Final report

DoorSine / Digital Assistant for Staff Office Door

Knut Sander Lien Blakkestad

University of Essex

CE301 Final Year Capstone Project

Logo, icon

Description automatically generated

# Acknowledgments

First and foremost, I want to thank my supervisor Dr. Shoaib Jameel. During my second year at Essex he was one of the supervisors for our team project, and through the year I got to know him as a professional lecturer with an abundance of wisdom and optimism.  
Upon learning about this project, and after a few discussions about the scope and goal of it, I decided this project was what I wanted my final year project to be. Working on this project has really helped me grow as a computer scientist and I want to thank Doctor Shoaib Jameel for giving me the opportunity to do so.

I also want to thank my family, who has supported me every step of the way. These three years of university has not been what I envisioned when starting, but with the guidance and support you have given me I have battled through and come out on top. I am forever grateful for having a family like you.

Lastly, I want to thank my friends who I have also relied on for emotional and sometimes physical help with problems and challenges. You guys are what has made these strange three years of university bearable and most of all enjoyable.

# Abstract

Title: DoorSine / Digital Assistant for Staff Office Door

Name: Knut S L Blakkestad

Reg No: 1904341

Location:  
The aim of the project was to create a digital assistant for an office door that would allow the owner to communicate and display information to anyone outside their office.  
Several similar products exist on the market, but they are more focused on managing entire  
workspaces.  
What makes this project stand out is that it is aimed at managing offices for individuals, rather than managing workspaces and rooms for corporations.  
It can be utilised by as few as a single office worker, and by as many as an entire corporation.  
The impact the project has is that it shrinks the time it takes to communicate with i.e. a lecturer from a day or two via email, to a quick message or a call.  
It also makes booking meetings and getting up to date information accessible and easy.  
I have learned so much during this project, and have had the opportunity to grow as a data scientist using the newest technology in app development.

Table of Contents

[1. Project Context 1](#_Toc101616876)

[1.1 Description 1](#_Toc101616877)

[1.2 Aims and Objectives 1](#_Toc101616878)

[1.3 Motivation 1](#_Toc101616879)

[1.4 Background Reading 2](#_Toc101616880)

[1.5 Related Technologies 4](#_Toc101616881)

[1.6 Implemented Technologies 5](#_Toc101616882)

[1.7 Sustainability 5](#_Toc101616883)

[1.8 Legal 6](#_Toc101616884)

[1.9 Ethical 7](#_Toc101616885)

[1.10 Intellectual Property 7](#_Toc101616886)

[1.11 Challenges 8](#_Toc101616887)

[2. Project Implementation 9](#_Toc101616888)

[2.1 Code Implementation 9](#_Toc101616889)

[2.1.1 Lib Folder 10](#_Toc101616890)

[2.1.2 Pages Folder 11](#_Toc101616891)

[2.1.3 Phone Pages 15](#_Toc101616892)

[2.1.4 Tablet Pages 17](#_Toc101616893)

[2.1.5 Models 18](#_Toc101616894)

[2.1.6 Widgets 18](#_Toc101616895)

[2.2 Visual Implementation 20](#_Toc101616896)

[3. Project Testing 27](#_Toc101616897)

[3.1 Shared Tests 27](#_Toc101616898)

[3.2 Phone Tests 28](#_Toc101616899)

[3.3 Tablet Tests 30](#_Toc101616900)

[4. Project Planning 32](#_Toc101616901)

[4.1 Planning and Execution 32](#_Toc101616902)

[4.2 Jira and GitLab Use 32](#_Toc101616903)

[4.3 Risk Management and Bugs 36](#_Toc101616904)

[5. Conclusion 37](#_Toc101616905)

[5.1 Achievements 37](#_Toc101616906)

[5.2 Future Development 38](#_Toc101616907)

[5.3 Final Words 38](#_Toc101616908)

[6. References 39](#_Toc101616909)

# 1. Project Context

## 1.1 Description

Below is the description of the project provided by Doctor Shoaib Jameel as the outline for this project on the Project Database.

The idea is to develop software which could run on a mobile device that connects to the network and takes commands remotely.   
The goal is to place this device outside every staff office door so that anyone who wants to visit the staff knows that status of staff - whether the staff is in the office or out of office, whether the staff is in a meeting, or on leave.   
We could do plenty of cool pieces of stuff with this device such as scheduling a meeting by directly interacting with the device and viewing your marks on this device using facial recognition (I am going too much overboard!).   
The initial goal would be to first build the software that works on a mobile device such as a mobile phone.   
If you are interested, please feel free to drop an e-mail to discuss what exactly you wish to work on including its scope[[1]](#footnote-1).

## 1.2 Aims and Objectives

After selecting this as my final year project, Doctor Shoaib Jameel and myself discussed and agreed upon exactly what the aims and objectives of the project should be. The initial goal was to develop the groundwork for the application and implement the following features.

* The staff should be able to set their availability so that anyone visiting their office could know if they currently are available or busy.
* A student can check when the staff is available next and book a meeting without disturbing the staff.
* Someone at the door could interact directly with the staff through messaging

Through the report, these features as well as additional ones will be highlighted and explained, showing how the needs of a potential user has been solved.

## 1.3 Motivation

App development has always intrigued me, but throughout the first two years of university there had been little to no focus on it. So, when the chance presented itself to try developing an application from scratch with freedom to do what I wanted with it, there was no doubt in my mind that this was the project I wanted to try developing.

Statistics from *Statista* highlight the importance smartphones and the effect the applications on them have on our life. A survey on the number of smartphone subscriptions worldwide from 2016 to 2027 shows that the amount has been increasing substantially since 2016 and is forecasted to keep increasing in the coming years. At the end of 2021, the number was at 6.259 billion and is forecasted to grow into 6.567 billion in 2022 [1].  
Another survey shows the increase of smartphone usage in different age groups between 16 and 65+. From 2012 to 2020. The percentages of the survey shows smaller increases in the combined age group 16-44, ranging from 12% at the lowest to 27% at the highest. While in the combined age group of 45+ the increase is even larger, ranging from 49% at the lowest to a 62% at the highest [2].  
These two surveys highlight that the usage and reliance of smartphones show no signs of slowing down, and provides the basis of my argument that the need for user friendly and reliable apps are higher than ever.

These surveys also support what I have experienced and continue to experience in my daily life. Born right before the turn of the century, I have experienced smartphones going from something that was non-existent to something that is used and often needed in the daily life of almost everyone. This provides another good reasons for why I wanted to do this project, and the reason why I believe this is a good time to do it.

## 1.4 Background Reading

Before looking into languages and environments to create the app in, I wanted to understand what makes an app good in the eyes of the user. I also wanted to look at the challenges app developers face and possibly how to overcome them.

A research paper titled “Real Challenges in Mobile App Development” from 2013 investigates the challenges in mobile app development both using qualitative data from 12 interviews with senior developers and quantitative data from 188 answers to a survey answered by people from the mobile development community.   
One of the main revelations of the study is that one of the biggest challenges in mobile app development is dealing with the various mobile platforms. Since the different platforms vary in build and functionality, the developers often found themselves creating a separate app for every platform and manually checking that the functionalities are preserved across the different versions [3].  
A criticism of the study could be that is older, and that there has been a lot of development in the field of mobile app development. A quick web search shows that this is still the case for most of app development. Articles naming the top 5, top 10, etc. programming languages like this *Medium* article[[2]](#footnote-2) or this *SpinxDigital* article[[3]](#footnote-3) all mention Java and Kotlin as the most used languages for Android development, and Swift as the most used for IOS development, showing that most app development focuses on separate languages for the different operating systems.

“Factors Influencing Quality of Mobile Apps: Role of Mobile App Development Life Cycle” is a paper published in October of 2014 and investigates what makes applications fail, and by proxy also describes what to do to not make an application that will fail.  
According to the report there were about 6.4 billion applications downloaded in 2009, with the number increasing at an accelerated rate to 76.9 billion by 2014 [4].  
This projected increase is further backed up by numbers from Statista, showing that the number of mobile app downloads in 2016 was at 140.68 billion and in 2021 was at 230 billion [5], showing that the mobile app industry is bigger now than ever.  
The report describes a bad app as having the following flaws:

* Poor design/UI
* Too much clutter on screen
* Poor navigation
* Does not meet the user requirements
* Does not address the specific issue
* Has security issues
* Fails at essential times
* Downloading issues
* Inconsistencies across platforms
* Compatibility issues
* High battery usage
* Slow replication function
* High ad frequency
* Not appropriately priced
* No endeavours made to solve any of the mentioned issues

In short, apps should be fast with a simple and understandable interface and should work as advertised without any issues relating to security, loading or battery consumption.

As previously mentioned, I had little experience with app development. I had no knowledge of the most used languages, what compilers to use, or what frameworks were the best ones. The background reading regarding this started by looking at articles like “Top 5 Programming languages for Mobile App Development”[[4]](#footnote-4)  
This article and others like it mention *Kotlin* and *Java* as leading languages for Android development, *Swift* is mentioned as the leading language for IOS development, and *JavaScript* is mentioned as the leading language for Web development,  
But throughout all of these articles, a language I had never heard about caught my attention called Dart.

