



Norwegian University of
Science and Technology

Security and usability of SSO and inter-app navigation on iOS and Android

Author(s)

Erling Moxnes Kristiansen
Truls Matias Torgersen
Toni Vucic

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

20.05.2019

Supervisor

Tomas Holt

Abstract

The goal of this study is to examine if several smaller mobile applications work as well as one big application, from a usability and security perspective. To answer this, we developed a cross-platform Single sign-on (SSO) library capable of providing a secure, invisible SSO experience. We also tested an implementation of inter-app navigation on 13 users and found that only 2/13 noticed a difference when solving two tasks in one big app, as opposed to three different apps. Using the System Usability Scale on our inter-app navigation implementation, we found the average usability of the three small apps solution to be just 5 points below the big app.

Abstract in Norwegian

Målet for denne oppgaven er å utforske om flere små apper er like brukbare og sikre som én stor app. For å finne ut av dette utviklet vi et kryss-plattform Single sign-on (SSO) bibliotek som tilbyr en sikker og usynlig SSO opplevelse. Vi testet også en implementasjon av inter-app navigasjon på 13 brukere og fant at bare 2/13 brukere merket en forskjell når de løste to oppgaver i én stor app, i motsetning til i tre små sammenlenkede apper. Ved å bruke System Usability Scale på vår implementasjon av inter-app navigasjons fant vi ut at den gjennomsnittlige brukbarheten for de tre små appene bare var 5 poeng lavere enn for den store appen.

Preface

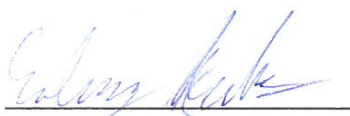
This is a research project belonging to the section of Applied Informatics (AIT) at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). Research is conducted for DIPS AS, Norway's leading eHealth provider.

DIPS had several problems that needed to be solved, but we chose to research mobile Single sign-on and inter-app navigation. This was chosen because of the many aspects of the task, suiting the interests of the whole team. The SSO solution required in solving the complex problem of balancing security and usability in a hospital context. The security requirements for an eHealth vendor dealing with sensitive patient data are high, requiring rigor when designing the system. Since the task also requests an investigation into inter-app usability, this provided opportunities for the team members to design, implement, test and evaluate mobile applications. All in all, the task provided opportunities for us to use many aspects of our education, including systems design, user testing, and security knowledge. It also provided a tough but motivating challenge to learn cross platform mobile development, API design and implementation, authorization and authentication with the OAuth2.0 protocol, more rigorous usability testing, and mobile security infrastructure.

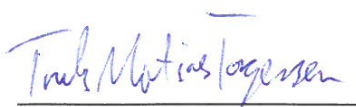
Acknowledgements

We want to thank the wonderful employees at DIPS for welcoming us into their workplace. Fun and enlightening discussions were had in lunch breaks, and we enjoyed getting competitive on the soccer field. We especially want to thank the development leader at DIPS, Runar Ovesen Hjerpbakk for mentoring us through the development process. We also want to thank Tore Mørkved, product manager for all things mobile at DIPS for helping us understand the context of our research, and for helping us connect to other professionals at DIPS, like Phuong Pedersen and Terje Sagmyr. We cannot forget Christer Brinchmann, who spent hours setting up a server we could use to test our SSO solution, and helping us when things went south. Lastly for DIPS, we want to thank the team members in Sri Lanka, for writing comprehensive documentation that helped us with OAuth2.0 and OpenID Connect. We also want to thank our supervisor Tomas Holt for helping us with the research aspect of our thesis, which were completely new to us. An additional thanks should be given to Dr. Kirsti Elisabeth Berntsen for helping us understand Design Science Research. Finally we want to thank Dr. Thomas Tullis from Bentley University for helping us interpret his research.

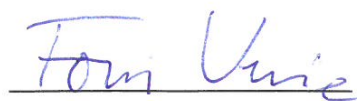
Trondheim, May 20th, 2019



Erling Moxnes Kristiansen



Truls Matias Torgersen



Toni Vucic

Summary

Norway's biggest eHealth vendor DIPS AS is bringing electronic patient journals to mobile. Electronic hospital solutions are infamously complex, and in order to avoid monolithic apps, DIPS wanted us to explore the feasibility of several smaller securely linked mobile applications.

The goal of this project was to assess if there is a change in usability or security when splitting a large app into smaller apps which combined provide the same overall functionality. To make the transition from one app to several as easy as possible for end-users would require mainly two things. An efficient and secure Single sign-on solution, and an intuitive and secure implementation of inter-app navigation.

Based on the newest Android and iOS API documentation, we have made a cross platform Single sign-on library for mobile, including an OAuth2.0 client that works with OpenID Connect. We also created two applications to verify it's functionality. The solution was made using Xamarin.Forms to enable the use of a single code-base for both iOS and Android. With our library an end-user only needs to sign-in once to to be signed into a group of apps. This is done by securely sharing an access token within the group of apps. The access token gives the holder permission to access protected resources, for example on an external server.

We wanted to verify that navigation between several securely linked apps was as intuitive as navigation within a big app. To do this we created 3 small apps and one big app with the exact same functionality, and asked 13 users to compare them using oral interview questions and the System Usability Scale.

Our conclusion is that it is possible to create an unobtrusive and efficient Single sign-on system that works on both Android and iOS. Our SUS scores and interviews indicate that we were able to create inter-app navigation that is almost as good as in-app navigation. However, more user tests are required for a definitive conclusion.

Report structure

Chapter 1 explains our assignment and the research questions derived from that. Chapter 2 contains an overview of important subjects and some theoretical background relevant to our research questions. Chapter 3 describes our user testing methodology and process, and our choices for engineering methodology. Chapter 4 will contain a detailed description of our test, process and engineering results. Chapter 5 will be a discussion of those results, lastly followed by Chapter 6 which contains the conclusions that can be drawn from our results and potential further work.

Appendices

Appendix *Available Technologies G*, is meant to function as an encyclopedia for the technologies and concepts that were crucial for the outcome of our results. It is heavily referenced throughout the report as a store of information.

We also relocated parts of the methodologies, results and discussions sections to the appendix for the sake of brevity. The information there is still relevant to our project and had an effect on the results, but was ultimately moved there to shorten an already long report. Each section in the appendix expands on the information in their respective sections in the main report. Every one of the sections in the appendix is referenced for easy readability.

Appendix B contains additional information about the user testing process.

Appendix C contains important reasoning's for our choice of process and development methods.

Appendix D contains further details about our use of the chosen engineering process methods.

Appendix E shows more results from our development.

Appendix F contains further discussion on the effects the chosen processes had on our results.

Contents

1	Introduction and relevance	11
1.1	Background	11
1.2	Assignment	11
1.2.1	Research questions and hypotheses	11
2	Theory/Literature Study	17
2.1	Authorization	17
2.2	Authentication	17
2.3	Principle of least privilege	17
2.4	Platform specifics	18
2.4.1	iOS	18
2.4.2	Android	18
2.5	App Sandboxing	19
2.6	Rooting/Jailbreaking	19
2.6.1	Android - Rooting	19
2.6.2	iOS - Jailbreak	20
3	Chosen Methodology	21
3.1	User testing methodology	22
3.1.1	Questions to be answered	22
3.1.2	User test design	23
3.1.3	Test execution errors and anomalies in methodology	32
3.1.4	User test reports	33
3.1.5	Data processing and storage	33
3.2	Engineering methodology	33
3.2.1	Method for collecting information	34
3.2.2	Design Science Research	34
3.2.3	Development methodologies	35
4	Results	36
4.1	User testing results	36
4.1.1	Presentation of the empirical data	36
4.1.2	User test reports	40
4.2	Engineering process results	40

4.2.1	Scrum	40
4.2.2	Version control	43
4.2.3	Pair programming	43
4.2.4	Mob programming	43
4.2.5	Mini-seminar	44
4.2.6	Cooperation with project lead and supervisor	44
4.2.7	Design process	44
4.3	Engineering artifacts results	49
4.3.1	SSO library	49
4.3.2	Apps for testing inter-app navigation UI	56
4.3.3	Apps for testing our SSO library	58
4.4	Administrative results	58
4.4.1	Timesheet	58
4.4.2	Gantt-diagram comparison	59
5	Discussion	61
5.1	Assumptions	61
5.2	From a user perspective, do several securely linked mobile applications function as well as one big mobile application, from a users' perspective?	62
5.2.1	There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.	62
5.2.2	Navigation between several securely linked apps is just as intuitive as navigation within one secure app	63
5.2.3	Given apps requiring user authentication upon start-up, the user does not need to re-authenticate when securely linked apps open each other.	65
5.3	Are several securely linked mobile applications as secure as one big secure mobile application?	67
5.3.1	When one app attempts to open another app by the same developer, a third app cannot hijack the inter-app navigation and get opened in stead of the intended app.	68
5.3.2	Data shared between securely linked applications is not accessible to other apps.	69
5.4	What is a secure well made cross-platform implementation of SSO for iOS and Android?	70
5.4.1	What are the requirements for the SSO-library to be a well made product?	70
5.4.2	What are the requirements for a secure SSO library?	72
5.5	Process evaluation	75
5.5.1	Progress plan evaluation	75
5.5.2	Time sheet evaluation	75
5.5.3	Scrum	75
5.5.4	Room configuration	75
5.5.5	Development and research methods	75

5.5.6	User testing method and process	76
5.5.7	Developer user testing	77
5.5.8	Evaluation of our use of DSR	77
5.5.9	Professional ethics considerations and team reflections . .	78
5.5.10	System perspective	78
5.5.11	The length of this report	79
6	Conclusion and further work	80
6.1	Engineering artifacts conclusion	80
6.2	Research questions conclusion	80
6.2.1	From a user perspective, do several securely linked mobile applications function as well as one big mobile application?	80
6.2.2	Are several securely linked mobile applications as secure as one big secure mobile application?	81
6.2.3	What is a secure well made cross-platform implementation of SSO for iOS and Android?	81
6.3	Further work	82
6.3.1	SSO-library	82
6.3.2	Inter-app navigation	82
	Appendices	89
	A Thesis assignment	90
	B User testing methodology	92
B.1	User tests changes and rewards	92
B.1.1	System Usability Scale changes	92
B.1.2	Changes caused by pilot tests	93
B.1.3	Details about anomalies that occurred during our user testing	93
B.1.4	Data processing and storage	94
B.1.5	Rewards	95
	C Engineering methodology	96
C.1	Process methodology	96
C.1.1	Scrum and agile development	96
C.1.2	Room configuration	98
C.1.3	Mini-seminar	99
	D Engineering process results details	100
D.1	Retrospectives	100
D.1.1	Decisions and changes from retrospectives	100
D.2	Pair-programming	101
D.3	Mob-programming	103
D.4	Cooperation with project lead and supervisor	103

E	Engineering artifacts results details	104
E.1	How to receive a fragment.	104
E.2	SSO-library dependencies	105
E.3	Test apps explanations and figures	105
F	Process evaluation details	108
F.1	Scrum	108
F.2	Developer user testing	108
F.3	Research and development methods	108
F.4	Room-configuration	109
G	Available technologies	111
G.1	Chosen by DIPS	111
G.1.1	Xamarin Forms	111
G.1.2	Authorization	112
G.1.3	Authentication	114
G.1.4	Biometric authentication	114
G.2	Our options	115
G.2.1	OpenID Connect flows	115
G.2.2	Local storage	118
G.2.3	Secure storage/encryption	119
G.2.4	Token sharing between apps	123
G.2.5	Inter-app Navigation	126
H	Vision document	130
I	Requirement documentation	143
J	System documentation	156
K	Diagrams	165
K.1	Class diagrams	165
K.2	Flow diagram	165
K.3	Different design stages	165
K.3.1	OAuth client designs	165
K.4	Test apps	165
L	User testing	173
L.1	Test plans	173
L.1.1	Test plan for non-doctors	173
L.1.2	Test plan for doctors	179
L.2	Consent form and non-disclosure agreement	185
L.3	Test reports	187
L.4	Empirical data	234

Chapter 1

Introduction and relevance

1.1 Background

DIPS currently develops and maintains two desktop applications and develops one mobile application for hospital systems. In the future they want to develop more mobile applications to cover some of the functionality given by their desktop applications, with the goal to add flexibility and mobility to the end-users workday.

Adding more functionality to an application leads to higher degree of complexity in its architecture and design, which results in a decrease in the ease of further development and maintenance.

1.2 Assignment

DIPS AS provided us with a thesis assignment. From this we made user stories which functioned as a specification of requirements (see subsection 1.2.1.3). Based on the user stories we made research questions and hypotheses. These are described below.

For the full thesis assignment, see A.

1.2.1 Research questions and hypotheses

Research questions are in bold, and their hypotheses are found in their subsequent indented list.

1.2.1.1 From a user perspective, do several securely linked mobile applications function as well as one big mobile application?

We define the applications to "function as well" if these hypotheses are found to be true.

- There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.
- Navigation between several securely linked apps is just as intuitive as navigation within one secure app.
- Given apps requiring user authentication upon startup, the user does not need to re-authenticate when securely linked apps open each other.

1.2.1.2 Are several securely linked mobile applications as secure as one big secure mobile application?

- Data shared between securely linked applications is not accessible to other apps.
- When one app attempts to open another app by the same developer, a third app cannot hijack the inter-app navigation and get opened instead of the intended app.

1.2.1.3 What is a secure well made cross-platform implementation of SSO for iOS and Android?

To answer this question we are we are going to build an SSO library meeting the requirements specified in the two next subchapters, and documenting our choices.

1.2.1.3.1 What are the requirements for a secure SSO library?

For all of these we assume the device is not a compromised device. We define a SSO library to be secure when:

- It is possible to remotely revoke user access to the application through the library.
- The sign-in session with the authentication and authorization server cannot be hijacked by a attacker.
- Tokens stored by the library cannot be extracted or modified by other apps on the device
- Tokens on the device are inaccessible/unusable when the device is locked
- The library has few, if any, external dependencies

1.2.1.3.2 What are the requirements for the SSO library to be a well made product?

Based on the requirement specification(see: I) and vision document(see: H), we consider an SSO library to be a well made product when:

- The library supports authentication and authorization with an external OAuth2.0/OpenID Connect server. This should be done through one library method, and the developer should be notified on successful login.
- The library can keep the user logged in across all the apps on the same device, created by the developer, without re-authenticating for each app.
- The library is able to prompt the user to re-authenticate if their session has expired, letting them continue where they left off before their authorization expired.
- The library should only prompt the user to log in if the user isn't already logged in.
- The library uses the same code base for both Android and iOS
- The library is easy to expand upon and re-use components from.

1.2.1.4 What is a secure way to implement inter-app navigation across Android and iOS?

To answer this question we are we are going to build one or more mock applications meeting the requirements specified in the next subchapter, and document our choices.

1.2.1.5 Further explanation of hypotheses

We recognize the need to further elaborate on parts of the hypotheses.

Several securely linked apps:

Several apps that can securely exchange data and navigate into each other without the risk of data leakage/theft or inter-app-navigation hijacking by a third app on an non-rooted/non-jailbroken Android/iOS device.

One secure app:

A normal app without elevated privileges on an non-rooted/non-jailbroken Android/iOS device. The app's private data is inaccessible to other apps on the device. This requirement is met for all apps using internal storage due to automatic iOS/Android app sandboxing.

Acronyms

DSR Design Science Research. 9, 34, 35, 77

OAuth2.0 OAuth2.0. 50–52

OIDC OpenID Connect. 49–52, 114, 117

PKCE Proof Key for Code Exchange. 73

SAML Security Assertion Markup Language. 112

SE Secure Element. 122

SSO Single sign-on. 2, 3, 5, 8, 12, 13, 21, 25, 32, 33, 43, 46, 48, 49, 52, 56, 58, 66, 70–73, 75, 77–82, 104, 105, 108, 166

SUS System Usability Scale. 22, 23, 25, 30–32, 36, 38, 39, 63, 64, 80, 81, 92–94

TEE Trusted Execution Environment. 66, 121, 122

Glossary

access token A string that can be used for verifying that the holder has access to a certain resource. 15, 16, 46, 53, 106, 107, 113, 116, 117

authorization code A string that can be exchanged for an access token and or refresh token. 113, 116, 117

back button Used for back navigation. 27, 28, 57

back navigation This has a different meaning on iOS and Android. On iOS, this means pressing the arrow button in the top left corner (not to be confused with the back to app button). This action navigates the user to the view that is one level up inside the app. On Android, back navigation is pressing the button on the bottom of the screen or swiping up on the edge of the screen. This action pops the navigation stack and shows that view to the user. 15

back to app button Only for iOS. The small black triangular button that appears in the very top left corner when the user has opened an app when being in another app. The button states the name of the previous app. Pressing it navigates the user back to the previous app in the state they left. Pressing the button is equivalent to opening multitasking and selecting the same previous app. The button disappears when the app navigated to goes to sleep. 15, 28, 37, 64, 65, 67, 95, 127

client credentials Usually a combination of a Client ID and a client secret. 16, 113

client secret A string that only the client should have access to. 15, 16, 51

comparative test Compare two or more systems to choose the best aspects from two or more designs. 23, 76

compromised device An Android or iOS device that has been rooted/jailbroken. Security features previously guaranteed by the operating system can no longer be relied upon.. 12, 72

confidential client An OAuth2.0 client capable of maintaining client credentials or in some other way adequately authenticate with the authorization server. 113

ID Token The ID Token is a security token that contains Claims about the Authentication of an End-User by an Authorization Server when using a Client, and potentially other requested Claims. The ID Token is represented as a JSON Web Token (JWT)]. 114, 116, 117

key alias A name given to a generated key. 20

personal information Information or considerations that can be linked to an individual. This can for example be certain audio recordings, name, address, phone number, car registration or birth date.. 94, 95

principal of least privilege See section 2.3. 69

proof of concept A small and incomplete version of a system used to prove and/or verify potential. 49, 78

public client A OAuth2.0 client that is unable to maintain a client secret. 113, 117

refresh token A token that can be used for requesting a new access token. 15, 46, 53, 113

response type The parameter in the authentication request that determines the flow to be used. 50, 116

unique ID From apple official document: "A 256-bit AES key that's burned into each processor at manufacture. It can't be read by firmware or software, and is used only by the processor's hardware AES engine. To obtain the actual key, an attacker would have to mount a highly sophisticated and expensive physical attack against the processor's silicon. The UID isn't related to any other identifier on the device including, but not limited to, the UDID.". 20

Chapter 2

Theory/Literature Study

This chapter introduces the reader to concepts relevant to answering the research questions in section 1.2.1, as well as the theoretical background our research builds upon. An understanding of the platform specific technologies might also be necessary to get the most out of the report. They are found in appendix G.

2.1 Authorization

The system needs access to resources owned by an end-user. The end-user has to grant the system access to their resources. Authorization is represented by a valid access token. The access token works similarly to a key to a door, in that it says nothing about the key holder.

2.2 Authentication

The end-users have to authenticate themselves with credentials only known to them. This means verifying to the system that they are who they claim to be. Usually using a combination of username and password. However, biometric authentication is also very common, especially on mobile devices. Authentication prevents unauthorized access to resources and provides the system a means of identifying the end-user.

2.3 Principle of least privilege

The principle of least privilege, also known as principle of least authority, is an important concept in computer security. It entails only giving subjects the privileges needed for the subjects to complete their tasks, and no more.

This can help to reduce the attack surface by removing unnecessary privileges which could have led to unforeseen exploits.

This principle can apply to access rights for users, applications, systems and processes on a computer.

2.4 Platform specifics

This section details information about some aspects of the two platforms that are relevant to our research and development.

2.4.1 iOS

On iOS every app has its own sandbox, see 2.5, which make it completely isolated from the other apps. This is the security foundation all apps on the device are built on.

2.4.2 Android

Android is an open source OS built on the Linux kernel[1]. On Android every app is treated as a user and are assigned their own unique user ID[2]. Every app is put in its own "sandbox", see 2.5. Android apps are limited in what they can do through a permission system. File-system permissions ensure that one user (app) cannot alter or read another user's (app's) files. This means that by default, every apps' internal storage is only accessible from the owning app.

2.4.2.1 Activity

An app can contain one or more activities. The activity class/concept facilitates the ability to only open parts of an app. Generally, each screen-view the user sees is provided by one activity. An app can launch specific activities in other apps. All this combined gives a the developer an opportunity to re-use activities which might already provide the needed functionality.

2.4.2.2 Task

A task is simply put a collection of activities. The activities in a task are put in a LIFO stack called the back stack.

The default behaviour of a back stack is this: An activity is added to the same task as the activity that opened it. The back button(hence the back stack name) finishes the active activity and pops it from the stack. The state of activities below the active one in the stack are saved. A whole task can be sent to the background with the state of all the activities saved. That is, unless the activities are destroyed by the OS when it does resource management.

The activities in a task doesn't have to be from the same app and every activity can be instantiated multiple times in different tasks or the same task.

2.5 App Sandboxing

Sandboxing is the concept of isolating an apps' process and all its resources to a location in the memory that is not accessible to any other apps or processes, except certain processes with root permissions. Which means no app can directly access another apps resources.

This is done to prevent malicious or buggy app code from compromising other apps on the system.[1]

Every app on iOS and Android has its own sandbox.

2.6 Rooting/Jailbreaking

An attacker which roots or jailbreaks a device can gain root permissions. This enables reading and changing all files on the device.

The scenario of an attacker gaining access to the hardware and rooting or jailbreaking it results in a inherently unsafe environment for the system to run in. Root access can also be gained through software exploits.

2.6.1 Android - Rooting

On Android, the vast amount of different implementations and the inability to guarantee the security of information on the device when its rooted makes it impossible to say that the device is secure when rooted[3]. See 2.6.1.1 for an example of an attack.

"A more robust approach to protecting data from root users is through the use of hardware solutions." [3]

Some of these solutions are presented in subsection G.2.3.2.

2.6.1.1 Security evaluation of Android Keystore

In February 2018 the University of Piraeus, Department of Digital Systems published a master's thesis titled Security evaluation of Android Keystore, by Georgios Kasagiannis[4].

The study conducted two experiments concluding that the KeyStore was not safe from attack for Android 5.0-7.0. The experiment results are presented here:

2.6.1.1.1 Attacking Qualcomm KeyStore on rooted device (Android 5.0 to 7.0)

The study found that it is possible change which android process was mapped to which keystore. A mapping of to key alias looks something like this:

```
<UID of the app> USRPKEY <key alias given inside app>  
<UID of the app> USRCERT <key alias given inside app>
```

”By copying the “keyblobs” from one app’s directory into another app’s directory and changing the other app’s UID, the other app can use the keys provided they are still on the same device. Copying the files to another device will not work because the private key can only be decrypted with the hardware-backed key that exists in the TEE”[4].

For Android 5:

- An attacker that has root permissions can easily use the keys of other apps on the same device by renaming the keystore files.
- Key pairs cannot be used outside of the device because the private data are encrypted with a device-specific key living inside the TEE that cannot be exported.

For Android 6 and 7:

- It is possible to request user input/consent before the key is to be used.
 - If the attacker knows the user’s password, it can access the key.

2.6.1.1.2 Attacking software based keymaster (Android 5.0 to 7.0)

Android 5, 6 and 7: Following a similar attack method on can extract software based RSA private keys as well. Which makes plain-text recovery possible with root access[4]. With root permissions its possible to simply change the ownership of the key files to another app.

2.6.2 iOS - Jailbreak

If the device is jailbroken by an adversary, the user can’t be certain that the device is 100% secure. This is because the adversary has root access to modify all files on the device. Apple states that ”iOS has additional encryption and data protection features to safeguard user data, even in cases where other parts of the security infrastructure have been compromised (for example, on a device with unauthorized modifications)” [5], where ”unauthorized modifications” could mean a jailbreak. This means that the iPhone is still somewhat secure, unless secure hardware is initially disabled by the removal of security measures like a pin code. The pin code together with the device’s in-built hardware key ensure security.

Chapter 3

Chosen Methodology

As mentioned in our vision document, our employer DIPS ASA faced the problem of "not being able to seamlessly navigate between apps with the same login credentials" [6] and "developing, maintaining and expanding big mobile applications simultaneously across several development teams." [6]

The task given to us was to "use the framework Xamarin Forms to make several mock-applications to support a clinical documentation-task. When the user is authenticated in one app, she should be authenticated in the others as well. The apps should support deep-linking between each other so that the navigation is experienced as efficient. The user should get authorization per scenario, and therefore also be remembered across the apps." [6]

While only creating the mock applications would fulfill the requirements of the task, we decided to take it one step further. We decided to create a stand-alone Single sign-on (SSO) library, and use it in our mock applications. The reason for building the SSO library was our employer's problem of "not being able to seamlessly navigate between apps with the same login credentials". [6] Our SSO library would function as the foundation for a long-term solution to that problem.

It is easier to re-use code with low coupling and high cohesion. The idea was that putting everything in it's own library separate from the logic of the mock apps forced us to write code adhering to those principles. For more on that, see the chapter on engineering methodology, section 3.2

3.1 User testing methodology

3.1.1 Questions to be answered

Research question:

Do several securely linked mobile applications function as well as one big secure mobile application, from a users' perspective?

To answer that question we formed the following hypotheses:

- There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.
- Navigation between several securely linked apps is just as intuitive as navigation within one secure app.

To evaluate those hypotheses we decided to ask the users directly, as well as make them compare the two solutions with the System Usability Scale (SUS), without letting them know what exactly we were measuring.

It is important to note that even though the questions are phrased in a very general way. That is, they imply we are seeking an answer to the question for every imaginable securely linked app. We do not expect our tests to answer these questions in a general way. We expect to only be able to answer the question to some degree for our own implementations of three "securely linked apps" (1.2.1.5) and one "secure app" (1.2.1.5).

3.1.1.1 What is the System Usability Scale?

Recommended by our user testing book[7]. System Usability Scale (SUS) is a type of likert scale containing 10 subjective questions. Being referenced in over 1300 articles and publications (7299 citations total) it is a industry standard for usability.

3.1.1.1.1 Why did we choose SUS?

Noted benefits of SUS are:

- It is a very easy scale to administer to participants
- It can be used on small sample sizes with reliable results[8]
- It is valid – it can effectively differentiate between usable and unusable systems[9]

The consideration of sample size is particularly important. Our goal was to test the system on 5 doctors and 10 non-doctors. SUS has been shown to be an accurate measure for small sample sizes. A highly cited study within the field (even cited by the creator of SUS, in his own SUS retrospective)[10] found that

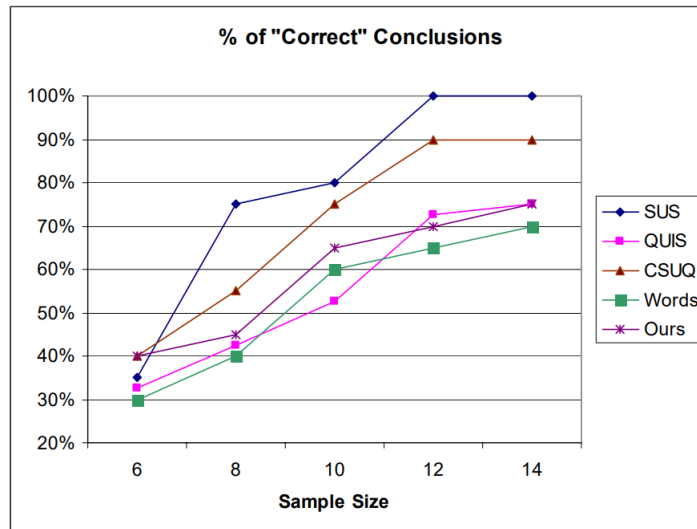


Figure 3.1: A comparison between other questionnaires on how well they predict the same answers, with fewer test subjects.

12 randomly chosen persons from a sample of 20 were able to predict the same result as the full sample size, 100% of the time. It also found SUS to be the most accurate measuring method for small sample sizes out of the 5 methods the study tested[8], see Figure 3.1.

3.1.1.1.2 SUS scoring

The score of SUS goes from 0 to 100, but must not be confused with a percentage. That is, the distribution of scores is not linear, so one system scoring 20 cannot be said to be 100% better than a system scoring 10. See Figure 3.2 for the distribution. A meta-study had found the average SUS score to be 68.[11] Meanwhile a score of 80 indicates that the system is better than right under 90% of other systems. See Figure 3.3 for ways to interpret the score.

3.1.2 User test design

Our user test is a comparative test. The goal of the test is primarily to discover if the test subjects experience a difference navigating between one app, and navigating between three apps.

The tests have a **test leader**, **observer**, and possibly also a cameraman. The test leader would ask the interview questions and handle the questionnaires. The observer wrote notes about inconsistencies in executing the test method and worked as a technical assistant preparing the apps. The cameraman filmed the iOS usability tests. The roles were rotated so that the test leader never

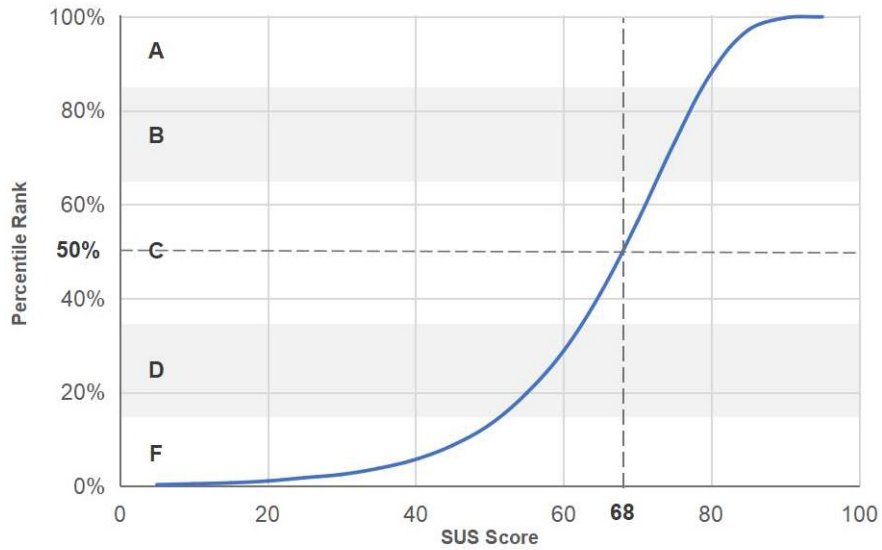


Figure 3.2: SUS scores and their percentiles based on a large dataset of SUS-scores. Source:[12], [13] The curve is partly based on data from [14].

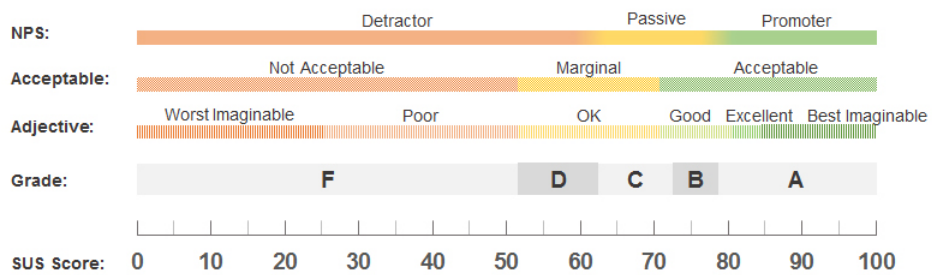


Figure 3.3: Different ways to interpret SUS score. NPS stands for Net Promoter Score. Source: <https://measuringu.com/interpret-sus-score/> (taken 19.05.2019)

interview someone they knew. The observer was often someone the test subject knew. We believe that the test subjects having someone they know in the room with them helping facilitate the test would make the test subjects more comfortable than if everyone were strangers. Then again, having people we know as test subjects is problematic in itself, because they might be inclined to give more positive answers to not damage relations.

The user test consists of five different elements.

1. Initial interview
2. Practical usability tests
3. System Usability Scale questionnaires
4. Demographic questionnaire
5. Comparative interview

The content of these are detailed in the doctor L.1.2 and non-doctor L.1.1 test-plans. The difference being that the doctor test-plan includes a short interview at the start, mentioned in 3.1.2.1.

The reasons for each part, and what they contain is detailed in the following sub-chapters.

3.1.2.1 Initial interview [Doctors only]

Throughout this project we have had the context of our solutions in mind. Our employer DIPS ASA delivers software solutions to hospitals covering most of Norway. To get a better understanding of the context our Single sign-on library and other research findings, we decided to ask the doctors we interviewed two questions.

- Can a phone be a useful work tool for you?
- In what contexts can the use of a phone be applicable?

While they are short and limited, we hoped that these questions would help us understand the context our solution would be used in better.

3.1.2.2 Practical usability tests

As detailed in subsection 4.3.2 of our results chapter, we created 4 applications to test the usability of inter-app navigation.

In this chapter we will describe the choices behind the designs, and how we conducted the tests.

3.1.2.2.1 App UI design

We did not consider the content of the applications to be particularly important. The reason for this was that we were testing inter-app navigation, not the functionality of a production-bound app. Some thought was put into the design to contextualize the experiment to a hospital setting. The app had to mimic hypothetical tasks a doctor would do on her hypothetical mobile device. DIPS already had an interactive mock-up for such a group of apps. We got permission to use the design elements in our own app.

In the interest of saving time we decided to take screenshots of the mock design and insert it as images and image-buttons into our own app in stead of re-creating it with Xamarin.Forms elements. The final designs are in 4.15.

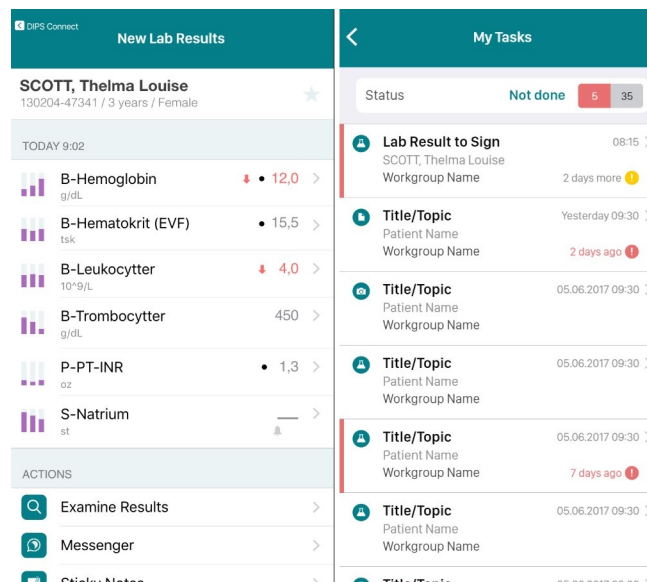


Figure 3.4: UI design from a DIPS app.

3.1.2.2.2 App content and navigation

To save development time and keep the experiment as simple and reproducible as possible, we limited our apps to one landing page containing two tasks, and two pages, one for each task.

To find relevant tasks that realistically could be separated into different apps we consulted project manager for all things mobile at DIPS, Tore Mørkved. Through him we had an informal meeting with quality consultant Terje Sagmyr, product owner of tasks and care plan Phuong Pedersen and Tore Mørkved. They suggested these tasks for us:

1. Choose a task from a list of tasks.
2. Sign off on some lab results.
3. Sign a medical document.

Content of My Tasks

A simple design was chosen to limit what the user could do during the test. Only displaying two tasks. See 4.15.

Content of Lab Results

The design would contain lab results taken from the DIPS mock prototype (Figure 3.4), and a custom made button for testers to sign the document. Once the document was signed, the user would be made aware that they had signed the document.

Content of Document Viewer

A medical document would be shown. The page would include a signing button just like Lab Results.

Keeping variables to a minimum

We wanted to test if the way we have designed our inter-app-navigation, including our technology choices and the OS limitations, leads to several small apps and one big app functioning equally well. Since the tasks are the same, only the changes in navigation implementation/design should impact our System Usability Scale scores and answers to our oral interview questions. To ensure this, one app and several small apps would be made as similar as possible. On Android, they should be 100% the same, except for the animation Android adds for inter-app navigation. On iOS they would be as similar as possible, without breaking Apple design guidelines. How this turned out, is described in subsection 4.3.2.

Navigation in one big app

iOS

We decided to use the standard behaviour of the app bar[15] for navigation in the big app. To return to the root app the user will need to press the back button in the top-left corner.

Several apps

Mock-up: Figure 3.5.

Since Apple provides no specific design guidelines for inter-app navigation, we

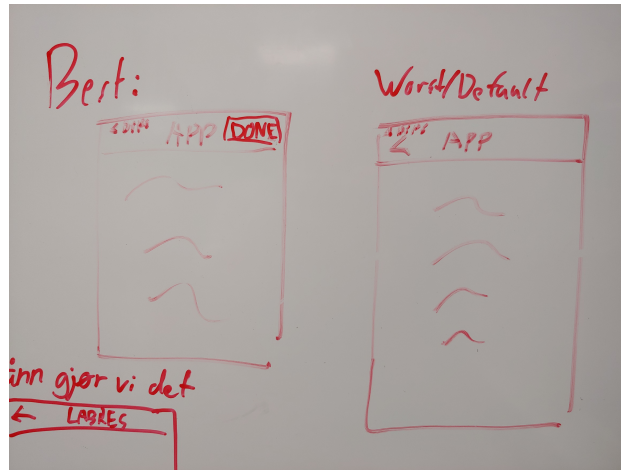


Figure 3.5: Whiteboard sketch of default (right) and our navigation design (left) for iOS.

decided to use a solution inspired by iOS modals. "A modal view typically includes completion and cancel buttons that exit the view." [16]. The same page says that a full screen modal (as opposed to non-fullscreen) should be used for complex tasks that can be completed within the context of the modal view. We were inspired by the iOS 12.2 Files app (Figure 3.6) where instead of having a back button, we had a button saying "Done" in the top left corner to return to the previous app. This is not the standard as modals are typically only used within the same app.

The done-button was placed in the left corner as opposed to the right corner, because the left corner is where the back button usually is. Since our app switches back to the previous view when the Done-button is pressed, our Done button works like a back button, so we put it there for consistency. The user also had the option of using the back to app button as it can not be removed. Since the back to app button is so small, we expected the user to use the Done-button.

Android

One app

We decided to use the standard behaviour of the app bar for navigation in the big app. To return to the root view the user will need to either press the up-button in the top-left corner or the back-button.

Several apps

For mock-up of navigation see 3.7.

Android has the back-button or **recent screen** for navigation between tasks.



Figure 3.6: After tapping a file in the iOS files app, this modal is shown.

Therefore, to give the user the impression of only being in one app we decided to make our own up-button which mimics the behaviour of the up-button in the app bar.

3.1.2.2.3 Usability test execution

The usability tests started with a short introduction, see L.1.1 for details. Usability tests were done by handing the tester an already opened app, and asking them to solve the tasks. When the users asked questions during the test, they would not be answered. Or if they were answered, the answer would be something along the lines of "What do you think?" as recommended by Practical user testing [7]. The book also recommends waiting four seconds before answering a question, as this will make the tester self-conscious and seek to escape the awkwardness by answering the question themselves. We decided to follow that advice.

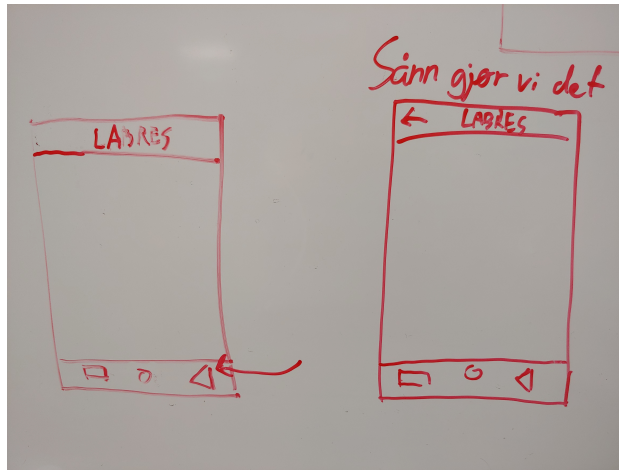


Figure 3.7: Whiteboard sketch of default (left) and our navigation design (right) for Android.

The tests were considered done when the user was sent back to the My Tasks view, and the task list was empty. A decision was made to not stop the subjects even though the test were done, to see if any interesting behavior was exhibited. If the subject said they were done, we said they were correct, and ended the test completely.

Irrespective of operating system choice, every other user would be given the big app to start, while others were given the starting app for the three small apps. This was done to eliminate bias in the System Usability Scale based on familiarity.

Rewards given to test subjects

See appendix B.1.5.

3.1.2.3 Questionnaires

To evaluate the usability of the system and help answer the questions

- There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.
- Navigation between several securely linked apps is just as intuitive as navigation within one secure app.

we opted for the System Usability Scale described in subsection 3.1.1.1.

There are two types of questionnaires. First, one completely identical System Usability Test after each user test. Then finally a demographic questionnaire including some questions about additional information.

The idea was to compare the SUS score of the two systems (one big app and three small apps) to see how the systems compared usability wise.

System Usability Scale changes

We decided to make some slight changes to the questions, see appendix B.1.1 for details.

3.1.2.3.1 Comparative interview

An oral comparative interview was done after both tests and System Usability Scale forms were completed. The goal of the interview was to discover what specifically the testers thought about the applications. The System Usability Scale only measures how usable a system is, but not why it is usable or unusable. That is why we asked questions like:

- "What do you think about the apps?"
- "What was hardest?"
- "What was easiest?"

We also wanted to know if these hypotheses were true or not for that particular tester:

1. There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.
2. Navigation between several securely linked apps is just as intuitive as navigation within one secure app.

To get an answer to these questions we asked two questions:

"How did navigating between the views in the apps work?"

If the users answered this by comparing navigation and finding differences, they would have noticed a difference and hypothesis 1 would be weakened.

If they described the two systems as one it could indicate that they experienced the navigation to be similar enough not to bother differentiating between the systems in their answer, strengthening hypothesis 1.

Regarding hypothesis 2, users could mention that they found one system harder to navigate within than the other, weakening the hypothesis.

If they answered positively without differentiating hypothesis 2 would also be strengthened.

”Did you notice a difference in navigation between the first and second test?”

This is a more straight forward yes/no question and either strengthens (yes) or weakens (no) both hypothesis 1 and 2.

3.1.2.3.2 Demographic questionnaire

To give some more context to the answers we decided to gather some pseudo-anonymous data about our testers. This was done with simple likert scale questions from 1-5. Categories were Age, occupation and self-evaluated technical knowledge. One additional question was asked to the doctors for us to get a better knowledge of the domain. See subsection L.1.2.

The age and gender data will be used to see if we have a representative sample of the population. We included occupation primarily to identify the doctors, because we value doctors more than non-doctors since doctors will be using the eventual implementation of our Single sign-on (SSO) library and inter-app navigation research results.

The questions relating to technical expertise were included to contextualize the the written answers, and the System Usability Scale scores in relation to other systems.

3.1.2.3.3 Pilot tests and modifications

We conducted two pilot tests which caused us to make some modifications before the actual tests. See B.1.2 for details.

3.1.3 Test execution errors and anomalies in methodology

We had to invalidate/remove test-person 13’s SUS score and timing data for the small apps test. This was because we did not ask the test person for their preferred navigation method on Android, and ran the test with without the normal Android navigation buttons (just gestures). Then we repeated the test, but decided that the SUS and timing were not usable due to learned behaviour.

The content and simplicity of the apps we used for testing also led to some confusion and discussion with the test subjects. The test values shown in one of the views (figure 4.17) were apparently not very realistic, and this was noticed by some of the doctors we tested. See L.3 for transcriptions of the comments made by the doctor. The lack of functionality also caused confusion among the test subjects.

Since we had decided to not reveal what we were actually testing, most of the subjects focused on the content and what could be done with it, or rather what they wanted to do, but could not.

There ended up being some additional inconsistencies in the execution of the tests. See B.1.3 for all details.

3.1.4 User test reports

During the tests the observer had a test report document at hand. This was used to write down any test anomalies, test results and user's actions during the usability tests. Relevant comments from the test subject were transcribed later. Other conversation during the user tests were not transcribed for all, since the transcriptions proved of little use in answering the research questions.

For all tests, the screen was recorded. This was done to verify test time and user actions.

To make sure all test were comparable, regardless of the OS, the test were considered started when the subject pressed one of the task presented. The test ended when the user arrived back to the empty list of tasks in My Tasks.

3.1.5 Data processing and storage

See appendix B.1.4 for details about data processing and storage.

3.2 Engineering methodology

To ensure the quality of our SSO library, and the rest of our engineering work, we chose the methodologies described in this chapter.

In general, we needed methodologies to help us cope with the fact that we knew almost nothing about the following:

- Android app development
- iOS app development
- iOS and Android operating system security
- Having different programs work together, open task inside each other and share data.
- Authentication and Authorization with OAuth2.0 and OpenID Connect
- The C# programming language
- MVVM
- Xamarin Forms

3.2.1 Method for collecting information

We mostly used API documentation written by Apple, Microsoft and Google employees to gather information for our technical solutions. We also used RFC 6749 - The OAuth 2.0 Authorization Framework, OpenID Connect Core 1.0 and RFC 8252 - OAuth 2.0 for Native Apps. We consider all these to be primary sources.

We did not rely too much on scientific papers because of the rapid year-to-year changes in mobile operating systems. Security flaws are patched frequently, so papers finding flaws need to be cross-checked with recent patches to see if they still exists. This is a labour intensive process that we did not consider relevant enough for our research questions. Instead we assumed that if the Android and Apple documentation claims that something is secure, it is secure.

