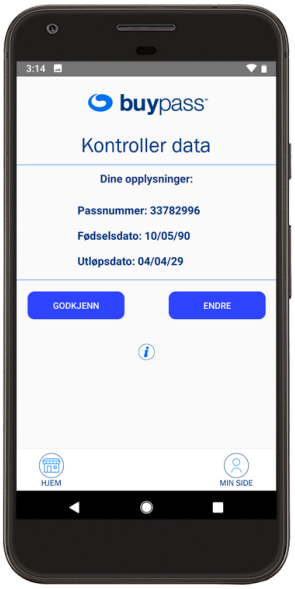
Figure 42: show the user page of the application. (a) shows a user with no verifications, and simply displays a message stating that the user has no verifications. (b) shows a user that has several verifications, and the colos of the text represents the status of the processes. (c) shows the verification report that provides the user with information as to what step in the verification process that went wrong, and why if there is a known error.



(a) (b)

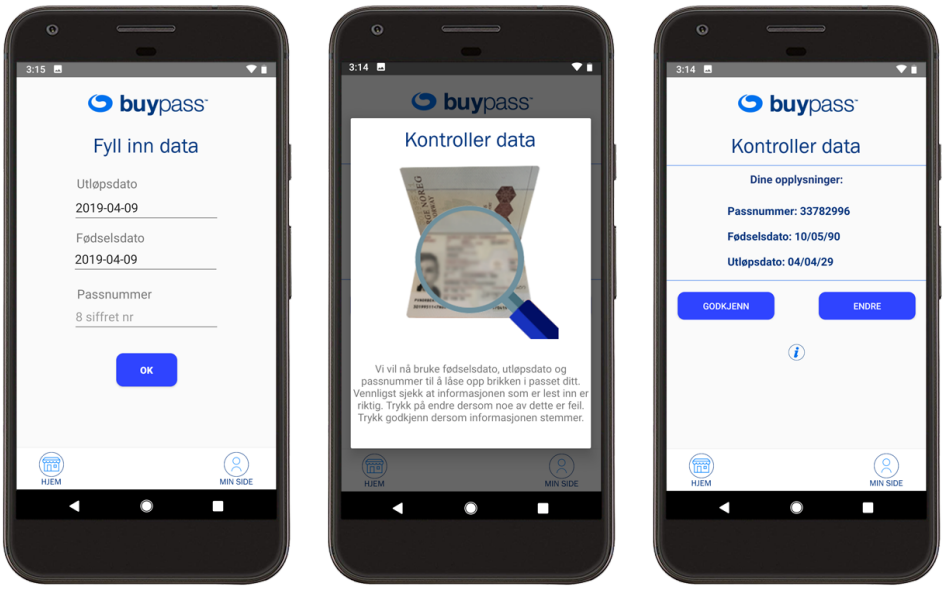
Figure 43: (a) Two alternatives for scanning the passport with NFC; one for obtaining the MRZ; the other one for manually insert of personal information. (b) Information regarding the NFC process.

(a) (b) (c)



(d)

Figure 44: (a) A message pop ups with information regarding how to scan MRZ. The data inside the markings (red rectangle) is what the MRZ scan needs (b) The process. (c) Information about checking the provided information is correct. (d) User may confirm or edit the inserted details from the scan.



(a) (b) (c)

Figure 45: (a) Manually or edit passport information. (b) Information about checking the provided information is correct. (c) User may confirm or edit the inserted details.

(a) (b)

Figure 46: (a) After confirming the provided information an animation shows how the user should scan the passport. (b) On successfully scanning the passport, it will show the result.

K.W.R, R.I.C, J.B, V.E.A

Digital verification of identity

NTNU
Kunnskap for en bedre verden

buypass