“Dart is a client-optimized language for fast apps on any platform”[[5]](#footnote-5). It is an open-source and object-oriented programming language released by Google in 2011 and has seen continuous improvements and development since its initial release. It was created to “offer the most productive programming language for multi-platform development”[[6]](#footnote-6) and provides developers with a programming language that can used to develop applications that can run on Android, IOs and web browser.   
It is meant to be used in conjunction with *Flutter*, which is a framework created by *Google* to build apps for any screen. The homepage states “Flutter transforms the app development process. Build, test, and deploy beautiful mobile, web, desktop, and embedded apps from a single codebase.”[[7]](#footnote-7) By looking at what other developers have been able to create using Dart and Flutter, and by undertaking several tutorials and testing the frameworks limitations myself, I was convinced that this was the language and environment I wanted to create and develop this project with.

## 1.5 Related Technologies

A big part of the background reading also consisted of looking at similar products already publicly available to gain insight into how they function, what their customer base looks like and to look at how they have succeeded. It was also helpful to gain inspiration for design and layout, and to look at what can be improved on in their features and design.

The major technologies looked at are *Door Tablet* with clients like *Michigan State University* and *Plymouth Marjon University*[[8]](#footnote-8), *Meetio* with clients like *Duchy Homes* and *Three*[[9]](#footnote-9), *Condeco* with clients like *Vodafone, Nestle* and *Comcast*[[10]](#footnote-10), and lastly *Pronestor* with clients like *United Nations* and *Siemens*[[11]](#footnote-11). These products are all mainly aimed at the office industry sector, but some of them have customers in the school sector as well.

Similarities between these apps and the project include some basic functionality. They all work on both phones and tablets, but this is true for most applications. They all provide the possibility of booking meetings, and provide the user with general information.

However there are more differences between this project and the related ones than there are similarities. Their main purpose is to manage the office space by giving the users the ability to book desks, rooms and more. As well as planning group meetings, accessing floor plans and displaying room information.  
The biggest difference that really makes this project stand out from the similar products is how it is aimed at managing one-to-one communication between a lecturer or office user and a person at their door. The person at the door can see the office users’ status, can get some basic information on them, and has the option of messaging, calling and booking meetings with them.

Looking at the similarities and differences, it is clear that similar products do exist on the market already and the different companies compete for the biggest share on the market. But the number of differences show that this project is something not yet available and would not struggle to compete with these other companies, but rather fill a gap in the market that currently is not filled by anyone.

## 1.6 Implemented Technologies

There are many requirements and functionalities needed to create a modern application, and having to design, code and create all of these from scratch would have taken away development time that should be used to create a highly functioning application. For this reason, several modern solutions have been implemented into the project to reduce the amount of time and work focused on smaller aspects of it. This however does not mean that the development has been easy. All these technologies still need to be understood and managed, as well as having to be implemented into the project using code. What it has done is allowing the development to be focused on creating a robust application with several implemented features without having to use a substantial part of the development on creating tools the application needs to work as intended. Below is a list showing the implemented technologies and what they are used for.

* Dart: A programming language created for cross platform app development[[12]](#footnote-12), used for the backend code of the app
* Flutter: A framework that uses Dart to create apps from a single codebase[[13]](#footnote-13), used for the frontend code of the app
* Firebase: An online database and analytics service[[14]](#footnote-14), used for storing user data, sending messages and manage user login
* Agora: An online hosting service for voice and video calling[[15]](#footnote-15), used to host the video calling functionality
* Heroku: A cloud computing service host[[16]](#footnote-16), used to host the token server that creates tokens for secure video calling
* Azure: A cloud computing service by Microsoft for application management[[17]](#footnote-17), used to connect with Microsoft and enable Microsoft login

## 1.7 Sustainability

The sustainability of a product like this is based on what work is done on it after release. If it were to be implemented and used at this very moment, there would be no software issues and it would work as expected. If it then was left at the current stage and was not sustained through updates and reworks, it would likely break within a year or two as the functionality of many of technologies it relies on would be updated and changed to no longer work with how the code for the application was written.  
The only way to keep it, and for that matter any application, working as expected for longer periods of time is through incremental updates and changes. Through its lifecycle, new versions of Android would be released, and the app would need to be adapted to these changes. Major upgrades and changes to Dart and Flutter could severely affect the structure of the application, potentially breaking it as older methods, variable types and frontend building blocks become deprecated. Firebase could change how data is accessed and stored and updated privacy rules could change what is permitted to store and access from it. Cloud services like Heroku and Azure could fundamentally and functionally change how they work and would in turn break the application.  
Application development is also a technology with constant development and incremental improvements, meaning any application that wants to stay relevant needs updates and changes. That is why a major company like Meta, formally Facebook, constantly updates Facebook and Instagram with new features. The same behaviour is mirrored by Google with constant updates to their applications, especially YouTube. They have managed to stay relevant and are still in use over 10 years after their creation because of technical updates and improvements through their existence.

## 1.8 Legal

One of the major legal problems any system with a login system and information storage has is that it must comply with GDPR. The app saves personally identifiable information (PII) such as name and email, which are both needed for the functionality and use of it.  
The system used for storing this is Firebase, more specifically Cloud Firestore and Firebase Authentication, both of which have passed the ISO 27001[[18]](#footnote-18), ISO 27017[[19]](#footnote-19) and ISO 27018[[20]](#footnote-20) certification processes as well as completing the SOC 1[[21]](#footnote-21), SOC 2[[22]](#footnote-22) and SOC 3[[23]](#footnote-23) evaluation processes.  
All data received from and sent to Firebase is encrypted using the HTTP protocol and the entirety of the data is only accessible to people with admin privileges.  
Firebase Authentication is additionally secured by only keeping logged IP addresses for a few weeks, while any other authentication information is kept until the Firebase customer (System administrator) initiates deletion, at which point the data will be removed from live and backup systems within 180 days. The same system is in place for Cloud Firestore.  
In addition the users have full access to the information stored on them via the app and can alter it if wrong or if it needs updating.

## 1.9 Ethical

GDPR plays an important role in the ethical aspect of applications, especially application like DoorSine that store and share the personal information of the users. This is mostly solved by the fact that the user chooses what to share, and that the application will be used within organisations where most of their personal details are already known. The remaining ethical issues consist of misuse of the application by outsiders.

A disgruntled student could choose to book many meetings after one another, thereby blocking other students from booking meetings themselves. In a similar fashion, they could also send messages or call the lecturer at inappropriate times of the day, disturbing them during their time of or even late at night.

During the day, these situations could be a problem but are unlikely to happen often. To solve the problem of messages and calls during the night, the simple and likely solution is that the devices would be taken down or disabled outside of working hours. There is also the possibility of adding a cooldown timer in the app for all these features, meaning after i.e., sending a message, calling, or booking a meeting the user would have to wait a set amount of time before doing it again. However, this would limit the usefulness of the app, and could cause confusion if a new user arrives and cannot book a meeting because someone else booked right before them. A better solution could be to disable certain functionalities outside working hours, so that there is no possibility of disturbing the lecturer when they are not working.

## 1.10 Intellectual Property

As mentioned in the related technologies section, there are several similar applications on the market already, but they only share some similarities with DoorSine. As an IP, it would stand out from the crowd and provide something that does not yet exist on the market. It would start out by being marketed towards learning institutions, with universities being the primary consumer.

To provide updates and fixes the app would require a monthly subscription of £6.99 or a yearly subscription of £69.99. These prices are subject to change but looking at the pricing of the similar products this is right in their price range. Discounts could be provided if larger portions of a workplace wanted to implement the product.

The application relies on several previously mentioned open-source technologies that provide free service to the consumer if the usage does not exceed a given amount. During testing and development of the app, the usage was only a fraction of the allowed amount. So, at launch and for the foreseeable future the application would still rely on them as they have provide reliable functionality during development. If DoorSine grew as a product, creating our own versions would be considered by comparing the cost of continued use of these products versus the cost and upkeep of creating our own versions of them.

## 1.11 Challenges

Creating an app from scratch has certainly not been an easy task. As mentioned several times already in this report, I had no prior knowledge of it and we have been taught little to nothing about app development during the years at the University of Essex.

Learning an entirely new programming language in Dart has been challenging. It, like most programming languages, have many common traits but also have many differences in certain areas. One of the more challenging aspects of Dart that I struggled with understanding at first is future and asynchronous operations.   
Understanding and dealing with variables and functions that have not happened yet or are in the progress of happing was a challenging concept to wrap my head around, but as more development was done on the app and more information was transmitted back and forth this became a vital part of how the app works.   
Almost no aspect of the application is without some aspect of this and learning it has been both challenging and fun.

Another aspect of the project that has been challenging is understanding and using all the implemented technologies mentioned. Although these technologies have made the project easier by providing functionalities I would have to make myself, they were not ready to be implemented without much background reading to understand how they work. There was a lot of setups needed for them to be ready for use, and since most of them are provided by separate companies, how the information is sent and received were different for each one.