3.2.2 Design Science Research

Design Science Research (DSR) is a research framework that among other things, considers the development of software as a method for generating knowledge about software engineering.

According to [17], Design Science Research consists of three cycles.

1. The Relevance Cycle: "Inputs requirements from the contextual environment into the research and introduces the research artifacts into environmental field testing." [17]
2. Design Cycle: "Supports a tighter loop of research activity for the construction and evaluation of design artifacts and processes." [17]
 - "The heart of any design science research project." [17]
 - Generates design alternatives and evaluates them against the requirements until a satisfactory design is achieved. [17], [18]
 - The requirements come from the Relevance Cycle, while the design and evaluation theories and methods come from the Rigor Cycle. [17]
3. The Rigor Cycle: "Provides grounding theories and methods along with domain experience and expertise from the foundations knowledge base into the research and adds the new knowledge generated by the research to the growing knowledge base." [17]

The cycles are illustrated in Figure 3.8.

3.2.2.1 Why did we choose to use DSR?

We had a project based in real world requirements (relevance). Through spiking and scrum we would do many design cycles. We would base our research on existing papers and documentation, as well as contributing ourselves to the knowledge base with our bachelor's thesis (rigor).

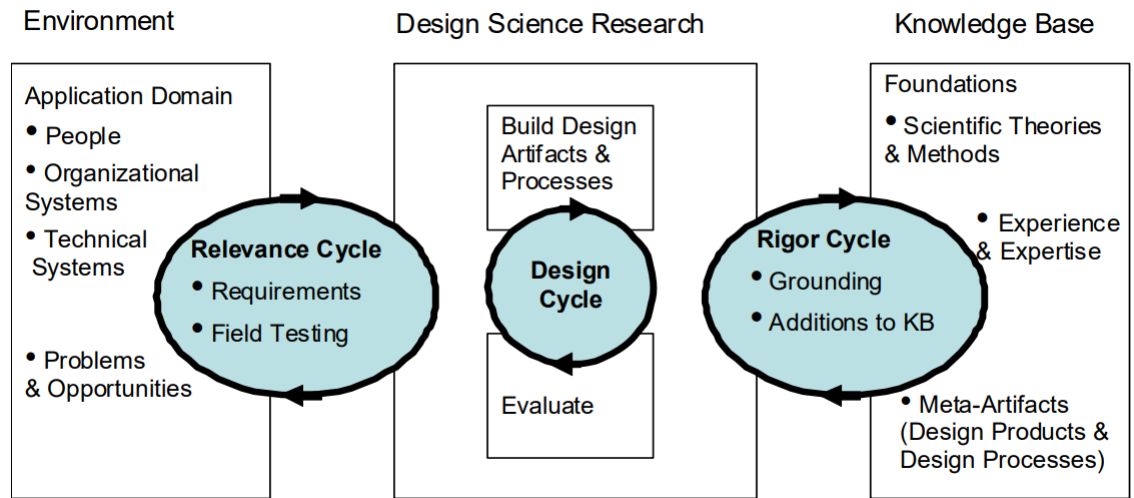


Figure 3.8: The Design Science Research Cycles according to [17]

3.2.2.2 How did we use DSR?

This is detailed in subsection 5.5.8

3.2.3 Development methodologies

Scrum, spiking, pair programming and mob programming were used during development. Detailed introductions to the methodologies are found in section C.1. We also used a special room configuration, and held regular mini-seminars. The reasoning for this and further details can be found in subsection C.1.2 and subsection C.1.3.

Chapter 4

Results

This chapter presents the results of our research and engineering work.

4.1 User testing results

We executed 13 user tests. This is not enough to draw definitive statistical conclusions based on the data. But the data might give an indication of what might be true.

4.1.1 Presentation of the empirical data

Of the 13 test subjects there were 4 doctors and 9 non-doctors. One of the non-doctors tests were not included due to being too much of an outlier. The subject we removed was 80+ years old without smartphone experience. He is not represented in the results. The SUS score for test subject 13's small apps test was invalidated due a methodology mistake described in subsection 3.1.3.

Every test was either done on iOS 12 or Android 9.0. All number values are rounded to one decimal. Time values are rounded to zero decimals. The age groups we tested include one over 60, four between 46-60, three between 26-35 and four between 15-25. The gender distribution was equal.

At the end of the user test we asked the test subject to fill out a questionnaire about their relationship with technology and phones.

As mentioned in paragraph 3.1.2.2.3 we did every other test subject with three small apps first. This was not divided into OSs, which led to us having a unevenly distributed amount of people using three apps first on the OSs. On iOS we got 2 with three apps first and 4 with one app first. On Android this number was 4 with three first and 2 with one first.

4.1.1.1 The data

4.1.1.1.1 Time to complete

The time used to complete the tasks on three apps across both OSs were on average 32 seconds and the median being 31 seconds. On the big app the time were on average 49 seconds, and the median being 31 seconds.

The average of the first test on both OSs were 44 seconds for the three apps and 1:10 minutes on the big app. The median was 43 seconds and 1:09 minutes respectively. The average of the second test were 22 seconds on three apps and 28 seconds on the big app. The median was 22 seconds and 25 seconds respectively.

4.1.1.1.2 Navigation type

Almost all the test persons used the same type of navigation on both tests. There were two exceptions. Test subject 4 claims that they did not tap the back to app button on purpose, and our recordings, see L.3 confirm that their finger was in the middle between Done and back-to-app. So we think it is fair to say that their intention was to click Done.

Test subject 6 used up-navigation in the big app and back-navigation when testing the small apps. When asked about it he said he did not do it consciously, see L.3.

We cannot merge the iOS and Android navigation data because with the exception of Android up-navigation (iOS Back-navigation), their buttons function differently.

4.1.1.1.3 Noticed a difference in navigation

Only two of the 13 testpersons noticed a difference between one big app and three small. Both of them were on iOS and doing the test with one big app first. The question we asked was "Did you notice a difference in navigation between the first and the second test?"

The the first subject answered first "No, actually not", but answer was changed to "No. Weeell, no more than that it went faster, because, well, uuuh. It said Done at the top, it wasn't on the first" when asked the leading question "nothing?" after. The second test subject said there was a difference right away. No followup question where asked as we had learned from the previous incident. The answer: "Yes, things somehow flowed better in the first [test], [it] was the larger transition from the picture to the picture in the other".

4.1.1.1.4 System Usability Scale scores

Valid test subjects: 3 doctors (Removed one outlier)
8 non-doctors (Removed one outlier)

Average and median scores across all test subjects:

	small apps	big app
Average	74.0 (71st percentile)	78.6 (85th percentile)
Median	77.5 (78th percentile)	82.5 (93rd percentile)

The percentiles are from Figure 3.2. Based on [13] both tests received a B on average.

Based on the median, the small apps scored a very strong B, while the big app scored an A.

Average SUS difference: 4.6 points

Median SUS difference: 5 points

This is a small difference in numbers, but a sizable difference in how well the apps compare to a large sample size of other systems tested using SUS.

The small apps are on average better than 71% of the sample SUS scores, while the big app is on average better than 85% of the sample SUS scores.

Reliability of the SUS scores

We base our reliability on the findings in [8] by Dr. Tom Tullis and Jacqueline N. Stetson. First they ran 123 comparison tests on two sites using one of 5 different usability testing methods, including SUS in each test. This produced 19-28 ratings for each usability testing method. With this data they established that Site 1 was better than Site 2. Then they showed that the SUS reached the correct conclusion 20/20 times when 20 random samples of 12 were drawn from a total of 19-28 SUS scores total. The study concluded that for the conditions of their study (website comparison) a SUS-sample-size of 12 gives reasonably reliable results.

In an email exchange we had with Dr. Tom Tullis he said that using SUS with 13 participants, we can be "pretty confident" in our data. He also recommended we do t-tests to get our confidence interval.

- A t-test on our median SUS-score for three small apps gives a 90% certainty that our population mean is 77.5 ± 5.7

- A t-test on our median SUS-score for the big app gives a 90% certainty that our population mean is 82.5 ± 3.5

When comparing the SUS score between the platforms, the big app got an almost identical score on iOS and Android, see 4.1.

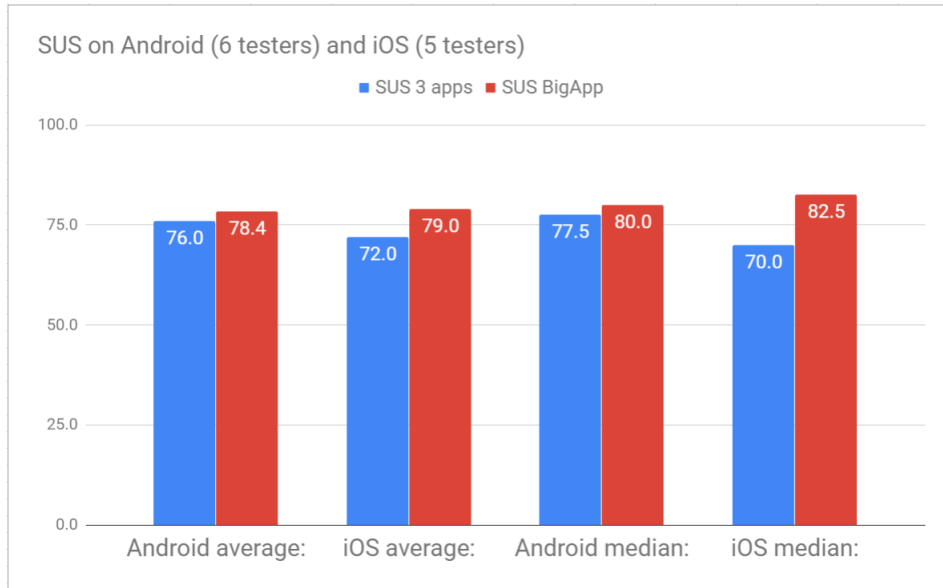


Figure 4.1: SUS comparison.

While the SUS score is slightly lower for the small apps on Android, the difference is bigger on iOS. The small sample size of less than 8 people per operating system, gives a SUS-accuracy of 35% or possibly less, based on Figure 3.1.

Looking more closely at the iOS test data:

Testnumber	Small apps first	Noticed difference	Small apps	Big app
4	No	No	85	85
5	Yes	No	60	85
11	Yes	No	55	62.5
12	No	Yes	70	82.5
14	No	No	90	80

Testperson 5 drags the iOS small app average down despite saying they did not notice a difference. Testperson 12 started with the big app first, yet rated it higher than the small apps. They (nr. 12) said they noticed a difference in how well the system flowed, full report in L.3.

Questionnaire answers

Although the questionnaire contained several questions, this is the only question relevant to our research questions:

- "On a scale from 1 to 10, how much do you want a cellphone as a workplace tool in the health sector?" This was a doctors only question. The average was 9.3 and median was 9.5

The full data is found at section L.4.

4.1.2 User test reports

See section L.3.

4.2 Engineering process results

4.2.1 Scrum

4.2.1.1 Backlog

We had our first meeting building the backlog on January 30th, 3 weeks into the project. On the same day we set up the first sprint, set to last 2 weeks. This first sprint planning meeting was documented with this picture: 4.2.

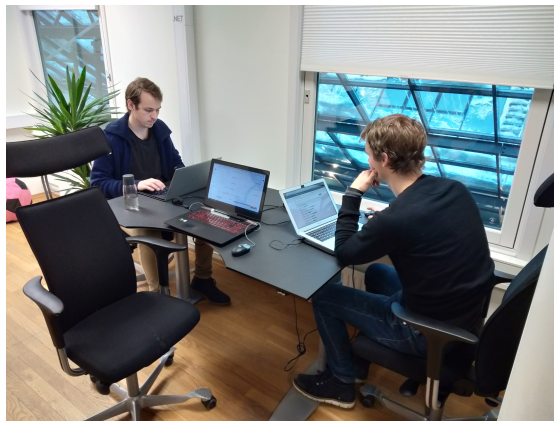


Figure 4.2: The team gets together on 30.01.2019 to create the backlog and plan the first sprint.

We started with 40 issues that can be found inside the vision document, found inside the vision document found at: H.

4.2.1.2 Sprints

We started the first scrum sprint 30.01.2019. The sprints were two week sprints with the exception of sprint 2 and the later mini-sprints. Sprint 2 was 3 weeks long due to it including the innovation camp we had in week 7.

We did not have a burn-down chart, in stead we used the progress bar on the GitHub projects overview page to see how the sprint was going. See 4.3

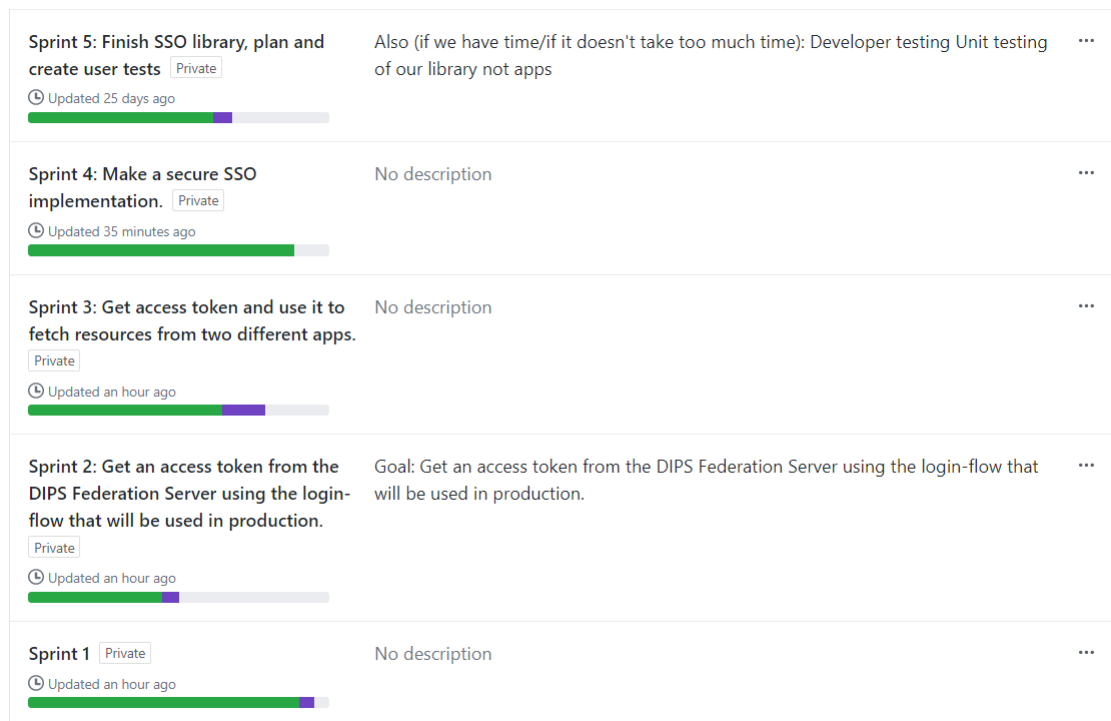


Figure 4.3: Spring completion, by number og issues resolved

We can see that sprints 1, 3 and 4 had almost all the planned issues resolved, while 5, 4, and especially 2 had many unresolved issues, fig 4.3.

As seen in the gantt diagram, fig 4.19, we planned to have 1 month exclusively for writing the report. Since sprint 1 became 3 weeks long instead of 2, the last development sprint was forced to only be one week long. We called this sprint a mini-sprint.

4.2.1.2.1 What did the sprints contain?

The goal of each sprint is written after the number with the exception of Sprint 1 in 4.3.

For Sprint 1 there was no specific over-arching goal, but looking at the issues it contains, it can be said to be about:

- Making a prototype of sending data between apps.
- Write the required project planning documentation.
- Learn Xamarin Forms and MVVM.

4.2.1.2.2 Mini-sprints

As shown in figure 4.20 we halved the durations of the sprints after sprint 5. This was partly due to fact that we had planned to use the last four week for this report and we wanted to start a new sprint when we started writing. We only had one week between the end of sprint 5 and the start of writing this report.

We also wanted shorter sprints to have the chance to have more frequent reviews and planning sessions so we could more easily track our progress towards the end.

The mini-sprints can be seen in Figure 4.4



Figure 4.4: Minisprint goals and completion

4.2.1.3 Retrospectives

Sprint retrospectives were held after each sprint. Each retrospective started with the scrum master introducing a game to get the team members into the right mood. See D.1 for details and additional insights.

4.2.1.4 Spiking

Spiking was used continuously throughout the entire project, but particularly in the first four sprints. In total **8** different apps were created in order to explore technologies or verify functionality, in addition to our user testing apps. Not all of them are described in the results chapter, but they can all be found in the GitHub repository.

4.2.2 Version control

We used GitHub as a version control for all our code. Pull request were executed when new functionality were going to be added to the master branch. Before a pull request was merged we completed a code-review to ensure the code was of high quality.

4.2.3 Pair programming

Pair programming was used around 10 times throughout the project. Sessions were done almost exclusively when the code to be written was essential SSO-library code. See D.2 for details and considerations.

4.2.4 Mob programming

Mob programming was attempted in the first two sprints. See: 4.5 and 4.6. After three sessions a discussion was had. The result of the discussion was that team members were unsatisfied with the efficiency of mob programming.

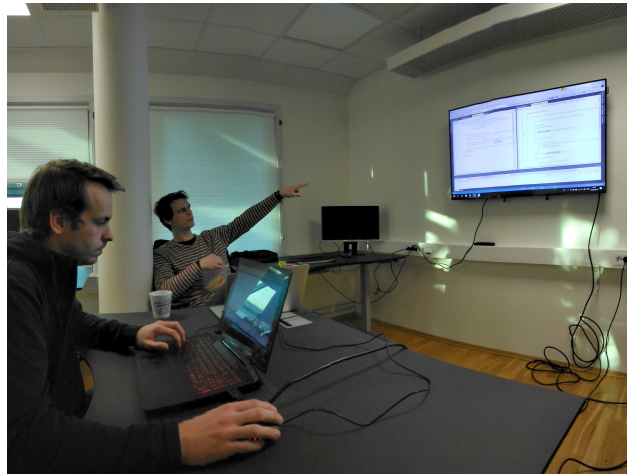


Figure 4.5: The team does mob programming on February 1st.

A decision was made to stop doing mob-research, and rather switch to pair



Figure 4.6: The team does mob programming on February 18th.

programming and pair research. This way, the last developer was freed up to do something else productive. See D.3 for reasonings why we thought this was a good decision.

4.2.5 Mini-seminar

Approximately 12 mini-seminars were held. In the earlier sprints, team members would research for days. At the end of such a research period, they would gather and present their findings to the rest of the group. Team members exclaimed that they enjoyed both holding and attending the seminars. Questions were asked to the speaker who had to defend his choices.

Due to the popularity(D.3) of mini-seminars it was formalized later in the project that each sit-down would include the question "Does anyone have anything they wish to share with the rest of the team?" If it was something of substance, the team would move to the mob/pair-programming station and and/or seminar would be held.

4.2.6 Cooperation with project lead and supervisor

See D.4 for details.

4.2.7 Design process

4.2.7.0.1 OAuth 2.0 Client

We went through several design iterations for the OAuth 2.0 client part of the library. See K.3.1 for some examples of early, simple sketches. We initially ended up deciding to make a class representing each possible type of flow in the

OAuth 2.0 and OpenID Connect specifications.

There would be a OAuth 2.0 class which is extended by an OpenID Connect class, as shown in figure: 4.7. Both of these will again be extended by all classes representing their possible flows. The OAuth2.0 flows are detailed in G.1.2.1, while the OpenID Connect flows are described in G.2.1.

Naming the classes based on flows was done because we thought developers would find it convenient and understandable to have classes corresponding to the flows in the OAuth2.0 and OpenID Connect specifications.

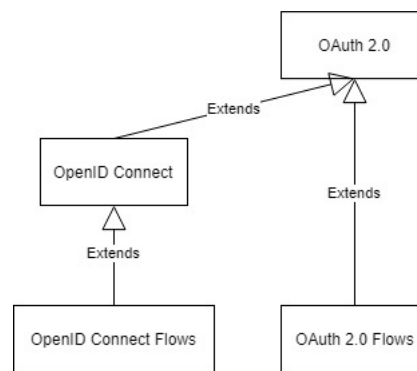


Figure 4.7: First draft of the authentication and authorization part of the library would be.

After a period of developing our initial design, we realized it would likely prove unfeasible and would lead to incomprehensible and unexpandable code, especially due to the necessary use of inheritance. When we reached double inheritance, see Figure 4.8, we had to reconsider our approach.

This led us to have a meeting with the development lead here at DIPS Trondheim, Runar to look at alternative design solutions for our code structure.

We considered a few choices:

Component pattern

We considered the Component pattern, which also proved unsuitable. It did not suit the standard and led to an unnecessarily complex model, see K.7 and K.8.

Builder pattern (Final design)

Instead of having a class for each flow, we had two classes, one targeting the Authorization endpoint, and one targeting the Token endpoint.

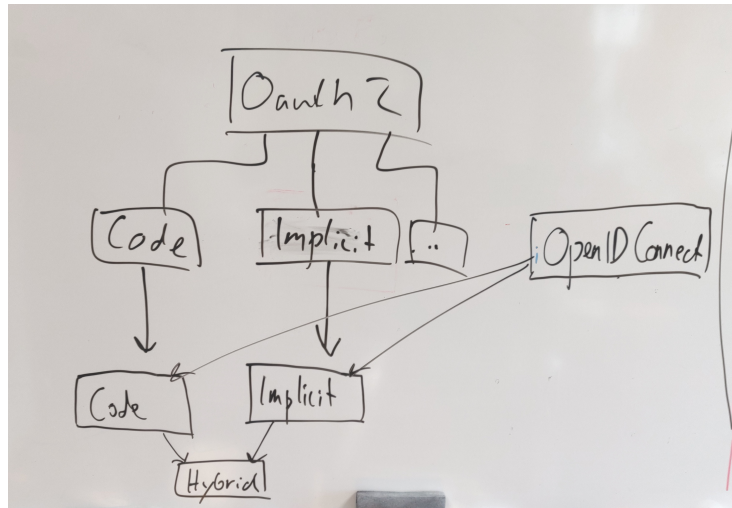


Figure 4.8: Inheriting from two classes. Not allowed in C#

We used the Builder-pattern for to add optional arguments and customization to the requests. This allowed the same functionality as our problematic flow-based structure, using only two classes.

An authentication and/or authorization flow consists of just the authorization request or both the authorization and token request. Since the difference between the flows is mostly what is returned at different points in the flow, it made more sense to structure the code around the two separate requests.

See K.3.1 for sketches of a design with Builder-pattern. The new structure design is shown in figure 4.12.

4.2.7.0.2 Token storage and sharing

Due to our considerable lack of knowledge in mobile development and all its possibilities we did not have a clear idea initially of how the token storing and sharing part of the library would be structured. We only had the idea to constrict the parts the user of our library would have to deal with to the single class; **TokensHandler**. This would ideally be all a developer would need to use to be able to have a SSO system. The **TokensHandler** would communicate with the necessary classes for storage and sharing, as shown in figure: 4.9.

The process to arrive at the final design involved several sketches of possible sharing flows. Figure 4.10 is an example of the whole flow from authentication to retrieving a new access token with the refresh token.

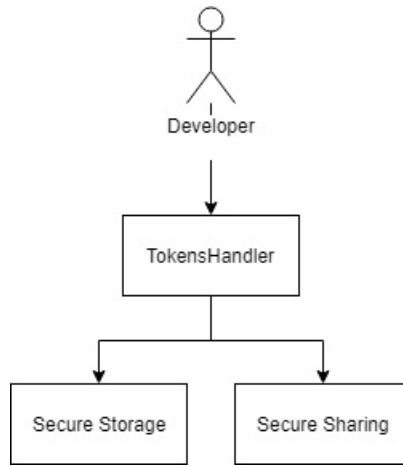


Figure 4.9: First draft of how the token sharing and storage part of the library would be.

4.2.7.0.3 Android

For Android we landed on using **ContentProviders**, see G.2.4.2.2. We had some doubts on how we should implement the system. One idea was to designate one of the apps to be the main entry point where all the tokens would be stored, then all the other apps could simply retrieve the tokens from that app. If the tokens were to expire any app could still start a authentication process, but the tokens would then have to be sent to the main app for encryption and persistent storage. This process was sketched to work like shown in figure 4.11.

We decided not to do this as this would lead to a lot of inter-process communication. Also, if the central app in the previous solution was uninstalled, the tokens would disappear. The final design were to give every app the possibility to encrypt and persistently store the tokens. See paragraph 4.3.1.2.2 for the final design.

4.2.7.0.4 iOS

We considered using either custom URL schemes or universal links to send the tokens between the apps when the end-user was redirected. But this idea was scrapped when we learned about the keychain on iOS.

Using the keychain would easily enable encryption and storage. Shared access to the keychain could be easily granted to only specific apps.

4.3 Engineering artifacts results

4.3.1 SSO library

This chapter aims to answer the question:

What is a secure way to implement a well made cross platform SSO library for iOS and Android?

We will examine if the library meets the requirements specified in subsection 1.2.1.3.

The result we have is more a proof of concept of what we initially hoped for, as the library doesn't contain all the functionality we planned to have and its stability is not tested beyond manual testing.

4.3.1.1 OAuth 2.0 client

To meet the requirements:

1. The library supports authentication and authorization with an external OAuth2.0/OpenID Connect server. This should be done through one library method, and the developer should be notified on successful login.
2. The library uses the same code base for both Android and iOS.
3. The library is easy to expand upon and re-use components from.
4. The sign-in session with the authentication and authorization server cannot be hijacked by an attacker.

We created our own OAuth2.0 client to be included in our library. Our initial plan for the structure of the OAuth2.0 client changed quite drastically, as described in 4.2.7.0.1, but currently it works like detailed below and as shown in figure 4.12.

The client part of the library is largely split into two parts: **AuthorizationRequest** and **TokenRequest**. See K.1 for a complete class diagram and K.3 for a diagram of the hybrid flow we used when authenticating.

4.3.1.1.1 Authorization Request

The **AuthorizationRequest** class models the authorization request defined in the OAuth 2.0 specification[19] and the authentication request in OpenID Connect specification[20]. The class currently supports:

- The authorization and authentication part of all three OIDC authentication flows using the device's browser as a user-agent:
 - Authorization code flow.
 - Implicit flow, should work but is currently not tested.

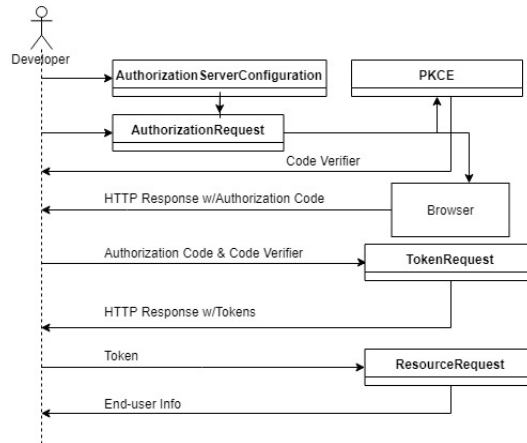


Figure 4.12: Simplified model of the OAuth 2.0 client part of the library. See K.1 for a complete class diagram.

- Hybrid flow.
- The authorization part of these two OAuth2.0 authorization flows using the device’s browser as a user-agent:
 - Authorization code flow.
 - Implicit flow, should work but are currently not tested.
- The use of PKCE.

Most of the optional parameters in the OIDC specification are currently not supported due to time constraints.

An **AuthorizationRequest** object is built using the Builder-pattern. The required parameters for the simplest possible authorization request call are required in the **AuthorizationRequestBuilder** constructor. Using the builder pattern means that additional parameters like nonce and state could be added in the future with eventual **SetNonce()** and **SetState()** methods.

How the **AuthorizationRequest** works

The developer specifies the wanted flow with the response type parameter of the **AuthorizationRequest** object and provides the **AuthorizationRequest** object with an **AuthorizationServerConfiguration** object. This is used to hold information about the authorization server the developer wants to use. PKCE can be added to the flow at this point if required. However, the library gives the responsibility of storing the code verifier for the token request to the developer to avoid having to keep a state in the library.

By calling method `RequestCodeUsingClientBrowser()` the `Xamarin.Essentials` method `Browser.OpenAsync()` opens the browser, the browser is then used to send the generated request so that either the authorization or authentication server can present the resource-owner with some kind of authentication and/or authorization method in the browser. For example, a login screen could be displayed in the browser where the resource-owner can authenticate with a username and password. This could be followed by giving the resource-owner a choice of approving the client to access their resources. If this happens successfully the authorization server will generate a redirect URI with the requested items and deliver it to the browser.

The browser can then redirect the resource-owner back to the app with either the tokens or the authorization code, or both. Receiving the URI in the app is not handled by our library.

Explanation of one way to receive the response from the server can be found here [E.1](#).

4.3.1.1.2 Token request

The **TokenRequest** class models the token request defined in the OAuth 2.0 specification[19] and the token request in OpenID Connect specification[20]. The class currently supports:

- The token request part of these OIDC flows:
 - Authorization code flow
 - Hybrid flow
- The token request part of these OAuth2.0 flows:
 - Authorization code flow
 - Client credentials flow, as long as the supported client authentication method is used.
- Client authentication is done by a combination of the client ID and the client secret. For this to be considered secure authentication there must be a way to safely store both the id and secret. As of now the responsibility for that is left to the user of the library.

Validation of the token response, as specified in [20][3.1.3.5] is currently not implemented due to time constraints.

How the TokenRequest works

The **TokenRequest** class handles the request for the token(s) and delivers the HTTP response to the developer.

The `TokenRequest` constructor parameters are whatever is necessary for the

flow chosen as defined by either the OIDC specification or the OAuth2.0 specification, and if PKCE was enabled it also requires the code verifier created in the **AuthorizationRequest**. The `HttpResponseMessage` is then passed to `TokensHandler` for processing, as specified in 4.3.1.2.1

4.3.1.2 Token storage and sharing (SSO)

To meet the requirements:

1. Tokens stored by the library cannot be extracted or modified by other apps on the device.
2. Tokens on the device are inaccessible/unusable when the device is locked.
3. The library can keep the user logged in across all the apps on the same device, created by the developer, without re-authenticating for each app.
4. The library is able to prompt the user to re-authenticate if their session has expired, letting them continue where they left off before their authorization expired.
5. The library should only prompt the user to log in if the user isn't already logged in.
6. The library uses the same code base for both Android and iOS.
7. The library is easy to expand upon and re-use components from.

We implemented the functionality presented below, which is also shown in figure 4.13. The complete class diagram is in appendix K.2.

4.3.1.2.1 How it works

The developer using the library only needs to use the **TokensHandler** and the **TokensContainer**. The actual encryption, storage and sharing of tokens is handled by the rest of the library.

Shared code (**TokensHandler** and **TokensContainer**)

The classes marked as "shared code" in figure 4.13 contains code that is used by both iOS and Android.

- The **TokensHandler** has all the necessary methods for storing and sharing the tokens between the apps.
 - **StoreTokensInAllApps()**: Takes the **TokensContainer** in **TokensHandler** and stores the tokens in an encrypted state in a location only reachable by the authorized apps.
 - **RetrieveStoredTokens()**: Retrieves the tokens currently stored on the device. The tokens will be returned in a decrypted state.

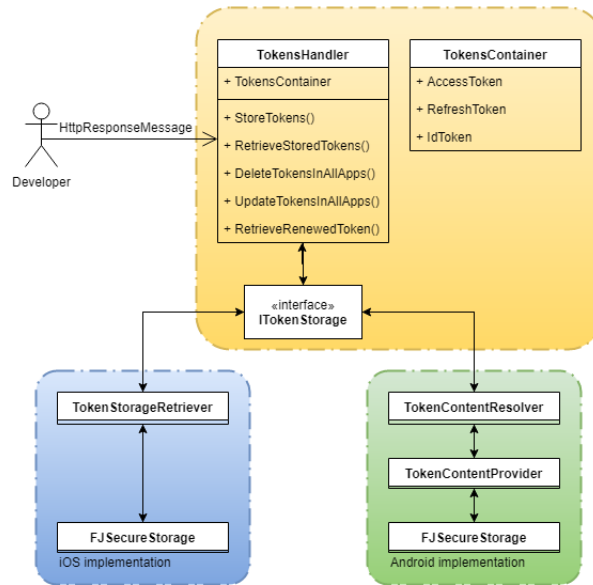


Figure 4.13: Simplified model of the storage and sharing components in the library. Yellow section is shared code, green is Android specific and blue is iOS specific. See K.2 for a complete class diagram.

- **DeleteTokensInAllApps()**: Deletes all tokens currently stored on the device. Can be used when end-user logs out.
- **UpdateTokensInAllApps()**: Removes currently stored tokens on the device and replaces them with new ones. Can be used when an old token is expired and a new one is fetched from the authorization server.
- **RetrieveRenewedTokens()**: Retrieves new tokens from the authorization server using the refresh token. Also updates the refresh token and id_token in that app if new ones are received. If the request should fail it returns the reason phrase sent in the **HttpResponse**.
- **RetrieveAccessTokenAsync()**: Retrieves the access token using **RetrieveStoredTokens()**, then checks if the access token is expired. If it is, **RetrieveRenewedTokens()** is called and a fresh access token is returned. If not, the access token is returned.
- The **TokensContainer** class holds all the tokens as strings and implements a method which check if the tokens are expired or not.

The developer have to send the **HttpResponseMessage** containing the access token as a parameter to the constructor of the **TokensHandler** object. The **HttpResponseMessage** is parsed and the tokens are extracted and put into

a **TokensContainer** object.

For Android the developer also have to provide a list of all the installed apps' application IDs, since these are used to identify each Android app's **Content Provider**. For iOS the developer needs to provide the name of the **KeychainAccessGroup** used by the iOS apps. If the apps is only developed for one platform, this can just be an empty string (iOS) or list (Android) for the OS not used.

Functionality for encrypting, storing and sharing the tokens between authorized apps requires platform-specific code. We use the Xamarin.Forms **Dependency Service**[21] to call platform-specific functionality from the shared code, as shown in figure 4.13. This is done through the **ITokenStorage** interface.

4.3.1.2.2 Android

The developer must define a permission with a given name and with **signature** level protection for the sharing system to be secure. Each authorized app must be signed with the same certificate and use the defined permission. See G.2.4.2.2 for explanation.

- **TokenContentProvider** implements a **Content Provider**, see G.2.4.2.2.
- **TokenContentResolver** contains the **Content Resolver** and all methods used to contact all the other apps' **Content Providers**.
- **FJSecureStorage** encrypt and decrypt tokens using a Android KeyStore provider and stores the tokens in **SharedPreferences**, see G.2.3.2.3 and G.2.2.2.3. This is mostly a copy of the Microsoft's **SecureStorage** open-source code[22]. We added **setUnlockedDeviceRequired()** to the generated keys to ensure the cryptographic keys can't be used to decrypt when the device is locked, see paragraph G.2.3.2.3.

When **StoreTokensInAllApps()** is called the calling app will encrypt and store the tokens in its own **SharedPreferences** file, then store it in the other apps by calling the **Insert()** method in all the other apps' **Content Providers**. Each app will encrypt the tokens with their own key and store them in their own **SharedPreferences** files.

When **UpdateTokensInAllApps()** is called the calling app takes the new tokens and goes through the same process as the store method did.

When **DeleteTokensInAllApps()** is called the calling app goes through all the apps' **Content Providers** and deletes any stored tokens.

When **RetrieveStoredTokens()** is called the calling app only checks its own **SharedPreferences** file by going through its own **Content Provider**. If the tokens are not there, then they won't be in any of the other apps. The developer

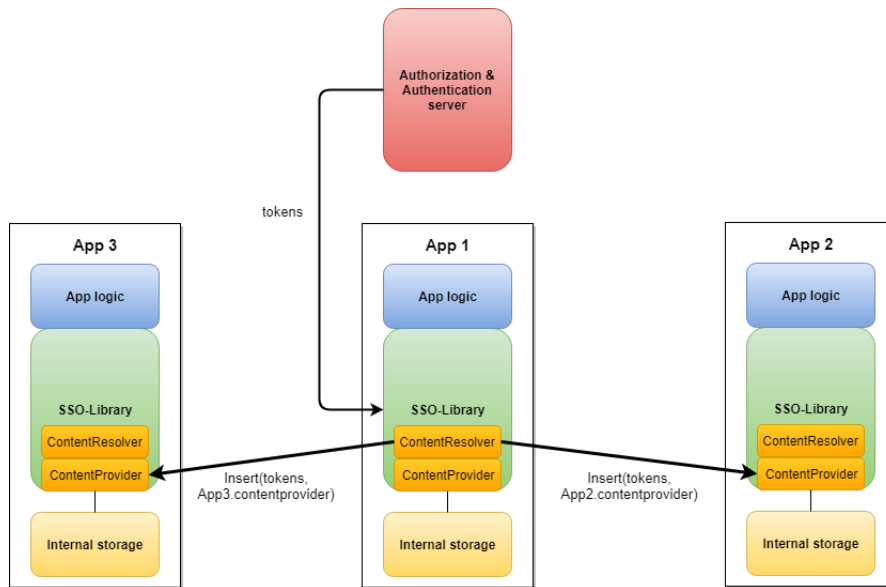


Figure 4.14: Final design on Android: Distributed token storage

would then need to get a new token from the authorization server. If they are there, the tokens are decrypted and returned.

Underlying storage mechanism can be replaced

A consequence of choosing to share data between apps with ContentProviders and ContentResolvers is that the underlying storage mechanism can be replaced without changing the way apps store and request data from each other.

As mentioned earlier, TokenContentProvider extends the ContentProvider class, which the TokenContentResolver uses. Currently secure storage is done by the FJSecureStorage class, storing encrypted data in SharedPreferences. But this can be replaced with for example a database, without requiring changes to the TokenContentResolver class.

Explanation of token sharing between apps

Each app stores the tokens encrypted in its own private internal storage. Every time a new tokens are retrieved from the authorization server the app which initiated the token request distributes the tokens to all the other apps. This limits the token sharing to only happen when a token has expired.

4.3.1.2.3 iOS

The class TokenStorageRetreiver is just an intermediate between TokensHandler and FJSecureStorage for iOS.

FJSecureStorage has methods for adding, fetching and removing items on the keychain. FJSecureStorage calls methods in the Keychain class, that talk directly to the keychain. To add an item to the keychain, you make and store something called a "record" (SecRecord (Secure Record)). This object has some properties which states everything about the item, such as the key, value, access group and data protection class. It also has a type. For us we chose GenericPassword. None of them matched our purpose perfectly, but GenericPassword was the closest and also had general use-cases.

At the bottom we use the static class SecKeyChain to interact with the Keychain using the record as a parameter into its methods.

What we added to Xamarin Essentials SecureStorage to make it our own

- We added the functionality to add an TokenContainer object directly as three different items on the Keychain.
- The default data protection class was changed from AfterFirstUnlock to WhenUnlocked.
- Most important: we added functionality for access groups. This was essential for apps to share securely stored data in a seamless way. Every app still has to enable access groups manually. See appendix J.

4.3.1.3 SSO library dependencies

One of the requirements for our library was: "The library has few, if any, external dependencies." See here for detailed list of dependencies E.2.

4.3.2 Apps for testing inter-app navigation UI

To answer the question: **From a user perspective, do several securely linked mobile applications function as well as one big mobile application?**

and help confirm or deny the following hypotheses we created the applications to be used for user testing specified in this section:

- There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.
- Navigation between several securely linked apps is just as intuitive as navigation within one secure app.

The whole test contains a total of three views, as shown in figure 4.17 and 4.15. It was designed to give the user two easy tasks to complete. There is a main

page with a list of the two tasks. The user navigates to the task by clicking the task in the list. A new view would then open. When the user completed the task they would need to go back to the task list so they can do the last task.

We created a total of four apps for user-testing, see appendix: K.4. One of the apps, BigApp, contains the entire test in one app and the three other apps, DIPSTasks, LabResult and DocumentViewer, combined contains the same test, where each app shows only one of the views. The latter was used to check if the user managed to navigate between several apps and if there was a noticeable difference between having one app or three different apps with the same functionality for the end-user.

4.3.2.1 iOS

On the iOS version of the small apps test we made a Done-button which takes the user back to the previous view, see figure 4.16. Basically providing the same functionality as the back button does in the big app test.

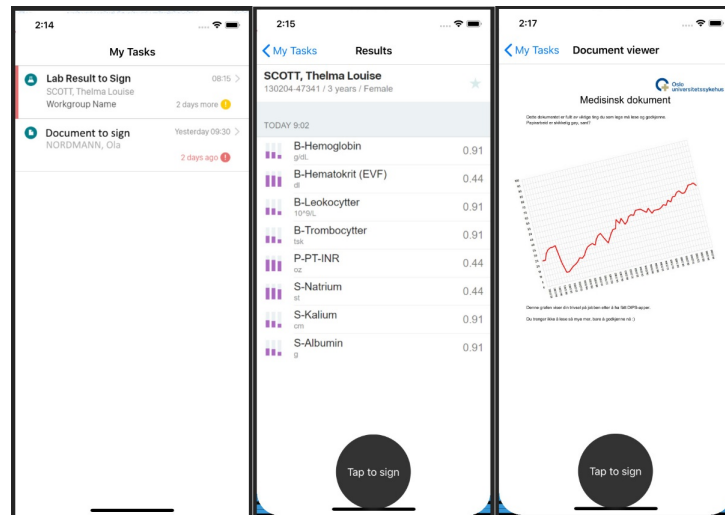


Figure 4.15: The three views in the big app on iOS.

4.3.2.2 Android

On the Android version of the small apps test we made an up-button with the same functionality as the up-button in the big app test. This means there is no visual difference between the big app test and the small apps test, see figure 4.18. The difference is only in the delay caused by starting a new process and loading the app, and the different animation when switching views.

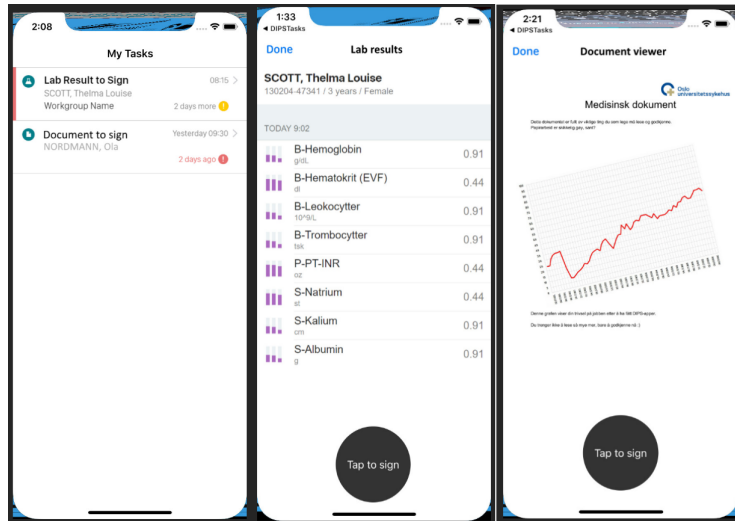


Figure 4.16: The three views in the small apps DIPSTasks, LabResults and DocumentViewer on iOS.

4.3.3 Apps for testing our SSO library

We made two apps to continually test the library's functions and its usability from a developer's perspective. Especially centered around verifying the functionality of the following:

- The library supports authentication and authorization with an external OAuth2.0/OpenID Connect server. This should be done through a few library methods, and the developer should be notified on successful login.
- The library can keep the user logged in across all the apps on the same device, created by the developer, without re-authenticating between each app.
- The library should only prompt the user to log in if the user isn't already logged in.

These apps were only made for development purposes and their visual design was not considered. See E.3 for details and figures.

4.4 Administrative results

4.4.1 Timesheet

Throughout the project we have logged the hours spent and divided them up in a few categories, we expected to reach somewhere between a total of 500-550 hours each. The categories were:

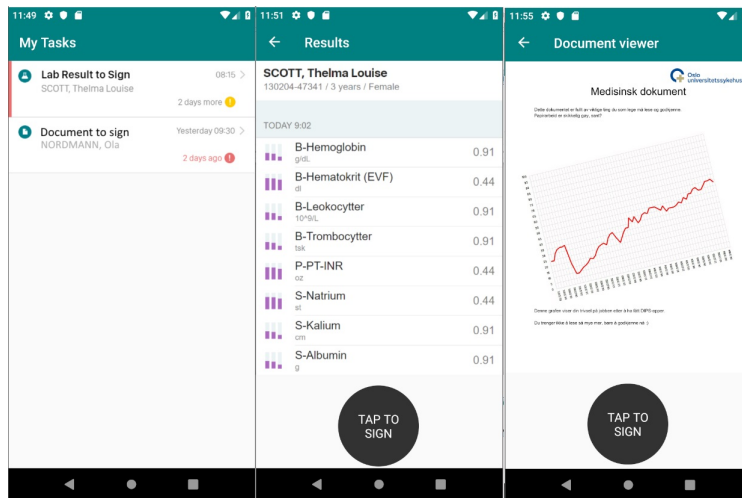


Figure 4.17: The three views in the BigApp on Android.

- Development
- Documentation
- Research/prototyping
- Planning/Meetings
- User-testing

We reached the higher parts of the expected interval with some variance between the team-members. See the project handbook for the details.

4.4.2 Gantt-diagram comparison

As seen in figures 4.19 and 4.20 the actual project trajectory ended up being different to the initial plan, in mainly these two areas:

1. The amount of research required was considerably larger than we expected.
2. User testing were delayed to the later than expected. We had hoped to be able to do several iterations of user testing so we could change the system according to results from the tests.

We also ended up making the last five sprint's duration only one week long(4.4).

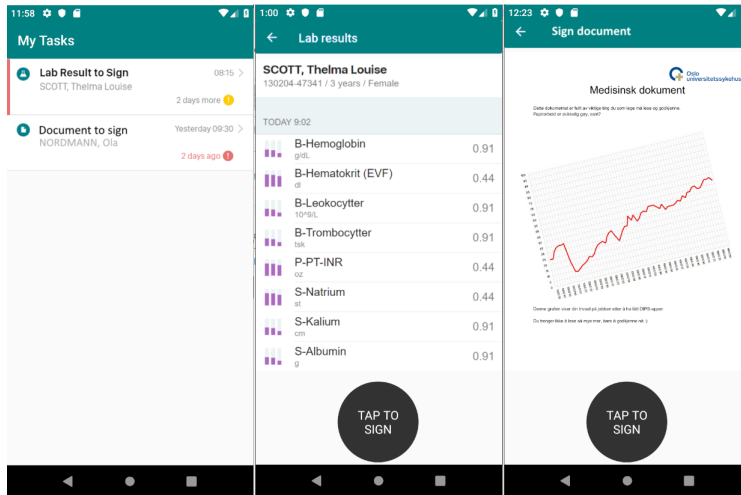


Figure 4.18: The three views in the small apps DIPSTasks, LabResults and DocumentViewer on Android.



Figure 4.19: Our initial Gantt-diagram.



Figure 4.20: Gantt-diagram of what actually happened.

Chapter 5

Discussion

In this chapter we will evaluate if the research questions have been answered. This is done by looking at each requirement and hypothesis one by one and evaluating it. The conclusions regarding each research question are presented in the conclusion chapter.

5.1 Assumptions

Three assumptions that have to be made to consider our solutions to be valid. The discussions in this chapter are based on these being true.

- The OS and the API provided is as secure as stated in the documentation. As there will most likely be discovered new security flaws, one should always use the newest version of the OS. Which means regular device upgrades might be necessary, since many manufactures stop supporting updates after a while. This problem is worse on some Android devices.
- The system is not running on a rooted or jailbroken device. See: 2.6
- An unauthorized user won't get access to an unlocked device. The lack of authentication between app switching might enable an unauthorized person to gain access to protected resources simply by picking up an unlocked phone, with the app opened and using the app as a authorized user would.

To be able to ensure a near absolute degree of security would require consideration of the specific context the device will be used in and the options available to ensure detection and prevention of rooting and jailbreaking.

5.2 From a user perspective, do several securely linked mobile applications function as well as one big mobile application, from a users' perspective?

We defined the applications to "function as well" in 1.2.1, if these three hypotheses are found to be true.

- There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.
- Navigation between several securely linked apps is just as intuitive as navigation within one secure app.
- Given apps requiring user authentication upon startup, the user does not need to re-authenticate when securely linked apps open each other.

5.2.1 There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.

This answers the question if the user will notice any difference between the solutions. Not if the user prefer the one or the other.

When doing user testing we found that two out of twelve answered that they noticed a difference. These were both on iOS doing the big app first.

The first "yes" to noticing a difference was given after a follow-up question was made when the test subject initially said no, see 4.1.1.1.3. This is a methodology fault but we still chose to consider it a "yes" for statistical purposes because a clear reason for noticing the difference was given. The test subject stated that they noticed that one of the apps had a Done-button while the other app did not.

As mentioned in 4.1.1.1.2, test subject 6 used up-navigation in the big app, and back-navigation (gestures) when testing 3 small apps. He answered that he did not notice a difference, but his actions indicate otherwise. His SUS scores however, were nearly identical (80 for the small apps, 77.5 for the big app). We think that the animations when switching apps on Android might have made him start using back-navigation unconsciously, because that is what he is conditioned to do when seeing an app switch animation. The animation can be replaced as described in paragraph G.2.5.2.1. We do not consider this one observation as significant enough to say that there is a noticeable difference on Android, especially since it is so easy to mitigate.

The user testing results indicate that there is no noticeable difference when using three apps on Android, and some noticeable difference on iOS. (2/6) subjects.

If on iOS the views in the big app were done more similarly to the small apps (like they are on Android) by using modal views with a "done"-button, the unsure yes from the first subject might have stayed a no. The second, more clear answer would probably not have changed.

5.2.2 Navigation between several securely linked apps is just as intuitive as navigation within one secure app

This does not answer the question if they notice any difference. Still, if the user does not notice any difference, its fair to assume that the user think both of them are just as intuitive.

To measure the intuitiveness of the navigation we timed every test and gave the subjects a System Usability Scale for each test.

How well the subject did on the test (time used) has only a modest correlation with the score given on the SUS test. This means that it only describes 6% of the results[23].

The usage of time as a metric to answer this hypotheses is questionable. Taking the time of a test has little value, as written in the book[7], since the amount of time for completion doesn't necessarily indicate how well the subject is doing. The second time the subject were always faster, because they learned the tasks. Since we started the tests with small apps first half the time, this difference is evened out.

Time will therefore not be considered in this discussion.

The System Usability Scale (SUS) together with the answer about the difference in navigation between the apps should however be a good indication to what is true regarding the hypotheses.

On Android everybody answered "no" on the question about a perceptible difference in navigation. The difference in SUS score between the small apps test and the big app test is just 1.6 average, 0.5 median. This is a very small difference, and with a sample size of just 6 people we are impressed with how similar scores the two systems got. Maybe the small difference is a result of perceiving an unconscious difference. We believe it is more likely that with a sample size of just 6 people on Android, this difference is due to an error margin inherent with so few test subjects and we do not consider it significant.

On iOS there were one third who answered "yes" on the question about a perceptible difference. The difference on the SUS score (7.0 average, 12,5 median) between the small apps test and the big app test was bigger than with Android. This indicates that the test subjects found the three apps less usable than the big app. With just 5 test subjects, the statistical significance is questionable, so we took a closer look. As mentioned in 4.1.1.1.4, one of the iOS test subjects noted that the small apps did not flow as well as the big app.

For the subjects that answered "no", the difference in SUS score could be because the subjects wanted to answer the same on both tests, but did not remember what they answered on the last test. In this case the difference could be attributed to noticing a difference unconsciously, or just randomness. Since they consist of 4/5 of the test subjects, we believe most of the difference on iOS can be attributed to the test subjects not remembering what they answered on the first test/randomness, because it does not match their oral answers. In total, we consider the three apps on iOS to be slightly less usable than the big app, but not to the degree shown in the average or median SUS scores.

See 5.2.1 on more details on how to interpret the "yeses" from the test subjects.

5.2.2.1 Can our results be generalized?

While it looks like the hypothesis is somewhat confirmed for our testing applications, does that answer the hypothesis in a general sense for all securely linked apps? It depends on if the other apps use the same navigation design and navigation component design as us. Our goal has always been to mimic in-app navigation as closely as possible.

5.2.2.1.1 On Android

The Material Design standard used by android simply states that: "The Back button allows users to navigate recently viewed screens in reverse chronological order." [24]. No inter-app navigation guidelines are given outside of this. While we support this back navigation, we also made up-navigation work in the same way, emulating how up-navigation works in our big app.

For apps using our navigation method, the results apply. For all other apps, they are less and less applicable as the design changes.

5.2.2.1.2 On iOS

Apple provides no specific design guidelines for inter-app navigation. Instead we follow the general navigation guidelines [25], and modals, see item 3.1.2.2.2. It is harder to generalize our solution for iOS, especially since we do not use the back to app button.

Why did we not use the back to app button

As mentioned in G.2.5.1 Apple automatically adds a back button when one app opens another. That means that our apps also have it. Why did we not use it?

There are two major issues and one minor issue with the button. The first major issue is that it is very hard to spot if the user does not know it is there. It is very small and is all black. We think that it blends into the the rest of the top bar where it is placed. The second issue also comes from the button's size and placement. Because it is so small, it might be hard to hit. This may be especially true in a stressful environment, which the hospital can be. The third issue is that iOS sometimes removes or replaces the button. The biggest ones are when screen recorder is on or you are calling someone. In those cases the button is replaced with a button contextual to those actions. If we solely relied on the back to app button in those cases, it would be impossible to navigate back to the previous app without using multitasking gestures.

A potential fourth issue is that this button can't be used to navigate back twice. The button disappears when the app displaying the button goes to sleep. So if you have gone from app A, to B, to C, then the user can press the button to go back to B from C, but then the button in B disappears, meaning the user can't go back to A. This could prove to be a challenge for the user.

Navigating to a sleeping app using custom URL schemes cause the app to restart leading the app to not keep state. In our testing we didn't have time the create apps that didn't get affected by this issue. This lead to the apps resetting the state every time and relying on data from the URL or elsewhere to set to the correct state. This again led to DIPSTasks displaying the wrong task if the user used a combination of these navigation options.

5.2.3 Given apps requiring user authentication upon start-up, the user does not need to re-authenticate when securely linked apps open each other.

Authentication and authorization is the same thing in this instance, since the only person authorized to use the app is the person the phone belongs to. Authorization also identifies the person, and is thus authentication. The words will be used interchangeably in this chapter.

The idea is that having securely linked apps open each other without requiring user authentication is more user friendly. It mimics how it would have been with one big app.

5.2.3.1 Android

The keys in the `AndroidKeyStore` are required to decrypt the tokens needed for calling restricted resources.

5.2.3.1.1 How to make sure the token is available to the app?

When generating or importing a key into the `AndroidKeyStore` it can be specified that the key is only authorized to be used if the user has been authenticated. We have found two options we found promising:

Option 1: Timed validity

First `setUserAuthenticationRequired()` (Android 6.0+ only) is used during key generation to protect the key. By default this would require a PIN, password, fingerprint or other trusted factor to access the decryption key, for each key use of the key. Also setting `setUserAuthenticationValidityDurationSeconds()` would let the key be available for the duration specified, after authentication.

Option 2: Device screen unlocked

Generated keys are protected with `setUnlockedDeviceRequired()` (Android 9.0+ only) during key generation. That means that key access is restricted to the secure lock screen protected by the TEE. Keys will be available as long as the device is unlocked, and inter-app navigation does not change that. The drawback here being that the key is available as long as the device is unlocked, not just when the app is opened. This would allow for automatic background token refresh. But since the requirement is user-authentication upon app opening, this will not satisfy the requirement. We chose this option for our SSO library.

Comparison

Only option 1 secures that the key can only be used when the app is open (because the request for key use will only be made within the app). That is, unless the timed validity extends beyond the user closing the app.

5.2.3.1.2 How to make sure the apps can open each other without needing re-authorization?

DIPS wants apps to require authorization when they are opened. They also want a seamless inter-app navigation experience. Can an app open another app and then not require authentication? It is possible, although we have not implemented any functionality regarding this.

A solution could be:

When an app is opened it checks if it has been opened by an intent or not. If it was opened by an intent, it will not require re-authorization. An attacker could try to exploit this by installing a DIPS app on his device and trying to use an

intent to bypass the authorization step. This can be prevented by requiring the DIPS apps to only accept intents to open from apps signed by the same certificate, as mentioned in G.2.5.2.1.

5.2.3.2 iOS

In Xamarin for iOS, the app's `OnStart()` gets called when the app starts up for the first time, `OnSleep()` get called when the app goes to sleep and `OnResume()` gets called when the app wakes up from a sleeping state. When an app gets navigated out of, it goes to sleep. When an app gets navigated into it calls the appropriate method depending on if the app is asleep or not running.

This means that the developer can not differentiate between using the back to app button and other methods of navigation.

The developer can however differentiate between the points of entry into the app when using custom URL schemes or universal links. This is done by having data from the URL checked in both the `OnStart()` and in `OnResume()` method. The data from the URL is a random string generated in the other app that is also shared through the keychain. If it matches, there is no authentication prompted, if it doesn't match authentication is prompted. When opening the app using another method the URL will be empty and therefore the data will not equal.

This has to be done if the developer want to authenticate the user when opening app from the home-screen when it is sleeping. Using this method will force the user to authenticate themselves if they go into multitasking and back to the app they were in. This is because the app goes to sleep when the multitasking is opened. The user will also have to authorize themselves when using the back to app button.

5.3 Are several securely linked mobile applications as secure as one big secure mobile application?

- When one app attempts to open another app by the same developer, a third app cannot hijack the inter-app navigation and get opened in stead of the intended app.
- Data shared between securely linked applications is not accessible to other apps.

5.3.1 When one app attempts to open another app by the same developer, a third app cannot hijack the inter-app navigation and get opened in stead of the intended app.

See here G.2.5 for details about the possible ways to navigate an end-user between apps.

5.3.1.1 Android

By using explicit intents you can be sure which activity is opened, see G.2.5.2.1. For hijacking to be possible when using explicit intents a malicious app would have to remove the legitimate app from the device, copy the unique identifier of the legitimate app and install the malicious app with that identifier. In this way an attacker could trick the end-user into exposing their username and password by creating a fake login page in a malicious app. This could be an actual problem based on what context the devices would be used in. If there is a chance for an adversary to gain access to a device and being able to modify its contents without detection, this could(to our knowledge) be a possible attack option.

With implicit intents a malicious app could copy the intent filter(G.2.5.2.1) of the legitimate app and the end-user may be prompted to choose the malicious app when switching, so this is not a good option. However, there is really no reason to use implicit intents if you know the package names or class names of the activities you want to launch, which you would in the context we have considered.

5.3.1.2 iOS

The use of custom URL scheme does not guarantee that correct app gets opened due to reasons explained in G.2.5.1.1. With the use of custom URL schemes an attacker could make the user open a malicious app instead. The fact that the identifier (see G.2.5.1.1) also differentiates between the schemes by consisting of your company name, makes is somewhat more secure. The identifier is still not enough to make it completely unique and therefore not secure. This malicious app could trick the user into exposing their username and password by creating a fake login page. The attacker could then get access to data in the original app using this information.

Apple says this about custom URL scheme: "While custom URL schemes are an acceptable form of deep linking, universal links are strongly recommended as a best practice." [26]

As Universal links is the same URL as to the website, it guarantees that the user will be sent to the correct app. This makes it the most secure way to navigate between apps.

Even though Universal links is the most secure way to navigate between apps, custom URL schemes is sufficient if you can guarantee that no new or old apps have the same URL scheme.

5.3.2 Data shared between securely linked applications is not accessible to other apps.

5.3.2.1 Android

When using the system we have made, described in 4.3.1, only apps signed with the same certificate[27] can access the sensitive data. Which means the data is as secure as the cryptographic algorithm used in creating the keys and certificate.

Another option could have been having the apps share a user ID. This wouldn't have been any less secure since the apps must be signed with the same certificate either way and access restriction on Android is managed by the permissions granted to a user ID so this would only be taking advantage of the foundation all security is based on. This could also lessen the inter-process communication necessary if you chose to let the apps share a process, which arguably could be an advantage. But the final decision for not using this was based on the preference of not having the apps share all their data with each-other. Based on our wish to follow the principal of least privilege and the fact that ContentProviders didn't introduce any new security weaknesses(at least that we know of) this felt like a better option.

5.3.2.2 iOS

If the keychain is shared directly by using keychain access groups or app groups this has the same level of security as storing it in an app's private keychain. This is also true when sharing files through app groups. In this case there are no additional security challenges when opting for a multi-app solution.

If the data is shared through a universal link, the data is also secure. No one, except the receiving app the read the link.

This is not true when using custom URL schemes. As another app can make an identical scheme and send the user to this other app instead. If confidential data were shared this way, the malicious app could read this data when being sent to their app.

So as long as the data is shared through one or more of the first methods and not through custom URL schemes, the data should be secure and not accessible to other apps

5.4 What is a secure well made cross-platform implementation of SSO for iOS and Android?

To answer this question we built an SSO-library. In this section we will discuss if the requirements specified in 1.2.1.3 have been met. The requirements are discussed one by one in this chapter.

5.4.1 What are the requirements for the SSO-library to be a well made product?

Based on the requirement specifications and vision document, we consider an SSO-library to be a well made product when:

5.4.1.1 The library supports authentication and authorization with an external OAuth2.0/OpenID Connect server. This should be done through one library method, and the developer should be notified on successful login.

The system we developed to satisfy this requirement is documented in 4.3.1.1. The AuthorizationRequest class enables an authorization and or authentication request to be sent to an OAuth2.0/OpenID Connect provider(4.3.1.1.1). The requirement for requiring only one method to send the request is met. However, receiving the response from the server containing the authorization code requires additional device-specific code outside of our library. A description of how we did our device-specific implementation is given in E.1.

Requiring the developer to write their own code to receive the authorization code from the server is not ideal. On Android, it might be possible to use a Service[28]. This was not seriously considered for our project due to confusion regarding the way Xamarin.Forms works in relation to services and Android activities. On iOS we did not find a way to reduce the amount of custom implementation that was needed to use our library. It might be possible, but we have not found a way.

Completing authentication requires swapping the authorization code for an access token in most OAuth2.0/OpenID Connect flows. This is handled with the TokenRequest class. This is a two step process, for details see 4.3.1.1.2. The reason for separating TokensRequest and TokensHandler is that we wanted the code to be as loosely coupled as possible. The HttpResponseMessage is self-contained and could have been passed somewhere else instead of the TokensHandler. An argument could be made that TokenRequest returning the tokens as strings would be a better idea. That would make the data somewhat unstructured as the string array does not say what it contains. An object containing strings could be returned and passed into TokensHandler. That would require TokensRequest and TokensHandler to know about the same type of custom object,

making them more tightly coupled.

5.4.1.2 The library can keep the user logged in across all the apps on the same device, created by the developer, without re-authenticating for each app.

The system we developed to satisfy this requirement is documented in 4.3.1.2. This part of the library has been repeatedly demonstrated to work using the DIPLogin(E.3.0.1) and TestApp2(E.3.0.2) apps.

The method `TokensHandler.RetrieveAccessTokenAsync()` is crucial in meeting this requirement. It is designed in such a way that if the refresh token stored on the device is still valid, a valid access token will be returned, first by checking local storage, and then by asking the server for a new access token if the one in local storage is expired. If the method fails because the refresh token is expired, an error message from the server is returned. The developer must then authenticate again with the OAuth2.0/OpenID Connect provider as described in 4.3.1.1.

On Android, the use of **Content Providers** as a means of sharing data facilitated a more flexible system than it would have been using any of the other available methods. There is no need for a main entry point app as there is no central storage. The end-user can start with any app in the system, and from there start the authentication and authorization process. There is no unnecessary inter-process communication, since the tokens are instantly sent to all the apps in the system after they are retrieved. The tokens are only sent whenever a access token is expired and a new one is retrieved. Also, if they were stored in one app's internal storage, see: G.2.2.2.1 and the app was uninstalled, all the other apps would lose their tokens. We also considered saving the encrypted tokens in external storage, see: G.2.2.2.2, but it would have made the encrypted tokens sensitive to modification or theft by other apps.

5.4.1.3 The library is able to prompt the user to re-authenticate if their session has expired, letting them continue where they left off before their authorization expired.

With the exception of including a method to sign in via the preferred device browser, see 4.3.1.1.1, our library does not provide any GUI elements. Other than the mentioned exception it is made completely separate from Xamarin.Forms or any other GUI dependencies.

Our library does not automatically open the browser when the refresh tokens expired, although this is easy to add (refdiscussion:automaticre-authentication). As of now, expiration of tokens can be checked with the library.

Our SSO library does not provide saving of application state. Saving of app

state when the browser is opened can be handled by Xamarin.Forms and the respective operating systems.

Should we have made the process of re-authentication automatic?

The problem would require navigating back to the page where such a re-authentication request was made and restoring state, which we have not implemented in our SSO library.

5.4.1.4 The library should only prompt the user to log in if the user is not already logged in.

The process of re-authentication is explained in 5.4.1.3. The reason for this requirement is to make sure that the authentication process is only started when it is required. Since our library does not initiate the authentication process on its own, this responsibility lies with the developer using our library.

5.4.1.5 The library uses the same code base for both Android and iOS.

As described in subsection 4.3.1 the library uses shared code where possible, and platform-specific code where necessary. Platform specific code is limited to persistent token storage and retrieval, and token sharing between apps. This is because the operating systems differ in the ways they deal with persistent storage, as mentioned in subsection G.2.2 and subsection G.2.4.

We do not see any ways to increase the amount of shared platform code in the library beyond what is already there due to the limitations mentioned.

5.4.1.6 The library is easy to expand upon and re-use components from.

As mentioned in 4.3.1.1.1, the `AuthorizationRequest` class is created through a Builder that allows for easy expansion in the future. Both `AuthorizationRequest` and `TokenRequest` return generic `System.Net.Http.HttpResponseMessage` objects. That means that any other part of the library that currently takes these as input, can be replaced.

5.4.2 What are the requirements for a secure SSO library?

For all of these we assume the device is not a compromised device. We define a SSO library to be secure when:

5.4.2.1 It is possible to remotely revoke user access to the application through the library.

Instantly revoking user access to the application(s) is currently not implemented. It is questionable if this should be a requirement. Implementing it would proba-

bly require the library to block the app’s view with some custom UI, or forcibly close the application. It seems to us that the control of those heavy-handed actions should be left to the developer implementing our library.

The back-end functionality for this is possible by reading the error messages found in the `HttpResponseMessage` objects returned from `AuthorizationRequest` and `TokenRequest`.

The OAuth2.0/OpenID Connect server can invalidate the access token, refresh token, authorization code and user credentials. This means that as soon as this is done, the app can no longer request new data from the protected external resource servers because the access token is invalid and a new one cannot be obtained from the OAuth2.0/OpenID Connect server.

In practice this means that an attacker only has access to the information already stored in the app when access is revoked. This could be very little, or a lot, depending on the app implementation. Implementing instant revocation of user access would provide additional security for sensitive data stored in the application, but should be the app developers responsibility.

5.4.2.2 The sign-in session with the authentication and authorization server cannot be hijacked by an attacker.

As mentioned in 4.3.1.1.1, our SSO client supports Proof Key for Code Exchange (PKCE). This is detailed in G.2.1.4.

Since our library supports this OAuth2.0 extension we consider this requirement to be met for clients using code and hybrid flow for authorization.

5.4.2.3 Tokens stored by the library cannot be extracted or modified by other apps on the device

Given that apps using the library follow the platform specific requirements specified in 4.3.1.2, tokens will be securely stored. The backing for this claim is explained below.

5.4.2.3.1 Android

Tokens are stored in the apps’ internal `SharedPreferences` file. They are encrypted using keys from the Android Keystore provider. An attacker could copy over the internal storage from the device, but the encrypted stored tokens would be unusable without the decryption keys stored in the Keystore. Since the Keystore is secure and the keys cannot be extracted, the tokens are secure.

The use of the Android `KeyChain` class would have provided the same level of security, see G.2.3.2.5. However, it requires end-user input, and we did not

want to involve the end-user any more than necessary to avoid potential mistakes.

5.4.2.3.2 iOS

Tokens are stored and shared on the keychain using access groups. Sharing the keychain directly made the sharing of tokens highly secure. There is only one place of storage and the data is only shared with the intended apps.

See 5.3.2.2 for more information on this subject.

5.4.2.4 Tokens on the device are inaccessible/unusable when the device is locked

5.4.2.4.1 Android

When the keys are generated in FJSecureStorage, they are made unavailable for use unless the device is unlocked, see 4.3.1.2.2. It is also not possible to extract the keys as specified in G.2.3.2.3.

5.4.2.4.2 iOS

The default data protection class in our library is set to WhenUnlocked. This means that the items on the keychain stored by our library by default only are accessible when the device is unlocked.

5.4.2.5 The library has few, if any, external dependencies

As described in G.1.1.1, dependencies introduce security vulnerabilities. As mentioned, our library only has four dependencies. Could we have had fewer dependencies?

While Newtonsoft.Json is a part of many C# projects, and considered safe by the developers here at DIPS, we could have avoided it by parsing out the content from the HttpResponseMessage ourselves. Still, we liked the simplicity of Newtonsoft.Json and DIPS had no objections, so we chose to include it.

Since Xamarin.Essentials is only used for opening the device-specific browser, the dependency can be removed if custom code is written for this. This was considered, but since we wanted as much cross-platform code as possible, and in the interest of time, it was kept. Both Xamarin.Essentials and Xamarin.Forms are both owned and maintained by Microsoft, and thus we consider the dependencies to be quite secure.

System.IdentityModel.Tokens.Jwt is made by Microsoft and therefore also considered quite secure.

5.5 Process evaluation

5.5.1 Progress plan evaluation

The discrepancies in the Gantt-diagrams in section 4.4.2 were due a few reasons:

1. We ended up researching and then developing one part of the library at the time. The process consisted of cycles of research with or without spiking and then developing. The complexity and possibilities were greater than expected so the research periods lasted longer, and the development process weren't as easy as we thought it would be
2. We delayed the user tests to finish the Single sign-on (SSO)-library. Due to this we ended up with only one iteration of tests.

5.5.2 Time sheet evaluation

At the start of the project we decided on a core time when everyone should be here of 09:00 - 14:00. We didn't manage to consistently meet this agreement, often due to other responsibilities we had. But on a whole we don't feel it affected the results negatively as we reached our goal of hours spent on the project, see the project handbook for details.

5.5.3 Scrum

The use of Scrum was on a whole a great benefit to the project and our ability to answer the research questions. It enabled flexibility in changing and adding new issues as they arose during research and development. The retrospectives also helped in building the team, which led to greater cooperation and progress and the solving of some inter-team problems(detailed in 4.2.1.3). Due to our lack of knowledge about many of the subjects affecting the research questions it wasn't reasonable to expect we would be able to predict the needed time required to find the answers, as evidenced by the Gantt-diagram in figure 4.20 and the varying degrees of sprint completions as described here 4.2.1.2. Another reason for the sprints not getting completed is that we wanted to constantly push ourselves, and rather add too many issues than too few. They were prioritized on GitHub so the bottom issues in each sprint were not must-do.

See section F.1 for a reflection on our use of the Scrum retrospective.

5.5.4 Room configuration

Our thoughts on the room configuration can be found in section F.4.

5.5.5 Development and research methods

See F.3 for our thoughts about how the methods used helped our project.

5.5.6 User testing method and process

Many of the problems mentioned in the user test results (paragraph 4.1.1.1.3), were likely caused by our very limited experience with user testing. The tests could also have benefited from more preparation. Several test subjects reported that the scenarios were not very realistic, especially the doctors. This focus from the testers on content, might have made the small differences in navigation harder to notice. Ideally, we should maybe have tested a system that was already known to the subjects so the only differences were what we actually wanted to test.

On the other hand, if we were to modify an already existing system, the testers could automatically be more skeptical to changes as opposed to the navigation they are used to. This pre-existing familiarity could have introduced bias into our results.

It would be interesting to see if the test subjects noticed the changes in navigation if the test apps had less distracting problems. If we could do this again, we would have ran an initial small round of tests to make sure our app content was realistic enough, and that users weren't confused with the signature button. We would also have included more test subjects to get more statistically significant results.

If we are going to time tests for usability again, we would probably ban the testers from speaking, so their comments don't impact the time it takes for them to do the tasks. We also had a disproportional amount of iOS users start with one app first, and the opposite for Android. If the experiment was conducted with a larger sample size this would probably have been evened out.

We believe all the testing issues were caused by a lack of time and our inexperience, not faults in the test methods.

5.5.6.1 Did we meet the requirements for comparative tests?

The book Practical user testing[7] outlines the following requirements for comparative tests:

- All tasks must be solvable in both systems
- A and B must be equally finished
- Testdata (shown to the user) must be identical
- All users must test both solutions
- Every other user must start with A, then B

All of these are fulfilled. The tests on iOS and Android were identical, as shown in section K.4. In our methodology chapter, 3.1.2 we outlined how all

the users would test both solutions, and start with every other solution. This is documented as done in our results found in paragraph 4.1.1.1.3, and in our complete user test data found at section L.4.

5.5.7 Developer user testing

We initially set a goal for testing our library on developer but decided not to, see F.2 for explanations.

5.5.8 Evaluation of our use of DSR

Initially we spent 3 weeks looking at existing SSO solutions and trying to discover if our research was relevant or not. This also involved learning about the technologies we would be implementing through structured collection of information. The information collection can be described as structured because we created two big thematically organized documents at around 50 pages each, containing our findings. Several smaller documents were also made. These documents can be found as attachments on NTNU Open.

Through our research into the environment we discovered that the sign-in part of Single sign-on was already solved through the open source AppAuth[29] library. Still, our employer wanted us to find out what works best for their context and use-case. That meant looking into different OAuth2.0 and OpenID Connect flows to find the best fit for them. This was done by rigorously following the OAuth2.0 and OpenID Connect specifications to create our own client. Actually implementing the specifications made us very comfortable with and knowledgeable about the protocols. Our sign-in client was created stepwise through several sprints.

When we had a working prototype that could log in to the DIPS server, we evaluated its expandability (Rellevance cycle), and discovered that we had classes with a complicated inheritance structure. We decided to re-evaluate our code design and developer API and do another design iteration, as mentioned in 4.2.7.0.1. This was our most significant design iteration, although many small iterations were made in other parts of the system.

While we have attempted in our paper to answer general questions, the scenarios we have chosen to implement and test are grounded in the needs of our task giver DIPS. This makes our solutions more relevant to DIPS, but possibly less generalizable. A problem with generalizing our usability research was the content of our apps. How can we make tasks that are general to everyone? DIPS is more interested in a hospital context, which is why we chose tasks based on suggestions from one of DIPS's consultants who used to be a nurse. This introduced a conflict of interests between making our research relevant for DIPS, and making it less relevant for DIPS, and more relevant for app development in general.

We hope that we have generated some useful knowledge that can be fed back to the knowledge base through our results and methodology evaluation.

5.5.9 Professional ethics considerations and team reflections

Professional ethics in computer science encompasses many subjects and problems, not the least conflict management within in the workplace. We have considered two of the more relevant ones from The Software Engineering Code of Ethics and Professional Practice[30].

1. Due to our inexperience and the relatively short time period we had to complete this, we can't guarantee that we have discovered all possible weaknesses in the system.
2. We also feel we haven't had the capacity or time to properly test, review or debug the software created in accordance with the seriousness of the context this can be used in.

We all feel handling of the conflicts that arose within the team were dealt with satisfactorily by discussions, especially during retrospectives, only within the team. There was no need for intervention from anyone outside the team. As a whole the teamwork has worked well. Through the many discussions of the relevant subjects throughout the project we all feel we have grown as software engineers.

5.5.10 System perspective

What we have made can be used as a guidance to what one should or shouldn't do when DIPS or someone else considers making a releasable SSO-system for mobile devices. It also stands as a proof of concept of having a SSO-system that is integrated with the DIPS authorization server.

5.5.10.1 Societal context

The results in this paper might add some weight in DIPS' decision to put more resources into mobile development. The potential mobile system might be a benefit to the hospitals and their employees, and in that way be of some small benefit to the patients.

5.5.10.2 Economic context

The reason for DIPS to give us this assignment was to create a secure SSO solution for mobile without any dependencies. By doing this possible DIPS would make developing multiple apps possible. This again will save them a lot of money as it is easier to develop and maintain.

5.5.10.3 Ethical implications

Developing code that could end up in a hospital setting has major ethical implications. To give a worst case scenario: Say someone is able to exploit a vulnerability in our SSO library to steal large amounts of patient data. Then they publish the patient data freely on the internet for anyone to search. Say a wife finds out that her husband has a sexually transmittable disease. Yet her husband was a virgin when they got together in high school. She does not have any STIs, so how did her husband get one? The logical conclusion is that the husband has most likely been unfaithful. This could lead to a divorce, having a negative impact on their children. In the end, the husband may take his own life after losing his children in a lengthy custody battle.

With this in mind, we should have tested the code more thoroughly, and had an independent security evaluation be done. However, our SSO library does not claim to be a software product ready for production in a high stakes environment. It is a working prototype for a hospital context. DIPS will not use our SSO library without heavy modifications, tests and external scrutiny.

5.5.11 The length of this report

The task we received from DIPS AS is twofold. It asked us to research both cross-platform SSO and inter-app navigation. Each of these would be enough to fill a standard NTNU bachelor's thesis of 20-40 pages. The requirement of always having to answer each question for both iOS and Android also contributed to a longer report size.

Chapter 6

Conclusion and further work

6.1 Engineering artifacts conclusion

As detailed in discussion 5.4, we met most of the requirements for a well made SSO library. We have a functioning OpenID Connect client supporting the safest solution for mobile, Hybrid Flow with PKCE. All the functionality dealing with token storage and sharing is securely implemented.

The result mostly only differs in degree of convenience for the developer implementing our library, and not in the overall planned functionality. We planned to automatically prompt the end-user to re-authenticate if the tokens are expired. This was not included since we left GUI creation to developers and since it would require saving the state of the app using our library.

6.2 Research questions conclusion

6.2.1 From a user perspective, do several securely linked mobile applications function as well as one big mobile application?

Only 2/13 user noticed a difference between using one app big app and using three smaller apps to solve the same tasks. They evaluated each solution independently using the System Usability Scale. We had 6 test subjects on Android, and 5 for iOS.

Our research indicates that there is no significant difference between using one big app, or three linked apps for users on Android. On iOS we consider the three apps on iOS to be slightly less usable than the big app, but not to the degree shown in the average or median SUS scores. With such a small sample

size, platform specific results are not very reliable and need further verification. When combining both operating systems however, the sample size is big enough to make a conclusion. Most of the subject did not notice a difference. However the big app is the preferred solution when using the SUS as a metric, scoring 4.6 points higher than the three small apps.

We have only shown our own inter-app navigation solution to be usable. This means that we only have managed to answer the research question for systems with similar a similar navigation solution to ours. To mitigate this we have attempted to follow Android and iOS design guidelines as much as possible.

6.2.2 Are several securely linked mobile applications as secure as one big secure mobile application?

On Android this is a definitive yes if the apps are protected by app signing. On iOS applications using the recommended universal links are secure, while applications using custom URL schemes are vulnerable to spoofing.

We did not find evidence of our SSO solution introducing any security challenges. It is built using secure APIs provided by the operating systems, so eventual security flaws will come from exploiting unknown vulnerabilities in our API.

6.2.3 What is a secure well made cross-platform implementation of SSO for iOS and Android?

We defined the requirements for the library to be secure, and well made, in subsection 1.2.1.3. Following our engineering artifacts, literature study and discussion, these conclusions can be made:

6.2.3.1 Authentication and Authorization

The OpenID Connect Hybrid Flow with PKCE and the browser as the user agent provides the most secure way of doing authentication and authorization in native Android and iOS apps, like the ones built using Xamarin.Forms.

6.2.3.2 Android

- Secure storage of tokens on Android is impossible without tamper-resistant hardware. Tokens need to be encrypted instead, preferably using the Android Keystore System, and stored in internal app storage.
- Sharing of tokens between apps on Android should be done with Content-Providers restricting access to certain apps using app signing.

6.2.3.3 iOS

- Secure storage on iOS should be done with the keychain. This data cannot be extracted since it is in secure hardware.
- Secure sharing of tokens on iOS should be done with a shared keychain group.

6.3 Further work

6.3.1 SSO-library

We found no way to satisfyingly authenticate the OAuth2.0 client to the authorization server, as it is considered a public client. It would require further research into possible solutions to be able to ensure the authorization server that it is communicating with a valid client. The library currently has no test coverage. This should be added before putting it to use or developing it further.

Android

It might be possible to create a more developer-friendly version of the SSO library by using **Services** on Android. This could provide a authentication and authorization flow that don't require the developer using the library to handle the response from the authorization request. It might then be possible for the browser to open a service, have the service parse out the code, and have the service then open the correct app[31]. We recommend further work in that direction.

6.3.2 Inter-app navigation

Any further testing should be done on a larger number of test subjects than we did to gain more statistically significant results.

We also only tested one possible solution to inter-app navigation. Further testing could include user tests only using the default inter-app navigation functionality provided by Android and iOS when the user is redirected between apps.

Further user tests could also use other types of navigation such as flat navigation or content-driven navigation.

Bibliography

- [1] Android Open Source Project, “Android enterprise security white paper”, Sep. 2018.
- [2] —, (). Android security, [Online]. Available: <https://source.android.com/security/app-sandbox>. (accessed: 01.05.2019).
- [3] A. O. S. Project. (). System and kernel security: Rooting of devices, [Online]. Available: <https://source.android.com/security/overview/kernel-security#rooting-devices>. (accessed: 03.05.2019).
- [4] G. Kasagiannis, “Security evaluation of android keystore”, Master’s thesis, University of Piraeus, Feb. 2018.
- [5] “. S. Guide”. (), [Online]. Available: https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf. (accessed: 13.05.2019).
- [6] E. M. Toni Vucic Truls Torgersen, *Vision document*, Apr. 9, 2019.
- [7] E. Toftøy-Andersen and J. G. Wold, *Praktisk brukertesting*, 1st ed. Cappele Damm AS, 2011, ISBN: 9788202343507.
- [8] T. S. Tullis and J. N. Stetson, “A comparison of questionnaires for assessing website usability”, Jun. 2006. DOI: https://www.researchgate.net/publication/228609327_A_Comparison_of_Questionnaires_for_Assessing_Website_Usability.
- [9] U.S. Department of Health Human Services. (). System usability scale (sus), [Online]. Available: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>. (accessed: 13.05.2019).
- [10] J. Brooke, “Sus: A retrospective”, *Journal of Usability Studies*, vol. 8, 2 Feb. 2013. DOI: http://uxpajournal.org/wp-content/uploads/sites/8/pdf/JUS_Brooke_February_2013.pdf.
- [11] J. Sauro, *A Practical Guide to the System Usability Scale: Background, Benchmarks Best Practices*. Measuring Usability LLC, 2011. DOI: <https://www.amazon.com/gp/product/1461062705/>.
- [12] Jeff Sauro, PhD. (Sep. 19, 2018). 5 ways to interpret a sus score, [Online]. Available: <https://measuringu.com/interpret-sus-score/>. (accessed: 19.05.2019).

- [13] —, (Jul. 10, 2018). Interpreting single items from the sus, [Online]. Available: <https://measuringu.com/sus-items/>. (accessed: 19.05.2019).
- [14] A. Bangor, P. T.Kortum, and J. T. Miller, “An empirical evaluation of the system usability scale”, *International Journal of Human-Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008. DOI: 10.1080/10447310802205776. eprint: <https://doi.org/10.1080/10447310802205776>. [Online]. Available: <https://doi.org/10.1080/10447310802205776>.
- [15] developer.apple.com. (). Human interface guidelines: Navigation bars, [Online]. Available: <https://developer.apple.com/design/human-interface-guidelines/ios/bars/navigation-bars/>. (accessed: 17.05.2019).
- [16] —, (). Human interface guidelines: Modality, [Online]. Available: <https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/modality/>. (accessed: 13.05.2019).
- [17] A. R. Hevner, “A three cycle view of design science research”, *Scandinavian Journal of Information Systems*, vol. 19, no. 4, 2 2007. DOI: <http://aisel.aisnet.org/sjis/vol19/iss2/4>.
- [18] H. A. Simon, *The Sciences of the Artificial*. MIT Press, 1996, ISBN: 0262193744.
- [19] E. D. Hardt. (). Rfc 6749: The oauth 2.0 authorization framework, [Online]. Available: <https://tools.ietf.org/html/rfc6749>. (accessed: 30.04.2019).
- [20] OpenID Foundation. (). Openid connect core spec, [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html. (accessed: 30.04.2019).
- [21] Microsoft. (). Dependency service in xamarin.forms., [Online]. Available: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction>. (accessed: 15.05.2019).
- [22] —, (). Xamarin.essentials securestorage., [Online]. Available: <https://github.com/xamarin/Essentials/tree/master/Xamarin.Essentials/SecureStorage>. (accessed: 15.05.2019).
- [23] Jeff Sauro, PhD. (Aug. 30, 2016). System usability scale, [Online]. Available: <https://measuringu.com/sus/>. (accessed: 13.05.2019).
- [24] Google. (). Understanding navigation: Reverse chronological navigation, [Online]. Available: <https://material.io/design/navigation/understanding-navigation.html#reverse-navigation>. (accessed: 17.05.2019).
- [25] developer.apple.com. (). Human interface guidelines: Navigation, [Online]. Available: <https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/navigation/>. (accessed: 17.05.2019).
- [26] “. Documentation”. (), [Online]. Available: https://developer.apple.com/documentation/uikit/core_app/allowing_apps_and_websites_to_link_to_your_content/defining_a_custom_url_scheme_for_your_app. (accessed: 13.05.2019).

- [27] Google Developers. (). Android app signing, [Online]. Available: <https://source.android.com/security/apksigning>. (accessed: 16.05.2019).
- [28] —, (). `IService`, [Online]. Available: <https://developer.android.com/guide/topics/manifest/service-element>. (accessed: 15.05.2019).
- [29] 25+ contributors on GitHub. (). Appauth, [Online]. Available: <https://appauth.io/>. (accessed: 18.05.2019).
- [30] ACM/IEEE-CS. (). The software engineering code of ethics and professional practice, [Online]. Available: <https://ethics.acm.org/code-of-ethics/software-engineering-code/>. (accessed: 17.05.2019).
- [31] T. Castelijns and d-man. (). Android start activity from service, [Online]. Available: <https://stackoverflow.com/questions/3606596/android-start-activity-from-service>. (accessed: 15.05.2019).
- [32] K. Finstad, “The system usability scale and non-native english speakers”, *Journal of Usability Studies*, vol. 1, no. 4, 4 Aug. 2006. DOI: http://uxpajournal.org/wp-content/uploads/sites/8/pdf/JUS_Finstad_Aug2006.pdf.
- [33] Jeff Sauro, PhD. (Aug. 30, 2016). Can you change a standardized questionnaire?, [Online]. Available: <https://measuringu.com/change-standardized/>. (accessed: 13.05.2019).
- [34] Agile Learning Labsh. (2010). The agile dictionary: Spike, [Online]. Available: <http://agiledictionary.com/209/spike/>. (accessed: 09.05.2019).
- [35] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, “Strengthening the case for pair programming”, *IEEE SOFTWARE*, 2000. DOI: <https://collaboration.csc.ncsu.edu/laurie/Papers/ieeeSoftware.PDF>.
- [36] W. ZUILL, “Mob programming – a whole team approach”, *www.agilealliance.org/*, 2016.
- [37] O. K. Aune, C. H.-P. K. Echtermeyer, and E. B. Sørensen, “Mob programming: A qualitative study from the perspective of a development team”, bathesis, Norwegian University of Science and Technology, Trondheim, Norway, 2018.
- [38] R. Andersson, “Pros and cons with mob programming – A Case Study”, bathesis, Högskolan Dalarna, Falun, Sweden, 2017.
- [39] Annie Murphy Paul. (). The protégé effect, [Online]. Available: <https://www.psychologytoday.com/intl/blog/how-be-brilliant/201206/the-prot-g-effect>. (accessed: 03.05.2019).
- [40] Agile Alliance. (). Glossary: Pair programming, [Online]. Available: <https://www.agilealliance.org/glossary/pairing>. (accessed: 11.05.2019).
- [41] Microsoft. (). What is xamarin.forms., [Online]. Available: <https://docs.microsoft.com/en-us/xamarin/get-started/>. (accessed: 15.05.2019).

- [42] —, (). What is nuget?, [Online]. Available: <https://www.nuget.org/>. (accessed: 16.05.2019).
- [43] T. Spring. (Jun. 2018). Malicious docker containers earn cryptomining criminals \$90k, [Online]. Available: <https://threatpost.com/malicious-docker-containers-earn-crypto-miners-90000/132816/>. (accessed: 16.05.2019).
- [44] T. Claburn. (Nov. 2018). Check your repos... crypto-coin-stealing code sneaks into fairly popular npm lib (2m downloads per week), [Online]. Available: https://www.theregister.co.uk/2018/11/26/npm_repo_bitcoin_stealer/. (accessed: 16.05.2019).
- [45] Microsoft. (). What is nuget?, [Online]. Available: <https://www.ehackingnews.com/2013/01/rubygemsorg-hacked-via-yaml-parsing.html>. (accessed: 16.05.2019).
- [46] M. Kumar. (Jan. 2019). Someone hacked php pear site and replaced the official package manager, [Online]. Available: <https://thehackernews.com/2019/01/php-pear-hacked.html>. (accessed: 16.05.2019).
- [47] A. Parecki. (). Oauth 2.0, [Online]. Available: <https://oauth.net/2/>. (accessed: 30.04.2019).
- [48] OpenID Foundation. (). Openid connect, [Online]. Available: <https://openid.net/connect/>. (accessed: 30.04.2019).
- [49] Google Developers. (). Fingerprintmanager, [Online]. Available: <https://developer.android.com/reference/android/hardware/fingerprint/FingerprintManager.html>. (accessed: 03.05.2019).
- [50] —, (). Biometricprompt, [Online]. Available: <https://developer.android.com/reference/android/hardware/biometrics/BiometricPrompt.html>. (accessed: 03.05.2019).
- [51] J. Thornsby. (). How to add fingerprint authentication to your android app, [Online]. Available: <https://www.androidauthority.com/how-to-add-fingerprint-authentication-to-your-android-app-747304/>. (accessed: 03.05.2019).
- [52] Google Developers. (). Biometricprompt.builder.setallowdevicecredential(), [Online]. Available: [https://developer.android.com/reference/android/hardware/biometrics/BiometricPrompt.Builder.html#setAllowDeviceCredential\(boolean\)](https://developer.android.com/reference/android/hardware/biometrics/BiometricPrompt.Builder.html#setAllowDeviceCredential(boolean)). (accessed: 03.05.2019).
- [53] —, (Oct. 2017). Oauth 2.0 for native apps: Oauth implicit grant authorization flow, [Online]. Available: <https://tools.ietf.org/html/rfc8252>. (accessed: 16.05.2019).
- [54] —, (). Save files to private directory, [Online]. Available: <https://developer.android.com/training/data-storage/files.html#PrivateFiles>. (accessed: 03.05.2019).