# 2. Project Implementation

## 2.1 Code Implementation

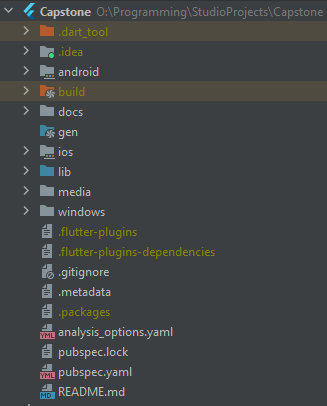
Much of the project structure is created based on the template provided when creating a new Flutter project in Android Studio. The following picture displays the project structure.

Figure 1: Project structure and content

Several of the folders like *android* and *iOS* contain files that are necessary for the application to run and compile on the respective platforms, and little development happens in them. There were however smaller changes to configuration files that were needed as the complexity of the project were changed and expanded.  
Some of the folders frequently used during development like *lib* were also provided when creating the project and contain the application class files and application tests respectively. A couple of folders like *docs* and *media* were created manually for document and media storage.

The *README.md* file in the project gives the viewer a quick description of the project as well as detailing the major features. The *lib* folder and all folders within it contain separate *README.md* files that give a short explanation of the folders content, while the class files themselves contain more detailed information and explanation of what their methods and classes do.

### 2.1.1 Lib Folder

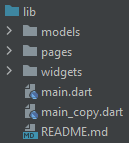
Most of the app framework is contained within the *lib* folder and it is structured like this.

Figure 2: Lib folder structure and content

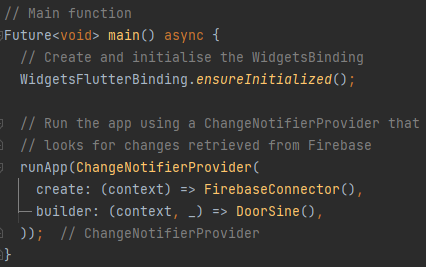
The only class files contained directly within *lib* is the *main.dart* file as well as a copy of it needed when running two simulations of the app at once. When running the app either through simulation or on a physical device, the process is started using the *main* method.

Figure 3: Application main function

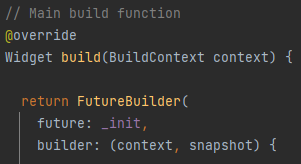
This initialises the *WidgetsBinding* which all the app widgets need to function and starts the app under a *ChangeNotifierProvider* which makes sure the app looks for changes retrieved from the custom class *FirebaseConnector*. Finally, it creates the app instance by calling the *DoorSine* class, which automatically calls the classes build method.

Figure 4: Build method using FutureBuilder

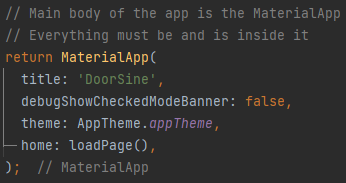
Since the build method of the *DoorSine* class needs to wait for the connection to *Firebase* to be established, it is structured using a *FutureBuilder*, which tells it to wait with certain actions when building. Several other classes use the same builder or variations of it and will be covered later in the report. These builders do not actually return anything themselves but rather has several widgets returned within them based on the status of what they are waiting for to load, their snapshot.

Figure 5: Material app construction

The main return of the builder is a *MaterialApp*, a constructor that uses material design to create an application base that all the other widgets build upon. The *MaterialApp* is given a title, a custom theme defined in *app\_theme.dart* and the home section is given a function call. This function then returns three different widgets based on the status of the snapshot. If the snapshot has an error, it displays an error message, if the snapshot is loading it displays a loading indicator, and if the snapshot has finished loading it returns the *LoginPage* class, effectively taking the user to the page where they can log in.

### 2.1.2 Pages Folder

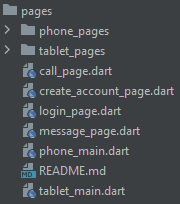
The *LoginPage* class is stored in the *pages* folder, which contains shared pages as well as individual pages for the phone side of the app and the tablet side of the app. These are stored in the *phone\_pages* folder and *tablet\_pages* folder respectively.

Figure 6: Pages folder structure and content

Other files in the *pages* folder are *call\_page.dart* which is a shared page that both the phone and tablet sides of the app uses for calling, *create\_account\_page.dart* where new user can create an account for themselves, *message\_page.dart* which works similarly to *call\_page.dart* but for messages, and *phone\_main.dart* and *tablet\_main.dart* which are the main pages for the phone side and tablet side of the app respectively.

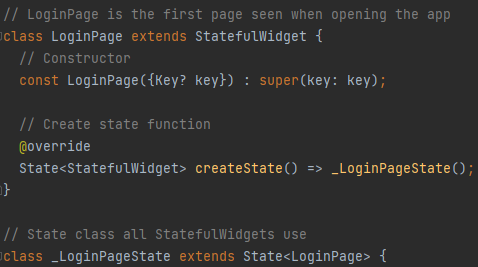
The *LoginPage* class is the first class encountered that extends *StatefulWidget*, since the *DoorSine* class extended *StatelessWidget*. The main difference between them is that a *StatelessWidget* is only built once and does not change through its lifetime, while a *StatefulWidget* is expected to change based on user input. The *StatefulWidget* does not have a build function like the *StatelessWidget* but has a function that creates a mutable *State*. This *State* contains the build function for the class, and when changes are applied to the *State* it is rebuilt with the changes applied.

Figure 7: LoginPage as a StatefulWidget and its State class

The *LoginPage* uses a form to take user input of their email and password, then sends a request to *Firebase* to see if the user exists. If the user does not exist or has entered the wrong email and password combination an error message is shown. If the user has provided the correct email and password combination, a pop up prompts the user to choose if they want to launch the phone or tablet side of the app. There is also an option to sign in using a Microsoft account if the accounts have been linked. This was implemented with the University of Essex usage of Microsoft services in mind, but anyone can choose to create their account in this manner.

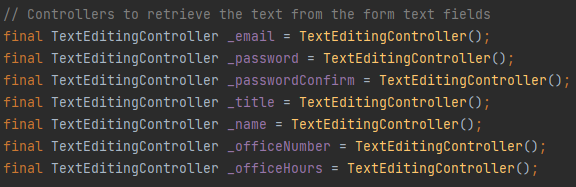
The *CreateAccountPage* works similarly to the *LoginPage* as it uses a form to take user input. If creating an account manually, all fields must be filled before the account can be created with an error message displaying what information is missing. This form and all others use a *TextEditingController* for every input field, which is used both to check that the input text is correctly formatted, as well as retrieving the text when creating the account. If the user wants to create an account using the Microsoft login the form does not need to be filled in, the app retrieves the necessary information from the Microsoft account.

Figure 8: TextEditingControllers used by the form for account creation

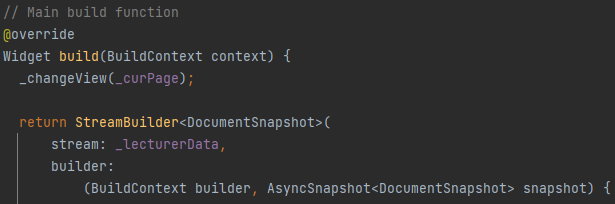
The *PhoneMain* and the *TabletMain* functions differently from each other. While the *TabletMain* functions like a homepage the *PhoneMain* is more of a wrapper for all the phone pages. It provides the app bar which displays the lecturers name and title, and the navigation bar at the bottom of the page to change between the different pages. It uses a different type of *FutureBuilder* called a *StreamBuilder* which uses a *Stream* to display information from a source and updates the information based on changes to the *Stream*. The *Stream* in this case is a direct connection to the lecturer database that stores information regarding the given user, and when the user updates the information on the client side it is reflected by changes to both the server side and client-side data.

Figure 9: Build method using a StreamBuilder

The *TabletMain* also uses the *StreamBuilder* to look for updates in the given lecturer’s database information and change the displayed information accordingly. As mentioned, it works like a home page with buttons taking the user to different pages with different features and information.

There are two joint pages that both the phone side and the tablet side of the app uses being *call\_page.dart* and *message\_page.dart*. Both sides of the application originally had separate pages for these features, but they were very similar in design and functionality, so by changing and optimising the code both now use the same two pages.

*CallPage* uses *AgoraIO* to host the video calls and uses *Heroku* as an online token generation service. Even with these two interfaces, there is a lot of initialisation and preparation needed to be done on the client side before the *CallPage* is ready. Every time either the lecturer or the visitor tries to join the channel it must be initialised as keeping it online constantly drains the battery of the device and uses valuable server space for other potential users. Event handlers for all events are also created so that when users joins, leaves or loses connection the app handles it correctly. Like most video calling applications, the caller fills the screen while the receiver can see themselves in a small video window in the corner. The build function of *CallPage* looks very simple but this is because most of the widgets are created in separate functions only to be combined in the build function. Below is one of these functions, showing how the large video view is created.