- [55] —, (). Save files to public directory, [Online]. Available: <https://developer.android.com/training/data-storage/files.html#PublicFiles>. (accessed: 03.05.2019).
- [56] Android Open Source Project. (). Android security, [Online]. Available: <https://source.android.com/security/trusty>. (accessed: 01.05.2019).
- [57] Google Developers. (). Android keystore, [Online]. Available: <https://developer.android.com/training/articles/keystore>. (accessed: 03.05.2019).
- [58] —, (). Android keystore, [Online]. Available: <https://developer.android.com/training/articles/keystore>. (accessed: 06.05.2019).
- [59] —, (). Keygenparameterspec.builder.setunlockeddevicerequired(), [Online]. Available: [https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setUnlockedDeviceRequired\(boolean\)](https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec.Builder#setUnlockedDeviceRequired(boolean)). (accessed: 16.05.2019).
- [60] (), [Online]. Available: https://developer.apple.com/documentation/security/keychain_services/keychain_items/sharing_access_to_keychain_items_among_a_collection_of_apps.
- [61] ". Documentation". (), [Online]. Available: https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_security_application-groups. (accessed: 13.05.2019).
- [62] —, (), [Online]. Available: https://developer.apple.com/documentation/uikit/core_app/allowing_apps_and_websites_to_link_to_your_content. (accessed: 13.05.2019).
- [63] ". Developers". (), [Online]. Available: <https://developer.android.com/guide/topics/manifest/permission-element>. (accessed: 14.05.2019).
- [64] —, (), [Online]. Available: <https://developer.android.com/reference/android/support/v4/content/FileProvider>. (accessed: 14.05.2019).
- [65] Google Developers. (). Android manifest:shareduserid, [Online]. Available: <https://developer.android.com/guide/topics/manifest/manifest-element#uid>. (accessed: 03.05.2019).
- [66] IBM Knowledge Center. (). Enabling the simple data sharing feature for android native applications, [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSHS8R_7.0.0/com.ibm.worklight.dev.doc/devref/t_enabling_simple_data_sharing_native_Android.html. (accessed: 03.05.2019).
- [67] D. Smith. (Feb. 17, 2016). Two key things to check before publishing your android app, [Online]. Available: <https://medium.com/developing-intent/android-developer-important-manifest-checks-8c86064013aa>. (accessed: 03.05.2019).
- [68] da...@gmail.com. (Nov. 8, 2008). Shareduserid cause upgrade issue - rendering data directory no longer accessible, [Online]. Available: <https://issuetracker.google.com/issues/36905922>. (accessed: 03.05.2019).

- [69] ". Documentation". (), [Online]. Available: https://developer.apple.com/documentation/uikit/core_app/allowing_apps_and_websites_to_link_to_your_content/handling_universal_links. (accessed: 13.05.2019).
- [70] —, (), [Online]. Available: https://developer.apple.com/documentation/uikit/core_app/allowing_apps_and_websites_to_link_to_your_content/enabling_universal_links. (accessed: 13.05.2019).
- [71] Google Developers. (). Intent:setflags, [Online]. Available: [https://developer.android.com/reference/android/content/Intent.html#setFlags\(int\)](https://developer.android.com/reference/android/content/Intent.html#setFlags(int)). (accessed: 15.05.2019).
- [72] Google Developer Training. (). Understanding activities and intents: Starting an activity with an explicit intent, [Online]. Available: https://google-developer-training.gitbooks.io/android-developer-fundamentals-course-concepts/content/en/Unit%5C%201/21_c_understanding_activities_and_intents.html. (accessed: 20.05.2019).
- [73] Google Developers. (). Startactivityforresult, [Online]. Available: <https://developer.android.com/training/basics/intents/result>. (accessed: 03.05.2019).

Appendices

Appendix A

Thesis assignment

Security and usability of SSO and inter-app navigation on iOS and Android

This was originally in Norwegian, but the translation was approved by our project supervisor at DIPS Trondheim.

Purpose of the assignment: There are good frameworks today for cross-platform development of mobile applications. Mobile phones are becoming more and more like fully fledged computers, which facilitates bigger and more complex applications like a patient journal system for hospitals. All new iOS devices and some Android devices support biometric authentication using fingerprints or face recognition.

Big applications have several drawbacks, in the form of increased complexity in architecture and design. This thesis shall examine whether several small applications can do as good of a job as one big application, and find an efficient way to implement single sign-on (SSO) on both iOS and Android using a shared code base for each app. The authentication shall support Face ID and Touch ID on iOS, and corresponding mechanisms on a modern Android-device.

Short description of the assignment suggestion: The students should use the framework Xamarin Forms to make several mock-applications to support a clinical documentation-task. When the user is authenticated in one app, she should be authenticated in the others as well. The apps should support deep-linking between each other so that the navigation is experienced as efficient. The user should get authorization per scenario, and therefore also be remembered across the apps.

OAuth2 should be the authentication mechanism. Tokens should be stored in Secure Enclave on iOS, and a correspondingly safe place on Android.

The same code base should be used on iOS and Android.

The solution shall be tested on real users to see if a multi-app solution can be as user friendly as one big app.

The students should also look at the differences between Android-devices, and consider safety and API-levels.

Complementary comments on what the thesis is about: DIPS AS is Norway's biggest supplier of e-health solutions for norwegian hospitals, with over 100 000 clinicians and administrative staff using our systems every day.

Today there is 1 app for nurses and doctors on iOS, that is being rolled out to Nordland Hospital. In the future we will create more apps, also for Android, and wish to evaluate strengths and weaknesses with 1 app, as opposed to several smaller apps. The technical and usability related questions are something we need to know before we choose strategy for further app development.

The students will receive knowledge of modern development frameworks, safety standards, and our development and user testing methodology.

Appendix B

User testing methodology

This section covers a few important topics about our user testing process:

- Our reasoning for making a change to the SUS.
- Changes made to our tests due to issues discovered in our pilot tests.
- A descriptions of anomalies that occurred during some of the user tests.
- Details of how we handled processing and storage of the sensitive data collected during testing.
- Notes about the use of a screen recorder vs camera.
- Details about the compensation the test subjects received.

See this section: 3.1, for the main topics of our user testing methodology and process.

B.1 User tests changes and rewards

B.1.1 System Usability Scale changes

We thought that the first question on the System Usability Scale, "I think that I would like to use this system frequently" did not apply to our applications. Most of the testers would be non-doctors, so it would not make sense for them to answer positively on this question. In stead we changed the question to: "The system felt fast and responsive."

A study has shown that non-native English speakers understand everything in SUS except for the word 'cumbersome'. The study recommends changing the word cumbersome to awkward but does not evaluate if this changes the accuracy of the SUS scale.[32] We used the word cumbersome in the first pilot test, but decided to change it to awkward since the tester asked what it meant, B.1.2.

Can you change the System Usability Scale?

Jeff Sauro, PhD in Educational Statistics and Research Methods claims that small changes in wording to a SUS don't necessarily alter its psychometric properties.[33] He points out that the first question on the SUS might not be relevant in many situations. He suggests some ways to check if the psychometric properties have been changed[33], but we did not take the time to verify that. We accept that changing the first question has made our System Usability Scale score somewhat less comparable to other systems, but accept this as our use of the SUS is to compare the two systems directly (using our modified SUS).

B.1.2 Changes caused by pilot tests

Two pilot tests were done to test the testers, test plan, questionnaire and apps. The first pilot test was done by Toni and Erling on Truls. As a result of observations and feedback from this pilot test we decided to do the following.

- Change the word "cumbersome" to awkward in the SUS.

The second pilot test was done on a person we knew with limited prior knowledge of the test apps. As a result of observations and feedback from the pilot test we decided to do the following.

- Remove the star (favorite) symbol from Lab Results. The button had no implemented functionality.
- Increase of contrast for the "Tap to sign" button on iOS
- Move the one non-doctor pre-test interview question to the final questionnaire.

B.1.3 Details about anomalies that occurred during our user testing

During the first test these errors were noted.

- The test leader was interrupted by the test subject and the observer, and was not able to say the 2nd part of the introduction.

During the second test these anomalies were noted:

- Truls stopped Erling from saying to the test subject that we were testing navigation. The latter had been specified in the test plan manuscript. The test continued normally.
- The test person was not asked if they used an Android or iOS device, leading to a change in operating system right before the test was to start.

After test 2 it was decided that the test plan manuscript had an error. The test introduction manuscript was changed to reflect that we do not inform the test subjects of the purpose of the test before the test. After this the tests continued normally.

There were several small cases of deviating from script during the interviews. They are documented in each test report, found at L.3.

The reasoning for many of the smaller deviances from script was that the formality of the tests sometimes led to an awkward mood. To facilitate a more natural conversation during the interview, the wording of some questions was changed during the tests. The wording was changed to better fit the rest of the conversation, mostly building on information provided in the previous answers.

B.1.4 Data processing and storage

Prior to the test beginning, test subjects were presented with a combined consent form and non-disclosure agreement based on the one found in "Practical user testing"[7]. It can be found in the consent-form: L.2.

With approval from our university advisor where there was doubt about legality we decided to collect the following data about the tests and test persons:

- Written non-specific demographic information
- Oral interview recorded on a Zoom H1 Handy Recorder.
 - Transcribed and deleted within 24 hours of the recording taking place.
 - Dialect re-written into standard norwegian Bokmål.
 - Recorder kept in locked office.
 - Recording never leaving the physical recorder.
 - Zoom H1 memory card to be destroyed upon project completion.
- Anonymous SUS score.
- Completely anonymous videos recorded by a screen recorder on the Android device we used for testing. Reasoning for this found at 3.1.4
- Video (without audio) of hands and screen for all the iOS tests. Reasoning for this found at 3.1.4

We tried to avoid dealing with personal information. While all of our other data cannot be linked to an individual (we don't have a key linking people to their user test results) the audio recordings are defined as personal information because it is possible to identify the speaker by their voice. For this reason we proposed a solution to our university advisor to only use audio recordings in order to accurately transcribe them immediately afterwards before deleting

them.

With these solutions we believe we are not storing any personal information and therefore are not obliged to report our research to Norwegian Centre for Research Data(NSD).

Notes about screen recording

For Android, the screen recorder was used. This was done to maximize anonymity for the test subjects. On iOS this was also wanted, but could not be done as the recording blocked the back to app button. Instead, video recording of the screen and the hands was done. The anonymity of including hands in the recording is questionable.

One unintended upside for recording both hands and screen was that it was possible to say when the test actually started. The internal screen recorder made it only possible to tell when the screen first was used.

B.1.5 Rewards

The book Practical user testing states that to avoid getting the testers "on your side" you should keep refreshments and food to a minimum. Any gift cards should be given to the user beforehand. This should be done so that the person getting tested has no motivation to make the tester happy in order to get a reward afterwards.[7] Although, we did not provide food or drinks for our test subjects, we gave the test subject a 'reward' of a chocolate bar, before the test began.

Appendix C

Engineering methodology

Choices of methods used during the engineering process, with explanations of how they work and why we thought them beneficial to use. See 3.2 for process methodology in main report.

C.1 Process methodology

C.1.1 Scrum and agile development

We decided to use Scrum as a development process as it lends itself well to changes in directions and specifications. This felt beneficial to a project involving so many unfamiliar topics. We acknowledged that we simply did not know the complexity or scope of the tasks we set out to do. So setting stringent and static requirements at that point made little sense.

We decided that the best idea would be a continuously updated backlog. As we learned more about our task, we would add new backlog items. To get the backlog items done we would work in two week sprints, with a central goal for each. We planned to have a retrospective and a review at the end of each sprint. Each new sprint should start with a planning session where experiences from earlier sprints would be taken into consideration.

C.1.1.1 Spiking

”A task aimed at answering a question or gathering information, rather than at producing shippable product.” [34]

Spiking is used to resolve questions relating to a particular technical question or a design problem. We chose this methodology because we did not feel confident in starting work on our final product at once. We did not know the programming language, nor the development platforms or protocols to be implemented, so we knew from experience that big mistakes would be made. We also knew

that any code design choices we made early on were bound to be challenged again once we acquired more knowledge. Therefore we focused on learning as much as possible as fast as possible.

In general development would work like this:

1. Take an issue from the sprint backlog.
2. Research the issue with the goal of finding a viable way to solve it.
3. Systematically write down important information found during research in one of our thematically organized shared research documents.
4. Make or update a mock/test app implementing the chosen way to solve the issue
5. Manually test and evaluate the chosen solution
6. If the knowledge gained was particularly exciting or useful for the rest of the group a mini-seminar would be held for the rest of the group about the proposed solution.

C.1.1.2 Pair programming

Pair programming is an agile software development technique in which two programmers work together at one workstation. They jointly produce one artifact (design, algorithm, code). One person is the driver, operating the keyboard and mouse while the their partner continuously and actively observes the driver's work. The partner watches for defects, thinks of alternatives, looks up resources, and considers strategic implications. The partners should deliberately switch roles periodically. Both are equal, active participants in the process. and wholly share the ownership of the work product.[35]

The study[35] found that almost the only defects that made it through unit and functional testing were written by solo programmers, as opposed to pair programmers.[35]

"A side effect of pair programming is a high order of staff cross training which removes additional long-term costs from development." [35] Since Truls would be programming mostly for iOS, and Erling and Toni for Android, we believed setting up mixed teams between the two platforms would help in ensuring everybody understands as much as possible about the entire product we were creating.

Studies show that over 90% of programmers enjoyed pair programming more than programming alone.[35] Truls had previously pair programmed in two smaller projects with positive experienced from both. This matches what would be expected from the research and is another reason for why we chose this route.

C.1.1.3 Mob programming

When doing mob programming, the entire team (at least 3 people) all work on the same problem. They work together in the same space to solve the same task.[36]

The study[36] found that mob programming is well suited for solving complex tasks. It also found one of the main benefits to be the transfer of knowledge between team members.[37][38] As previously stated, we wanted every member of our team to know as much as possible about the product and the theory behind it. We also wanted high code quality code, and studies[38] have found that mob programming leads to fewer bugs.

C.1.2 Room configuration

Not much thought was put into the room configuration initially. The developers sat with everyone facing the walls in the room. See figure C.1 for illustration.

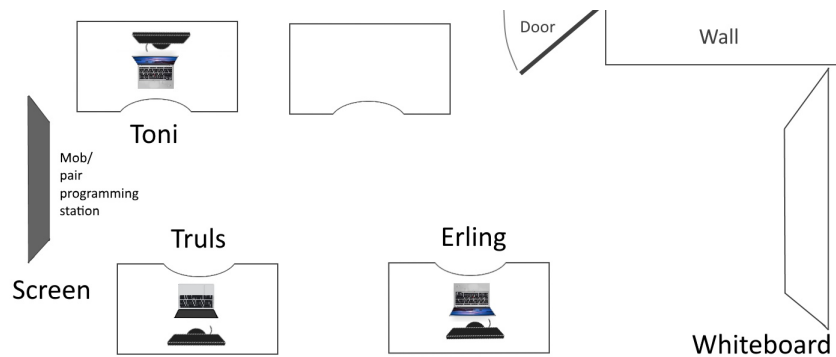


Figure C.1: Initial development room setup

Early in the project it was brought up in a sprint retrospective that communication between team members was a little hard and everybody felt a little isolated. This prompted us to devise our own setup which is illustrated below in C.2.

We moved the tables into two groups. In the middle of the room: one workstation with a group of three tables, and the members facing each other. On one side of the room: an TV station with a table near the TV for sessions when everybody was move over to this table and look at the TV together. On the other side of the room we had a whiteboard for sketching ideas and explaining things to each other either on the fly or in a mini seminar.

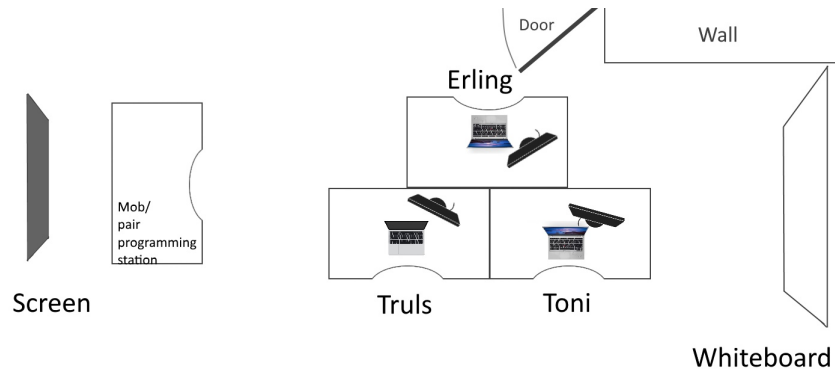


Figure C.2: Development room setup

C.1.3 Mini-seminar

Gathering everyone in the group to teach a specific subject. Makes it so that everyone is updated on the important topics and so that only one person needs to research a topic. This frees up time to the rest of the group, while still keeping everybody up to date.

A side effect of this is that the person holding the mini-seminar quickly finds out how well they know the subject when forced to explaining it and answer questions about the topic. Also it is easier to remember stuff that you have explained for others. This is called the protégé effect[39].

Appendix D

Engineering process results details

Adds additional details, insights and considerations to some of the subjects discussed in 4.2.

D.1 Retrospectives

We had a warm-up exercise before each retrospective. For example, skittles (a type of colored candy) were bought, and each skittle represented a question or command. The team members picked a skittle in turn and answered pre-decided questions based the color of the skittle. Questions were of the sort: "What is a long term goal you are working towards?" "Tell us something good that has happened recently." "Has anything bad happened recently?" "Tell a story where you felt embarrassed." This brought the team closer together.

D.1.1 Decisions and changes from retrospectives

Information regarding things that worked well and things that didn't work well were often brought up during retrospectives. It was beneficial in helping us determining what we should keep doing and what we should try to change. One retrospective brought some issues to light which prompted us to make some changes, figure D.2.

An interesting discussion was brought up during a sprint retrospective; we discovered an unintended consequence of choosing to make a library. As we developed with that as a goal we felt we put more thought into every part of the library and how it should be structured. The discussions and critical thinking which followed resulted in us improving as developers. If we only made proof of concepts we would have put less thought into the design and future proofing.



Figure D.1: Erling organising post-its after a sprint retrospective

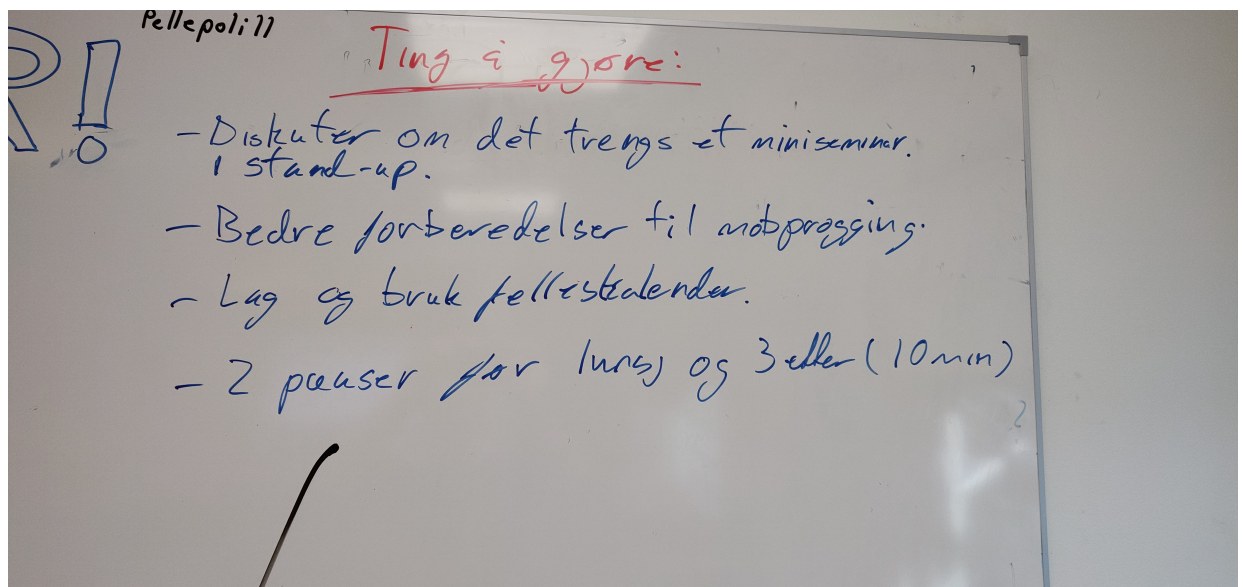


Figure D.2: Some goals we made due to issues brought up in a retrospective.

D.2 Pair-programming

The team members mentioned in the sprint retrospectives (D.3) and during development that they enjoyed pair programming, and that it seemed to improve productivity. Pair programming sessions were often done after an intense research period, which decreased the amount of interruptions to check documen-

tation.

The team members mentioned to each other that they enjoyed learning about each other's work through the pair programming sessions.

According to Agile Alliance, one should switch roles every few minutes or so.[40] The team did not do this. Instead one team member was the "driver" the whole session, maybe with some exceptions. One of the main reasons for this was that one of the team members was a Mac user, while the others were using Windows. If one of the team members were using the other OS, it felt really clumsy. Committing and pushing to git just to switch roles often, did not seem natural or worth it.

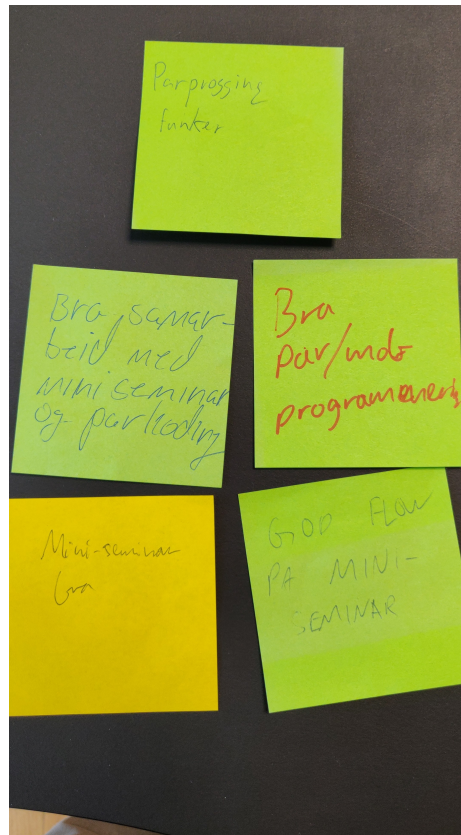


Figure D.3: Post-its mentioning how well pair-programming and mini-seminars worked.

D.3 Mob-programming

The reason for us not continuing this was that whenever the "driver" got stuck, everything stopped. The driver could not continue because he did not know how to do a specific task. Since all the team members had about the same amount of knowledge about the issues we were trying to solve, and the technologies chosen to solve them, nobody could help the driver continue. In stead, all the members sat down and started researching the issue, so that programming could continue.

This mob-research was experienced as inefficient. The reason was that having 3 people try to discover the answer to a question on the web, did not feel much more efficient than having one or two people do it. The developers would often read the same sources from Google, and arrive at a conclusion at approximately the same time. If the other team members did not find an answer before the first person who found one did, their time felt wasted.

D.4 Cooperation with project lead and supervisor

This sections details the meetings and interactions we had with both our project supervisor at DIPS, Runar Ovesen Hjerpbakk and our supervisor, Tomas Holt. See project handbook for minutes of meetings.

D.4.0.1 Runar Ovesen Hjerpbakk

The first meetings were more formal with a written meeting invitation, an written agenda and a referent, but the team soon realized that this wasn't the norm at DIPS. Most of the meetings were informal and spontaneous, using Slack to find a suitable time. The meeting still had talking points, but not a formal written agenda. There were no referent, but the team members took notes.

D.4.0.2 Supervisor

We had three formal meetings with our supervisor, including the kick-off meeting. All meetings had a written meeting invitation, an written agenda and a referent from the team. We also had a presentation of our research question. Our supervisor was there together with other supervisors and other teams. After the presentation we got feedback on our research question.

In the normal meetings we got answers to and had the opportunity to discuss important questions we had about the bachelor. Both how to write the bachelor thesis and how to work to get the best results.

Appendix E

Engineering artifacts results details

This section details the test apps we made for testing our SSO-library and an overview of its dependencies. Also, an example of how we received the URI from the browser after the authorization request. See 4.3.1 for a detailed explanation of how our SSO-library functions.

E.1 How to receive a fragment.

What follows are examples on how to receive the fragment of an URI when the end-user is redirected into your app from the browser.

For Android

The receiving app need to listen for an intent that matches the one generated by the browser when it receives the redirect URI, see: G.2.5.2.1. This is done by specifying an **IntentFilter** in an **Activity**, see: 2.4.2, with the correct DataScheme and DataHost.

For our testing application DIPSSLogin, DataScheme=com.dipssso.dipslogin and DataHost=callback. These must match the redirect URI registered on the server for that client. On our server this is com.dipssso.dipslogin://callback. When these match the browser will open the app. If several apps are set to handle the same URL scheme, the user will be prompted to choose which the browser should open.

Once your app has been opened, extract the fragment from the scheme and use the library function *TokensHandler.GetAuthorizationCodeFromFragment(string fragment)* to get an authorization code from the fragment.

For iOS

Specify a custom URL scheme in your app's Info.plist that the browser automatically opens upon receiving the response from the server. The custom URL scheme for the app must match that registered on the server for that client. When the app is opened by the operating system, a fragment can be extracted from the custom URL scheme that opened the app. The fragment contains the authorization code. This fragment can be passed to our library function *TokenHandler.GetAuthorizationCodeFromFragment(string fragment)* to extract the code from the fragment.

E.2 SSO-library dependencies

Our library has the following dependencies:

- **Newtonsoft.Json (12.0.1)**: For parsing HttpResponseMessage into C# objects.
- **System.IdentityModel.Tokens.Jwt (5.4.0)**: Used for parsing jwt tokens.
- **Xamarin.Essentials (1.1.0)**: For cross platform browser opening. Could be replaced by custom implementation without too much effort.
- **Xamarin.Forms (3.6.0.293080)**: For DependencyService

E.3 Test apps explanations and figures

An overview of the apps we made to continually test the SSO-library.

E.3.0.1 Login app (DIPSLogin)

This was used to test the whole login flow. When opened it immediately sends a authorization request using our library's **AuthorizationRequest**, as detailed in 4.3.1.1.1. The login page shown in figure E.1 is sent by the authentication server. After successful login the end-user is asked to authorize the client for access to the resources.

If the authentication and authorization is successful the browser redirects the end-user back to the app's main page. If the authorization and authentication were successful the received authorization code is displayed on the the bottom of the app. The four different buttons, fig E.2, tests the other functionality of our library.

- **Request tokens** does the token request to the authorization server. The tokens in the response is then sent to all the apps where they are separately encrypted and stored.

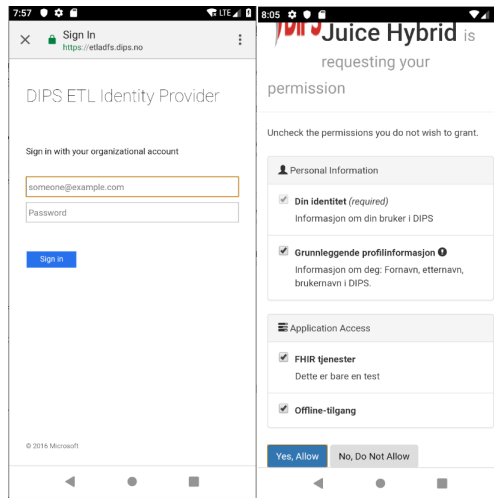


Figure E.1: The login page and authorization page.

- **Open other app** opens TestApp2 with an empty **intent**.
- **Delete all tokens** goes through every app's **Content Provider** and deletes all the stored tokens on the device.
- **Get access token** just retrieves the token from the app's storage, decrypts it and, if successful, displays the token as in fig E.2.

E.3.0.2 TestApp2

This app was used for testing successful sharing of an access token and general navigation between apps. It has two buttons as shown in fig E.3:

- **Get token** retrieves a shared token from the storage of this app, as detailed in 4.3.1. Used to quickly see if transfer was successful. If it was the access token is displayed as shown in fig E.3.
- **Go back to other app** just opens the previous app again.

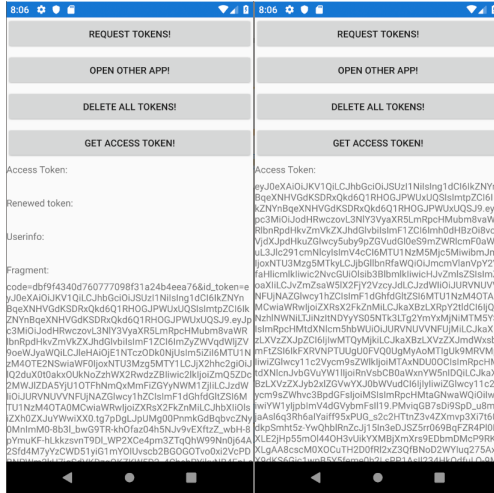


Figure E.2: The main page of the login test app with code and with access token.

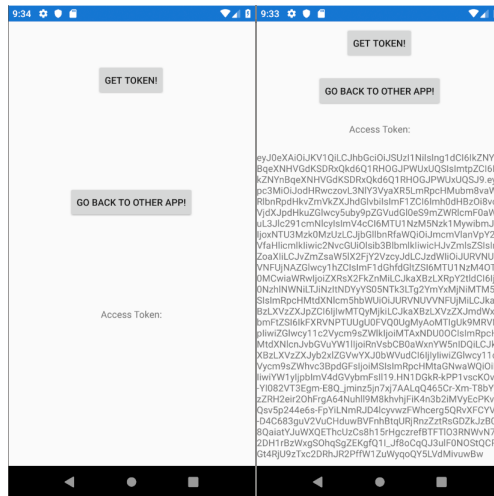


Figure E.3: The main page of TestApp2 with and without the access token.

Appendix F

Process evaluation details

An expansion of the process discussion in 5.5. Describes some additional thoughts we had about the use of some of the chosen research and development process.

F.1 Scrum

Our retrospectives often lasted over an hour, but it is difficult to say if every hour was well spent. Especially considering we always had a warmup exercises before each retrospective. What we can say is that we grew closer as friends. It allowed us to share how we felt about the current state of the project, and the group.

Since project manager Runar participated in the first two retrospectives, we also got to know him better. This helped ease the communication between the project manager and the team.

F.2 Developer user testing

We originally had plans for testing the SSO library on developers to see how easy it was to use. This was to verify that our SSO library was a good product. We ended up not doing it. The reason for this was that it didn't coincide with our earlier research questions. The earlier research questions asked us only to discuss the issues of SSO in a generalized sense, and not evaluate our library. For that reason, we decided we didn't have time to do it. Doing it would probably have made us better library programmers due to the feedback we would have received.

F.3 Research and development methods

Spiking was invaluable to us in reaching the project's current state. It enabled us to test and consequently verify or discard the potential solutions we discov-

ered during our research periods. It fit well with agile development, although few of the spikes could be directly integrated into our project. That was because the code was so hastily built and purpose-made for what we were testing.

Pair-programming proved a big benefit in helping us solve difficult problems and aid each other in learning new concepts. The collaborative nature of pair-programming also gave us a mental boost.

Mini-seminars were beneficial in giving all the team-members the same foundation of knowledge when discussing subjects and problems. It's difficult to say if they had any more direct impact on the results, but the relatively small time sacrifice was definitely worth the benefits, detailed in 4.2.5.

Mob-programming was stopped due to the reasons detailed here 4.2.4. It didn't have a positive impact on our progress.

F.4 Room-configuration

We brought up during several retrospectives that we liked the new room configuration. See Figure F.1. The mob/pair programming station worked well. The person not a part of pair programming could still be asked questions and was able to read the code on the TV screen and come with feedback. The TV screen was also used successfully for mini-seminars.

The location of the whiteboard was handy and it was used to write down agreed-upon solutions to be implemented. This provided a quick reference with answers to complex questions. It was also used to discuss solutions, and explain our problems and solutions to the our project supervisor at DIPS, Runar Ovesen Hjerpbakk.

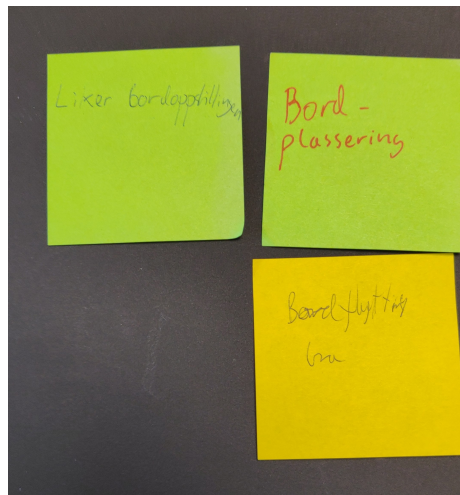


Figure F.1: Post-its mentioning how well the change of the room-configuration worked.

Appendix G

Available technologies

This chapter presents the technologies we and DIPS ASA have considered to help us answer these research questions:

- From a user perspective, do several securely linked mobile applications function as well as one big mobile application?
- What is a secure way to implement SSO across Android and iOS?
- What is a secure way to implement inter-app navigation across Android and iOS?

G.1 Chosen by DIPS

This section gives an introduction to the technologies and principles chosen by the oppgavestiller DIPS ASA.

G.1.1 Xamarin Forms

”Xamarin.Forms is a cross-platform UI toolkit that allows developers to efficiently create native user interface layouts that can be shared across iOS, Android, and Universal Windows Platform apps.”[41]

G.1.1.1 NuGet packages

NuGet is the package manager for .Net[42]. NuGet packages contain compiled code (DLLs), a descriptive manifest, and other files related to the code. Over 100,000 unique packages are hosted on nuget.org.

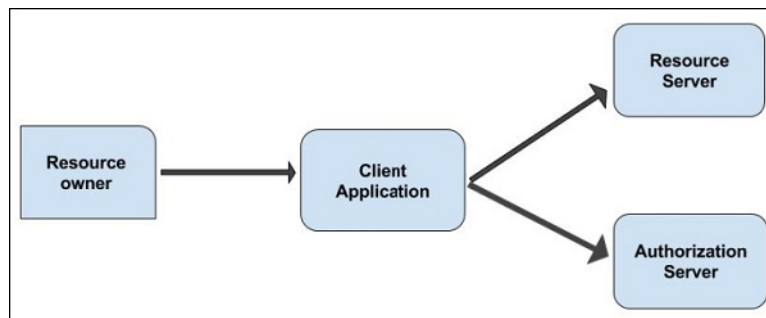
Using NuGet packages introduces a security risk. The packages themselves could contain malicious code[43], their repositories could be hacked[44], or the distribution service itself could be hacked[45], [46].

G.1.2 Authorization

OAuth 2.0 is the industry-standard protocol for authorization[47]. Replacing the earlier standard Security Assertion Markup Language (SAML) It enables a third-party application to obtain limited access to an HTTP service.[19]

G.1.2.1 OAuth 2.0 Glossary

- **Resource owner:** For example a person owning a picture stored on a server.
- **Client:** For example an app, or desktop program.
- **Resource Server:** For example a REST server containing user info or pictures.
- **Authorization Server:** Able to give access to the resource server. For example by providing an access token.
- **Third party** An external agent like Facebook, Twitter or Google. Or the application the user is using, in that case a "third party" client.



Source: <https://www.tutorialspoint.com/oauth2.0/images/roles.jpg>

Figure G.1: OAuth 2.0 roles

In the traditional client-server authentication model, the client application uses the resource owner's credentials (like username and password) to authenticate with the server. This means that the client application gets access to the user's username and password.

Another situation: Say a third party wants access to a resource owner's private files on some resource server. Before OAuth the resource owner would in some cases give the third party his username and password so that the third party could authenticate with the resource server and work with the resource owner's files. A real example was giving yahoo your email login details so it could fetch your contacts.

For security reasons it is a bad idea to give anyone the username and password combination.

OAuth and OAuth2.0 solve this issue by letting resource owners authorize clients and third parties without giving them the resource owner's password. Instead "clients obtain an access token – a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner." [19][Chapter 1: Introduction]

OAuth2.0 has the following flows for requesting and getting authorization:

- **Authorization Code Grant** Lets a confidential client request an authorization code. Then that code is exchanged for an access token and or a refresh token.
- **Implicit Grant** Lets a public client request an access token without having to get an authorization code first.
- **Resource Owner Password Credentials Grant** The resource owner password credentials grant type is suitable in cases where the resource owner has a trust relationship with the client, such as the client being the device operating system or a highly privileged application. [19][Chapter 4.3] An access token and or a refresh token is received immediately.
- **Client Credentials Grant** The client can request an access token using only its client credentials.
- **Extension Grants** See [19][Chapter 4.5]

G.1.2.2 Client profiles

OAuth 2.0 defines two different types of client profiles based on their ability to authenticate with the authorization server.

G.1.2.2.1 Public

Clients that are unable to keep client credentials confidential, and incapable of authenticating in any other way.

Credentials stored in the source code of an app on a device in an insecure location is inherently insecure. The credentials might be readable by decompilation. A native app on a mobile device is therefore considered a public client, unless there exist another adequate client authentication method.

Public clients are exposed to impersonation attacks. A malicious app might be able to extract the credentials and successfully authenticate itself to the authorization server.

G.1.2.2.2 Confidential

Clients that are able to keep client credentials confidential or capable of adequate client authentication in some other way.

G.1.3 Authentication

OpenID Connect (OIDC) is a standard built on top of OAuth2.0. While OAuth2.0 deals with authorization (access), OpenID Connect deals with authentication (who you are). "OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol. It also allows clients to obtain basic profile information about the End-User in an interoperable and REST-like manner" [48].

All OIDC flows requires the server to authenticate the end-user in some way, usually by asking for a username and password. The authorization server can delegate this task to another server. This enables using authentication servers in entirely different domains, with an existing database of users. This is an example(K.3:

1. The authorization server passes on the request from the client to the authentication server.
2. The authentication server redirects the end-user to a login page in the browser.
3. If the end-user is successfully authenticated the authentication server sends the authorization server an ID Token.
4. The authorization server will then make a response to the client dependent on the response type in the initial request made by the client.

In addition to adding authentication to the authorization flows specified in OAuth2.0, OIDC has added the new Hybrid flow, see section G.2.1.3.

G.1.4 Biometric authentication

Biometric authentication support is different based on both operating systems and the phone hardware.

G.1.4.1 iOS

iPhone 8 and prior has Touch ID (fingerprint identification) and the newer devices has Face ID (face identification). When used by a third party developer, both technology is implemented in the same way and the biometric data is processed using the Secure Enclave in both cases. The app using biometric authentication never gets access to the user-data[5].

The developer can ask for biometric authentication at any time, i.e. when opening an app.

The developer can choose to have a fall back option or not, i.e. a pin code.

The developer can choose to bind access to a keychain item to biometric authentication.

G.1.4.2 Android

On Android, biometric authentication can be used to unlock the `AndroidKeyStore`. Android provides two classes for biometric authentication. This is not a complete overview of all functionality provided, see the full Android documentation. For API level 23 (Android 6.0) to API level 27 (Android 8.1) `FingerprintManager`[49] was used. From API level 28 (Android 9.0) `BiometricPrompt`[50] is used while `FingerprintManager` is deprecated.

G.1.4.2.1 FingerprintManager

Android's first official biometric API[51]. Requires developers to implement their own user interface, only providing the underlying API calls. The method `authenticate()` "warms up the fingerprint hardware and starts scanning for a fingerprint." [49][Public methods]

G.1.4.2.2 BiometricPrompt

Shows a system-provided dialog upon starting authentication. The method `authenticate` warms up the biometric hardware and starts scanning for a biometric.[50][Public methods].

G.1.4.2.3 BiometricPrompt in Android Q

One gripe with `BiometricPrompt` in Android 9.0 is that it does not provide a fallback in case biometric authentication does not work or is not provided. This is solved in Android Q with the following method:

`BiometricPrompt.Builder.setAllowDeviceCredential(boolean enable)`. "The user will first be prompted to authenticate with biometrics, but also given the option to authenticate with their device PIN, pattern, or password." [52]

G.2 Our options

This section details the available options to meet the requirements of the assignment and answer the questions in section 1.2.1.

G.2.1 OpenID Connect flows

We have to consider the best authentication and authorization flow for our system and its requirements.

How it works

The value of the response type parameter in authentication request, specified in the specification [20], informs the authorization server which flow should be used.

G.2.1.1 Authorization Code flow

The choice of the response type **code** enables the authorization code flow.

Authentication request

During the first request the client only identifies with the client ID. A successful authentication response will include an URL with the authorization code in the fragment.

Token request

A client will authenticate itself to the authorization server using the method registered with its client ID at the server. One example is with a hashed combination of its client ID and client secret.

G.2.1.2 Implicit flow

The choice of one of these two response types enables the implicit flow:

- **id_token token**: A successful authentication response will include both an ID Token and an access token.
- **id_token**: A successful authentication response will only include an ID Token.

Authentication request

During this flow the client only authenticates using its client ID. A successful authentication response will include an URL with the requested items in the fragment.

Token request

There is no separate token request for this flow, which makes it susceptible to attacks where other clients can pose as this. The authorization server only checks if the request contains a valid client ID, not where it comes from.

G.2.1.3 Hybrid flow

The choice of one of these three different response types enables hybrid flow.

- **code id_token**: A successful authentication response will include both an authorization code and an ID Token.

- **code token:** A successful authentication response will include both an authorization code and an access token.
- **code id_token token:** A successful authentication response will include an authorization code, an ID Token and an access token.

Authentication request

The client authenticates in the same way as done in the authentication code flow. A successful authentication response will include an URL with the requested items in the fragment.

Token request

Works the same way as the authentication code flow.

G.2.1.4 Proof Key For Exchange(PKCE)

All three OIDC flows are susceptible to attackers intercepting either the code or the tokens from the fragment in the URL returned by the authorization server. PKCE ensures that an attacker intercepting the authorization code cannot exchange it for an access token, because the attacker does not possess PKCE's dynamically generated secret.

Without PKCE the access token in an intercepted authentication response can be used by the attacker to access restricted resources.

For flows where the access token is only returned in a token request an attacker would also need the client ID and secret. For a client on a mobile device both the client secret and client ID will be stored in the app, which makes them discoverable to an attacker by decompiling the app. A native app is defined by OAuth 2.0 to be a public client.

By intercepting the code from the URL and with access to both the client ID and secret, an attacker will be able to successfully do a token request and thereby gain unauthorized access to the resources.

PKCE was introduced to remove this attack vector from the hybrid and authorization code flows where only the code is returned in the authentication response. It does this by requiring these steps:

1. The client must add a hash of a randomly generated string to the authentication request.
2. Then the client will add the non-hashed version of the string to the token request.

The authorization server will only respond with a token if the hashed version of the string received in the token request matches the hash sent in the authentication request. The string will not be a part of the source code of the client and therefore not discoverable by decompiling.

PKCE does not protect the Implicit Flow, which is one of the reasons why Implicit FlowG.2.1.2 is not recommended for native mobile clients.[53]

G.2.2 Local storage

This subsection details some of the possible ways to persistently store data on both iOS and Android.

G.2.2.1 iOS

G.2.2.1.1 Keychain

By default, iOS stores passwords, certificates, secure notes, encryption keys, credit card and other highly sensitive small texts in the keychain. As a developer you can store any small text on the keychain. Data is stored in "items" as an implementation similar to SQL Lite. By default the data is only available for the app that stored the data. [5]

G.2.2.1.2 Files

To store data on files may also be an option. Every file on iOS is assigned a data protection class, which makes it secure [5]. The developer should make sure to set class of wanted protection, as the default may not be the best in their case.

G.2.2.2 Android

G.2.2.2.1 Internal storage

The internal storage for an app simply means the data is stored somewhere in that apps' private file tree. By default, this file tree can't be read from anywhere else. Data saved in internal storage is removed once the app is uninstalled.

G.2.2.2.2 External storage

The external storage is not private to an app and can be accessed from all other apps and is accessible through the main file tree of the device. Files stored here can be viewed and moved by anyone. On some devices the external storage can be located on a removable SD-card, meaning that it might not always be accessible[54].

G.2.2.2.3 SharedPreferences

For storing small amounts of app-private data that don't require structure. If you have a relatively small collection of key-values that you would like to save, you should use the SharedPreferences APIs. SharedPreferences uses XML files that persist across user sessions, even if the app is killed. Data can be specified to be private or shared. Each app has its own SharedPreferences.