Figure 10: Function that returns a Widget containing the recipient video feed

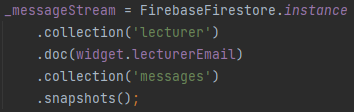
1*MessagePage* uses a database connected to the given lecturer database in *Firebase* to store messages sent between the user and the visitor. When a message is sent it is uploaded to the database and the new message is retrieved and shown for both users using a *StreamBuilder*. The messages are created using a custom created widget called *Message* and are displayed differently based on which user sent the message. Below is a screenshot showing the *Stream* that retrieves the messages.

Figure 11: Message stream retrieving messages from the database

### 2.1.3 Phone Pages

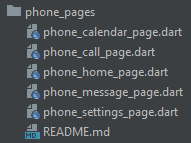
All pages used by the phone side of the app are stored in the *phone\_pages* folder. As mentioned these differ from the other pages because they all share the *PhoneMain* as their wrapper/parent widget.

Figure 12: Phone pages folder contents

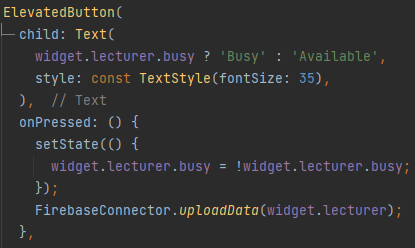
The first page shown after choosing the phone side of the app when logging in is the *phone\_home\_page.dart* with its class *PhoneHomePage*. It contains buttons where the user can change their status between busy/available and in office/out of office. This functionality is implemented using two buttons that show what the users’ current status is.

Figure 13: Button that updates the lecturers status

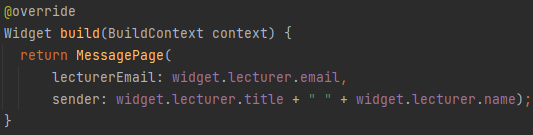
The next page on the phone side of the app is *phone\_message\_page.dart* with the class *PhoneMessagePage*. It has very little code in it as most of the code is provided by the shared *MessagePage* class. It differs slightly from its counterpart *TabletMessagePage* by the fact that the sender is set as the lecturer.

Figure 14: Build method of PhoneMessagePage

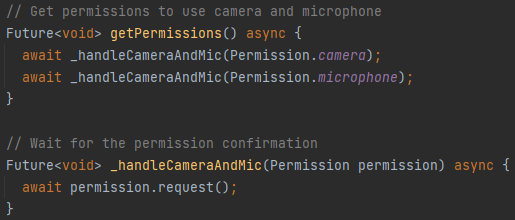
The middle page of the phone side of the app is *phone\_call\_page.dart* with the class *PhoneCallPage*. It also has very little code in it as most the work is done in *CallPage*. When the user uses the app for the first time, it asks for permission to use the camera and microphone, which then is saved in the application, and will launch without asking for permission the next time it is initiated.

Figure 15: Functions that asks user permission to use camera and microphone

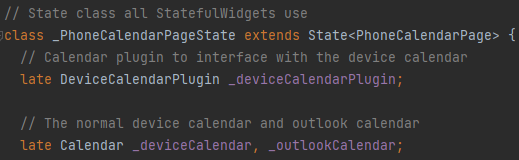
The second to last page of the phone side of the app is *phone\_calendar\_page.dart* with the class *PhoneCalendarPage*. It interfaces with the lecturers calendar and adds events to it. It has limited functionality by design as the actual interaction with the calendar is done on the tablet side of the app, but it provides the user with an overview of their calendar without having to leave the app. In the picture below is the *State* of this page, where another use of asynchronous coding is used, the late variable. This is on non-null variables where the data they store will be retrieved later in the code and is used to inform the compiler that the variables will be given a value before they are accessed.

Figure 16: Phone calendar page state, with late variables

The last page of the phone side of the app is *phone\_settings\_page.dart* with the class *PhoneSettingsPage*. It imports the lecturers data from the *Firebase* online database and displays what information is currently saved. The information is displayed in a form that the lecturer can update if needed and updates sent to the online database is automatically reflected on both the phone side and tablet side of the app. There is also the option to log out if needed.

### 2.1.4 Tablet Pages

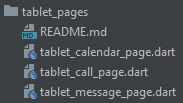
Besides the main page, the tablet side of the app has three pages which are *tablet\_calendar\_page.dart*, *tablet\_call\_page.dart* and ­­*tablet\_messages\_page.dart*. The two later pages looking and working very similarly to their counterparts on the phone side of the app.

Figure 17: Tablet pages folder content

The *TabletCalendarPage* class contained in the *tablet\_calendar\_page.dart* file shares the functionality of interfacing with the users’ calendar to display when they are available and not. The added functionality it provides is that a visitor can book meetings during the available times in the lecturers’ calendar, ensuring it does not overlap with any existing events. To book a meeting, the app uses a combination of text fields, a clock widget and a calendar widget, making it easy and intuitive to book a meeting at the desired time and date. Below is the code of the date picker.

Figure 18: Date picker button on TabletCalendarPage

### 2.1.5 Models

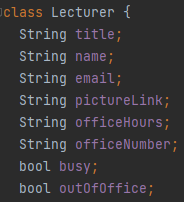
The *models* folder contains a single file, *lecturer.dart*, which provides the data structure that stores the information about the lecturer. It also provides two helper functions to pack and unpack it to and from a JSON format.

Figure 19: Lecturer model structure

### 2.1.6 Widgets

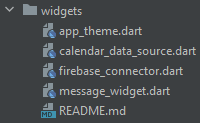
 The *widgets* folder contains an assortment of widgets used through the app, some of which have been mentioned already like *message\_widget.dart* and *app\_theme.dart*.

Figure 20: Widgets folder content

The file *app\_theme.dart* contains the class *AppTheme* that provides a custom theme used by the app. It uses a colour swatch, a type of gradient, to provide the app with a simple, elegant, and modern design.

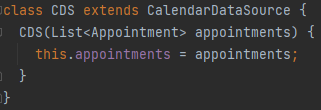
The file *calendar\_data\_source.dart* with the class *CDS* is a helper class needed for the app to retrieve and store events from the users calendar.

Figure 21: CDS class structure

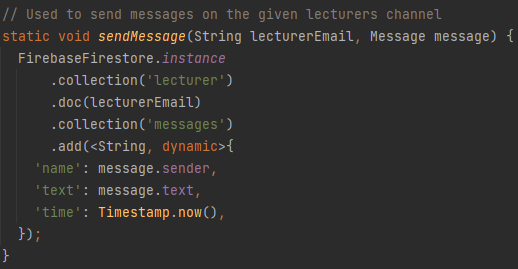
The custom made class *FirebaseConnector* is stored in the file *firebase\_connector.dart* and provides the app with a connection to the online *Firebase* services. It makes sure the connection is initialised and stays operative while the app is in use and helps retrieve and send data between the app and the online database. Below is one of its custom methods that uploads a message to the database.

Figure 22: Send message method

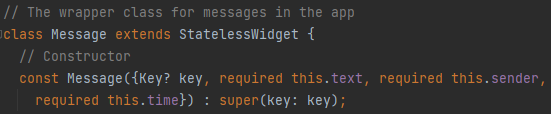
The class *Message* stored in *message\_widget.dart* is used as a wrapper to store and display messages when either of the message pages are displayed. It stores the content of the message, who sent it and when it was sent.

Figure 23: Message class structure

## 2.2 Visual Implementation

This section includes pictures of the app, including some shared pages, as well as the pages only used by the phone side and the pages only used by the tablet side of the application.

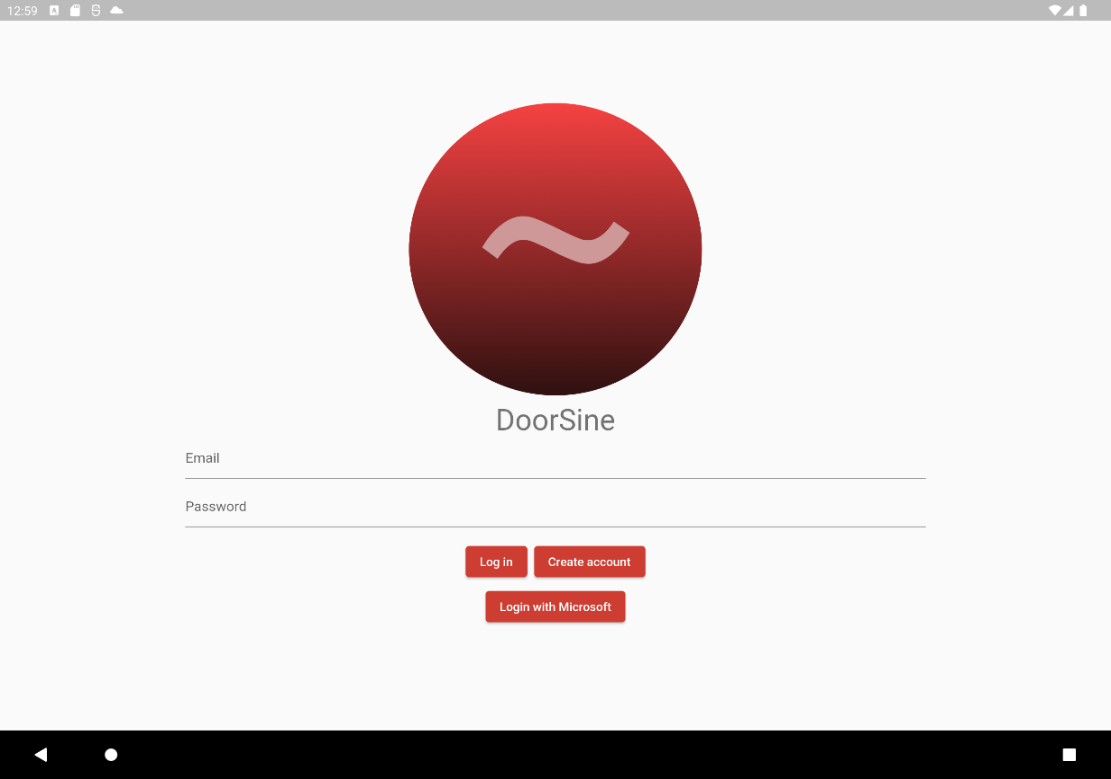
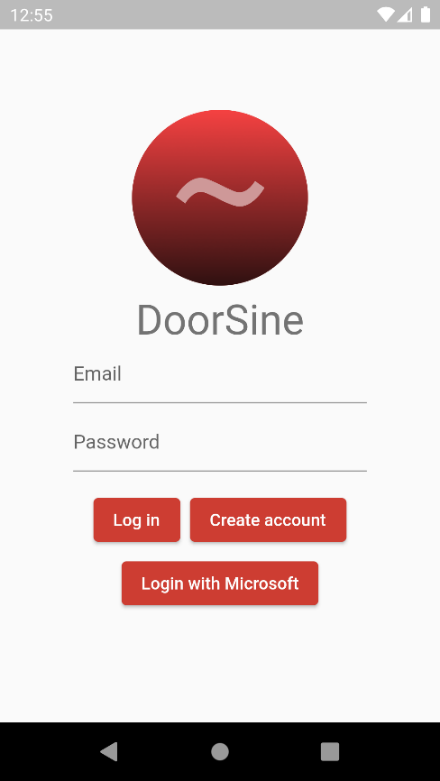
The login pages for the phone and tablet share the same page, but the size of the objects on the page is altered based on the size of the device. Here the user can choose to login using either an email and password combo or using a Microsoft account. If the user does not have an account yet, they can create one on the account creation page.

Figure 24: Phone login page

Figure 25: Tablet login page

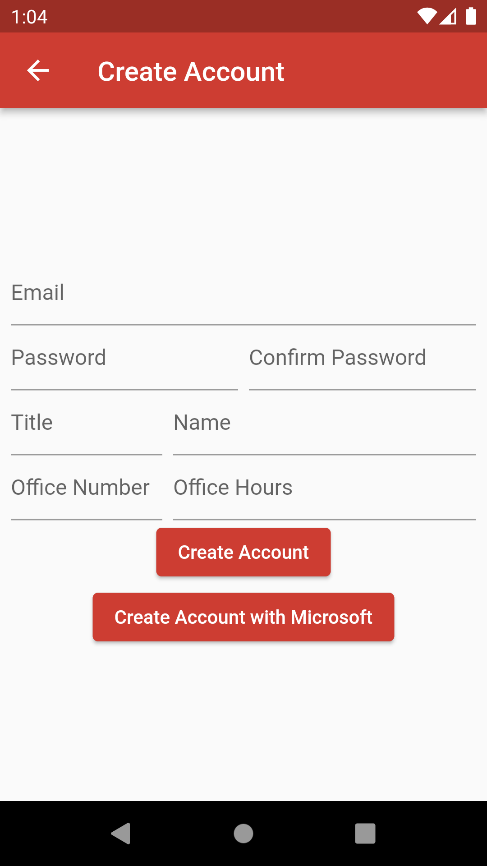
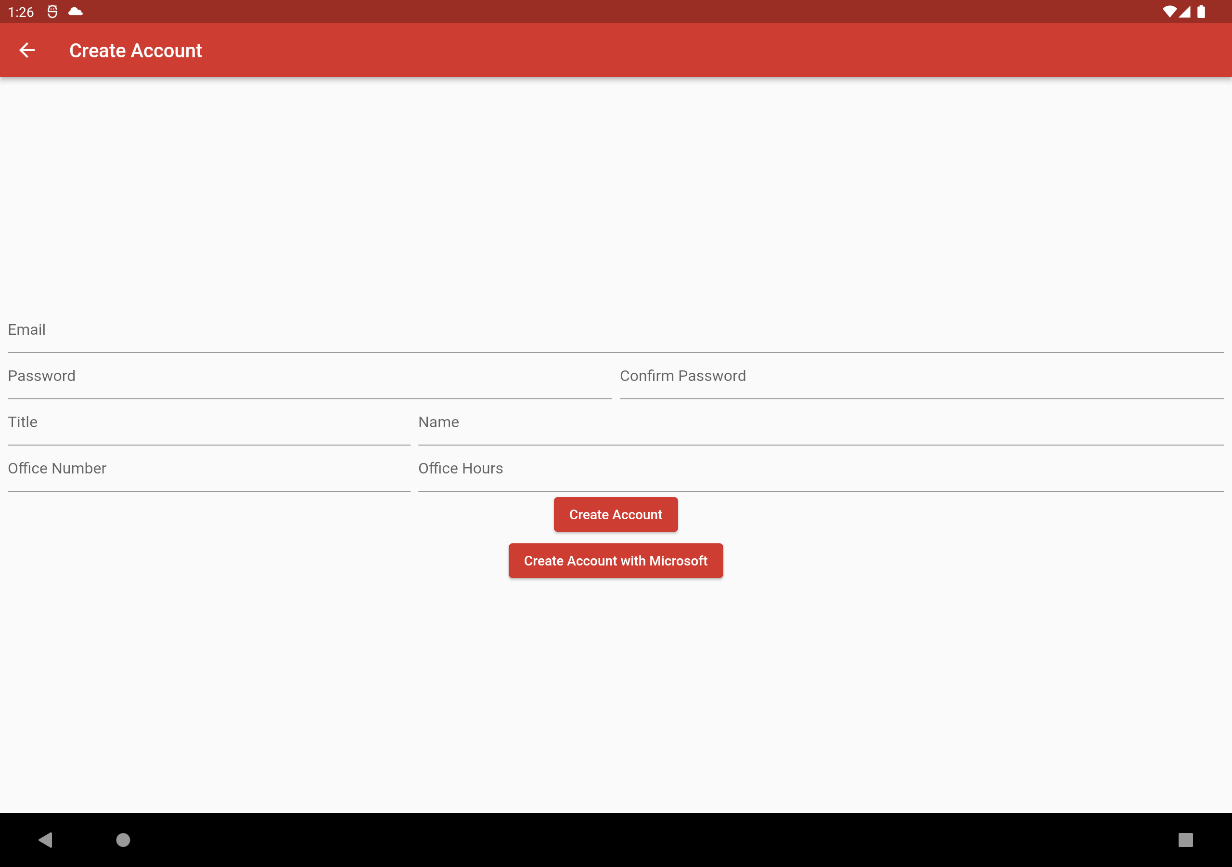
The account creation page works the same way as the login page where both the phone and the tablet use the same page but rezised to fit the individual screen size. Here the user has the option of creating an account by providing their own information or by linking their Microsoft account.