G.2.2.2.4 Databases

Creating a database in the app's internal storage provides a means of storing large amounts of app-private structured data. It is also possible to store the database in public external storage.

G.2.2.2.5 Files

Files can be saved in 4 ways from a OS-perspective.

- **Internal** Files stored in internal storage are saved in a private directory and deleted when the app is deleted. They are only accessible from the app that saved them. User can't access these from anywhere else. Although in earlier API levels you could set them to be accessible to other apps.
- **External public:** "Files that should be freely available to other apps and to the user. When the user uninstalls your app, these files should remain available to the user. For example, photos captured by your app or other downloaded files should be saved as public files." [55]
- **External private:** "Files that rightfully belong to your app and will be deleted when the user uninstalls your app. Although these files are technically accessible by the user and other apps because they are on the external storage, they don't provide value to the user outside of your app." [54]
- **Cache:** It is also possible to store data temporarily in an internal cache, although they might be deleted by the OS.

G.2.3 Secure storage/encryption

The systems needs a way to securely store the access token. This is to prevent the token from getting in the hands of an adversary. This can be achieved by storing the token in a place where only the authorized apps can access it or encrypting it with a key only known to the authorized apps. Ideally the encrypted data cannot be retrieved, and the encryption key cannot be used by an adversary. But we will see later that this is not always the case.

This subsection details how you can ensure secure storage and encryption on both iOS and Android.

Availability	File Data Protection	Keychain Data Protection
When unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
While locked	NSFileProtectionCompleteUnlessOpen	N/A
After first unlock	NSFileProtectionCompleteUntilFirstUserAuthentication	kSecAttrAccessibleAfterFirstUnlock
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode enabled	N/A	kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly

Figure G.2: Caption

While iOS-devices with processors newer than the A7 (iPhone 5s) provide hardware-backed secure storage, only certain Android devices do.

G.2.3.1 iOS

G.2.3.1.1 Keychain

All decryption of the keychain is done by the secure enclave[5]. The keys inside the secure enclave is inaccessible, and their is no way of importing keys. Encryption are done using the dedicated AES-256 crypto engine for high efficiency. Keychain stores its items in the most secure way available on iOS devices.

As with every file on iOS the Keychain items gets encrypted and assigned a data protection class, as shown in figure G.2. The availability determines when the Keychain item is accessible. It is synced to iCloud, if not specified as "this device only" (non-migratory). An attempt to read the Keychain item on another device won't works, because because it is encrypted using the UID.

Keychain items can also be protected with Touch ID or Face ID, to be released by the Secure Enclave only by a successful match or the device passcode[5].

Secure enclave

The Secure Enclave is hardware on a iOS device that is completely separate from the rest of the hardware and makes encryption intact even if the security of the rest of the phone has been breached. To achieve this level of security the encryption key is stored inside the enclave and can't be viewed by anyone, even trusted entities. This also means that a key to decrypt something can not be created outside the secure enclave[5].

Keychain access control

With **Access control lists**, policies for access to a keychain item can be set. Authentication with biometrics or the device passcode can be required. Access can be revoked if certain events occur, like the deletion or addition of biometric data. These lists are evaluated inside the Secure Enclave and are therefore not changeable by unauthorized users[5].

G.2.3.2 Android

G.2.3.2.1 TEE (Trusted Execution Environment)

All "Android devices that support a lock screen and ship with Android 7.0 Nougat and higher have a secondary, isolated environment called a Trusted Execution Environment (TEE)" [1] A Trusted Execution Environment(TEE) on an Android device can be provided by having an additional OS on the device. This OS can be separated from the Android OS by both hardware and software, but they will both run on the same processor and using the same memory[56]. On devices with a TEE there can be areas of the hardware the Android OS will not have access to. Therefore, a TEE can be used to securely encrypt data by not exposing the key anywhere other than in the TEE. Access to the TEE is available through a driver in the Android-kernel.

In the case of Android, this means that compromising the Android OS (or its kernel) would not result in compromising the processes running in the TEE.

The TEE is responsible for the following functionality:

- **Lock screen passcode verification:** Does verification for devices supporting a secure lock screen. Unless an even more secure environment like tamper-resistant hardware[1] is available.
- **Fingerprint template matching:** For devices with fingerprint sensors running Android 6.0 or higher.
- **Protection and management of KeyStore keys:** The requirement being at least android 7.0 and support for a secure lock screen.

G.2.3.2.2 Hardware secure storage/Secure Element(SE)

A Secure Element(SE) is separate hardware on an Android device just for security functions and operations. It will have its own CPU and memory, which is inaccessible from the rest of the device. A key used for encryption cannot exist outside this secure hardware, making it impossible to extract for malicious use. The data to be encrypted or decrypted will instead be sent into the SE. The use of a key is restricted by some form of authentication, like a password or ID. This is also sometimes called Tamper-resistant hardware.[1]

G.2.3.2.3 Android KeyStore

The KeyStore enables the app to generate and use cryptographic keys. As long as the app process isn't compromised no one else can read, use or extract the keys[57].

"The Keystore system is used by the KeyChain API as well as the Android Keystore provider feature that was introduced in Android 4.3 (API level 18)." [58]

The KeyStore offers a few different security features:

- The actual keys are never in the app. Use of the keys is done via calls to a system process which carries out the cryptographic operations. If the app's process is compromised, the attacker may be able to use the app's keys but cannot extract their keys.
- Keys can be bound to "secure hardware" (or). This makes them impossible to extract from the Android OS or the device's internal storage[58].
- The KeyStore also offers ways to restrict how the key is used. A developer can specify which algorithms and operations a key can be used for, the length of the validity of the key or requiring user authentication when using the key.

Changes to the Android keystore system in Android 9.0

Android 9.0 added additional functionality to the classes in the Android keystore system. Some highlights are presented below.

- Additional security can be added by using a hardware backed StongBox Keymaster[57].
- It is possible to make KeyStore keys unavailable for decryption unless the device is unlocked, using the `setUnlockedDeviceRequired()`[59] method in the `KeyGenParameterSpec` Builder.

Even with these features the key can be misused if a malicious app somehow manages to pose as the legitimate app or compromise the legitimate app. The malicious app can ask the KeyStore to decrypt the sensitive data[58]. This might be possible if an attacker manages to gain superuser/root privileges.

KeyStore and secure hardware

If the generated keys are bound to secure hardware (Trusted Execution Environment (TEE), Secure Element (SE)) the following applies: The data to be decrypted is moved to the secure hardware and decrypted in the memory there. By default, only the app that required the key generation can use the key. When binding to secure hardware is enabled for a key, its key material is never exposed outside of secure hardware." [58]

G.2.3.2.4 KeyStore provider

Using a KeyStore provider enables an app to store keys and credentials only that app can access. This doesn't require end-user consent.

G.2.3.2.5 KeyChain

The KeyChain API lets apps use system-wide credentials. "When an app requests the use of any credential through the KeyChain API, users get to choose, through a system-provided UI, which of the installed credentials an app can access. This allows several apps to use the same set of credentials with user consent." [58][Section: Choose between a keychain or the Android keystore provider] Users can add their own credentials to the

G.2.4 Token sharing between apps

The system needs a way to share the access token retrieved by one app with the other approved apps. This is to prevent the user having to re-authenticate and re-authorize apps between app switching. This must be done in a way so unauthorized apps can't read or use the access token.

By default there are limitations of sharing stored data between apps, see 2.5. However, in certain cases, several apps can share stored data, but in the case of iOS only if the apps are from the same developer. This subsection details possible ways to share data between apps on both iOS and Android.

G.2.4.1 iOS

There are many ways to share a token between apps, and a few to share it in a secure way. Due to sandboxing(see 2.5), stored data cannot be shared between apps from the same developer, unless it is explicitly chosen.

G.2.4.1.1 Keychain and access groups

"Access groups" lets apps share keychain items with other apps from the same developer. The apps still has the option to store data in their own private keychain group. From the keychain perspective the private keychain and the access group is two entirely separate keychains groups on an equal level. See Figure G.3 and Figure G.4 for a visualization.

An app may be apart of many access groups and app groups. A keychain item may only be a part of one access group. [5] The app's access groups is stored in an array of access groups. This array always contains of the application identifier (app ID), which is recognized as the private access group for the app. The first item in this array is the default access group. This means that if nothing else is specified, this is where the item will be stored. If an keychain access group is specified then, it's name will be added in the beginning of the array, making it the default access group. When an access group from an app group is added, it is added at the end of the array, making it so that it never can be the default.

"This is managed by requiring third-party apps to use access groups with a

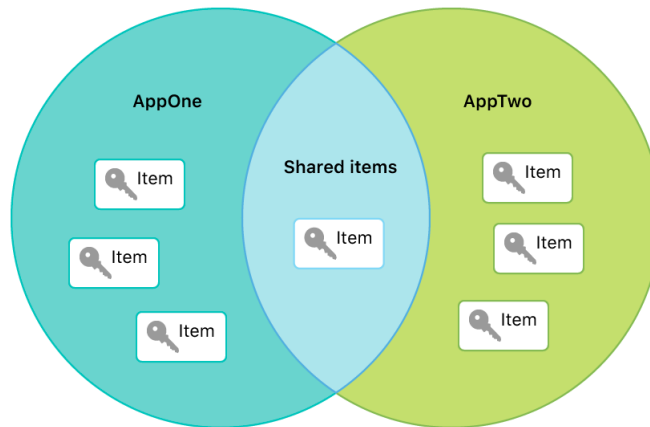


Figure G.3: A shared keychain access group from the apps' perspective.
Source: [60]

prefix allocated to them through the Apple Developer Program through application groups. The prefix requirement and application group uniqueness are enforced through code signing, Provisioning Profiles, and the Apple Developer Program.”[5] This means that even if another app uses the same prefix, it will not get access to the same data. A daemon determines which app have access to which data by looking at “Keychain-access-groups” (Keychain access group), “application-identifier” (basically app id), and “application-group” (App group).

G.2.4.1.2 App groups

App groups let you share the keychain within the group, in addition to extra sharing functionality between apps like sharing files. ”Apps within a group can communicate with other members in the group using IPC mechanisms including Mach IPC, POSIX semaphores and shared memory, and UNIX domain sockets.”[61]

You also need to use app groups to make extensions. The keychain items that is shared in the app group are not shared in the same access group as the items that is shared in the access group just for sharing the keychain. From the keychains perspective, the private keychain, the access groups for only keychain sharing and the app groups are three entirely different entities. See Figure G.3 and Figure G.4 for a visualization.

G.2.4.1.3 Universal Links

In the universal link, data may also be sent. This is done securely. No one can read the link as it is unique for the receiving app [62]. See G.2.5.1.2 for more

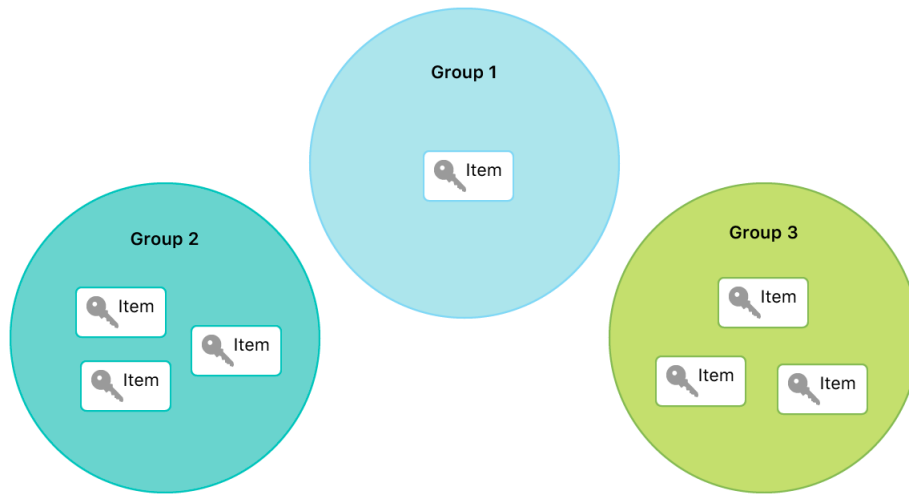


Figure G.4: Shared keychain access groups from the keychain's perspective.

Source:

https://developer.apple.com/documentation/security/keychain_services/keychain_items/sharing_access_to_keychain_items

information.

G.2.4.1.4 Custom URL schemes

Extra data can be attached to the URL. This has security issues. See G.2.5.1.1 for more information.

G.2.4.2 Android

G.2.4.2.1 Intents

Data can be attached to an **Intent**. See G.2.5.2.1 for more information.

G.2.4.2.2 Content Provider

ContentProviders is the intended way on Android to share data between apps. It enables an easy to use interface to an apps' storage. While the overridable methods of a ContentProvider is built to mimic calls to a database you can implement them to work with whatever storage method you chose. Another app can use a ContentResolver to access this interface and then retrieve, store or update data in the other app.

Access to a Content Provider can be restricted to only apps with a specific permission. A permission can have different levels of protection[63], and it can be user defined.

The **signature** protection level provides granular access to a Content Provider and could be especially relevant to us. "A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval." [63] This unobtrusively and securely enables data sharing.

G.2.4.2.3 File Provider

A File Provider is a subclass of Content Provider which enables sharing of files by creating and sending URIs pointing to the file to share [64].

G.2.4.2.4 External Storage

Files in external storage can be read by any app. See subsection G.2.2

G.2.4.2.5 Shared User ID

At install time, Android gives each package a distinct Linux user ID that remains constant for the duration of the package's life on that device. Apps can share user ID using the `android:sharedUserId` [65] attribute in `AndroidManifest.xml`'s manifest tag. All apps with the same user ID can access each other's data and, if desired, run in the same process. A requirement for sharing UID is that the apps are signed by the same certificate [65].

"Upgrading an application that modified its `sharedUserId` is not allowed by the Android operating system for security reasons." [66] "Not adding one [`sharedUserId`] to your app before launching to Google Play (or any other distribution channel) will make it impossible to add one at any point in the future through an update, since adding after an initial install will make your applications data inaccessible (under `/data/data/package_name`) [67], [68].

G.2.5 Inter-app Navigation

This section details aspects relevant to redirecting the end-user between different apps on the device. The information is crucial to the result of the research question **From a user perspective, do several securely linked mobile applications function as well as one big mobile application?**.

On Android this can be optimized by taking advantage of the flexibility provided by activities and tasks, see 2.4.2.

On, iOS this can be done using modal views.

G.2.5.1 iOS

When navigating from one app to another the OS automatically displays the back to app button. This makes it so that if you navigate from app A to app B to app C, you can move back to app B using the black arrow, but not to app A because there is no button.

The back to app button is a triangle pointing the same way as the normal iOS back button. In some cases, like during a phone call or screen recording, it is replaced by a contextual button relevant for those actions.

When it opens the previous app, the `OnResume` method gets called and if the app has not been killed, it maintains its previous state.

G.2.5.1.1 Custom URL scheme

Every app may register an URL scheme for opening that app. Unfortunately there is no check on if this scheme is one of a kind, so there may be two apps with the same scheme. Apple does however deny the developers to register a scheme that is "well known" for instance schemes for opening mail and facetime [26].

The iPhone checks if there are any schemes matching this scheme and opens the corresponding app. The developer may check in advance if the scheme will open something. If there isn't any matching schemes, nothing happens. If there are two or more identical schemes, the identifier still makes the schemes unique. The identifier is a reverse DNS string that incorporates your company's domain and app name. Although the identifier is best practice there are still nothing that stops another app to register the same URL scheme and handle the corresponding links. Which means a malicious app might be able to read an URL sent to another app.

G.2.5.1.2 Universal links

Universal links makes it so that an app may have an corresponding website. When using universal links and being on this website in Safari, a banner will show up on the top of the screen asking you to install the app. If the app is installed, the user will open this app in the same context as in the website. This is called "deep linking" into the app.

Earlier this functionality was based on custom URL schemes. The JavaScript on the website had a timeout on the request of opening the app. If the request timed out, the user was sent to the app store. This technology relied on blocking in JavaScript, which was no longer supported after iOS 9.3. With this Apple pushed Universal links as the only method to deep link into an app from a website.

If a universal link is opened when the app is not installed on the phone, the user

will be sent to the corresponding website. It's the receiving app's responsibility to handle the links. This is a possible attack point, so it's important that the app is handling also malformed URLs[69].

By being the same link as to the corresponding website, universal links ensures that there is only one possible app to link to on the phone and thereby that no other app can read the link[62]. It does not check the corresponding website when switching app using an Universal link.

requirements

A website with "paired" with your app. This is required, even though iOS does not check the website when deep-linking to an app that is installed on the phone[70].

G.2.5.2 Android

When navigating from one app to another the end-user will have to use the back-button or find the previous app in the **recent app** screen if a new task was started.

See 2.4.2 for more information about tasks and activities.

G.2.5.2.1 Intents

An app can communicate with other apps using **Intents**. If an app needs some functionality given by another app this can be stated in the intent. The app states its intent to the OS, the OS then finds the appropriate app and launches it. In the Android application manifest, it can be specified that only apps signed with the same certificate can open each other. Data, like a string, can be attached to the intent and in that way be sent to another app. Intents contain flags telling the operating system how to open the Activity that the intent intends to open. These are set with `setFlags()`. The different flags and what they do is detailed here: [71]. Examples are: deciding if the app should be restarted or resumed, and if the activity should be added to the current Task or not.

There are two different types of intents:

Explicit intents

As mentioned, intents can be used to open other apps/activities. With explicit intents one can specify which activity should be opened with either a package name or fully-qualified component class-name. Hijacking an explicit intent is not possible, as the intents use unique app identifiers. Two apps with the same package name cannot be installed on the same Android device.

Implicit intents

With implicit intents the OS finds the app that matches the intent, if there is one. If there are more than one the end-user may be prompted to choose between them.

Intent filter

Each activity in an app can have one or more **IntentFilters**. These are used to specify what **Intents** the app can handle.

Starting activities

There are two ways to start activities on Android.

StartActivity(intent) The new activity appears on the screen, and the originating activity is paused.[72].

StartActivityForResult(intent, TEXT_REQUEST)[73] *TEXT_REQUEST* is used to identify each intent with

If intent.SetFlags(0) is used before calling StartActivityForResult, the transitioning animation will be replaced in—from-bottom animation on Android 9.0. There may be more ways to change the animation.

Appendix H

Vision document

Team Free Juice, DIPS Single-Sign-On Vision document

Version <1.2>

Revision History

Date	Version	Description	Authors
31/01/19	1.0	First draft	Truls Matias Torgersen, Erling Kristiansen and Toni Vucic
09/04/19	1.1	Review in preparation for meeting with Tomas 09.04.2019. Very minor changes.	Toni Vucic
19/05/19	1.2	Cleaned up the sections: User environment and product overview. Added the needs of DIPS based on early meetings and the bachelor task. Added nurse/doctor needs. Added the GitHub issues we had after creating our first backlog. Added the non-functional needs based on the requirement specifiacion, the bachelor assignment document, hypothese and informal talks with the develompent leader at DIPS Trondheim, during the first month..	Toni Vucic

Introduction

The purpose of this document is to outline the requirements and goals for our software engineering bachelor's project. The project consists of three small rudimentary cross-platform (iOS/Android) applications, one big application, and an underlying cross-platform Single-Sign-On library. The goal is to investigate if several securely linked mobile applications function as well as one big secure mobile application, both from a users' perspective and a security perspective. The document outlines the stakeholders, target users and features of the system.

What is "our system"? The Single Sign-On library is the system we are developing. Other than that, four mock applications will be made in order to user-test navigation. A few more apps will be made to verify that the technology used in the Single Sign-On library works.

Table of contents

Introduction	1
Table of contents	2
Summary of problem and product	3
Single-sign-on problem summary	3
Specific problem summary	3
Product summary	3
Stakeholder and User Descriptions	4
Stakeholder Summary	4
User Summary	4
User Environment	5
Workflows	5
Other systems	6
Hardware in place today	6
Summary of user needs	6
The needs of DIPS	6
Needs of nurses and doctors	7
Alternatives to our product	7
Product Overview	7
Product role in user environment	8
Conditions and dependencies	8
Product Features	8
Non functional features and other requirements	10
Code and technology	10
Security	10
Design and usability	11

Summary of problem and product

Single-sign-on problem summary

The problem of	not being able to seamlessly navigate between apps with the same login credentials
affects	the end users experience of the system
the impact of which is	hard to use systems requiring new log-in every time the user navigates between apps
a successful solution would be	a secure single-sign-on solution

Specific problem summary

Exploring the useability of several smaller apps versus one larger app all having the same functionality. The larger app produces a problem:

The problem of	developing, maintaining and expanding big mobile applications simultaneously across several development teams.
affects	mobile application developers and companies working on complex applications
the impact of which is	slow expensive development where product releases are hard to coordinate and teams lock each others progress due to overlapping boundaries
a successful solution would be	several small mobile applications with high cohesion and low coupling, maintaining the same usability and security as one big application

Product summary

For	DIPS AS.
Who	wants a secure single sign on solution, without dependencies.

That	keeps you logged on between all of DIPS' applications on the same mobile device.
Unlike	DIPS's current/previous solution of one large app providing all the functionality.
Our product	keeps the user logged on, and still just as safe.

Stakeholder and User Descriptions

Stakeholder Summary

Name	Description	Responsibilities
DIPS	<ul style="list-style-type: none"> - The company we write our bachelor at. - Interested in knowing the results of our research. 	<ul style="list-style-type: none"> - Write the project assignment - Sets the demand for the features of the project - Approves the final product
NTNU	<ul style="list-style-type: none"> - Our university 	<ul style="list-style-type: none"> - Guide us through the process, make sure all the formal stuff is ok and give us the information we need. - Approves the final product
Toni, Truls and Erling	<ul style="list-style-type: none"> - Us 	<ul style="list-style-type: none"> - Develop the software. Document the software and its development in our bachelor thesis

User Summary

Name	Description	Responsibilities under development	Represented by
Workers at the hospital (physicians and/or nurses)	<ul style="list-style-type: none"> - will be the user of this solution. This user group would like to interact with our solution as 	<ul style="list-style-type: none"> - May be a part of user testing 	Themselves

	little as possible so they can do the actual work that the solution protects.		
DIPS	<ul style="list-style-type: none"> - Develops and maintains the current patient-data solution in several hospitals - might implement our SSO-solution 	<ul style="list-style-type: none"> - guidance - decides requirements 	Manager and head of the Trondheim office, Runar Ovesen Hjerpbakk

User Environment

This describes the environment our users will use our system in.

Workflows

The environment is a busy hospital with a large variety of tasks to be done. Nurses are checking up on patients and doing and logging daily tasks. Doctors are retrieving and saving patient data to make critical decisions. Today, using desktop computers, the patients are cared for, then the completed tasks are logged on a computer somewhere else.

Today the systems rely on a connection to the internet and many supporting systems.

Other systems

There are many systems at the hospital. We expect our SSO solution to interact with the current DIPS authentication and authorization servers. There are currently no other systems that need to be taken into account.

Hardware in place today

Electronic patient journals are currently handled on desktop Windows computers. Since we are creating a solution for Android and iOS, these computers cannot be used.

Every hospital employee has a private phone. Some hospitals also have phones that the doctors and nurses use for work only. At some Norwegian hospitals currently not served by

DIPS, like St. Olavs Hospital, they have outdated and unresponsive Android devices. We will not be using any of these.

Summary of user needs

The needs of DIPS

These are based on the project proposal and talks with the development lead at DIPS Trondheim.

Needs	Priority	Today's solution	Proposed solution
Share access tokens securely between apps on the same device. (One authorization per login scenario)	High	None	Build a library providing this functionality on top of Android and iOS operating system method calls.
Store tokens securely, using Secure Enclave on iOS, and a correspondingly safe place on Android.	High	Unknown on iOS, none on Android.	Build a library providing this functionality on top of Android and iOS operating system method calls.
For apps to sign in to the DIPS authorization & authentication server using OAuth2.0/OpenID Connect, and obtain an access token in a secure and user-friendly way.	High	Possibly insecure OAuth2.0/OpenID Connect sign-in code in the in-development DIPS Visit App, based on the open source AppAuth library.	Our own OAuth2.0/OpenID Connect client supporting the most secure OAuth2.0/OpenID Connect authorization & authentication flow.
Mock apps made in Xamarin Forms, supporting deep linking so that navigation is experienced as efficient	High	None	Make the mock apps using the functionality provided by the Android and iOS operating systems. Then user test to see if it is experienced as efficient.
To know if several small applications function as well as a big application, from a user perspective.	High	None	We will do a literature study and create test applications to verify that it is true, as well as usability tests.
An understanding of relevant security differences across Android-versions and devices.	Medium	None	We will do a literature study and consider this in our report, where applicable.

Needs of nurses and doctors

The need for an SSO library does not stem from the needs of the doctors and nurses. It is DIPS ASA who want to see if several small apps can work well together. The nurses and doctors are more interested in how the apps implementing our SSO library will work.

Needs	Priority	Today's solution	Proposed solution
The navigation between apps to be intuitive	Medium	There is no solution with navigation between several apps.	Do user testing to ensure our inter-app navigation design is as good as the navigation in one app.
An efficient and painless sign-in experience	Medium	Sign-in via the browser, redirecting back into the DIPS Visit app.	Sign-in via the browser, redirecting back into the app.

Alternatives to our product

The alternative to our SSO solution is one large app with all the functionality the small apps have combined. This would eliminate the need for a SSO solution but introduce development difficulties that follows with one large app.

There are currently several off-the-shelf solutions available like Auth0 and the solutions from okta.

Product Overview

The product is a single sign on solution for multiple apps on the same mobile device. No authentication when switching apps is preferred. The user may still have to use fingerprint, face ID or write a pin code when switching between apps, but not the full login with username and password. Authentication and authorization is done using through the OAuth 2.0 protocol with the DIPS authentication server.

Product role in user environment

The main goal is to make life easier for the workers at the hospital. The working environment for the users may be stressful, so our solution must be as seamless as possible and not be noticeable if possible.

The new mobile system DIPS is developing, built on top of our SSO technology, will facilitate a more effective and natural workflow. Logging and information gathering can be done from anywhere in the hospital, which is an advantage in a busy workflow.

Conditions and dependencies






A requirement for our SSO solution to work are OAuth2.0-enabled DIPS authentication and authorization servers.

Product Features

With permission from our supervisor Tomas Holt, we defined the product features as GitHub issues. All the 40 original issues are pasted here. Those marked as enhancement are directly related to the product. There are 24 such issues.

<input type="checkbox"/>	🔔 40 Open ✓ 2 Closed	Author ▾	Labels ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	🔔 FaceID for authentication enhancement iOS #42 opened 14 minutes ago by tonivuc						
<input type="checkbox"/>	🔔 Fingerprint for authentication Android enhancement iOS #41 opened 15 minutes ago by tonivuc						
<input type="checkbox"/>	🔔 Run iOS apps on Windows via Truls's machine enhancement #40 opened 17 minutes ago by tonivuc						
<input type="checkbox"/>	🔔 Sequence diagrams? documentation #39 opened 2 hours ago by tonivuc						
<input type="checkbox"/>	🔔 Domain model? documentation #38 opened 2 hours ago by tonivuc						
<input type="checkbox"/>	🔔 Prepare for workshop on tuesday documentation #37 opened 2 hours ago by ErlingMK						
<input type="checkbox"/>	🔔 Securely store a token enhancement iOS #36 opened 3 hours ago by Trulsmatias						
<input type="checkbox"/>	🔔 Securely store a token Android enhancement #35 opened 3 hours ago by Trulsmatias						
<input type="checkbox"/>	🔔 Implement automatic background token refresh Android enhancement iOS #34 opened 3 hours ago by tonivuc						
<input type="checkbox"/>	🔔 Use same access token for more than one app enhancement iOS #33 opened 3 hours ago by ErlingMK						
<input type="checkbox"/>	🔔 Write public API documentation enhancement #32 opened 3 hours ago by tonivuc						
<input type="checkbox"/>	🔔 Use same access token for more than one app Android enhancement #31 opened 3 hours ago by ErlingMK						

<input type="checkbox"/>	🕒 Use access token to fetch resources enhancement iOS #30 opened 3 hours ago by Trulsmatias
<input type="checkbox"/>	🕒 Use access token to fetch resources Android enhancement #29 opened 3 hours ago by Trulsmatias
<input type="checkbox"/>	🕒 Create Timeout exception interface enhancement iOS #28 opened 3 hours ago by Trulsmatias
<input type="checkbox"/>	🕒 Create timeout exception interface Android enhancement #27 opened 3 hours ago by Trulsmatias
<input type="checkbox"/>	🕒 Single log-out enhancement #26 opened 3 hours ago by tonivuc
<input type="checkbox"/>	🕒 Find out if it is secure to NOT write pin code when switching apps. Android question #25 opened 3 hours ago by Trulsmatias
<input type="checkbox"/>	🕒 Find out if it is secure to NOT write pin code when switching apps. iOS question #24 opened 3 hours ago by Trulsmatias
<input type="checkbox"/>	🕒 Do a live git squash example question #23 opened 3 hours ago by tonivuc
<input type="checkbox"/>	🕒 Set-up meeting with Runar on how to use services in Xamarin question #22 opened 3 hours ago by ErlingMK
<input type="checkbox"/>	🕒 Login to DIPS using OAuth2 and verify it worked enhancement iOS #21 opened 3 hours ago by tonivuc
<input type="checkbox"/>	🕒 Login to DIPS using OAuth2 and verify it worked Android enhancement #20 opened 3 hours ago by tonivuc
<input type="checkbox"/>	🕒 Do mob-coding for a MV-VM example #19 opened 3 hours ago by ErlingMK
<input type="checkbox"/>	🕒 User testing documentation #18 opened 3 hours ago by Trulsmatias

40 Open ✓ 2 Closed		Author ▾	Labels ▾	Projects ▾	Milestones ▾	Assignee ▾	Sort ▾
🔔	Do a live pull request example documentation						
#17	opened 3 hours ago by ErlingMK						
🔔	Stay signed in when switching from one app to another enhancement iOS						
#16	opened 3 hours ago by tonivuc						
🔔	Stay signed in when switching from one app to another Android enhancement						
#15	opened 3 hours ago by tonivuc						
🔔	Send data/string from one app to another Android enhancement						
#12	opened 4 hours ago by tonivuc						
🔔	Send and receive data between two apps enhancement iOS						
#11	opened 4 hours ago by Trulsmatias						
🔔	Open one app from another app enhancement iOS						
#10	opened 4 hours ago by Trulsmatias						
🔔	Open another an app from within another app Android enhancement						
#9	opened 4 hours ago by tonivuc						
🔔	Meet the people implementing OAuth 2.0 at DIPS question						
#8	opened 4 hours ago by ErlingMK						
🔔	Write phase plan document documentation						
#7	opened 4 hours ago by tonivuc						
🔔	Finish requirements document documentation						
#6	opened 4 hours ago by Trulsmatias						
🔔	Finish Gantt documentation						
#5	opened 4 hours ago by Trulsmatias						
🔔	Make initial meeting plan with Tomas documentation						
#4	opened 4 hours ago by ErlingMK						
🔔	Write a first draft of the vision document. documentation						
#3	opened 4 hours ago by tonivuc						
🔔	Setup CI enhancement						
#2	opened 23 hours ago by Sankra						
🔔	Lage Solution for første app enhancement						 1
#1	opened a day ago by Sankra						

Non functional features and other requirements

Code and technology

- The SSO code should have high cohesion and low coupling
- The SSO library is easy to expand upon and re-use components from
- The library uses the same code base for both Android and iOS
- The mock apps use Xamarin Forms
- Recommended: The mock apps are built using the MVVM principle

Security

- Tokens on the device should be inaccessible/unusable when the device is locked
- Tokens stored by the library cannot be extracted or modified by other apps on the device

- Data shared between securely linked applications is not accessible to other apps.
- The sign-in session with the authentication & authorization server cannot be hijacked by an attacker
- It is possible to remotely revoke user access to the application through the library.
- Data shared between securely linked applications is not accessible to other apps.
- When one app attempts to open another app by the same developer, a third app cannot hijack the inter-app navigation and get opened instead of the intended app.
- The library has few, if any external dependencies
- Only needs to be secure on un-rooted/non-jailbroken devices.

Design and usability

- The library supports authentication & authorization with an external OAuth2.0/OpenID Connect server. This should be done through one library method, and the developer should be notified on successful login.
- The SSO library should be unintrusive to end-users (doctors and nurses)
 - The library should only prompt the user to log in if the user isn't already logged in.
 - The library can keep the user logged in across all the apps on the same device, created by the developer, without re-authenticating for each app.
 - The library is able to prompt the user to re-authenticate if their session has expired, letting them continue where they left off before their authorization expired.
- Navigation between several securely linked apps is just as intuitive as navigation within one secure app
- There is no noticeable difference for a user when changing between views in one secure app as opposed to when changing between several securely linked apps.

Appendix I

Requirement documentation

Single-Sign-On Requirement Specificaion

Version <1.0>

Revision history

Date	Version	Description	Author
29.01.2019	<0.1>	Started translating from Norwegian to English and writing the introduction	Toni Vucic
30.01.2019	<0.2>	First proper draft. Includes a few user stories for "user" and "developer" and some content still in Norwegian.	Toni Vucic
19.05.2019	<1.0>	Added all the sequence diagrams we used while planning the project. Polished the document.	Toni Vucic

Table of contents

Introduction	2
Purpose	2
Scope	2
User Stories	2
App user	2
Developer	4
Sequence diagrams	6
Authorization and Authentication	6
Token storage and sharing	10

1. Introduction

The purpose of this document is to lay out the requirements for our cross-platform Single-Sign-On implementation. It also contains two pictures functioning as requirements for the user testing UI-application navigation design.

1.1 Purpose

The requirement document is based on goals outlined in the Vision document. It will describe the requirements for our system, and some of the most important user scenarios.

1.2 Scope

The requirements in this document are based on meetings with DIPS ASA, who have requested a survey of Single-Sign-On solutions and a working, user-tested implementation of such a system. More details are to be found in the Vision document.

2. User Stories

2.1 App user

```
As a <role>  
I wish to <goal>  
So that <advantage>
```

```
As an app user  
I wish to sign in  
So that I have access to the app functionality  
Scenario: Sign in  
Given that I have opened the app  
  And I am not signed in  
Then I am presented with a sign-in page  
  When I input my credentials and submit  
Then I am signed into the app  
  And am presented with the application's functionality
```

As an app user
I wish to use the application after not using it for a while
So that I can perform my daily work tasks

Scenario: User timed out
Given I want to resume work in the app
 And the login timeout has expired
When I open the app
 Then I am prompted to verify my identity
When identity has been verified
 Then I should be able to continue exactly where I left off

As an app user
I wish to [do a task in a different app]
So that I [can do the task]

Scenario: Switch between apps
Given that I am is signed in
 And wishe to do a task in a different DIPS app
When I click on a link/button to another app
 Then I should be redirected to the other app
 And automatically be signed in
 And be able to continue the task I set out to do with the context
from the previous app transfered.

2.2 Developer

As a developer

I wish to have users log in via an external service/server

So that I can request information that user is authorized to have

Scenario: User login

Given a user opening the app and not yet being authorized

When user opens the app

I call a function to load the external log-in web-page within the app

Given user successfully authorized by server

When the user's browser is redirected back to the application

Then the SSO API should let me know the user has been authorized

Given user unsuccessfully authorized by server

The embedded webpage provided by the Authorization Server will handle additional attempts and not re-direct back to the application until the user has been authorized.

WARNING, we need to investigate the possibility of the server not acting like we expect.

As a developer
I wish to have users be signed in to all my application once they sign in on one
So that their workflow is not disturbed by more sign-ins.

Scenario: Open another app from app tray
Given a user already signed into one app recently
And opens an app he has not signed into recently
When user opens app
Then the developer calls a function to check if the user is signed in
If the user is signed in
Then the function returns true
If the user is not signed in
Then the function suggests a web-page be opened for the user to sign in

Scenario: Open app from within another developer-owned app
Given a user is currently signed into an app
When the user presses a button/link
Then the developer calls a `switchApp(appIdentifier, data)` function from our API
And the user is automatically redirected to the other app
It is important that the developer saves the current app state before switching

As a developer
I wish to revoke the access the user has to the application
So that confidential data is protected

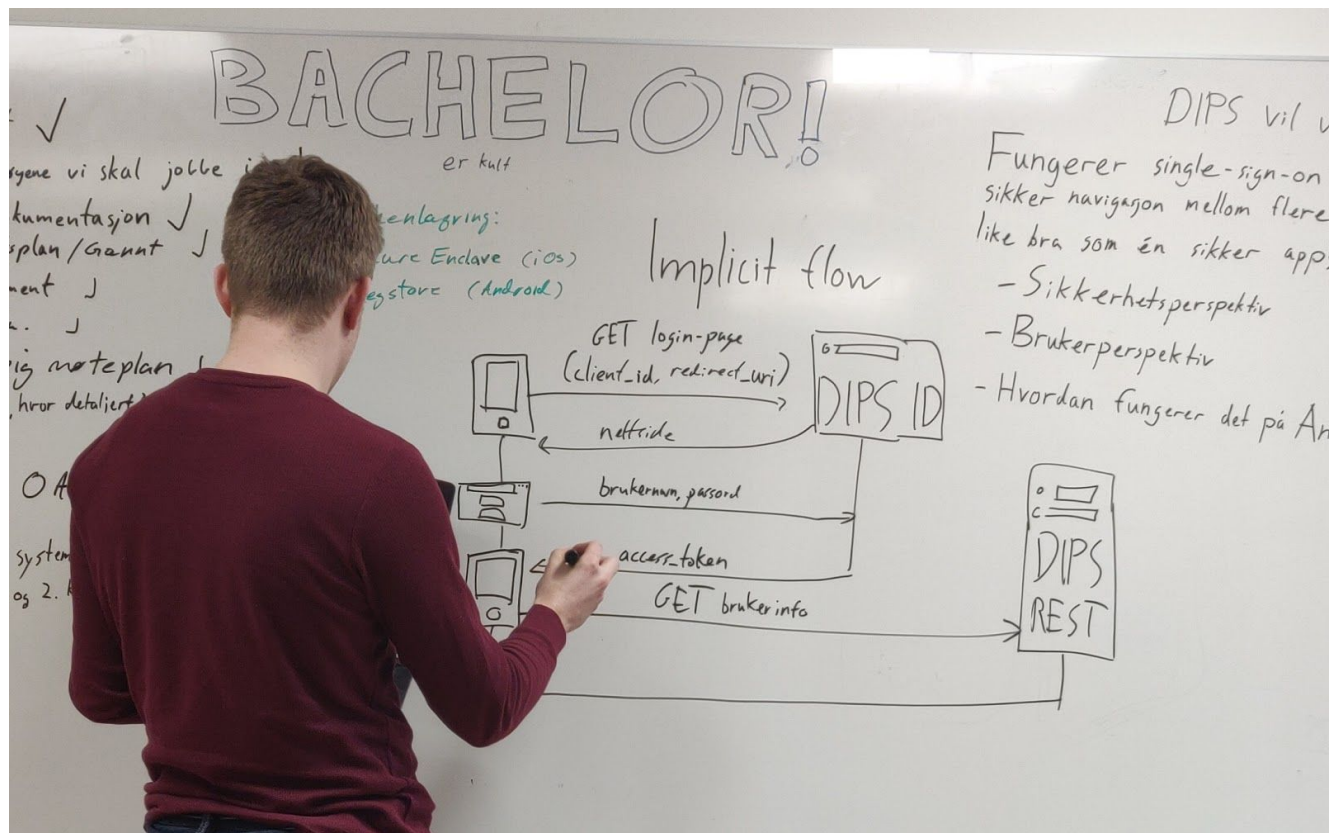
Scenario: User kicked from app
Given a user is signed into and using an application
And their access to remote resources has been revoked
When the user attempts to read or write to a remote resource
I call on the resource using the token the SSO API provided.
And get an error back that I handle

3. Sequence diagrams

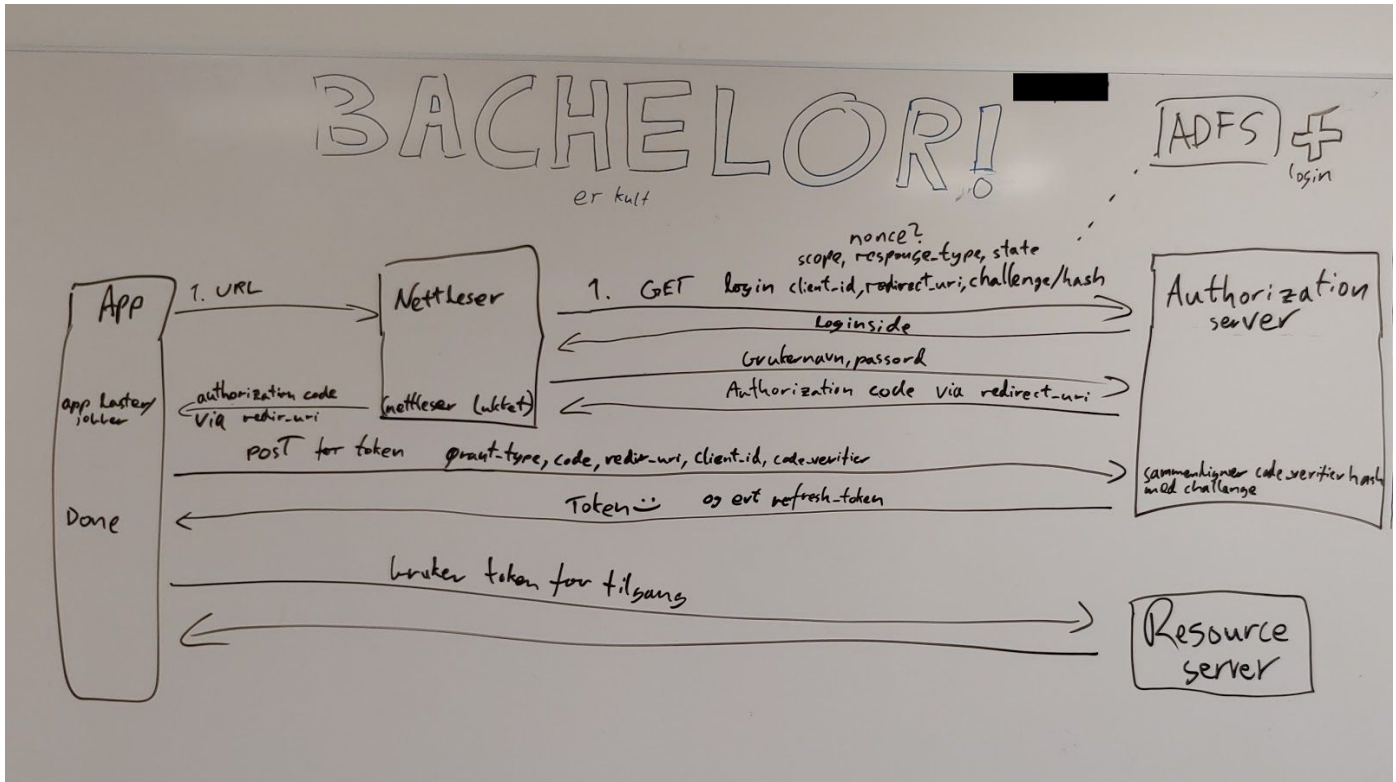
3.1 Authorization and Authentication

To get an understanding of how authentication and authorization works, these sequence diagrams were created early on in the project. The first two sequence diagrams do not mirror reality perfectly because of our lacking knowledge of how authorization and authentication worked. The third sequence diagram resembles the final product more closely. The fourth starts getting down to the method level. The diagrams were used to get a quick overview of the functionality we still had to implement.

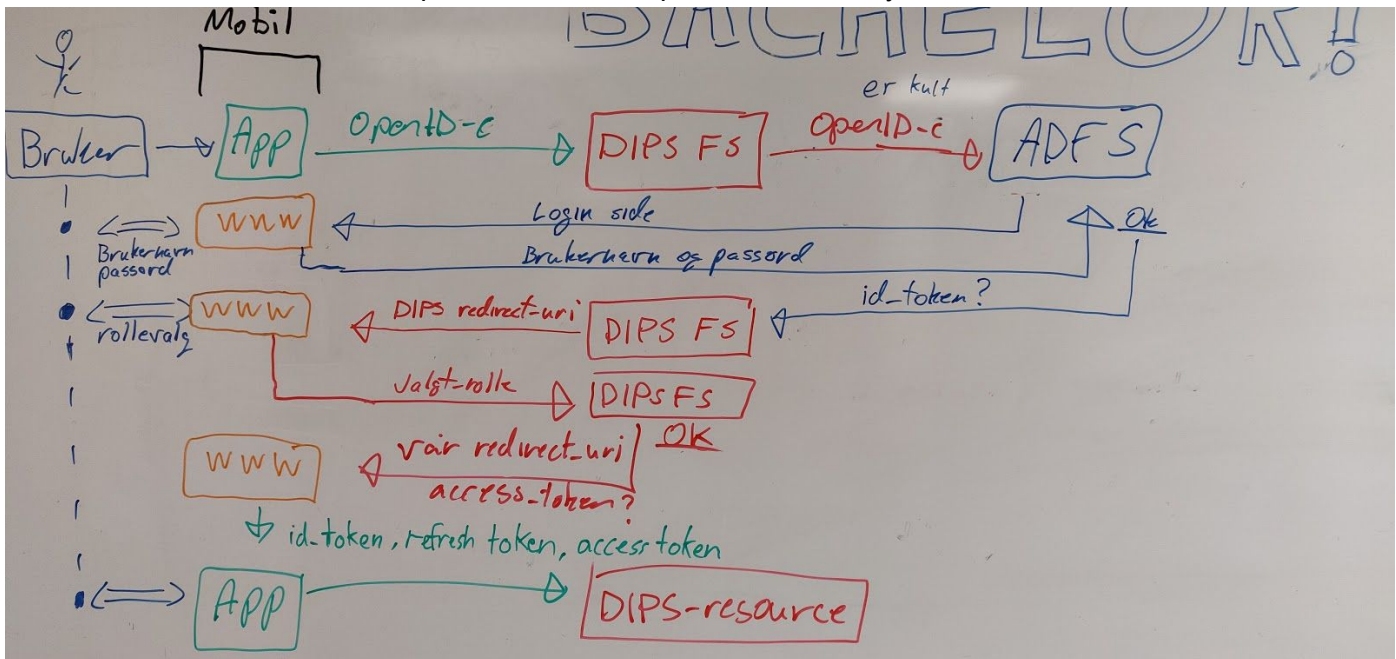
Picture taken 06.02.2019. Early model of the OAuth2.0 implicit flow, which we later decided not to use.