Figure 26: Phone account creation

Figure 27: Tablet account creation

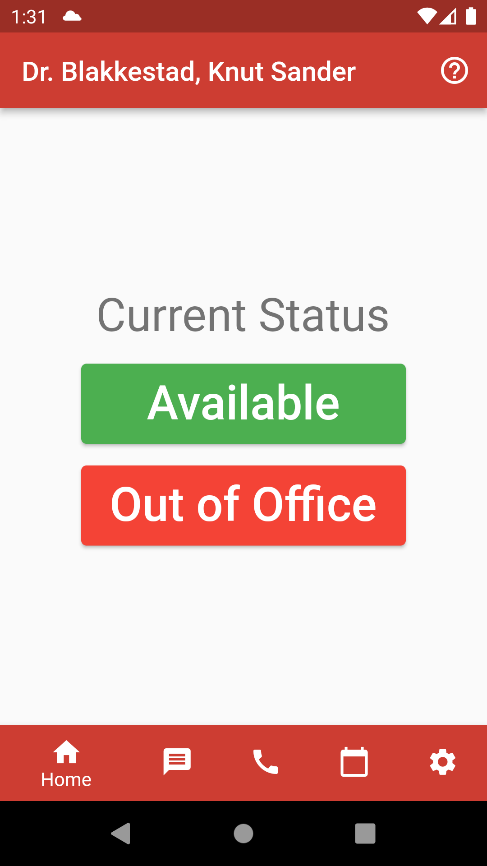
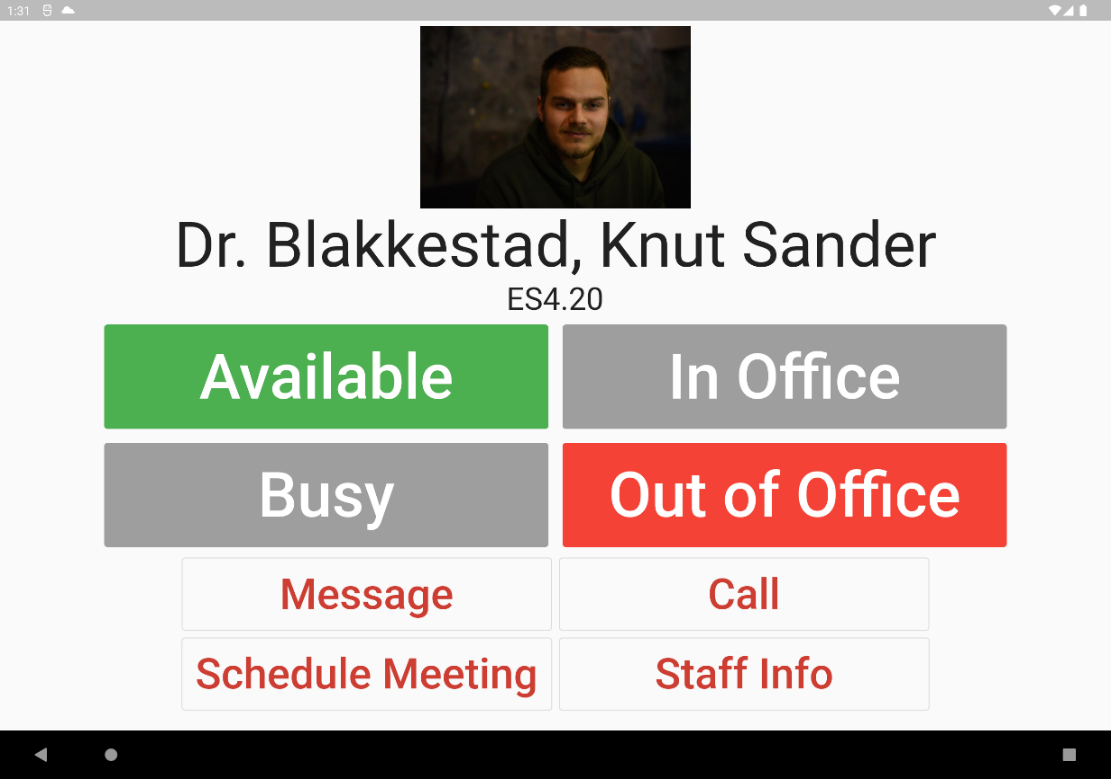
The pages the user is taken to after logging in is either the phone home page or the tablet main page based on what they choose in the pop-up menu that appears. As mentioned earlier in the report the phone side of the app has a main page, but it only contains the top and bottom bar. The content between these two parts are provided by separate pages, the pictured one being the home page. This provides updates to the tablet main page which displays the lecturers information, as well as providing buttons to navigate to other pages.

Figure 28: Tablet home page

Figure 29: Phone home page

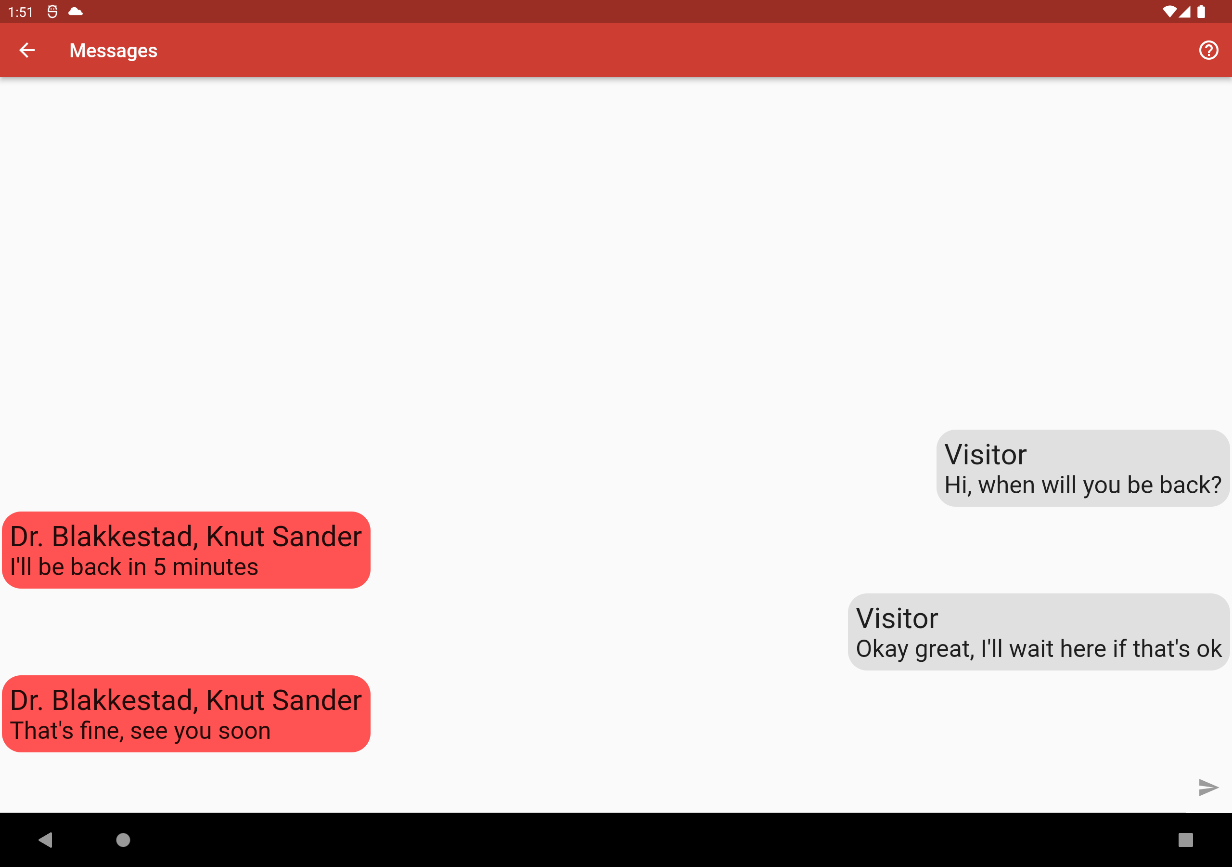
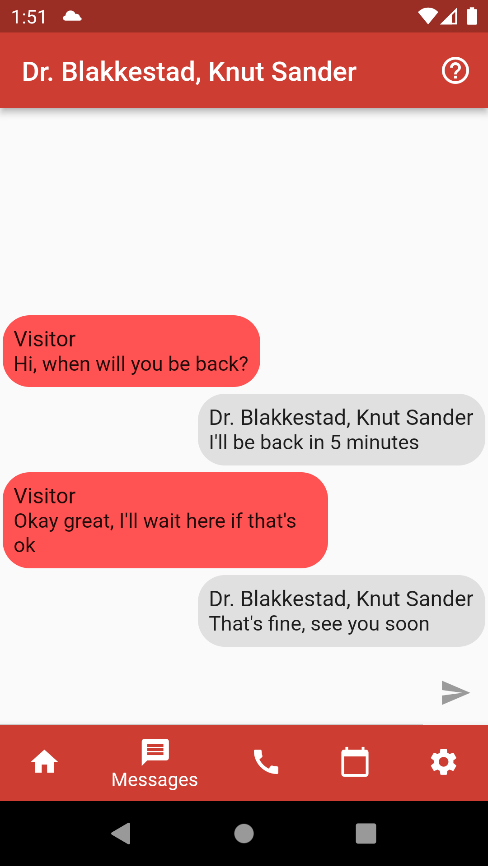
The message pages of both sides of the app share most of their functionality, with the messages sent from the device displayed on the right side of the screen and the messages received displayed on the left side of the screen. They are also sorted based on when they were sent, with newer messages being displayed towards the bottom and older messages displayed towards the top.