Picture taken 21.02.2019



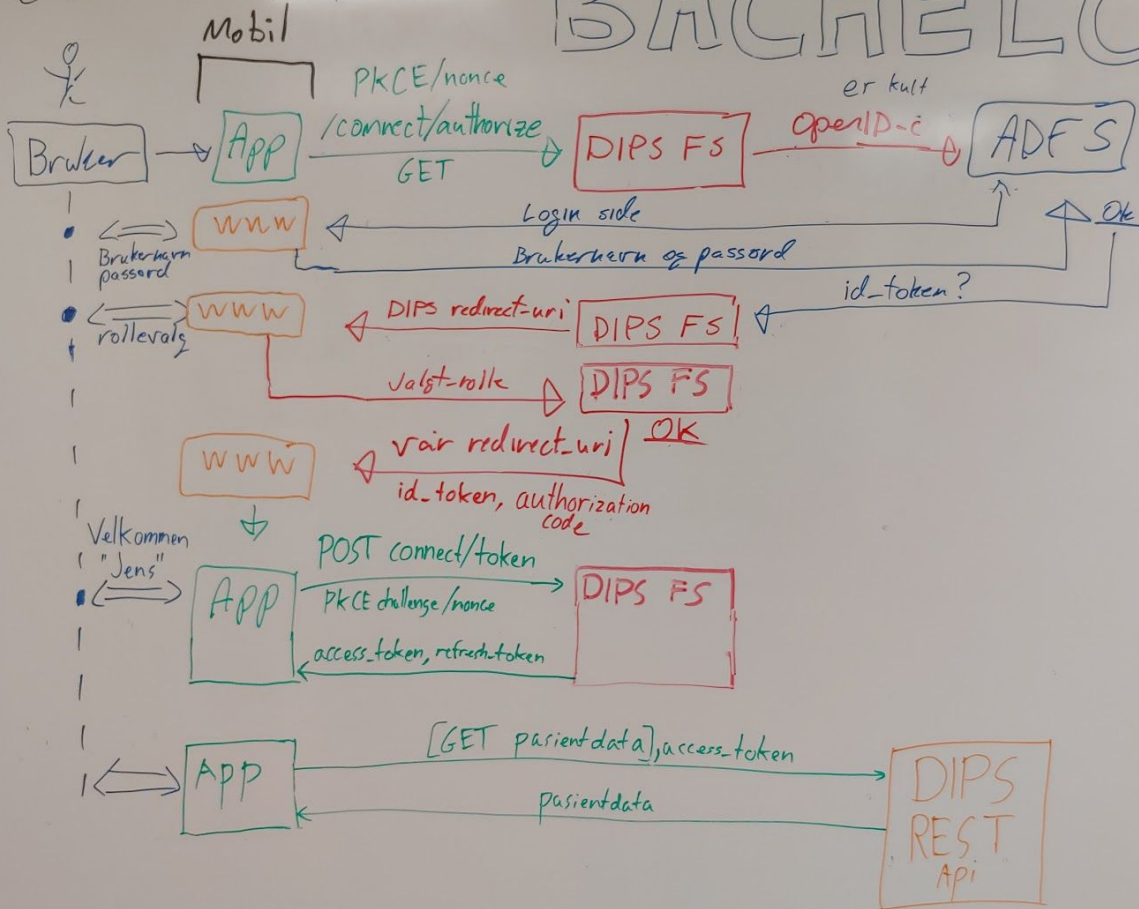
Picture taken 22.02.2019. Attempts to model the OpenID Connect Hybrid flow.



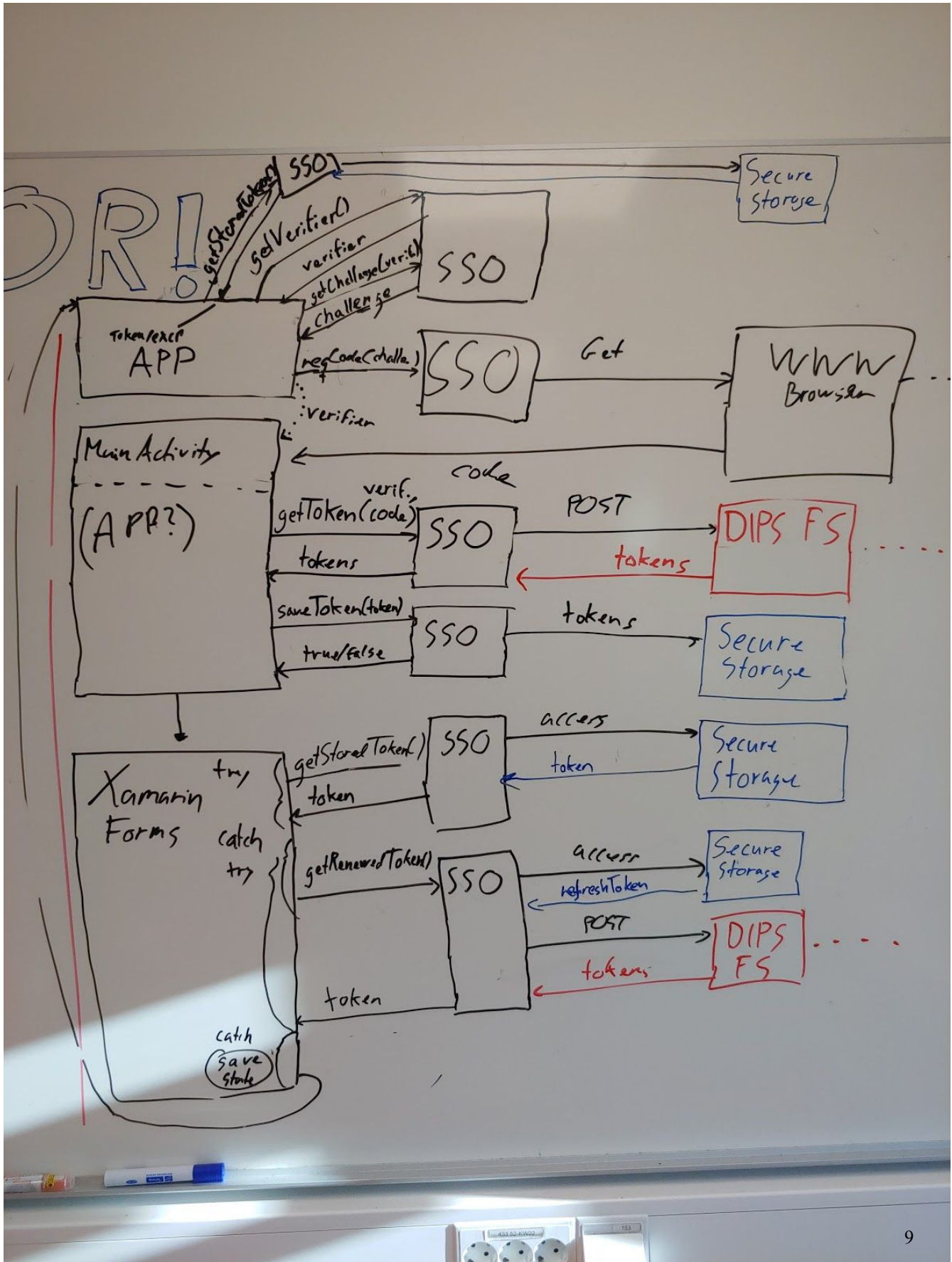
Picture taken 26.02.2019. A more complete model of the OpenID Connect Hybrid Flow with PKCE

The Middleware bois present:

BACHELOR!

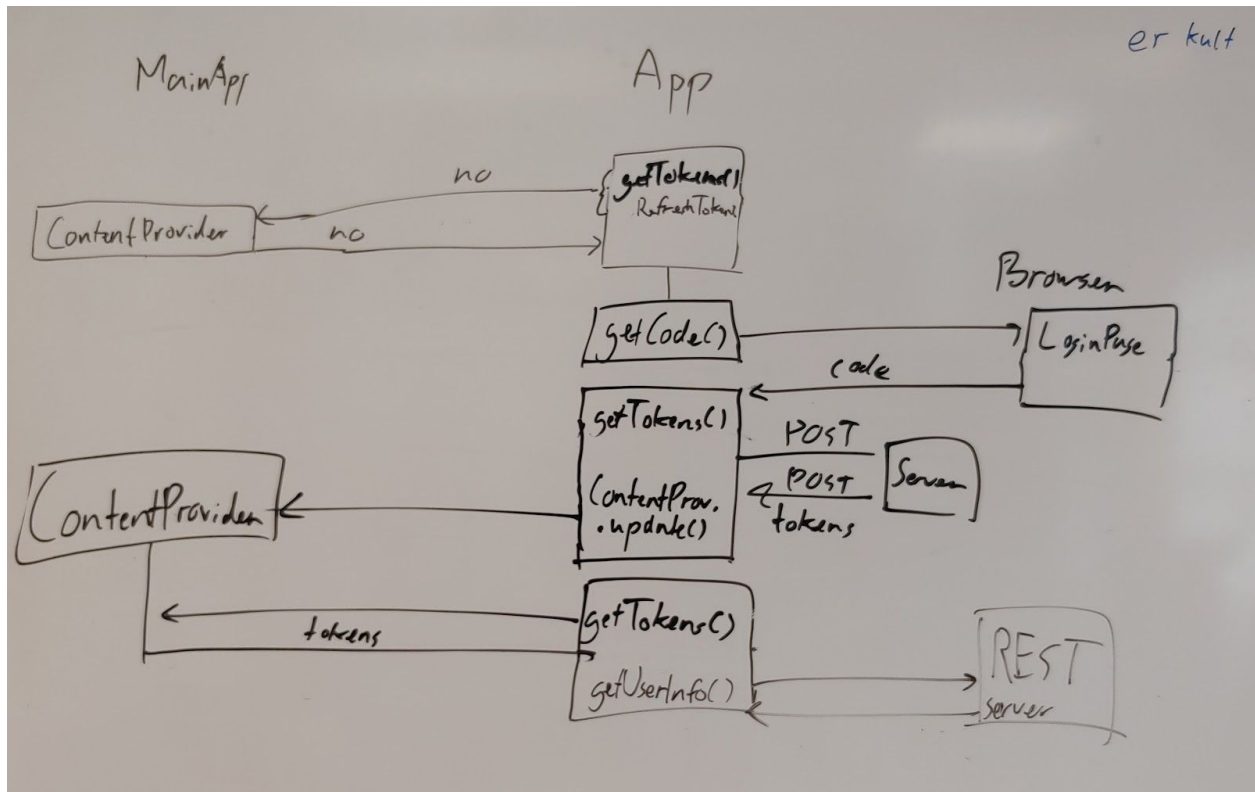


Picture taken 15.03.2019. Tries to model persistent token use.



3.2 Token storage and sharing

Picture taken 28.03.2019. Models storing tokens in one central app. (Something we decided against later)



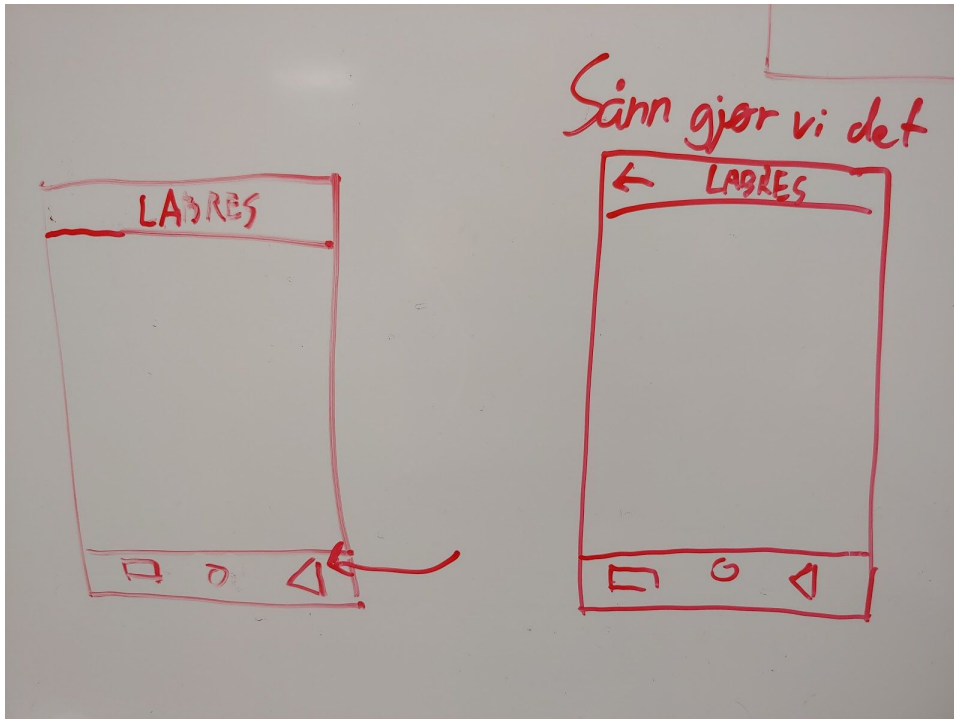
4. Visual prototypes

Our SSO implementation provides a service to developers. We do not provide any UI elements, and thus we will not be prototyping the user interface for our SSO implementation. During the project we will be making mock-applications to test functionality, and will possibly make some slightly more advanced mock apps to be used for user testing. But as these are prototypes in themselves, we will not be prototyping them earlier.

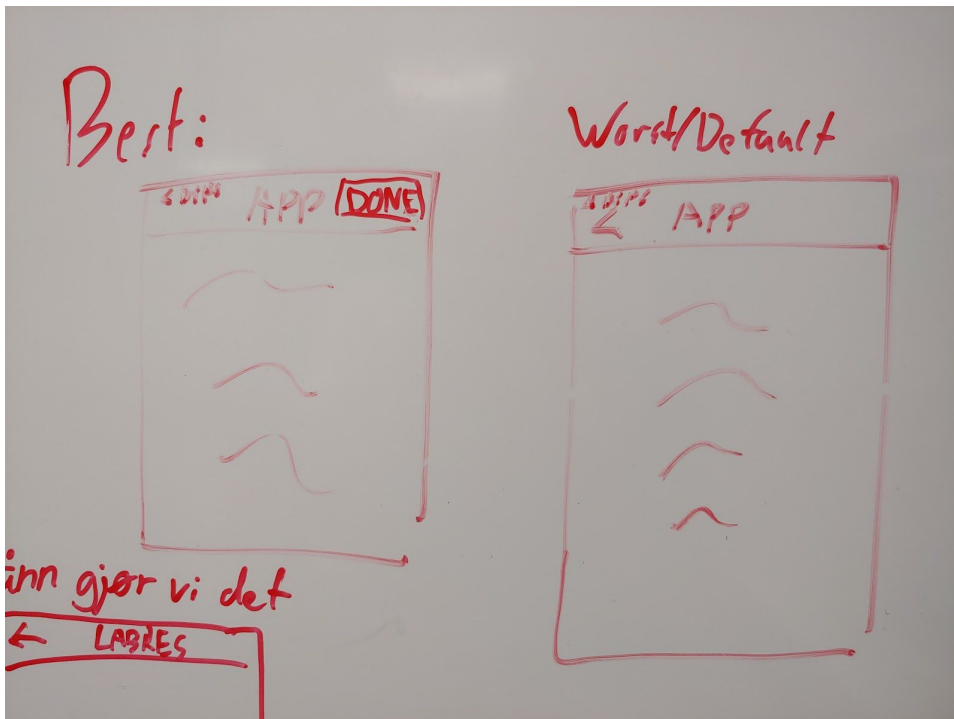
Edit 19.05.2019:

Some visual prototypes were made for the user testing applications, seen below.

Android:



iOS:



Appendix J

System documentation

Bachelor thesis
System documentation

Version <1.0>

Revision history

Date	Version	Author
<20/05/19>	<1.0>	Erling Moxnes Kristiansen

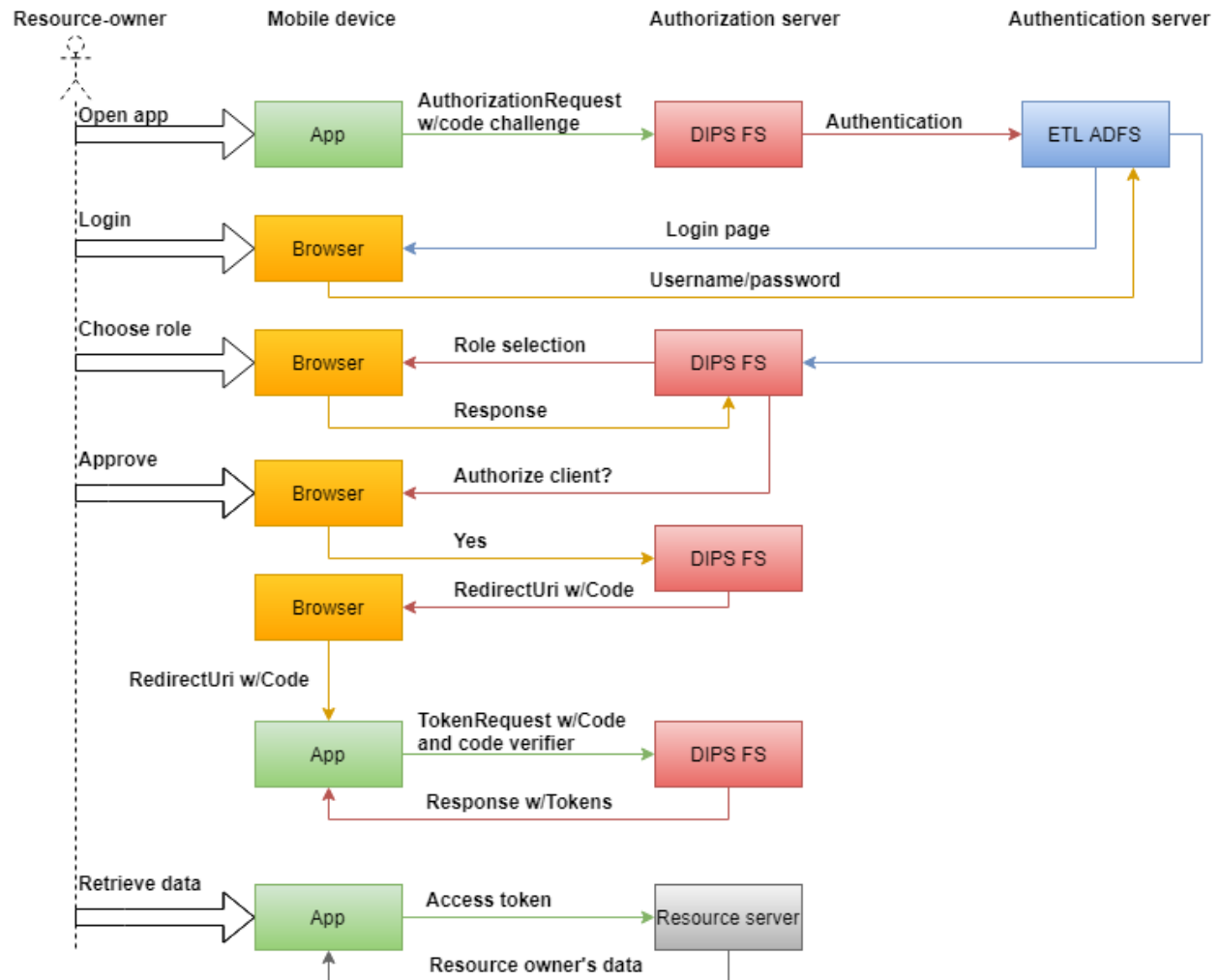
Contents

1.	Introduction	3
2.	Architecture	3
3.	Project structure	3
4.	Class diagrams	5
5.	Usage	7
6.	References	8

1. Introduction

This document contains information about the library we have developed as a part of our bachelor thesis and models of how it interacts with different servers and the apps. For additional information see the main report.

2. Architecture

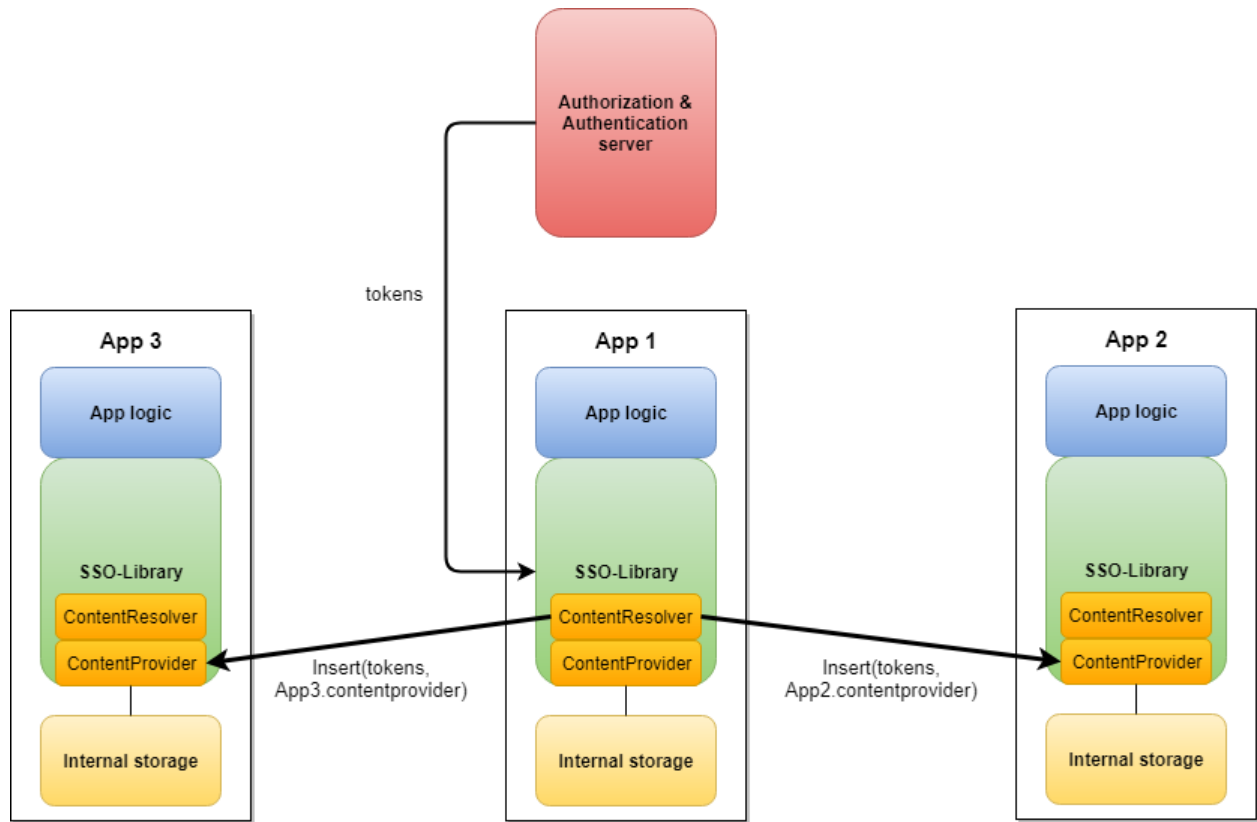


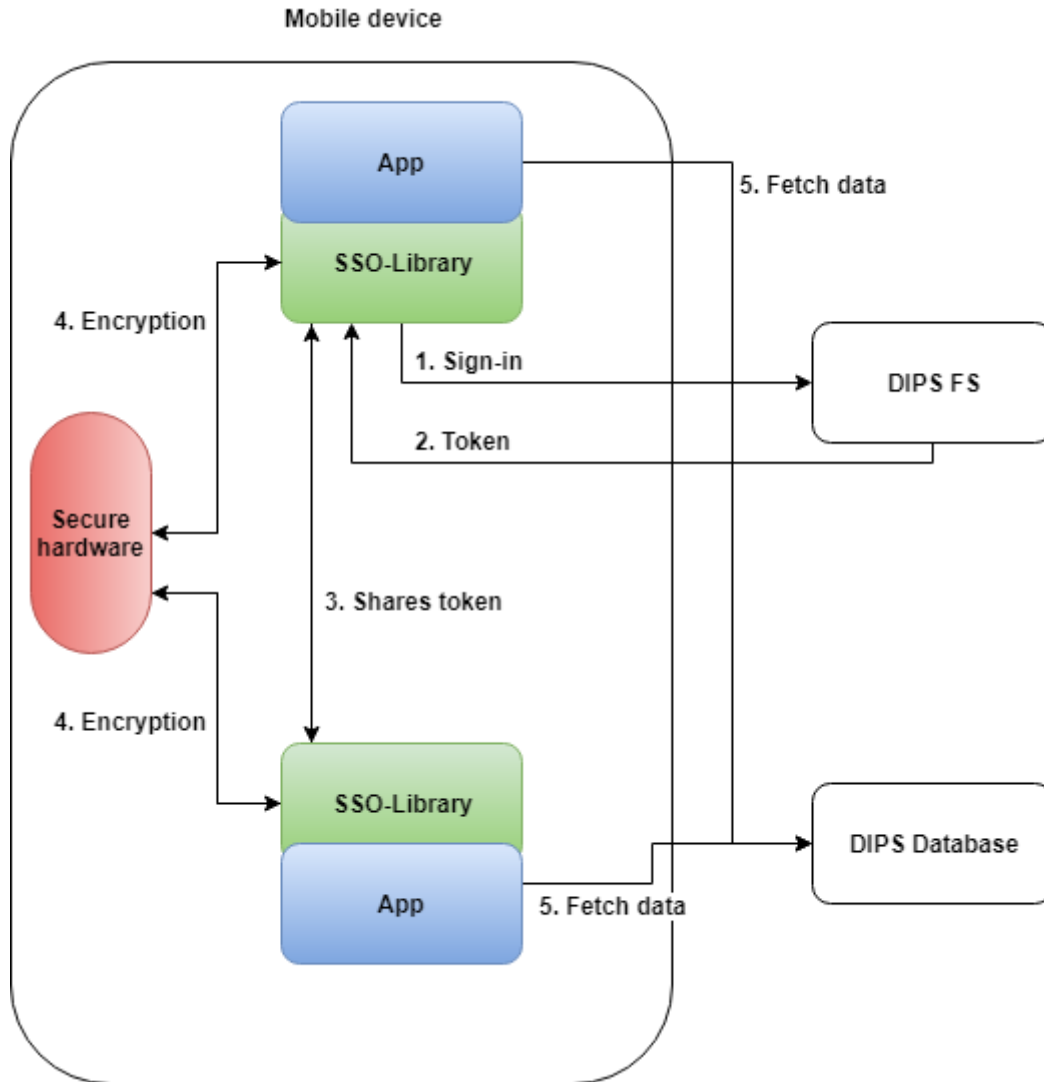
A flow diagram of the authentication and authorization process we used in our apps. The hybrid flow defined in the OpenID Connect specification is used. DIPS FS and ETL ADSF are servers made and maintained by DIPS ASA.

3. Project structure

The library simply consists of one folder for the shared code and one folder each for iOS and Android specific code. There are no additional subfolders.

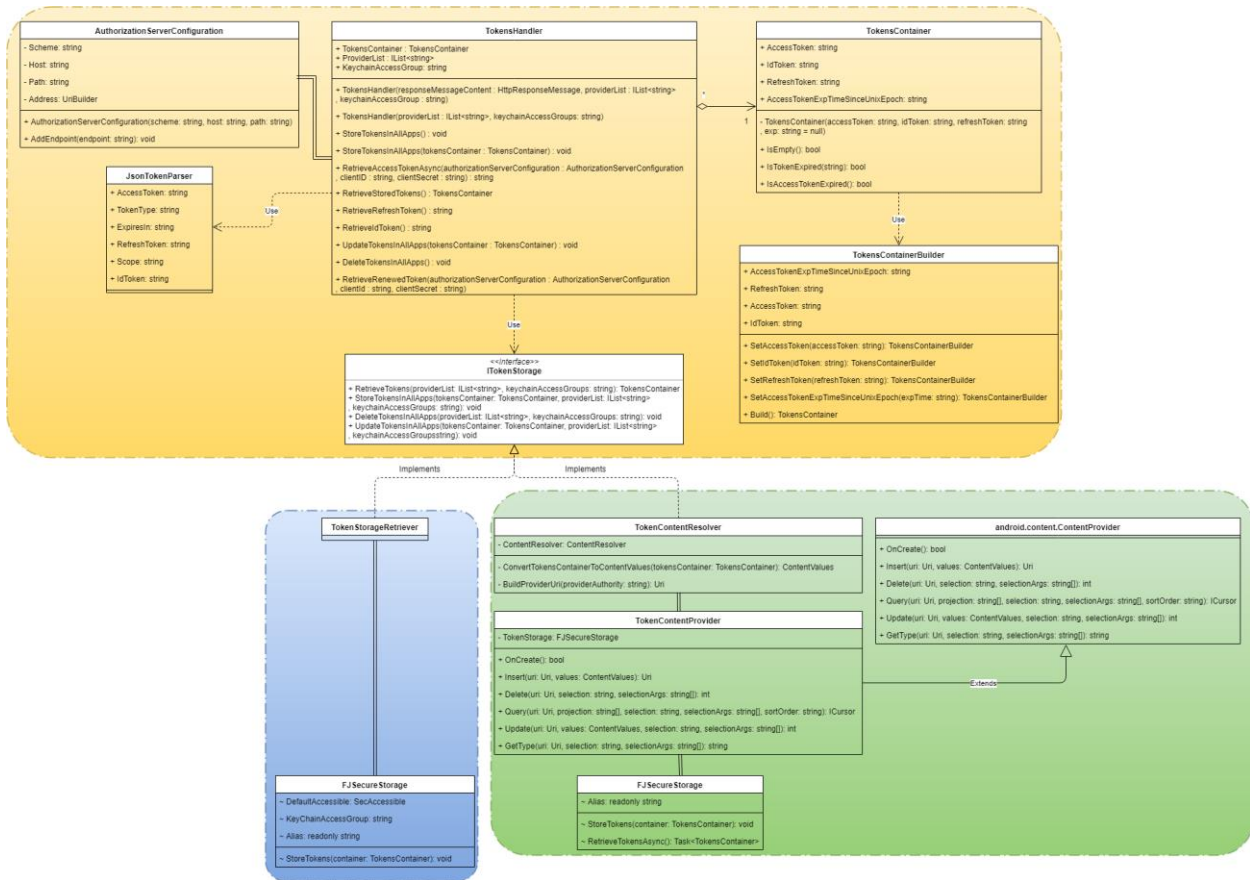
Here are two examples of how the library interacts with several Android apps during the process of sign-in, sharing, encryption and storage.



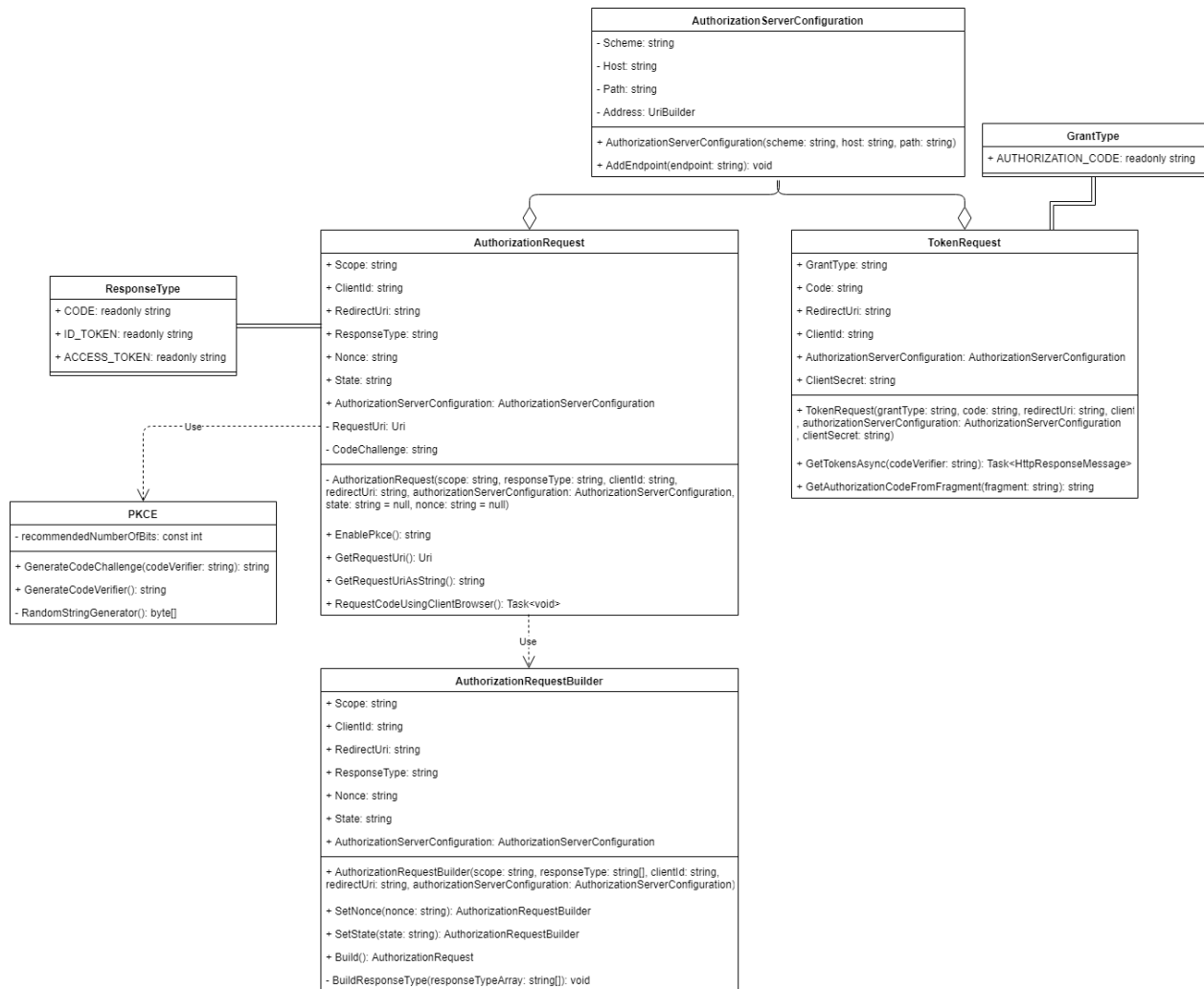


4. Class diagrams

A class diagram for the components of the library that handles the encryption, storage and sharing of the tokens on a mobile device. The yellow part is code shared by both iOS and Android, the blue part is iOS specific code and the green part is Android specific code.



A class diagram of the components of the library that functions as an OAuth2.0 client. The two main classes AuthorizationRequest and TokenRequest are modeled to respectively target the Authorization endpoint and the Token endpoint defined in the OAuth2.0 RFC 6749.



5. Usage

The library is currently not made into a dll file. Running it would require copying the folders in the lib folder from GitHub. Placed the “Shared” folder in the shared code of a Xamarin.Forms solution, the “iOS” folder in the iOS project and the “Android” folder in the Android project, and then follow these instructions to use it:

1. iOS

To be able to open the app after a successful login, the OpenURL method in AppDelegate.cs must be implemented. To enable sharing of the keychain, certain steps has to be taken. These steps are described on these websites <https://docs.microsoft.com/en-us/xamarin/ios/deploy-test/provisioning/capabilities/?tabs=macos> <https://docs.microsoft.com/en-us/xamarin/ios/deploy-test/provisioning/entitlements?tabs=macos>

2. Android

To enable sharing of data you have to go through a few steps:

3. Define a new permission in the manifest of one the apps: [assembly: Permission(Name = "com.dips.app.providerpermission", ProtectionLevel = Android.Content.PM.Protection.Signature)]
4. Every app must ask to use this permission. This is done by adding this to the manifest: [assembly: UsesPermission(Name = "com.dips.app.providerpermission")]

Every app MUST be signed with the same certificate.

The providerList parameter required by TokensHandler must be a list of strings where each string is an app's applicationId with ".tokenprovider" appended. Like so: + ".tokenprovider". Every app which needs to share data

must be listed.

6. References

GitHub repository can be found here: <https://github.com/Sankra/xamarin-forms-ss0>

Appendix K

Diagrams

K.1 Class diagrams

K.2 Flow diagram

K.3 Different design stages

K.3.1 OAuth client designs

K.4 Test apps

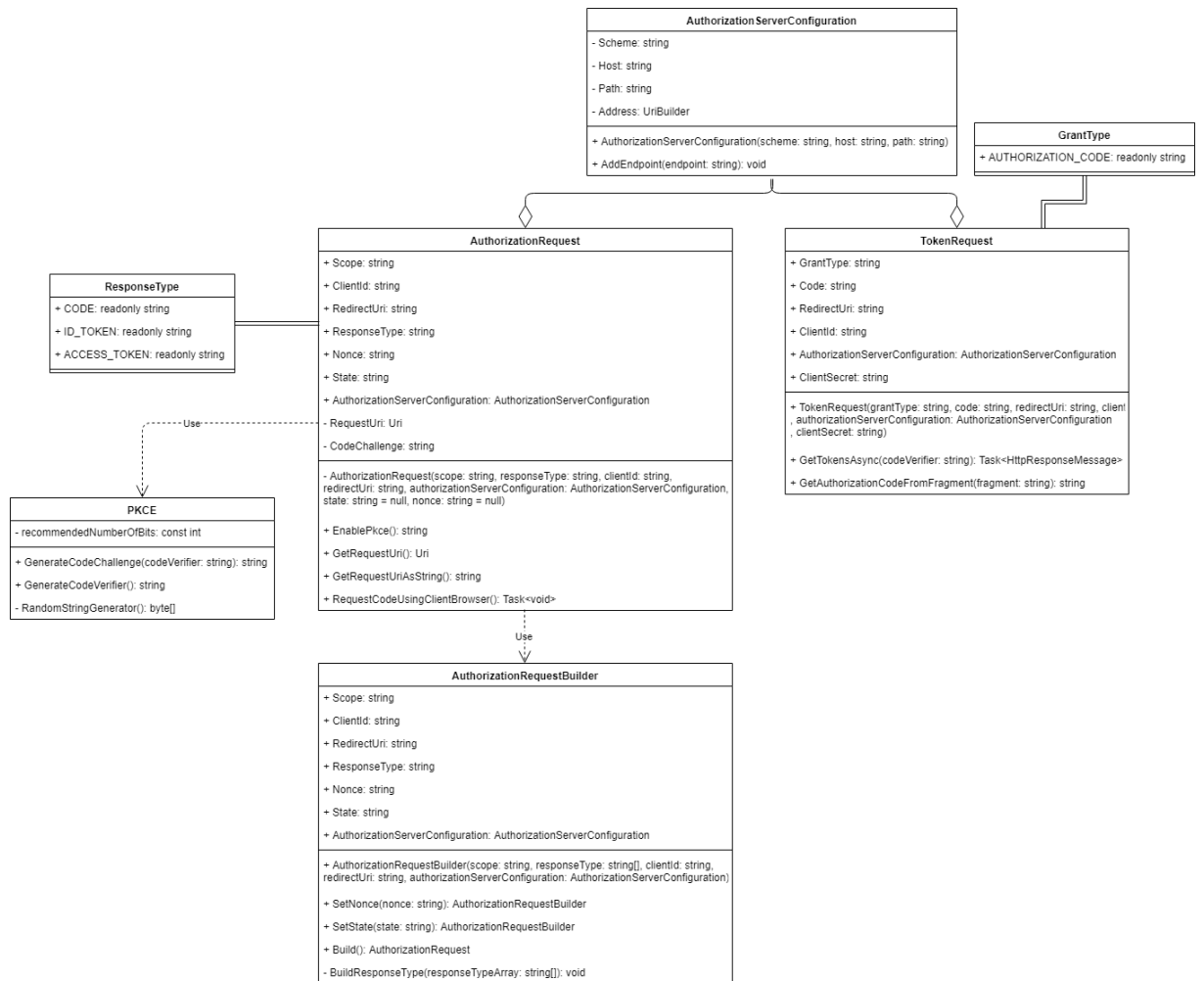


Figure K.1: A complete class diagram of the OAuth2.0 client which is included in the SSO-library.

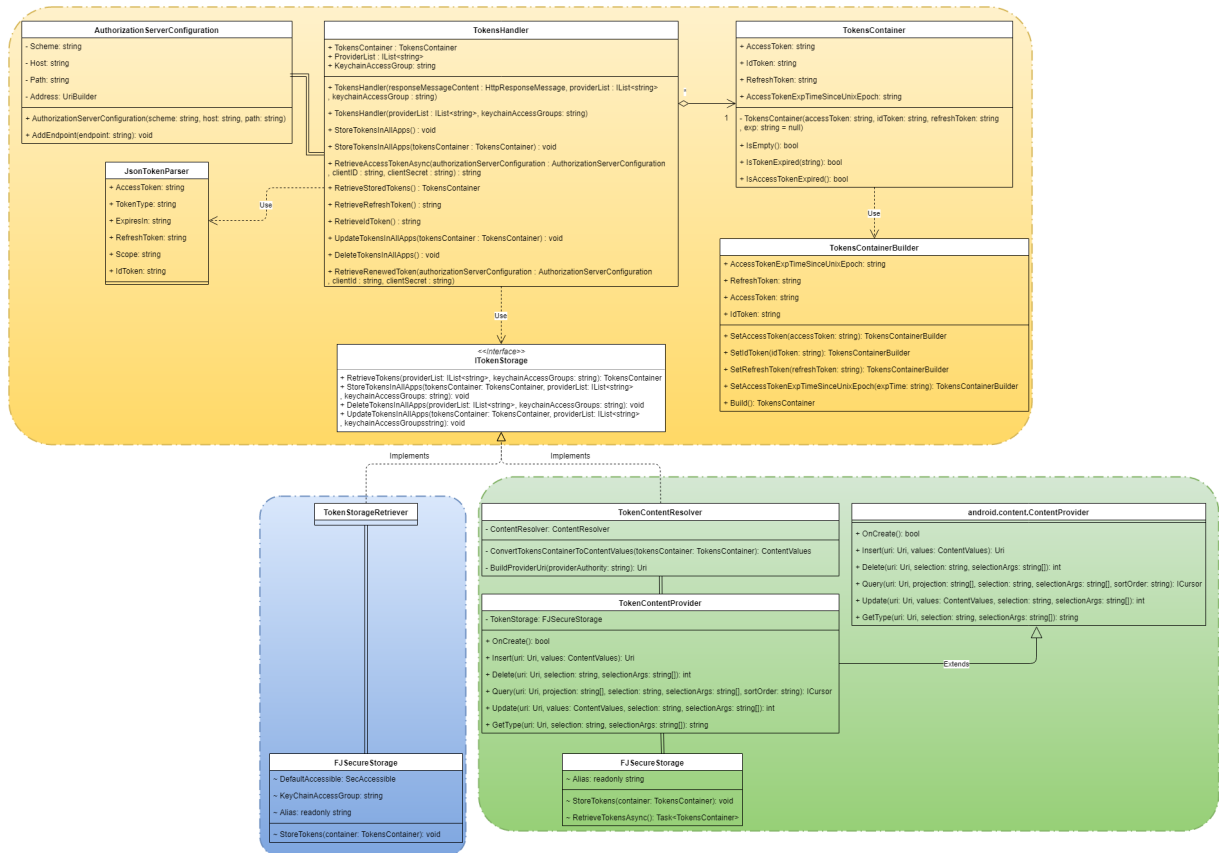


Figure K.2: A complete class diagram of the token encryption, storage and sharing components of the library. Areas marked with yellow is shared code for both iOS and Android, the blue area is iOS specific and the green area is Android specific.

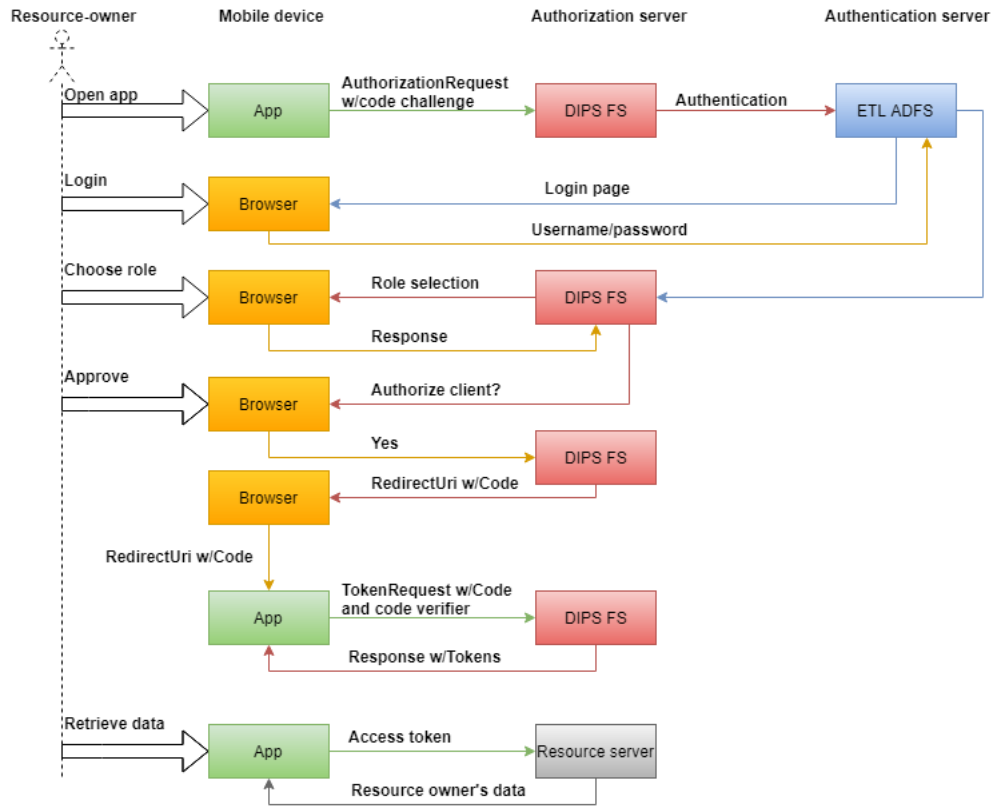


Figure K.3: Flow-diagram of the hybrid flow authentication and authorization we used with DIPS' servers.

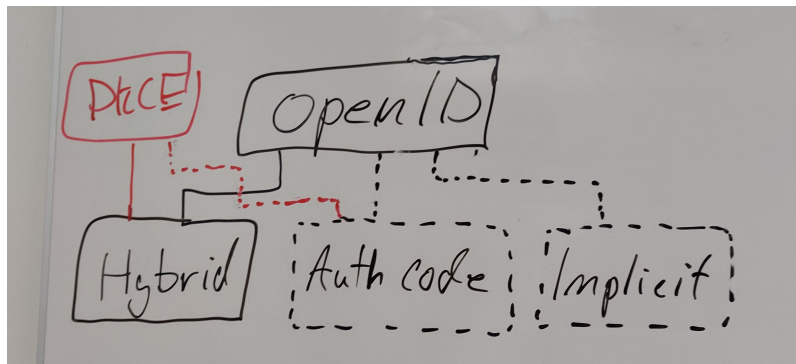


Figure K.4: A design suggestion for the OAuth 2.0 client.

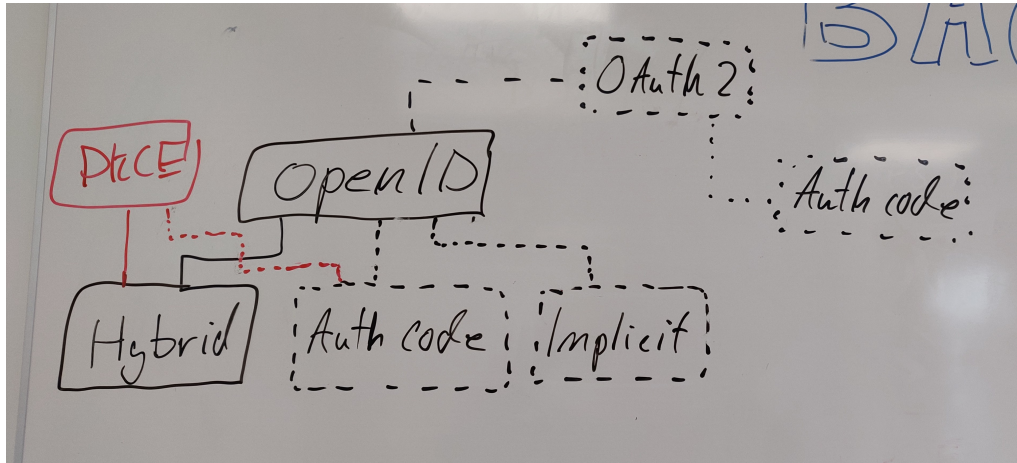


Figure K.5: The design suggestion for the OAuth 2.0 client expanded.

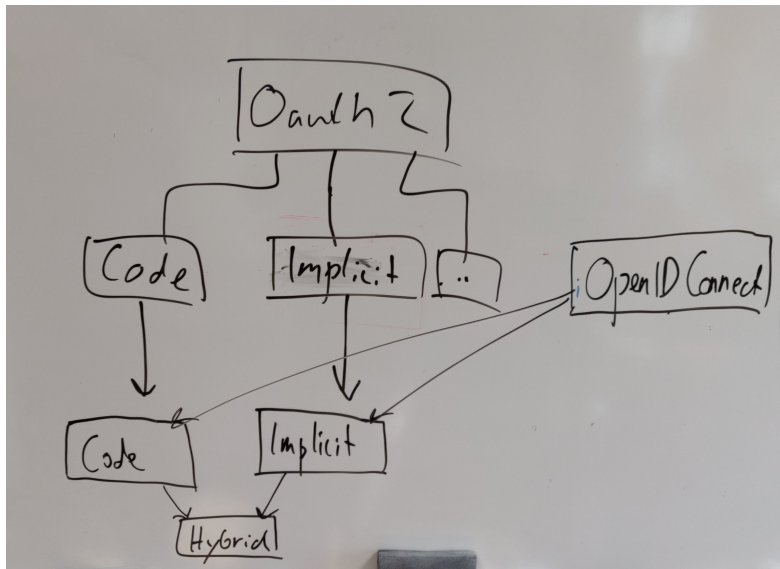


Figure K.6: Another design suggestion for the OAuth 2.0 client.

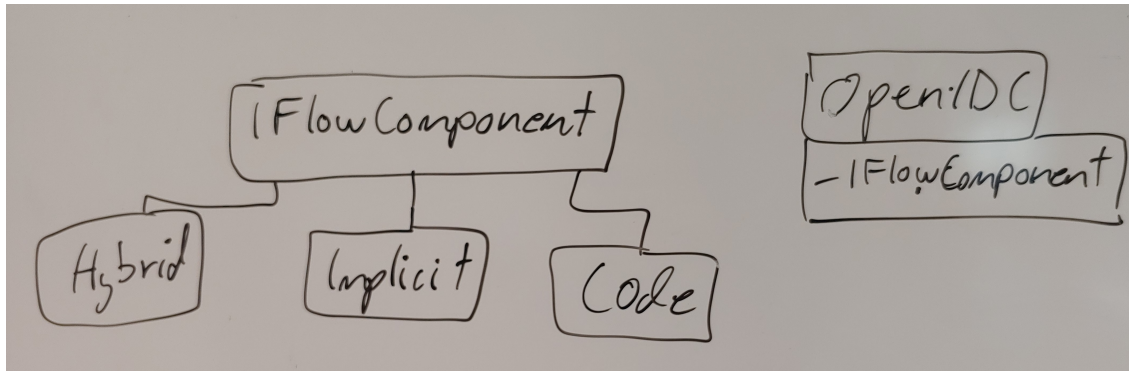


Figure K.7: We considered using the Component pattern to build the flows.

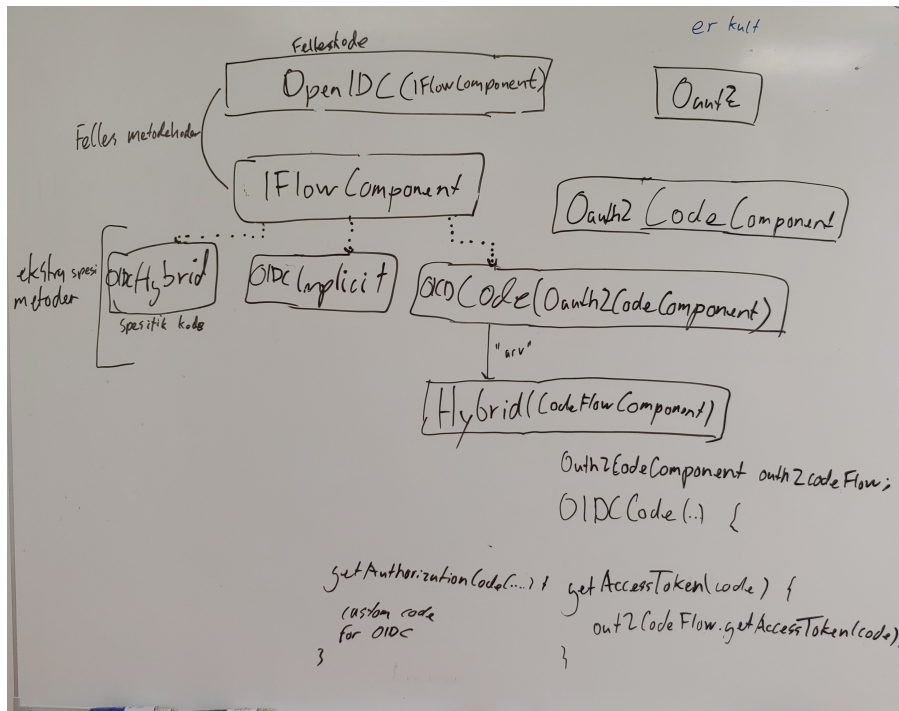


Figure K.8: We considered using the Component pattern to build the flows.

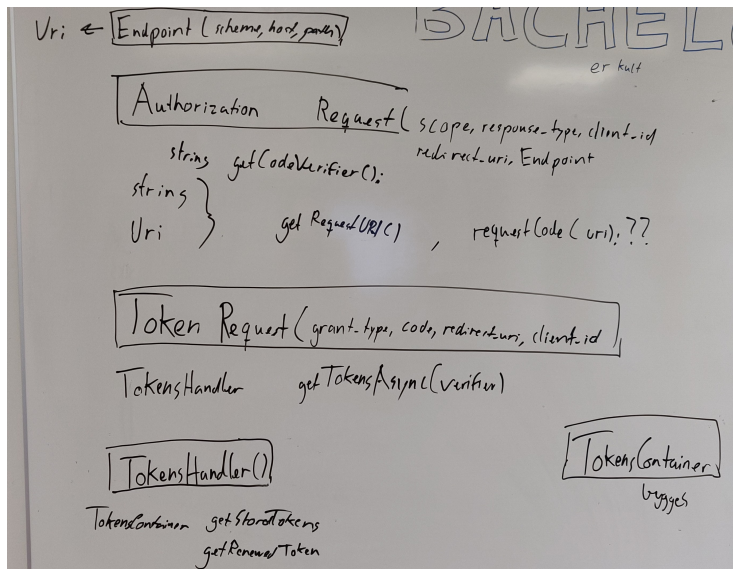
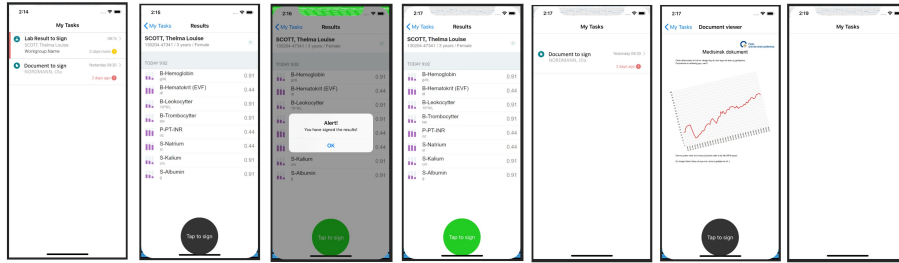
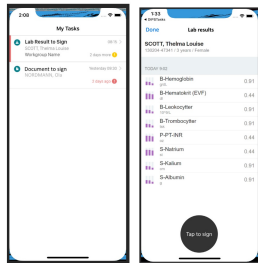


Figure K.9: A sketch of the final OAuth 2.0 client design.

BigApp iOS



3 små iOS

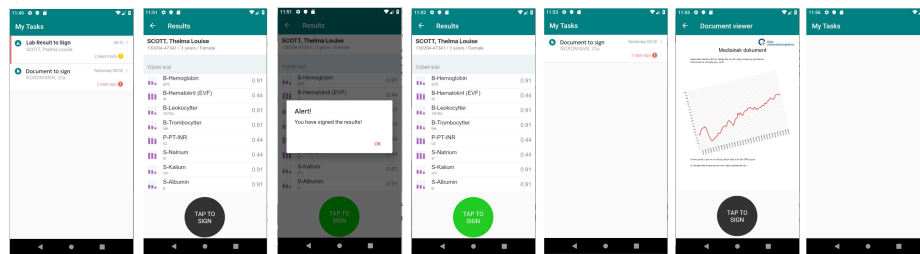


Samme som over

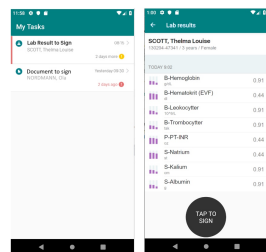


Samme som over

BigApp Android



3 små Android



Samme som over



Samme som over

Figure K.10: The flow and design of the test apps.

Appendix L

User testing

L.1 Test plans

L.1.1 Test plan for non-doctors

Testplan for ikke-leger

Planlegging

Hvor?

Enten på kontoret til DIPS, ute hos bruker eller sekundært over skype.

Når?

Avtales direkte med personene eller med hjelp fra utviklingsleder hos DIPS Trondheim, Runar.

Hvordan?

1. Testene gjennomføres og tester fyller ut et skjema per test
2. Til slutt spørsmål om testene.

Formålet med testen

Vurderer om en bruker fra målgruppen behersker navigeringen mellom mer enn én app. Er det forståelig? Er det en hindring?

Hvem?

Hvem som helst som ikke er leger.

Nødvendig utstyr for testene:

Test i samme fysiske lokasjon

- En måte å ta opp lyden.
- En av våre pre-oppsatte Android eller iOS-telefoner
 - Operativsystem avhenger av hva testeren bruker til vanlig.
- Veldig ønskelig med funksjonalitet for skjermopptak. Lar seg ikke gjøre på iOS, da skjermopptakknappen fjerner tilbake-til-forrige-app-knappen. Har derfor et kamera pekt på telefonen for å ta opp knappetrykk.

Moderert fjerntest:

- Datamaskin med Skype, Google Hangouts eller lignende for å ta opp lyd og kommunisere
- Testeren må ha en mobiltelefon vi kan fjern-installere apper på.

Øvrige spørsmål

- Skal vi bruke gestures eller tradisjonelle knapper? La bruker velge.

Oppgavebeskrivelse

Brukeren skal se over og signere både noen testresultater og et dokument. Annenhver bruker begynner med den store appen.

Vi tester én stor app vi har laget mot en løsning med flere små apper som vi har laget.

Gjennomføring

Inledning til hele opplegget:

Velkommen! I dag skal vi teste brukervennligheten til appene vi utvikler som en del av bacheloroppgaven vår for DIPS. Du skal teste to forskjellige løsninger, og etter hver test vil du bli stilt noen korte spørsmål og til slutt fylle ut et kort spørreskjema. Intervjuet vil bli tatt opp slik at vi kan skrive det ned etterpå. Er det noe du lurer på før vi starter?

Belønning

Gis med en gang slik at testpersonen ikke føler hen må bligjøre testerne for å få belønning. Alle testere får en liten sjokolade som takk for at de var med.

Selve testen:

Estimert tidsbruk per person (ikke inkludert opprigg og omrigg):

Ca. 30 minutter

Innledning del 1:

Testen innledes med en fast innledning:

3 små apper:

“Løs de to oppgavene du får presentert. Gjerne tenk høyt underveis, ved å si f.eks. ting du lurer på, hva du leter etter, etc. Det er systemet vi tester, ikke deg. Vi er ikke så interessert i tilbakemelding på oppgavene, de er bare midlertidige, men heller hvor intuitive systemet er i bruk.”

1 stor app

“Løs oppgavene du får presentert. Gjerne tenk høyt underveis, ved å si f.eks. ting du lurer på, hva du leter etter, etc. Det er systemet vi tester, og ikke deg. Vi er ikke så interessert i tilbakemelding på oppgavene, de er bare midlertidige, men heller hvor intuitiv systemet er i bruk.”

Innledning del 2 (felles):

Vi ønsker å gjøre opptak av testen for å få mer nøyaktig data av hvordan appene brukes. Det består av følgende:

- Skjermopptak på mobilen,
- Lyd-opptak.

Skjermopptaken er anonyme. Lydopptakene transkriberes og slettes innen 24 timer. Du skal løse oppgavene som er presentert på skjermen.

Har du noen spørsmål?

Testpersonale lukker appene på mobilen.

Praktisk gjennomføring:

Etter innledning og eventuelle spørsmål fra testeren startes kamera, skjermopptak og video-opptak. Notatbok-appen skal åpnes og testerens tester-ID skal skrives inn. Så starter man skjermopptak. Når testeren verifiserer at den er klar får den en ulåst test-mobil med appen åpnet i hendene. Testen har da begynt.

Avslutning test 1:

Når brukeren har løst de to oppgavene er testen over. Testeren blir informert om at testen er over og at de har gjort en strålende jobb. Testpersonale lukker appene på mobilen.

Spørreskjema

Testeren blir nå bedt om å fylle ut et spørreskjema.

Spørreskjema ÉN app:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNURDRXQzMzUkMwREYzOFMyOEK5TzZYWVdUSS4u>

Spørreskjema TRE apper:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNUNDJaU0VZNjiNODM0S1BOT0FQTK1IQUZOQy4u>

Overgang til å teste den andre løsningen

Hvis testerene har testet 3 små apper skal den nå teste én stor app, og omvendt.

“Du skal nå teste en annen versjon av samme system. Det meste vil være likt. Vi vil etter testen spørre deg om å sammenligne systemene.”

Testing av den andre løsningen

Foregår på samme måte som testing av den første løsningen.

Spørreskjema for den andre løsningen.

Spørreskjema ÉN app:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNURDRXQzMzUkMwREYzOFMyOEK5TzZYWVdUSS4u>

Spørreskjema TRE apper:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNUNDJaU0VZNjINODM0S1BOT0FQTK1IQUZOQy4u>

Intervju

Testerene blir nå spurt noen spørsmål angående systemet de har testet.

- Hva synes du om appene?
- Hva var vanskeligst?
- Hva var lettest?
- Hvordan fungerte det å navigere mellom visningene i appene?
- Merket du en forskjell i navigeringen mellom den første og andre testen?
- Har du forbedringsforslag eller tilbakemeldinger?
- Noe annet du tenkte på?

Spørreskjema etter at begge testene er fullført

Spørreskjemaet består av 3 deler, der de to første har identiske spørsmål og burde nå være utført. Det siste spørreskjemaet handler om dem og består av følgende spørsmål:

- Aldersgruppe
- Kjønn
- Stilling/Yrke
- På en skala fra 1 til 5, hvor teknologisk interessert er du?
- På en skala fra 1 til 5, hvor god er du til å bruke teknologi?

Link:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNUMkIxVDE2VFBYMU1HRFQ0M001VkZGRVdUQy4u>

Våre praktiske oppgaver etter testen.

Noter eventuelle problemer. Skill mellom observasjon, tolkning og løsningsforslag.

1. Hvordan utfører du de to oppgavene til vanlig?

Transkriber opptakene til en mappe.

L.1.2 Test plan for doctors

Testplan for leger

Planlegging

Hvor?

Enten på kontoret til DIPS, ute hos bruker eller sekundært over skype.

Når?

Avtales med hjelp fra utviklingsansvarlig på DIPS Trondheim, Runar.

Hvordan?

Begynner med et intervju, deretter testen og til slutt spørsmål om testen.

Formålet med testen

Vurderer om en bruker fra målgruppen behersker navigeringen mellom mer enn én app. Er det forståelig? Er det en hindring?

Hvem?

Leger er førsteønsket. Terje på DIPS kan fungere godt for en pilottest.

Nødvendig utstyr for testene:

Test i samme fysiske lokasjon

- En måte å ta opp lyden.
- En av våre pre-opsatte Android eller iOS-telefoner
 - Operativsystem avhenger av hva testeren bruker til vanlig.
- Veldig ønskelig med funksjonalitet for skjermopptak. Hvis det ikke lar seg gjøre må vi ha et kamera pekt på telefonen for å ta opp knappetrykk.

Moderert fjerntest:

- Datamaskin med Skype, Google Hangouts eller lignende for å ta opp lyd og kommunisere
- Testeren må ha en mobiltelefon vi kan fjern-installere apper på.

Øvrige spørsmål

- Skal vi bruke gestures eller tradisjonelle knapper? La bruker velge.

Oppgavebeskrivelse

Brukeren skal se over og signere både noen testresultater og et dokument. Annenhver bruker begynner med den store appen.

Vi tester én stor app vi har laget mot en løsning med flere små apper som vi har laget.

Gjennomføring

Inledning til hele opplegget:

Velkommen! I dag skal vi teste brukervennligheten til appene vi utvikler som en del av bacheloroppgaven vår for DIPS. Du skal teste to forskjellige løsninger med tilhørende spørreskjema. Før det skal vi stille deg to spørsmål om ditt mobilbruk, og etter testen vil du bli stilt noen korte spørsmål og til slutt fylle ut et kort spørreskjema. Intervjuet vil bli tatt opp slik at vi kan skrive det ned etterpå. Er det noe du lurer på før vi starter med intervjuet om mobilbruk?

Innledende intervju: (med lydopptak)

1. Kan en telefon være et nyttig arbeidsverktøy for deg?
2. I hvilke kontekster kan det være aktuelt å bruke en telefon?

Selve testen:

Estimert tidsbruk per person (ikke inkludert opprigg og omrigg):

Ca. 30 minutter

Innledning:

Testen innledes med en fast innledning:

3 små apper:

“Løs de to oppgavene du får presentert. Gjerne tenk høyt underveis, ved å si f.eks. ting du lurer på, hva du leter etter, etc. Det er systemet vi tester, ikke deg. Vi er ikke så interessert i tilbakemelding på oppgavene, de er bare midlertidige, men heller hvor intuitive appene er i bruk.”

1 stor app

“Løs oppgavene du får presentert. Gjerne tenk høyt underveis, ved å si f.eks. ting du lurer på, hva du leter etter, etc. Det er systemet vi tester, og ikke deg. Vi er ikke så interessert i tilbakemelding på oppgavene, de er bare midlertidige, men heller hvor intuitiv appen er i bruk.”

Innledning del 2 (felles):

Vi ønsker å gjøre opptak av testen for å få mer nøyaktig data av hvordan appene brukes. Det består av følgende:

- Skjermopptak på mobilen,
- Lyd-opptak av dine reaksjoner og meninger.

Skjermopptaken er anonyme. Lydopptakene transkriberes og slettes innen 24 timer. Du skal løse oppgavene som er presentert på skjermen.

Testpersonale lukker appene på mobilen.

Praktisk gjennomføring:

Etter innledning og eventuelle spørsmål fra testeren startes kamera, skjermopptak og video-opptak. Notatbok-appen skal åpnes og testerens tester-ID skal skrives inn. Så starter man skjermopptak. Når testeren verifiserer at den er klar får den en ulåst test-mobil med appen åpnet i hendene. Testen har da begynt.

Avslutning test 1:

Når brukeren har løst de to oppgavene er testen over. Testeren blir informert om at testen er over og at de har gjort en strålende jobb. Testpersonale lukker appene på mobilen.

Spørreskjema

Testeren blir nå bedt om å fylle ut et spørreskjema som er felles for de to løsningene.

Spørreskjema ÉN APP:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNURDRXQzMzUkMwREYzOFMyOEK5TzZYWVdUSS4u>

Spørreskjema TRE APPER:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNUNDJaU0VZNIjINODM0S1BOT0FQTK1IQUZOQy4u>

Overgang til å teste den andre løsningen

Hvis testerene har testet 3 små apper skal den nå teste én stor app, og omvendt.

“Du skal nå teste en annen versjon av samme system. Det meste vil være likt. Vi vil etter testen spørre deg om å sammenligne systemene.”

Testing av den andre løsningen

Foregår på samme måte som testing av den første løsningen.

Spørreskjema for den andre løsningen.

Er det samme som for den første løsningen.

Spørreskjema ÉN app:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNURDRXQzMzUkMwREYzOFMyOEK5TzZYVWdUSS4u>

Spørreskjema TRE apper:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNUNDJaU0VZNjlnODM0S1BOT0FQTK1lQUZOQy4u>

Intervju

Testerene blir nå spurt noen spørsmål angående systemet de har testet.

- Hva synes du om appene?
- Hva var vanskeligst?
- Hva var lettest?
- Hvordan fungerte det å navigere mellom visningene i appene?
- Merket du en forskjell i navigeringen mellom den første og andre testen?
- Har du forbedringsforslag eller tilbakemeldinger?
- Noe annet du tenkte på?

Spørreskjema etter at begge testene er fullført

Spørreskjemaet består av 3 deler, der de to første har identiske spørsmål og burde nå være utført. Det siste spørreskjemaet handler om dem og består av følgende spørsmål:

- Aldersgruppe
- Kjønn
- Stilling/Yrke
- På en skala fra 1 til 5, hvor teknologisk interessert er du?
- På en skala fra 1 til 5, hvor god er du til å bruke teknologi?
- På en skala fra 1 til 10, hvor mye ønsker du å ha mobil som arbeidsverktøy i helsevesenet?

Link:

<https://forms.office.com/Pages/ResponsePage.aspx?id=cgahCS-CZ0SluluzdZZ8BUNq8wuhSbVGvecK8r4WjmNUN1pBWFdTUTQxUDJINvc3Q04xQTVSV002Ni4u>

Belønning

Alle testere får en liten sjokolade som takk for at de var med.

Våre praktiske oppgaver etter testen.

Noter eventuelle problemer. Skill mellom observasjon, tolkning og løsningsforslag.

1. Hvordan utfører du de to oppgavene til vanlig?

Transkriber opptakene til en mappe.

L.2 Consent form and non-disclosure agreement

Samtykke og taushetserklæring

Jeg deltar frivillig i brukervennlighetstesting av nytt navigasjonssystem for DIPS AS sine apper. Som testbruker har jeg rett til å avbryte testen når som helst uten begrunnelse. Jeg og testens data blir anonymisert, og mine personalia og kontaktinformasjon slettes innen 20. April 2019. Dersom jeg ønsker det, har jeg rett til å få slettet eventuelle opptak før det. Som kompensasjon for deltakelse mottar jeg en liten sjokolade.

Samtykke til opptak

Under testen vil det bli gjort opptak av mobilskjermen. Samt lydopptak av meg og kamera-opptak av at jeg gjennomfører testen (utsnitt av mobil og hender med fokus på hva jeg trykker på). Jeg samtykker til at disse opptakene kan brukes til brukervennlighetsanalyse, og jeg fraskriver meg herved alle rettigheter til opptaket. Lyd-opptak vil transkriberes og slettes innen 24 timer av at testen er gjennomført. Skjermopptak og eventuelt opptak av mobilskjerm og hender slettes senest 20. April.

Taushetserklæring

Den informasjon og kunnskap om systemet som jeg tilegner meg, erklærer jeg herved at jeg ikke skal dele med andre.

Navn

Sted/dato

Signatur

L.3 Test reports

April 2019

Testrapport lege testperson 2

Bachelor DIPS våren 2019

Testleder: Erling Moxnes Kristiansen
Testobservatør: Truls Matias Torgersen
Kameraoperatør: Toni Vucic
Transkribent: Toni Vucic

Testperson nr.: 2

Produkt testet: Versjon 2.0 av BigApp, LabResults, DocumentViewer og DIPSTasks.
Test-enhet: Truls sin iPhone XR
Operativsystem: iOS 12.2

Opplevdes navigasjonen forskjellig?

Notater fra observatør: Tester sa hen merket forskjell etter ledende spørsmål. Sa nei først.

Øvrige kommentarer: Truls jobbet med å få appene på iOS og det ble derfor ikke tatt noen notater. Toni mente det ikke skjedde noe utenom det vanlige. Erling nevnte at han ble avbrutt av Truls i del 2 av introduksjonen.

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
Nei	01:28	00:21	iOS 12.2	N/A	done	done	up-nav	up-nav	ja	25	40

Transkribert intervju og observasjoner:

Innledning:

1. Kan en telefon være et nyttig arbeidsverktøy for deg?
"Ja"
 - a. Har du noen eksempler på hva du...?
 - i. "Jeg bruker det til å lese mail, daglig, universitetsmail men ikke sykehusmail, den får jeg ikke på telefon."
2. I hvilke [andre] kontekster kan det være aktuelt å bruke en telefon?
 - a. "Ja det kunne vært for eksempel at du overførte røntgenbilder når man er på vakt"
 - b. "Du slipper å logge deg inn på den fryktelige PCen som sykehuset stiller med hjemme"

Test 1 (Én stor)

Fikk spørsmål fra tester om dette er til helseplattformen før testen gikk i gang. Det var det ikke, da DIPS er konkurrent.

Erling ble avbrutt i innledningen av at testeren fikk en telefonsamtale. Samt at Toni tok litt styringen. Eneste innledning gitt var: "Da skal du løse oppgavene", sagt av Toni

Testen starter

DIPSTasks:

"Da skal jeg løse, den øveste først da sikkert?"

tester tapper Lab result to Sign

LabResults:

Toni: "Bare tenk høyt hvis du lurer på noe eller synes noe er forvirrende"

"Hemoglobin på 0,91, da er du død. Hematokritt på 0,84 det er jo greit. Skal vi se..."

Kan ikke noe å trykke på der nei."

tester prøver å trykke på blodtestresultatene

"Det der er jo helt vilt, det der er helt vilt, og det der er helt vilt"

tester peker på de forskjellige blodresultatene mens hen snakker

"Ja, det er noen kommafeil her tror jeg"

Toni: "Innholdet er ikke det aller viktigste, det er mer navigasjonen vi tester"

tester prøver igjen å trykke på et blodprøveresultat

tester trykker på Tap to sign

"Tap to sign"

alerten kommer opp

"Oops, nei det her går ikke an"

**tester prøver å trykke utenfor alerten* (Vi gjetter at det er for å få den bort)*

"kan ikke signere alt det her"

April 2019

tester trykker ok i alerten

"Jeg har nå signert nå da, men jeg har ikke uttalt meg noe om det"

Toni: "Nei.."

tester trykker Tap to sign igjen, får opp alert, trykker ok igjen

"Det gjør det ja"

tester trykker Tap to sign igjen, får opp alert, trykker ok igjen

"ja"

tester trykker My Tasks

"da skal jeg gå tilbake"

DIPSTasks:

tester trykker Document to Sign

"dokument"

DocumentViewer:

tester tilter telefonen opp fra bordet og er stille en stund

det kan virke som hen leser dokumentet

"Hm"

tester trykker Tap to sign, så ok

trykker på My Tasks-knappen

DIPSTasks:

"Yes, da var jeg ferdig?"

Toni: "Mhm"

tester trykker på My Tasks-tittelen

"Ikke noe mer da?"

Erling: "Det var det"

Mellom testen:

"Okei, dere kommenterer ikke noe videre hvorfor de resultatene var helt ville??"

Erling: "Nei, det er bare navigasjonen som vi tester"

"Åja, okei"

Erling: "Resultatene er sånt sett ikke relevante, men vi har et skjema hvis du bare kunne..., i forhold til den testen"

Test 2 (små apper)

DIPSTasks:

"Thelma Louise igjen ja"

Trykker Lab Result to Sign

LabResults:

Trykker Tap to sign, så ok

trykker Done

DIPSTasks:

trykker Document to Sign

DocumentViewer:

trykker tap to sign

“poenget at jeg lærer av å ha gjort det før sikkert?”

trykker ok

trykker Done

DIPSTasks:

trykker på My Tasks-tittelen

“Daså?”

Toni: “Mhm, ja”

Erling: “Det var det”

Etter testen

“Du mener det var framgang?”

Toni: “Det gikk jo fortere :)”

“De var ikke noe bedre de prøvesvarene”

Toni: “Hehe nei”

“Pasienten er fortsatt død”

Alle ler

Avsluttende intervju

- Hva synes du om appene?
 - “Jeg synes at det hele var veldig forvirrende fordi jeg skjønnte ikke hva jeg skulle gjøre med de prøvesvarene. For vanligvis bruker vi å gjøre noe annet enn å signere dem hvis det er såpass, hvis de er sykelige. Hvis der er helt OK så bare signerer vi.”
- Hva var vanskeligst?
 - “Vanskeligst var å skjønne hva jeg skulle gjøre med det.”
- Hva var lettest?
 - “Lettest var jo å signere det”
- Hvordan fungerte det å navigere mellom visningene i appene?
 - “Det var greit, helt ok når en først skjønnte hva en skulle gjøre med svarene, ikke skulle gjøre mener jeg.”
- Merket du en forskjell mellom navigeringen fra den første testen og den andre testen?
 - **“Nei, faktisk ikke”**
 - Erling: “Ingenting?”
 - **“Nei, Jaaa, ikke annet enn at det gikk fortere, fordi at, ja, uuuh. Det sto nå Done øverst, det var det ikke på den første.”**
- Har du forbedringsforslag eller tilbakemeldinger?
 - “Når du ser på blodprøvesvar og du er i tvil om verdien så burde du trykke på svaret også får du opp referanseverdien, normalvariasjon, referanseverdiene. Det er jo et minstekrav. Ellers så... tja. Det kunne vært interessant å gå rett

April 2019

inn på pasientjournalen og lest om pasienten fra de svarene, og ikke bare se svarene. Så kan man lese seg opp på problemstillingen.”

- Noe annet du tenkte på?
 - **Dette spørsmålet ble ikke stilt.**

Testrapport lege testperson 3

Bachelor DIPS våren 2019

Testleder: Erling Moxnes Kristiansen
Testobservatør: Truls Matias Torgersen
Kameraoperatør: Toni Vucic
Transkribering: Toni Vucic

App-versjon: 1.0 (pga. stjernen som ikke er fjernet)

Testperson nr.: 3

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
Ja	00:19	00:34	Android 9.0	knapper	up-nav	up-nav	up-nav	up-nav	nei	70	62,5

Notater fra observatør:

Husk å spørre om de bruker android eller iPhone FØR vi starter!

Tror Erling glemte å si at vi ikke tester legens kunnskaper, men appen.

Har glemt å fjerne stjernen på Android

Tror kanskje det at vi sa at det var bra at hen fortsatte etter hen var ferdig oppfordret hen til å gjøre det samme på android-testen etterpå.

Tror ikke vi bør si høyt om det er én app eller flere.

(Tester lurer på om hen er konsistent nok på hva hen svarer på skjemaet)

Holdt på ekstra når hen egentlig var ferdig for å se om det var noe lureri eller noen "feller" som var der.

Øvrige kommentarer: Ingen

Transkribert intervju:

Innledning:

1. Kan en telefon være et nyttig arbeidsverktøy for deg?

- a. "Absolutt."
 - b. Erling: "Har du noen eksempler på når det kunne vært nyttig?"
 - i. "Det er jo forberedelse til den jobben man har. Lære nye teknikker. Få en gjennomgang av hva som har kommet inn av litteratur. Bruke den som et kunnskapsverktøy, og det er jo fote du trenger å se på en video, hvis du skal gjøre noe du ikke har gjort før, eller noe du har glemt. Kjempefint å bruke mobilen til alt mulig. Du kan bruke den til å snakke med kollegaer, det er noe som har overtatt for datamaskinen egentlig nå. Så alt du trenger å vite kan du stort sett finne på nettet via mobilen.
2. ~~I hvilke kontekster kan det være aktuelt å bruke en telefon?~~
3. "Er det noen spesifikke kontekster ellers i arbeidshverdagen din, der det kunne vært aktuelt å bruke en telefon? Der den kunne erstatte noe som fins?"
- a. "Tja det er jo alt som har med PC å gjøre kan du jo gjøre på en mobil. Det er jo skjermen som blir litt mindre, men man har jo blitt veldig vant til å bruke det formatet, sånn at jeg syns jo, hvis du ser på røntgenbilde, lese gjennom mailer, alt mulig er jo veldig greit på en telefon, for **den har du jo med deg og du slipper å logge på en ny PC**, sånne ting da. Men hvis man skal sette seg ned og bruke tid, så blir jo mobilskjermen litt liten, til å se på og fingranske. Eller sitte å planlegge noe som tar litt mer tid, så er kanskje skjermen litt liten. Man vil jo kanskje kunne bruke en mobil i fremtiden til å ta notater via talegjenkjenning eller ett eller annet sånt, hvis man er ute å traverer i gangene. I stedet for at man må stoppe opp, finne en maskin, logge deg på, finne pasienten, sånne ting vil jeg jo se for meg at fremtiden kanskje blir litt enklere.

Innledning til test 1.

Erling: "Denne gangen blir det 3 små apper Toni" ...

Test 1 (små apper, iOS)

DIPSTasks:

"Da skal jeg bare trykke på ett eller annet her, og se på..."

Trykker på Lab Result to sign

LabResults:

Prøver å trykke flere ganger på B-Hemoglobin.

Trykker tap to sign

Tar en liten pause på 1-2 sekunder

"You have already..."

Trykker ok

Flytter fingeren litt rundt uten å trykke på noe

"Der var det ikke så lett å skjønne hva man..."

Trykker på B-Hematokrit

Trykker på symbolet til venstre for B-hematokrit

Trykker på B-Hematokrit igjen

Trykker tittelen til resten av resultatene fra øverst til ned og nederst til opp
Trykker Tap to sign igjen

Trykker ok

Trykker Done

DIPSTasks:

“Nå forsvant det.”

Trykker Document to sign

DocumentViewer:

Prøver å pinch-zoome på bildet

Trykker Tap to sign

Trykker ok

Trykker Done

DIPSTasks:

“Hva gjør man nå da?”

“Nå er jeg litt sånn rådvill hvor jeg går videre i den her”

Erling: “Det var kun de to oppgavene”

Nå er egentlig testen over, men vi lar tester fortsette.

Trykker på iOS back-nav-knappen som sier Document...

DocumentViewer:

Trykker Done

DIPSTasks:

Trykker på iOS back-nav-knappen som sier Document...

DocumentViewer:

Trykker Tap to sign

Trykker ok

Trykker Done

DIPSTasks:

“Sånn at det, det var ikke så mye mer spennende å se på den der”

Trykker på iOS back-nav-knappen som sier Document...

DocumentViewer:

Gjør swipe-up gesture og havner på app-hjemskjermen

App-Hjem-Skjermen:

Åpner DocumentViewer

DocumentViewer:

Trykker Done

DIPSTasks:

Venter litt

Trykker på iOS back-nav-knappen som sier Document...

DocumentViewer:

Trykker Done

DIPSTasks:

Trykker på iOS back-nav-knappen som sier Document...

DocumentViewer:

Gjør swipe-up gesture og havner på app-hjemskjermen

App-Hjem-Skjermen:

Åpner LabResults

LabResults:

Trykker Done

DIPSTasks:

Trykker på iOS back-nav-knappen som sier LabResults

LabResults:

Gjør swipe-up gesture og havner på app-hjemskjermen

App-Hjem-Skjermen:

Åpner documentviewer

DocumentViewer:

Trykker Done

DIPSTasks:

Gjør swipe-up gesture og havner på app-hjemskjermen

App-Hjem-Skjermen:

Åpner BigApp

BigApp DIPSTasks:

Trykker Document to sign

BigApp DocumentViewer:

“Ja?”

“Nei jeg har ikke så mye mer å, jeg finner ikke noe mer spennende på de der”

Testen avbrytes.

Tidsbruk for hoved-delen av denne testen: 00:38

Etter Test 1

“Nå skal det sies at jeg aldri har brukt Apple-iPhone.

Truls: “Du bruker Android til vanlig?”

“Mhm”

Truls: “Da burde vi kanskje teste Android, det burde vi spurt om på forhånd”

“Jeg er helt Apple-dust”

Test 1 (små apper) (På nytt pga. feil operativsystem i starten), Android

DIPSTasks:

Trykker på Lab Result to Sign

LabResults:

Prøver å bla i test-resultatene

Trykker Tap to sign

Trykker ok

Skal til å trukke up-nav knappen men ombestemmer seg og trykker på stjerne-symbolet i stedet flere ganger

Trykker på up-nav-knappen

DIPSTasks:

Trykker Document to sign

DocumentViewer:

Prøver å pinch-zoom på bildet

Trykker tap to sign

Trykker ok

Trykker på up-nav-knappen

DIPSTasks:

Testereren har løst oppgavene men får ikke beskjed om det.

Venter litt

Trykker på back-nav-knappen

LabResults:

Trykker på back-nav-knappen

DIPSTasks:

(Nå har visningen i DIPSTasks blitt resatt til slik det var når den åpnet appen)

Trykker på multi-tasking-knappen

Multi-taskingsvisning:

Blar til LabResults

Blar til Notes

Blar til LabResults

Blar til DIPSTasks

Går inn i DIPSTasks

DIPSTasks:

Trykker Document to sign

DocumentViewer:

Trykker up-nav

DIPSTasks:

Trykker Lab Results

LabResults:

Trykker tap to sign

Trykker ok

Trykker up-nav

DIPSTasks:

“Var ikke så mye mer å gjøre nå. My Tasks, none”

Testen er over

Test 2 (Én app)

DIPSTasks:

Tapper Lab result to Sign

LabResults:

Prøver å tappe B-Haemoglobin

Prøver å scrolle i resultatene

Trykker tap to sign

Trykker ok

Trykker på up-nav

DIPSTasks:

Trykker med en gang på Document to sign

DocumentViewer:

Trykker med en gang å Tap to sign

Trykker ok

Trykker tap to sign

Trykker ok

Trykker på up-nav

DIPSTasks:

Trykker på back-nav

Havner på hjemskjermen

“Ja, signert, kvittert.”

Testen er over

Etter testen, på starten av SUS-undersøkelse:

“Det var ikke store forskjellen på det der da, spørsmålet er om jeg er konsistent nok i hva jeg svarer”

Avsluttende intervju

- Hva synes du om appene?
 - “Nei, syns og syns. Jeg skjønnte ikke helt hva jeg skulle gjøre. Jeg skulle signere noen dokumenter. Jeg hadde lyst til å vite litt mer om verdiene, trykker litt på de forskjellige stedene for å se om det lå noe mer info, noen kurve over tid, men den ga ikke så mye, det kom liksom ikke noen forklaring hvis du trykket på noe dødt. Var det bare dødt? Heller ikke så lett å se hvis du signerer noe så er det ofte greit å få opp noe, “you have checked” ett eller annet, du får en hake eller ett eller annet symbol som forsvinner når du trykker det bort igjen. Sånn at det satt liksom ikke noe spor når du trykket, du bare var, at du hadde signet det ut da. Men det var jo, helt oversiktlig og greit, og det e jo bare å spinne det rundt fingeren, men jeg vet jo søren hva jeg holdt på med. Jeg føler jo at det er en sånn der dobbel, paranoid beredskap.

Her er det noe lureri i den settingen av den appen. Nå skal vi testes på ett eller annet. Det er noe en ikke skjønner, man får litt den følelsen, men det er jo en test-setting da. Menmen, det skal jeg leve med. Hvis jeg er lurert opp i stry så skal jeg leve med det.”

- ~~Hva var vanskeligst?~~
- Var det noe du fant vanskelig?
 - “Nei, næh”
- ~~Hva var lettest?~~
- Var det noe du synes var lett?
 - “Sitte å trykke på en telefon og bla litt frem og tilbake, det, det er jo lett. Det var kanskje, det var liksom ikke så mye, det sto mine oppgaver, trykk på det, ferdig. Hvis det var oppgaven, så var det veldig lett. Hvis det er en skjult agenda, med noen skjulte bunner og greier, så blir det litt mer spennende.”
- Hvordan fungerte det å navigere mellom visningene i appene?
 - “Jeg synes jo ærlig talt at det ikke var den store forskjellen. Hvis jeg skal være helt ærlig så følte det ut som akkurat det samme, som at det var samme appen to ganger. Jeg så i hvertfall ikke noen forskjell på det. Det var jo akkurat det samme.”
- ~~Merket du en forskjell i navigeringen mellom den første og andre testen?~~
- Da er neste spørsmål om du merket noen forskjell kanskje litt unødvendig.
 - “Det gjorde jeg ikke”
- Har du forbedringsforslag eller tilbakemeldinger?
 - “Nei jeg synes jo det er veldig greit å se at du har vært et sted, at det kommer opp et grønt merke eller en kryss rubrikk som sier at du har signert her i stedet for at det bare kommer opp hvis du prøver å trykke på “you have already signed”, liker å sette spor. Det synes jeg i hvertfall er greit, og hvis man klikker flere ganger og ikke noe skjer, sånn at det kommer opp en eller annen forklaring på at det er ikke mere data rundt hemoglobinmålingen, det er ikke noe mer data rundt den kurven, du kan ikke gjøre noe mer, det er ikke noen vits å sitte å trykke. For av og til så sitter man og trykker også har man bare trykket litt skeivt på noe som er et aktivt panel eller ett eller annet sånt. Jeg vet ikke.”
- Noe annet du tenkte på?
 - “Nei. Litt spennende å vite hva dere egentlig testet?”