Figure 30: Phone message page

Figure 31: Tablet message page

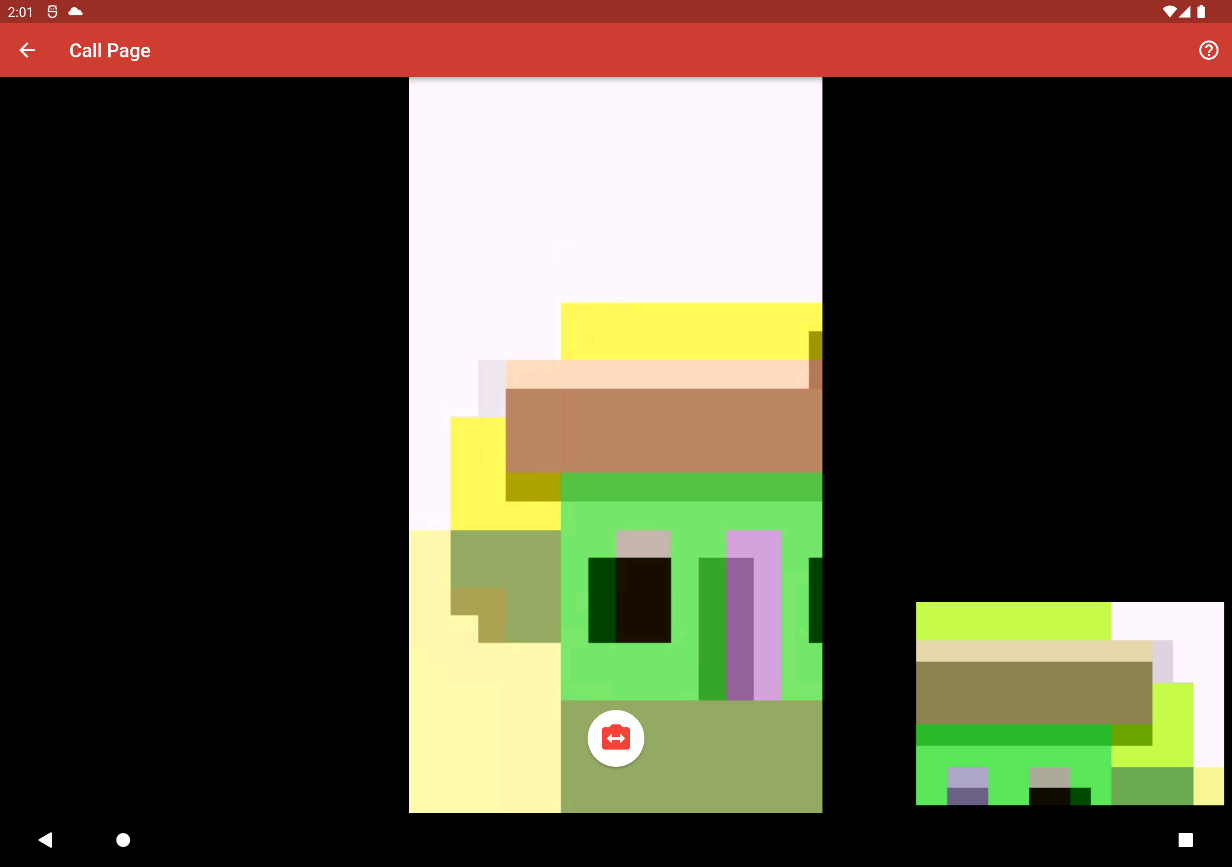
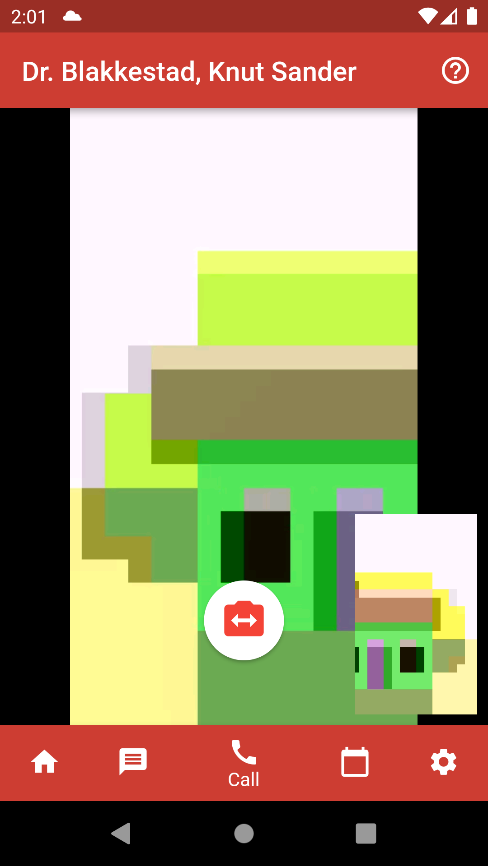
The call pages of both sides of the app also share most of their functionality, with the device user being shown in a smaller video window in the lower corner, while the caller is shown central on the page. The user of either device also has the option of flipping their cameras if necessary.

Figure 32: Phone call page

Figure 33: Tablet call page

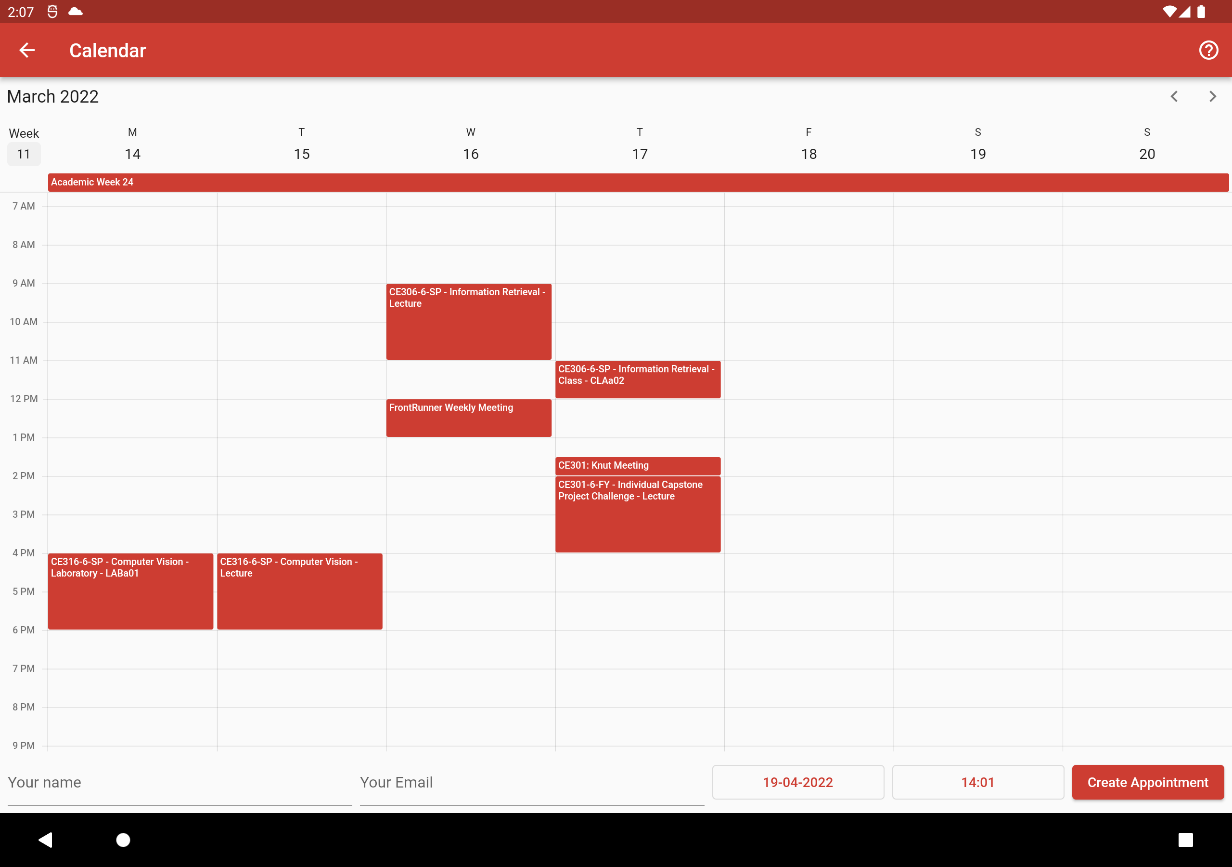
The calendar pages of the two sides of the app differ in function, as the phone calendar page is used for conveniences sake so the user does not have to close to app to check their work calendar. The tablet calendar page however is used to book meetings with the lecturer, giving a name and an email, and then choosing an available time on a given date. These meetings are automatically added to the lecturers calendar, and they can then contact the person who wants a meeting for further planning.