Testrapport lege testperson 4

Bachelor DIPS våren 2019

Testleder: Erling Moxnes Kristiansen
Testobservatør: Truls Matias Torgersen
Kameraoperatør: Toni Vucic
Transkribent: Ukjent og Toni Vucic

Testperson nr.: 4

Notater fra observatør:

Ikke si hva vi tester (altså "navigasjon mellom apper"). "hvor intuitive de er i bruk" er fint

Øvrige kommentarer:

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
nei	00:20	00:22	iOS 12.2	N/A	back-to-app	done	back	back	nei	85	85

Transkribert intervju:

Innledning:

1. Kan en telefon være et nyttig arbeidsverktøy for deg?
"Ja, absolutt. Felleskatalogen, tar en del bilder for dokumentasjon."
2. I hvilke kontekster kan det være aktuelt å bruke en telefon?
"Nei, ikke noe jeg har tenkt på, men det er det sikkert"

Test 1 (Én app)

Tester en

"Skal jeg bare begynne å trykke på det? Hva er det dere vil ha meg til å gjøre?"

"Men hva er oppgaven"

"Her, my tasks"

"Å signere laben, det er sikkert det"

"Det var den"

Går så inn på neste oppgave

“Det var den”
“Er det flere oppgaver?”

Test 2 (små apper)

DIPSTasks:

LabResults:



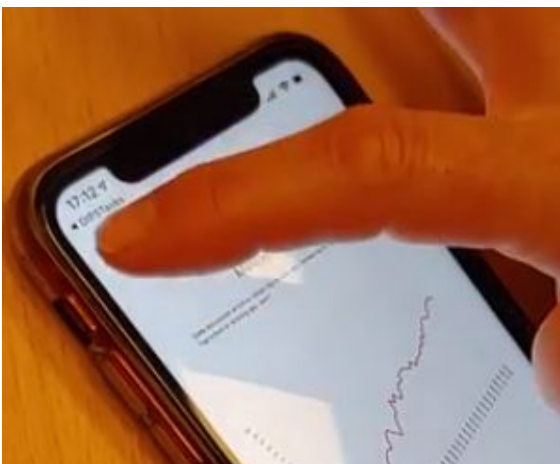
Tar seg litt tid før hen trykker på **Back-to-app**-knappen.
Kan virke som hen prøvde å trykke Done. Men siden hen trykket back-to-app neste gang hen var på LabResults tenker vi likevel det er mer sannsynlig at hen trykket back-to-app med vilje første gangen. Må også sees i sammenheng med intervjuet på slutten.

DIPSTasks:

DocumentViewer:

Signerer.

Tapper **Done**-knappen



DIPSTasks:

Vi stopper å time testen her.

Lab-results kommer tilbake pga. en navigasjons-bug.

Tapper lab result to sign

LabResults:



Tapper Back to App-knappen

Avsluttende intervju

- Hva synes du om appene?
 - “Jo altså det var, det var oversiktlige resultater på blodprøvene og enkelt å signere ut. Savnet på den andre, den der kurven at du kunne gjøre den litt større så du leste teksten litt bedre. For sånne gamle damer som meg så er det greit med større tekst. Men enkelt å signere ja.”
- Var det noe du fant vanskelig?
 - “Nei.”
- Var det noe du synes var lett?
 - “Ja, når jeg bare hadde skjont hva testen gitt ut på så var det jo lett.”
- Hvordan syns du det var å navigere mellom visninger i appene?
 - “Ja, det minner om andre ting man gjør, øverst i venste hjørne er ganske vanlig, intuitivt.”
- Merket du en noen forskjell mellom navigeringen i den første og den andre testen?
 - “Nei, jeg lurte egentlig på hva som var forskjellen. Jeg er ikke helt sikker hva det var.”
- Har du forbedringsforslag eller tilbakemeldinger?
 - “Nei, hvis man hadde muligheten til å lage bildet større, det tror jeg er det viktigste. Blodprøvesvarene var veldig store og tydelige.”
- ~~Noe annet du tenkte på?~~

Andre spørsmål:

Toni: “Jeg har et spørsmål. Jeg merket at ene gangen du brukte appene så trykket du “Done” og andre gangen trykket du på den svarte tilbake-knappen. Tenkte du noe på det?”

Tester: “Den ene gangen så kom jeg tilbake til bildet der jeg allerede hadde signert ut, da trykket jeg kanskje litt upresist”

Truls: “Det er egentlig en feil i appen vår som gjør at det blir sånn”

Tester: "Åja, for da tenkte jeg, hva trykket jeg nå? Men den kom opp en gang til, 'det her har jeg jo svart på'"

Toni: "Her har du de to mulighetene"

Tester: "Den var tasks ja, neeeei, nei jeg tror at den siste gangen så trykte jeg på den, og det var fordi at jeg trodde at jeg kanskje hadde trykket på den 'done' sist og det funket ikke for da fikk jeg den ikke tilbake. Så jeg tror kanskje..."

Toni: "Hva mener du med at du fikk den tilbake?"

Tester: "Så du ikke at jeg fikk labben en gang til?"

Toni: "Ah sånn ja"

Tester: "Så det var den siste gangen jeg fikk labben at jeg prøvde den andre for å få det bort."

Toni: "Okei"

Tester: "Jeg tror det har med det å gjøre."

Truls: "Det jeg lurte på var, for du sa at du, du nevnte en pil oppe i venste hjørne som du var veldig vanlig. Tenkte du på den svarte eller tenker du på en sånn..."

Tester: "Neei, jeg tror ikke at jeg tenkte så nøye på det, men om det er melleboken til Trondheim Kommune eller hva det er for noe så er 'Ferdig' øverst i det hjørnet. Så det er veldig naturlig å gå oppi der og trykke på en knapp."

Truls: "Ja så..."

Tester: "Jeg tror at jeg egentlig, dere har det jo på film, men jeg tror at jeg egentlig brukte den 'Done' men at jeg tror at jeg kanskje siste gangen prøvde den der [back-to-app] for å liksom, fordi den ene kom tilbake en gang til, eller det var tilfeldig. Jeg tror kanskje at det var derfor. Det var ikke noe jeg tenkte på der og da."

Toni: "Men du kom deg jo tilbake."

Tester: "Jeg gjorde jo det, og det fungerte jo."

litt snakk om hva vi testet

Tester: "Den første gangen var det bare én knapp, og den andre to eller?"

Truls: "Det var bare én knapp første gangen."

Tester: "Det var det, ja. Nei for jeg syns den blåe 'Done' er vel den mest naturlige å bruke"

Truls: "Mhm"

Tester: "Tror jeg. Den ligner mest på de knappene jeg er vant til å bruke"

Truls: "Ja riktig."

Tester: "For den var stor og den var tydelig, det andre var jo veldig smått, og oppe i hjørnet."

Testrapport lege testperson 5

Bachelor DIPS våren 2019

Testleder: Erling Moxnes Kristiansen

Testobservatør: Truls Torgersen

Kameramann: Toni Vucic

Transkribent: Toni Vucic

Testperson nr.: 5

Produkt testet: Testapper v. 2.0.1

Notater fra observatør:

Har ikke "vet ikke" knapp på skjema (gjelder spesielt "well integrated")

"Er det tilfeller der du vil vise ting til pasienten under visit" er vel ikke en del av manus? Helt greit at du går utenfor for oppfølging og utfylling.

Øvrige kommentarer:

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell ?	SUS stor app	SUS små apper
ja	00:24	00:43	iOS 12.2	N/A	done	done	back	back	nei	85	60

Transkribert intervju og observasjoner:

Innledning:

1. Kan en telefon være et nyttig arbeidsverktøy for deg?
 - a. "Ja."
 - b. Erling: "Har du noen eksempler?"
 - i. "Det er jo det allerede i dag, jeg bruker det som, jeg har jo langt inn oppslagsverk på det som jeg bruker. Jeg bruker av og til å ta bilder, google hvis det er ting jeg ikke kommer på i farten. Ja egentlig bruker jeg det til ett eller annet hele dagen."
2. ~~I hvilke kontekster kan det være aktuelt å bruke en telefon?~~

3. “Er det noen kontekster i dag der du kan se for deg at telefonen kan erstatte noe annet som du bruker?”
 - a. “Altså det er jo snakk om at man skal få journalene inn på telefonen for eksempel når man går visitt da. Det hadde vært greit å kunne slått opp røntgenbilder når du var inne på rommet, hatt alle journalnotater tilgjengelig, for eksempel. Men om det var min telefon eller en iPad eller hvasomhelst, det hadde jo egentlig vært, men i hvertfall hatt det tilgjengelig da.”

Test 1 (små apper)

DIPSTasks:

LabResults:

Trykker Done

DIPSTasks:

DocumentViewer:

Trykker Done

DIPSTasks:

Testen er egentlig ferdig nå og timer stoppes.

Sitter en stund og ser på skjermen.

Trykker til slutt på back-to-app-knappen

DocumentViewer:

Signerer og trykker **Done igjen**.

Test 2 (Én app)

DIPSTasks:

LabResults:

Back-button

DIPSTasks:

DocumentViewer:

Back-button

DIPSTasks:

Avsluttende intervju

- Hva synes du om appene?
 - “Det var jo, hvis du får en app hvor folk kan måle blodprøvesvar så er jo det kjempegreit. Det som dere burde hatt inn var jo normalverdiene, i appen, fordi, ja det er sånn at man ikke går ut og husker normalverdiene på alle blodprøver for eksempel, så det må være en mulighet å trykke på kjapt. Så det er jo greit. Ellers så var det helt okei.
- ~~Hva var vanskeligst?~~

- Var det noe du synes var vanskelig?
 - “Nei! Nei ikke når jeg skjønnte at det var bare at jeg skulle trykke så, nei det var ikke noe vanskeligere enn det, hehe. Jeg tror jeg skal få til det her.”
- ~~Hva var lettest?~~
- Var det noe du syntes var lett?
 - “Ja når jeg skjønnte hvordan det funket så var det helt lett å bruke den.”
- Hvordan fungerte det å navigere mellom visningene i appene?
 - “Nei det går jo kjapt det, hvis du bare skjønner, ser hvor du skal trykke hen. Så det gikk helt fint. Hvis dere lager en så enkel app så tror jeg nok at det går an å bruke den. Med noen små sånne ting og tang som man tenker at doktoren trenger, for eksempel normalverdi på blodprøver som ikke er i verden mulig å huske.”
- Merket du en forskjell i navigeringen mellom den første og andre testen?
 - “Nei, egentlig ikke så mye, nei.”
- Har du forbedringsforslag eller tilbakemeldinger?
 - Toni: “Kan jeg spørre en ting?”
 - **Tester:** “mhm?”
 - Toni: “Når du sa egentlig ikke så mye, vil det si at det var en forskjell der?”
 - **Tester:** “Nei jeg tenkte ikke på det. Jeg tror at, jeg så på det, mulig at det var en forskjell men det er jo ikke, jeg så på det mer som at, til å begynne med så var jeg bare litt sånn usikker på, altså om det var noe mer som lå i det. Eh ja, men når jeg fant ut at det var jo bare å trykke også var det ikke noe mer, da var det, det er jo ofte sånn at man tenker at det, ja, mhm.”
 - **Tester:** “Forslag til forbedringer forøvrig, nei den, hvis den er enkel og brukervennlig, det som er saken er at man må ha tilgang til den informasjonen man trenger i en app så man ikke trenger noe annet i tillegg, det er jo en sak. Også er det nok sånn at det er lurt å ha den kanskje på en større skjerm, egentlig. I hvertfall sånn hvis man bruker det i visitt-sammenheng og sånn så kan jo kanskje være egnet til en iPad eller noe sånn fordi at det blir litt liten skrift da. Altså hvertfall hvis folk har mobiler med, det er jo ikke alle som har sånn max-skjerm på 6 tommer eller sånn, så ville nok være mer hensiktsmessig å ha det på en sånn iPad eller en Huawei-skjerm hvisomhelst da, enn ett nettbrett, kanskje.
 - Erling: “Er det tilfeller der du trenger å vise noe til pasienten, når du er på visitt?”
 - **Tester:** “Ja, røntgen for eksempel er kjempegreit å kunne vise til pasientene, og da også er det bedre å ha en større skjerm. Men det vi gjør nå er at vi printer ut røntgenbildene og tar dem med opp til pasienten og viser dem i papirform, mens målet er å være elektronisk og uten papir. Også i nettbrettsform, kjempegreit. Men da må du ha tilgang til røntgenbildene og at de kommer opp smooth og kjapt. Og røntgen er jo et annet program, det er jo ikke integrert foreløpig i hverken DIPS eller doclive [?] i hvertfall. Men om det blir integrert inn i Epic det gjenstår jo å se da.
- ~~Noe annet du tenkte på?~~

Etter testen

Tester viser oss den nåværende mobilen de har på sykehuset som er en tykk Android-mobil med utdatert OS som er veldig treg på å vise nye data når det blir forespurt.

Testrapport ikke-lege testperson 6

Bachelor DIPS våren 2019

Testleder: Truls Matias Torgersen

Testobservatør: Toni Vucic

Transkribent: Truls

Dobbeltsjekk av tidsbruk: Toni

Testperson nr.: 6

Produkt testet: Versjon 2.0 av BigApp, LabResults, DocumentViewer og DIPSTasks.

Test-enhet: Erling sin OnePlus 6

Notater fra observatør: Testeren fikk se handlingsforløpet på skjermen til Truls 2. gangen hen fikk SU-skjema. Riktignok bare i 2-3 sekunder, men nok til å spørre “er det for min del det er split-screen?” (Spørreskjema og testforløp hadde 50% av skjermen hver)

Øvrige kommentarer: Ingen

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativ system	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell ?	SUS stor app	SUS små apper
nei	00:50	00:30	android 9.0	gestures	back	back	up	up	Nei	77,580	

Testens handlingsforløp med transkriberte svar

Test 1 (stor app) kommentarer

“Begynne øverst da.”

Gikk inn i LabResults

Gikk rett tilbake til DIPSTasks etter bare et sekund eller to

“Åja, sånn ja. Det er en oversikt over oppgaver.”

Gikk inn i LabResults igjen

Signerte, trykket ok, trykket up-navigation-knappen og havnet på DIPSTasks
"Okei, you have signed the results. Ingen bankID, ingenting."

Utsagnet under kom pga bildet brukte litt tid på å laste inn.

"Må jeg tappe en gang til? Nei."

Gikk inn i DocumentViewer

Leste alt i dokumentet ordrett.

"Okei, greit."

Signerte, trykket OK

"Flott."

Så på siden

"Men da er den grønn."

Trykket på signerings knappen en gang til. Alert popper opp.

"You have signed the results. Ok, greit."

Går tilbake til DipsTasks

"Kanskje gjort den knappen ikke trykkelig når jeg er ferdig trykka? Da er jeg tom for tasks."

Test 2 (små apper) kommentarer

"Ehh, ja."

Gikk inn i LabResults, signerte

Gikk tilbake med back-knappen.

Gikk inn i documentviewer med en gang.

Signert og gikk tilbake med back-knappen.

Kommentarer etter testene

"Jeg merka ikke forskjell på dem overhodet liksom."

Intervju etter test

1. Hva synes du om appene?

- a. "Jeg synes dem var veldig straightforward, det var jo ikke mye veldig mye jeg skulle gjøre. 1, 2, 3, 4, 5, 6 klikk? Men dem så neat ut. Som jeg sa, hvis den tap to sign knappen blir grønn og er fortsatt tap-bar, kanskje hatt en annen farge ikke vært tap-bar, ville det gitt mer mening, men hva vet jeg?"

2. Hva var vanskeligst og lettest?
 - a. "Vanskeligst var vel umiddelbart å få en oversikt over hva farger, varsler, labels og alt mulig betydde. Men det gikk seg til ganske fort. Lettest? Var vel å signere ting."
3. Hvordan fungerte det å navigere mellom visninger i appene?
 - a. "Ehh, visninger?" *Truls klargjør hva vi mener* "Greit, som i de fleste andre gode apper jeg kjenner til."
4. Merket du en forskjell mellom navigeringen innad i én app og navigeringen mellom flere apper?
 - a. "Ikke i det hele tatt."
5. Har du forbedringsforslag eller tilbakemeldinger?
 - a. "Nei."
6. Noe annet du tenkte på?
 - a. "Tror ikke det."

Ekstra kommentarer til slutt

Truls: Du sa det var noen ekstra millisekunder på bytting når det var tre apper istedenfor en.

Person: Jeg tenkte vel over det første gang og, men jeg hang meg så gæli opp i den knappen. Men det var mer umiddelbar respons første gangen.

Truls: Ja ja, riktig. Det er litt sånn rart siden du sa du ikke merka no forskjell egentlig, i det hele tatt, men når du tenker over det gjør du det.

Person: Merkelig nok falt det ikke meg inn at det skulle være et måle kriterium.

Toni: Det er jo bra for våres del. For hvis du hadde merka det underbevisst kunne det kommet fram på dem forskjellige spørreskjemaene. Det kan hende.

Person: Men der igjen var jeg så perpleks i at dem var så lik i opplevelsen, at ja. Men jeg vet jeg svarte forskjellige på dem to skjemaene. Men jeg vet ikke, var kanskje fordi jeg nyanserte litt.

Testrapport ikke-lege testperson 7

Bachelor DIPS våren 2019

Testleder: Erling Moxnes Kristiansen

Testobservatør: Toni Vucic

Transkribent: Toni Vucic

Testperson nr.: 7

Produkt testet: Versjon 2.0 av BigApp, LabResults, DocumentViewer og DIPSTasks.

Test-enhet: Erling sin OnePlus 6

Notater fra observatør: Erling sa det var bra at hen ikke merket forskjell på testene, før han begynte å spørre sammenligningsspørsmålene. Ikke ideelt.

Øvrige kommentarer:

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell ?	SUS stor app	SUS små apper
ja	00:32	01:02	android 9.0	knapper	up-nav	up-nav	up-nav	up-nav	nei	75	77,5

Transkribert intervju & observasjon:

Test 1 (små apper) kommentarer

Bruker et par sekunder til å se DIPSTasks

LabResults:

Ser over, signerer LabResults, leser alerten høyt, prøver å scrolle opp,

trykker på up-navigasjonknappen

DIPSTasks:

Trykker på Sign Document-knappen

DocumentViewer

Ser over grafen, prøver å trykke og pinch-zoome på grafen, viser at teksten er for liten til

testeleder, signerer, trykker ok,

trykker på up-navigasjonknappen

DIPSTasks:

Sier det er tomt for oppgaver, trykker likevel back-knappen nederst.
Testen er egentlig over nå.

LabResults:

Alle oppgavene er her igjen siden state ikke har blitt tatt vare på.
trykker på up-navigasjonknappen

DIPSTasks:

Testen avsluttes.

Test 2 (stor app) kommentarer

DIPSTasks:

Trykker LabResults

LabResults:

Signerer

up-navigasjon

DIPSTasks:

Trykker Sign Document

DocumentViewer:

Signerer

up-navigasjon

DIPSTasks:

Testen er ferdig.

Etter test

“Men jeg så ikke noen forskjell mellom versjon 1 og versjon 2”

Erling: “Nei, det er bra, vi kan vise deg, forklare deg hva forskjellene er når vi er ferdige.”

Intervju etter test

1. Hva synes du om appene?
 - a. “Dette var veldig enkelt. Jeg trodde det skulle være noe mer komplisert, med litt flere detaljer.”
2. Hva synes du var vanskeligst?
 - a. “Det var ingen ting som var vanskelig.”
3. Hva var det som var lett?
 - a. “Det var lett å skjønne at dette var blodverdier til pasienten og at jeg skulle underskrive at jeg har sett dem. Den andre, der var det litt små bokstaver. Det er eneste. Men jeg kunne lese det hvis jeg anstrengte meg. Men det er ikke sikkert jeg ville gidde hvis det var mange sånne.”

4. Hvordan fungerte det å navigere mellom visninger i appene?
 - a. "Veldig enkelt, ikke noe problem."
5. Merket du en forskjell mellom navigeringen innad i én app og navigeringen mellom flere apper?
 - a. "Nei, for meg så de helt identiske ut."
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. "Bortsett fra størrelese på teksten, ikke noe annet."
7. Noe annet du tenkte på?
 - a. "Langt spørreskjema etter en kort oppgave."

Post-test briefing:

Tester har blitt informert om hva som ble testet, og hvordan det var små apper i den ene testen og en app i den andre.

"For meg, jeg så ikke noen forskjell"

Testrapport ikke-lege testperson 8

Bachelor DIPS våren 2019

Testleder: Truls

Testobservatør: Erling

Testperson nr.: 8

Kommentarer: Har sett en versjon av systemet før.

Produkt testet:

Notater fra observatør:

Øvrige kommentarer:

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
nei	2:12	00:31	Android 9.0	Knapp	up-nav	up-nav	up-nav	up-nav	nei	82,5	82,5

Transkribert intervju og observasjoner:

Test 1 kommentarer

Viser usikkerhet over hva hen skal gjøre.

DIPSTasks: Leser over oppgavene. Går inn i labresults

LabResults: Leser over bildet. Signerer. Trykker ok. Går tilbake.

DIPSTasks: Trykker på doc viewer.

DocumentViewer: Bemerket at det ikke går å zoome. Bruker litt tid på å lese. Signerer den og.

Går tilbake

DIPSTasks: Skjønner at hen er ferdig

Test 2 kommentarer

DIPSTasks: Trykker seg inn på lab

LabResults: Signerer og går tilbake

DIPSTasks: Velger doc viewer
DocumentViewer: Signerer og går tilbake
DIPSTasks: Er ferdig

Intervju etter test

1. Hva synes du om appene?
 - a. Det var bra, det var enkelt å bruke. Ja dere fortalte jo at det er til helsepersonell. Det kom jo forsovet fram når det sto signer her og sånn. Og så var det noen prøvelabresultater og sånn
2. Hva var vanskeligst?
 - a. Ingenting
3. Hva var lettest?
 - a. Det var lettest å signere
4. Hvordan fungerte det å navigere mellom visningene i appene?
 - a. Det var det eneste tingen jeg tenkte på som jeg syntes var litt awkward. Det er bare sånn bitte litt bare da. Men da man har signert, og man i tillegg får et varsel om at man har signert, så vet ikke jeg da, for jeg er ikke lege og vet ikke hvordan de jobber med det til vanlig, men når jeg trykker ok der, så er det naturlig for meg å gå automatisk tilbake, istedenfor å måtte *trykke* tilbake
5. Merket du en forskjell i navigeringen mellom den første og andre testen?
 - a. Jeg merket ikke forskjell
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. *kommer med tilbakemelding på at dokumentet var vanskelig å lese og at det burde finnes en bedre måte å løse det på
7. Noe annet du tenkte på?
 - a. Nei, bare det at det er mer naturlig å bli sendt rett ut etter å ha signert.

Post-test briefing

Truls: som du egentlig visste allerede da, så er det vi tester hvordan det føles å navigere mellom appene. Og hva du trykker du på; bruker du denne knappen oppe eller bruker du denne knappen nede f.eks.. Og om du merker noen forskjell og sånne ting. Men du visste vel alt dette allerede? Eller hadde du glemt noe av det?

Testperson: Nei, jeg visste det. Jeg merket jo egentlig ikke noen forskjell, det var først når jeg fikk den nye appen at jeg tenkte "åja, nå vet jeg at det er en forskjell", men det var ikke sånn merkbart liksom.

Truls: Men selv om du visste at det var en forskjell, la du merke til at det var en forskjell?

Testperson: ja, da la jeg merke til det.

Truls: ja riktig

Testperson: ja

Truls: på hvilken måte da?

Testperson: Nei, det er bare animasjonen som er annerledes.

Truls: å, ok

Testperson: At den kommer opp, istedenfor fra siden

Testrapport ikke-lege testperson 9

Bachelor DIPS våren 2019

Testleder: Truls Torgersen

Testobservatør: Toni Vucic

Testskribent: Truls

Tid verifisert av: Toni Vucic

Testperson nr.: 9

Produkt testet: Testapper v. 2.0.1

Notater fra observatør: Første test måtte startes på nytt siden vi glemte å slå på riktige gestures. Tester kom til Document viewer

Øvrige kommentarer: Vi glemte å spørre om tester brukte knapper eller gestures. Første test av 3 apper ble dermed avbrutt halvveis. Den testen er IKKE dokumentert her. Test 1 som blir beskrevet under er en gjentakelse med knapper aktivert. **I forhold til tidstaking**, så er test 1 sin tid brukt siden dette er førsteinntrykket og navigasjonsmetoden testpersonen brukte er lik i begge testene. Dette gjelder fram til DocumentViewer. Derfra bruker vi tidsdata fra andre tagging av test 1, som da fortsatt er førsteinntrykk.

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
ja	00:00:23	00:01:01	Android	knapp	up	up	up	up	nei	82,5	77,5

Transkribert intervju og observasjoner:

Test 1 (små apper) kommentarer

DIPSTasks: Inn i labresults

LabResults: Signerer en gang, prøver å dra ned siden for å oppdatere. Signerer deretter flere ganger. Tilbake med **up-knapp**.

DIPSTasks: Inn i documentviewer.

DocumentViewer: Prøver å zoome inn. Signerer to ganger og tilbake med **up-knappen**
DIPSTasks:

Testen er egentlig ferdig nå.

Trykker på back knappen og blir sendt til labresults. Går tilbake til DIPSTasks med back-knappen. Fortsetter å gå frem og tilbake mellom visningene selv om alle oppgavene er gjort. Tester uttrykker usikkerhet over hva som skal gjøres da listen er tom. Sier seg ferdig.

Test 2 kommentarer

DIPSTasks: Inn i labresults

LabResults: Signerer to ganger og prøver å trykk flere steder på skjermen.

DIPSTasks: Inn i documentviewer

DocumentViewer: Prøver å zoome igjen. Signerer to ganger og tilbake med up-knapp.

DIPSTasks: Er ferdig

Intervju etter test

1. Hva synes du om appene?
 - a. Savnet en oppdatering etter jeg hadde trykket på signer. Den endra farga, en man kunne fortsatt trykke på den og det var kanskje forvirrende. Det var et eller annet trøbbel når man gikk tilbake. Men ellers gikk det superkjapt.
2. Hva var vanskeligst?
 - a. Egentlig å vite at man var ferdig med oppgaven.
3. Hva var lettest?
 - a. Å bli ferdig med oppgaven uten å vite det.
4. Hvordan fungerte det å navigere mellom visningene i appene?
 - a. Det fungerte veldig bra i basis funksjonalitet. Som inn i en ting og ut fra den. Men et eller annet som skjedde med tilbake knappen når du sto i hjem skjermen. Gikk tilbake til en sånn som kanskje ikke skulle eksistert lenger.
5. Merket du en forskjell i navigeringen mellom den første og andre testen?
 - a. Nei, ikke som jeg kommer på.
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. Ikke klikkbar knapp etter man har signert. Og disposing av modale views når man går tilbake. Det var alt, ellers var det greit.
7. Noe annet du tenkte på?
 - a. Det gikk ikke an å zoome i oppgave 2. Det kan ha vært vanskelig å lese teksten for enkelte, eventuelt mer informasjon på historikk på verdi målingene i den første. Kunne vært viktig å se før man signerte.

Kommentarer etter alt

Toni forklarer hva vi tester

Tester: Tenkte ikke på at det var flere apper i den ene løsningen. Trodde det var modale views som dukka opp. Da ville jeg si det er ekstremt sømløst.

Testrapport ikke-lege testperson 10

Bachelor DIPS våren 2019

Brukes ikke i analysen av data siden det er en outlier.

Testleder: Erling Moxnes Kristiansen

Testobservatør: Ingen

Kameramann og test-veileder: Toni Vucic

Transkribent: Toni Vucic

Testperson nr.: 10

Produkt testet: Testapper v. 1.1 (Pga. stjernesymbolet)

Øvrige kommentarer:

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
nei	07:52	01:46	iOS 12.1	N/A	up-nav	up-nav	up-nav	up-nav	nei	25	40

Transkribert intervju og observasjoner:

Test 1 kommentarer

Denne testen ble så lang (over 10 minutter) at bare deler er transkribert.

DIPSTasks:

Brukte mange minutter før hen turte å trykke på noe etter lett oppmuntring og en kort forklaring av hva slags app det var. Det ble sagt at den er laget for leger på sykehus og at de har noen oppgaver de skal løse.

LabResults:

Visste ikke hva hen skulle gjøre i flere minutter. Ba hen lese opp de mulige valgene hen hadde. Hen leste opp "Tap to sign". Det gikk litt mer tid uten at hen gjorde noe.

Toni: "Se for deg at du er lege og at du har fått resultatene her. Tror du det er noen handlinger du kan gjøre nå?"

"Jeg har ikke greie på det i det hele tatt"

Toni: "Du nevnte en signatur?"

"Ja her da?"

Toni: "Kan det være noe? Hva står det på den knappen?"

"Tap to sign, da har vi gjort det. Aner ikke hva det menes"

Toni: "Hva er det som står der da?"

Hen trykket så tap to sign.

"Signert for resultatet? Det skjønner jeg jo"

Toni: "Står det noe mer?"

"Okei står det nedenfor"

Toni: "Eh ja"

Toni: "Prøv å tenke..."

Hen trykket tap to sign men ingenting skjedde.

"Nei ingenting"

Hen trykker ok

"Da går vi tilbake der igjen"

"Haha dette blir dårlig resultat"

Toni: "Neida"

"Haha"

Toni: "Tror du det er noe mer du kan trykke her nå?"

"Nei, for å si det rett ut, nei ærlig talt nei ikke"

Toni: "Hva med øverst på visningen?"

"Jeg kan prøve å trykke på det her da? Hva skjer da?"

Hen trykker på My Tasks-knappen (iOS back) og havnet i DIPSTasks igjen.

DIPSTasks:

Lurte på hva hen skulle gjøre. Endte med å trykke Document to sign.

DocumentViewer:

Brukte nesten like lang tid her. Slet med å lese dokumentet. Tok telefonen opp mot ansiktet heller enn å gjøre pinch-to-zoom. Men signert og trykket ok etter en stund. Trodde hen kom til en ny side etter å ha trykket ok. Begynte å lese alt på nytt. Signerte og trykket ok igjen. Trodde hen kom til en ny side men sa alt var likt og ble litt rådvill. Skjønte heller ikke nå at hen skulle navigere tilbake før hen fikk beskjed om å forklare mulighetene hen hadde.

Da trykket hen selv på My Tasks-knappen (iOS back).

DIPSTasks:

Skjønte ikke at hen var ferdig. Prøve å trykke My Tasks et par ganger. Trykket under My Tasks-teksten der det var en usynlig knapp og havnet på Lab Results igjen.

LabResults:

Testen ble avbrutt

System Usability Scale, én stor app

Testeren fikk en laptop i fanget med spørsmålene, samt fikk spørsmålene lest opp høyt, med svaralternativene. Hen fikk beskjed om at hvis hen ønsket oversettelse av spørsmålene kunne Toni gjøre det for hen. Toni trykket på svaralternativene på skjermen som testeren valgte muntlig, siden testeren aldri hadde brukt en moderne datamaskin før.

F.eks.

Toni: "I think that I would need the support of a technical person to be able to use this system."

"Jaja, jeg tror jeg trenger."

Toni: "Somewhat agree? Eller strongly agree?"

"Some"

Toni klikker somewhat agree

Toni: "I found the various functions in this system were well integrated"

"Oh no"

Toni: "No? OK"

"Strongly disagree eller somewhat disagree?"

"Some"

Når testeren indikerte positivt eller negativt svar ble de to positive eller negative svaralternativene lest opp.

System Usability Scale, små apper

Samme opplegg som første SUS-spørreskjema

Test 2 kommentarer

DIPSTasks:

"Hva skal jeg gjøre her da?"

Toni: "Jeg kan ikke si noe mer gangen her enn jeg gjorde forrige gang."

"Det er bare å prøve seg fram."

Tester trukker Lab result to sign

LabResults:

"Her får vi igjen det samme"

"Det her må jo være beregnet på en, samme som tidligere"

"Jeg prøver det der da"

Trykker tap to sign

"Hva skal jeg gjøre nå da?"

Toni: "Jeg vet ikke"

"Jeg bare trykker jeg, haha, aner ikke hva det er"

Tester trykker ok

“Hun må jo være farmasaut hun damen der” (Snakker om Thelma Louise, person i appen vår)

“Jeg bare trykker der, hva skjer da?”

Tester trykker på Done

DIPSTasks:

“Jaha”

Trykker document so sign

DocumentViewer:

“Huff,å her, meer graf. Medisinsk dokument. Og så liten skrift, nei det ser jeg ikke”

Tester signerer, trykker ok

Tester trykker på Done

DIPSTasks:

“Kom tilbake til det der, akkurat det samme. Og den var blank”

Toni: “Hva tenker du nå?”

“At jeg skjønner svært lite av dette, hahaha”

Testen avsluttes

Intervju etter test

1. Hva synes du om appene?
 - a. “Appene? Jeg er ikke den rette til å svare på det egentlig. Jeg kan knapt nok finne det aller enkleste.”
2. Hva var vanskeligst?
 - a. “Det var jo, ja, hva som var vanskeligst? Sannheten er jo det at jeg har alt for lite bakgrunn for å kunne svare på det. Det hele var jo vanskelig. Jeg er jo overhodet ikke vant til å bruke noe av dette.”
3. Hva var lettest? Edit: “Det var kanskje ikke noe som var lettest?”
 - a. “Neeei egentlig ikke.”
 - b. Toni: “Hvis vi stiller spørsmålet en gang til nå.”
 - c. “Hva var lettest”
 - d. “Hva var lettest? Nei jeg vet ikke, nei jeg kan ikke svare på det. Jeg syntes ikke det var spesielt lett.”
4. Hvordan fungerte det å navigere mellom visningene i appene? Edit: Og da mener vi de forskjellige skjermbildene da. Hvordan fungerte det?
 - a. “Hadde jeg brukt brettet mitt hadde jeg ringt han! (Toni)”
5. Merket du en forskjell i navigeringen mellom den første og andre testen?
 - a. “Skal jeg være ærlig så skjønnte jeg ikke så veldig mye av noen av delene.”
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. “Jeg er ikke kompetent til å uttale meg om det. Neineinei.”
7. Noe annet du tenkte på?
 - a. “Altså, jeg har vel ikke noen særlig formening om noe av det, skal jeg være ærlig.”

Testrapport ikke-lege testperson 11

Bachelor DIPS våren 2019

Testleder: Toni Vucic

Testobservatør: Erling

Transkribent: Truls

Dobbeltsjekking av tidsbruk: Toni Vucic

Testperson nr.: 11

Produkt testet: Testapper v. 2.0.1

Notater fra observatør:

- Gikk tregt å komme i gang siden vi var uforberedt.
- Toni glemte å fjerne notatene for gjennomføring av testen når bruker fikk data for å svare på skjema.
- Truls fikk beskjed om ikke å avbryte testen når han kom inn med sjokolade.

Øvrige kommentarer:

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
ja	00:00:23	00:00:32	iOS	N/A	Done	Done	Back	Back	Nei	62,5	55

Transkribert intervju og observasjoner:

Test 1 kommentarer

DIPSTasks: Gikk inn i labresults

LabResults: Signerer en gang og bruker Done for å gå tilbake.

DIPSTasks: Inn i documentviewer

DocumentViewer: Signerer to ganger og bruker Done for å gå tilbake

DIPSTasks: Trykker på back-to-app knapp og blir dermed sendt til documentviewer. Trykker på Done og er tilbake i DIPSTasks.

Test 2 kommentarer

DIPSTasks: Gikk inn i labresults

LabResults: Signerer en gang og går tilbake

DIPSTasks: Gikk inn i documentviewer

DocumentViewer: Signerte en gang og går tilbake

DIPSTasks: Ferdig

Intervju etter test

1. Hva synes du om appene?
 - a. Jeg skjønnte ikke helt hva det var først, men det var greit etterhvert. Når det sto at for å signere må du trykke på knappe, så har du gjort det. Da var det effektivt og greit.
2. Hva var vanskeligst?
 - a. Å skjønne hva appen hva for noe. Hvilken situasjon ville du brukt den.
3. Hva var lettest?
 - a. Å trykke seg videre.
4. Hvordan fungerte det å navigere mellom visningene i appene?
 - a. Det gikk ganske bra.
5. Merket du en forskjell i navigeringen mellom den første og andre testen?
 - a. Nei.
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. Nei.
7. Noe annet du tenkte på?
 - a. Nei. Det blir sikkert en bra app.

Kommentarer etter alt

Toni forklarer hva vi tester.

Testrapport ikke-lege testperson 12

Bachelor DIPS våren 2019

Testleder: Toni Vucic
Testobservatør: Erling
Transkribent: Erling

Testperson nr.: 12

Produkt testet: Testapper v. 2.0.1

Notater fra observatør:

Øvrige kommentarer: I videoen fra test med stor app kommer testnr og app opp til slutt.

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
nei	00:30	00:18	iOS	N/A	Done	Done	Back	Back	ja	82,5	70

Transkribert intervju og observasjoner:

Test 1 kommentarer

DIPSTasks: Tester er usikker på hva som skal trykkes på. Men trykker på labresults

LabResults: Signerer 2 ganger, går tilbake med back

Tester: Hva går det her ut på egentlig?

DIPSTasks: Inn i docs

DocumentViewer: Signerer en gang

DIPSTasks: Ferdig

Test 2 kommentarer

DIPSTasks: Velger labresults

LabResults: Signerer en gang, trykker done

DIPSTasks: Velger docs

DocumentViewer: Signerer gang, trykker done

DIPSTasks: Ferdig

Toni: Du kan si ifra når du føler du er ferdig.

Tester: Ja, jeg er ferdig. Tror jeg.

Intervju etter test

1. Hva synes du om appene?
 - a. Nei, de var jo ganske enkle. Kjekke, sikkert.
2. Hva var vanskeligst?
 - a. Det vanskeligste var jo å skjønne hva jeg skulle gjøre. Jeg skjønnte jeg hadde signert noen greier. Vet ikke om jeg er den appene er lagd til.
3. Hva var lettest?
 - a. Det letteste var jo å finne fram til de tingene man skulle trykke på.
4. Hvordan fungerte det å navigere mellom visningene i appene?
 - a. Veldig bra.
5. Merket du en forskjell i navigeringen mellom den første og andre testen?
 - a. Ja, ting på en måte fløt bedre i den første, var større overgang fra bildet til bildet i den andre.
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. Nei.
7. Noe annet du tenkte på?
 - a. Egentlig ikke.

Kommentarer etter alt

Toni: Forklarer hva vi har laget og hva som ble testet.

Testrapport ikke-lege testperson 13

Bachelor DIPS våren 2019

Testleder: Toni Vucic

Testobservatør: Erling Moxnes Kristiansen

Transkribent: Erling Moxnes Kristiansen

Testperson nr.: 13

Produkt testet: Testapper v. 2.0.1

Notater fra observatør:

Øvrige kommentarer: Vi tok test av små apper på nytt da gestures var i bruk ved første gjennomgang, da bruker er vant med å bruke knapper. Så det ble totalt 3 tester: små apper(gestures) -> stor app(knapper) -> små apper(knapper). Sus og tid kan derfor ikke brukes for tre apper siden personen enten har fått feil versjon av systemet i første testen eller at hen har blitt vant med systemet og derfor muligens vil gi høyere skår eller fullføre oppgavene fortere enn ellers. Knappen som brukes og sus fra test to er likevel god informasjon.

Tabellen er fylt med data fra test 2 og 3

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
ja	45 sek	30 sek	Android	knapper	back	back	back	back	nei	82,5	85

Transkribert intervju og observasjoner:

Test 1(m/gestures) kommentarer

DIPSTasks: inn i lab results

LabResults: Signerte to ganger, prøver å trykke på prøveresultat. Gikk tilbake med up knapp

DIPSTasks: Inn på docs

DocumentViewer: Signere en gang og tilbake til tasks.

DIPSTasks: Er ferdig.

Toni: Hva tenker du nå?

Tester: Det var greit, når jeg hadde trykka sign var det ikke helt intuitivt at jeg måtte trykke tilbake. Kanskje skulle skjedd av seg selv.

Test 2 kommentarer

DIPSTasks: Går inn i labres

LabResults: Signerer og bruker back knapp

DIPSTasks: Går rett inn i docs

DocumentViewer: signerer og bruker back knapp igjen

DIPSTasks: Ferdig

Tester viser litt usikkerhet når begge oppgavene er gjort.

Toni: Hva er status nå?

Tester: Står jo ingen ting her nå. Hva er egentlig forskjellen?

Toni: Kan jeg ikke si helt ennå.

Test 3 kommentarer

DIPSTasks: Inn i results

LabResults: Signerer en gang

DIPSTasks: Inn i docs

DocumentViewer: Signerer en gang

DIPSTasks: Ferdig

Intervju etter test

1. Hva synes du om appene?
 - a. Det var greit, var lett å bruk på en måte.
2. Hva var vanskeligst?
 - a. Vet egentlig ikke.
3. Hva var lettest?
 - a. Det var veldig lett å bare trykke, så har du signa så er du ferdig. Har ikke no bedre svar.
4. Hvordan fungerte det å navigere mellom visningene i appene?
 - a. Bra.
5. Merket du en forskjell i navigeringen mellom den første og andre testen?
 - a. Egentlig ikke.
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. Det jeg synes ikke var helt intuitivt var at du ikke hoppa tilbake til lista med oppgaver når du har gjort ferdig en.
7. Noe annet du tenkte på?
 - a. Nei, egentlig ikke.

Kommentarer etter alt annet

Tester blir forklart og vist hva vi testet.

Toni: hvorfor brukte du up-knapp i test 1, men back-knapp i test 2?

Tester: Nei, er vel bare vane sikkert.

Erling: Du bruker vanligvis back-knappen?

Tester: Ja, for da slipper du å strekke fingeren.

Testrapport ikke-lege testperson 14

Bachelor DIPS våren 2019

Testleder: Truls Torgersen

Testobservatør: Toni Vucic

Transkribent: Toni Vucic

Testperson nr.: 14

Produkt testet: Testapper v. 2.0.1

Notater fra observatør: Glemte å skru på opptakeren før midt i første test.

Øvrige kommentarer:

Små apper først	Tidsbruk stor app	Tidsbruk små apper	Operativsystem	Back-navigasjonstype	Nav back fra LabResults	Nav back fra DocumentViewer	Nav back fra stor app Results	Nav back fra stor app DocViewer	Merket Forskjell?	SUS stor app	SUS små apper
nei	01:40	00:10	iOS 12.2	N/A	done	done	up	up	nei	80	90

Transkribert intervju og observasjoner:

Innledning:

Test 1 (stor app) kommentarer

DIPSTasks:

LabResults:

Brukte VELDIG lang tid på å signere, siden testeren trykket Tap to sign om og om igjen. Sikkert 20 ganger.

Trykker vanlig iOS back-knapp.

DIPSTasks:

DocumentViewer:

Trykker vanlig iOS back-knapp.

DIPSTasks:

En av oss sier: "Du skal kanskje tilbake"

Hen: "Åja, ja"

"Det var det?"

Test 2 (små apper) kommentarer

DIPSTasks:

LabResults:

Trykker Done (Var ikke så lett å se på videoen)

DIPSTasks:

DocumentViewer:

Trykker Done

DIPSTasks:

"Ja, trykker på LabResults, Done"

"Også går jeg på neste og gjør det samme med den"

"Jepp"

Intervju etter test

1. Hva synes du om appene?
 - a. Den første var litt vanskelig å forstå. Etter å ha trykket DONE kunne jeg ikke fortsette å signere på samme skjema, og da tok det litt tid før jeg forsto at jeg måtte gå tilbake selv for å gå videre til neste dokument
2. Hva var vanskeligst?
 - a. Å forstå hvordan det fungerte, hadde ikke kunnskap om appen før bruk eller hva jeg skulle gjøre
3. Hva var lettest?
 - a. "Trykke, signere, bare ved ett tastetrykk var veldig lett"
4. Hvordan fungerte det å navigere mellom visningene i appene?
 - a. "Det var lettere på app nr. 2 syns jeg. Fordi du fikk mulighet til å trykke DONE i venstre hjørne, og litt mer intuitivt"
5. Merket du en forskjell i navigeringen mellom den første og andre testen?
 - a. "Jeg syns de fungerte likt, like kjapt, men det var lettere å forstå at du skulle navigere i test nr. 2"
6. Har du forbedringsforslag eller tilbakemeldinger?
 - a. "Når jeg begynte med den første så synes jeg det manglet at det kanskje etter du har signert så blir du flyttet direkte tilbake til oppgavelisten
7. Noe annet du tenkte på?
 - a. Nei

Ekstra-intervju:

Tester la ikke merke til at hen ble sendt mellom forskjellige apper.
Merket heller ikke den lille svarte back-to-app-knappen.

L.4 Empirical data

Link to sheet:

<https://docs.google.com/spreadsheets/d/1FwNk7mdu0hyP1aD9GisxQ1qc1QUWlihgYK64rHVZkbQ/edit?usp=sharing>