Figure 34: Phone calendar page

Figure 35: Tablet calendar page

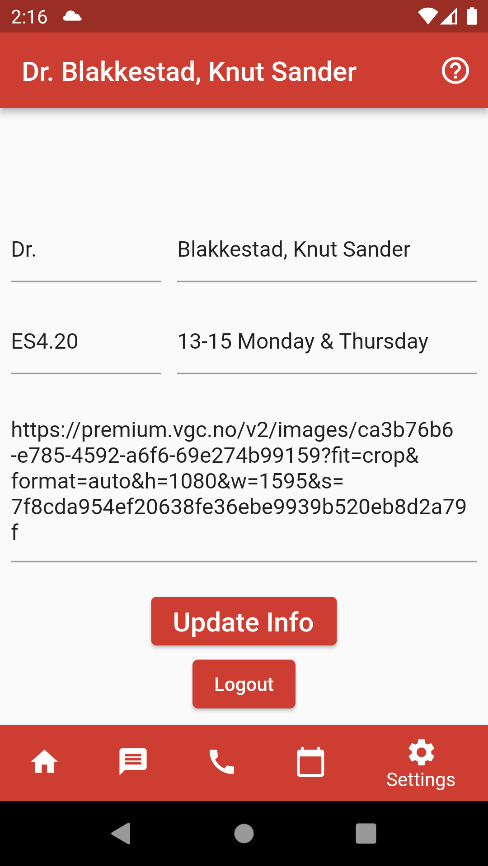
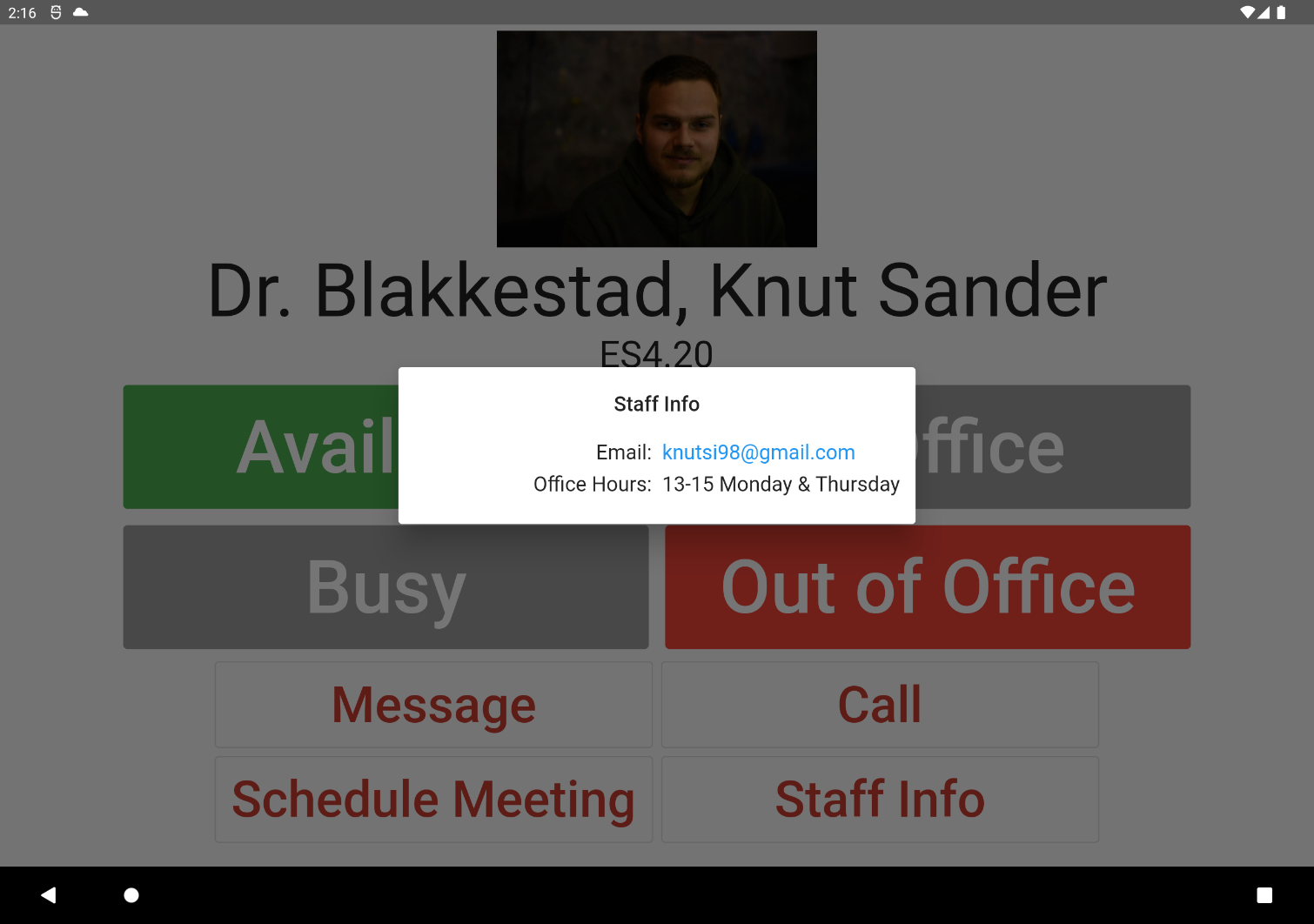
The last page of the phone side of the app is the settings page where the lecturer can change their displayed information. A logout button is alsi provided for convenience.  
The last button on the tablet main page does not lead to a different page, but displays a pop-up with information that is not displayed on the main page.

Figure 36: Tablet information pop-up

Figure 37: Phone settings page



# 3. Project Testing

With most modern projects, testing is an essential part of the development process. Testing that the application works like intended is important, and there are mainly two ways to do this. The first is creating tests using code and running them using code coverage, making sure that everything runs smoothly and returns the correct values and objects. The second is to test using manual tests by creating scenarios and expected results and then write down the results, potentially discovering bugs that need to be fixed in the process.

Testing an application using unit tests is possible, but with a highly interactive application like DoorSine it is difficult to ascertain from simple inputs and outputs if the code has executed correctly. The code could run and return the expected output, but without visualising the result it is impossible to determine whether the output looks like it should. For this reason, manual testing has been chosen as the primary and only way of testing DoorSine.

From the manual testing a few bugs were discovered and solved, but most of the applications functionality worked as expected. This is because most bugs were discovered either during or right after something was implemented, meaning there was little need to login to Jira to create a bug that could be solved in the time it takes to login to the Horizon webserver. More bugs would likely have been reported on Jira if it was not for this.

## Shared Tests

* Testing *LoginPage* functionality
  + Typing neither an email nor a password
    - PASS
    - User is asked to provide email and password
  + Typing a non-existent user email and password combo
    - FAIL
    - User is not logged in, but no error message is displayed.
    - STATUS: Solved 18/04/2022
  + Using an unlinked Microsoft account to log in
    - FAIL
    - Microsoft user is prompted to choose device, but this crashes as there is no data to retrieve
    - STATUS: Solved 18/04/2022
* Testing *CreateAccountPage* functionality
  + Not providing information in form fields
    - PASS
    - Error messages display what information is missing
  + New account using form
    - PASS
    - New user is created and sent to log in screen
  + New account using Microsoft account
    - PASS
    - New user is created and sent to log in screen
  + New account with weak password
    - PASS
    - Error message displays that the chosen password is too weak
  + New account with existing email
    - PASS
    - Error message displays that the email is already in use
  + New account with existing Microsoft account
    - FAIL
    - Signing up with an existing Microsoft account overrides existing data
    - STATUS: SOLVED 18/04/2022

## 3.2 Phone Tests

* Testing *PhoneMain* functionality
  + Using the bottom navigation bar to change pages
    - PASS
    - Navigation works as expected and changing between the pages is quick and loads the pages quickly. Trying to change pages rapidly is not a problem.
  + Page info display
    - PASS
    - The info button displays the correct information based on the displayed paged
* Testing *PhoneHomePage* functionality
  + Pressing the buttons updates the info on the database
    - PASS
    - Availability and location is updated correctly and quickly.
  + Pressing the buttons rapidly still updates the data correctly
    - PASS
    - Data was updated correctly even when spamming
* Testing *PhoneMessagePage* functionality
  + A message without content cannot be sent
    - PASS
    - The send button is disabled when no content is given
  + Sent message displays as expected
    - PASS
    - Sent messages is displayed on the right side
  + Received messages are displayed correctly
    - PASS
    - Received messages is displayed on the left side
* Testing *PhoneCallPage* functionality
  + Page is loaded and displays the user
    - PASS
    - Page loads and ask the user for permission to use camera and microphone if its used for the first time. Displays the user in the bottom right corner.
  + Caller is displayed and audible
    - PASS
    - When a call is received the caller is visible and audible on the call page.
  + User is displayed and audible
    - PASS
    - User is visible and audible on the callers device
* Testing *PhoneCalendarPage* functionality
  + Page loads and displays the calendar
    - PASS
    - Pages loads and displays the users calendar. Only relevant information from the calendar is retrieved and displayed.
* Testing *PhoneSettingsPage* functionality
  + Information is updated correctly
    - PASS
    - Updating the information on the phone uploads it correctly to the database. The updated data is also displayed in the app
  + Information must be provided to update the database
    - FAIL
    - Providing no input in any of the fields is allowed.
    - STATUS: Solved 18/04/2022
  + Logout button logs the user out and returns them to the login page
    - PASS
    - Logout button works as expected

## 3.3 Tablet Tests

* Test *TabletMain* functionality
  + Changes on the database is reflected on the main page
    - PASS
    - Information is updated, displayed correctly and quickly
  + Buttons lead to the correct pages
    - PASS
    - All buttons takes the user to the correct page or displays the correct info
* Testing *TabletMessagePage* functionality
  + A message without content cannot be sent
    - PASS
    - The send button is disabled when no content is given
  + Sent message displays as expected
    - PASS
    - Sent messages is displayed on the right side
  + Received messages are displayed correctly
    - PASS
    - Received messages is displayed on the left side
  + Information button displays page info
    - PASS
    - Information relevant to the page is displayed
* Testing *TabletCallPage* functionality
  + Page is loaded and displays the user
    - PASS
    - Page loads and ask the user for permission to use camera and microphone if its used for the first time. Displays the user in the bottom right corner.
  + Caller is displayed and audible
    - PASS
    - When a call is received the caller is visible and audible on the call page.
  + User is displayed and audible
    - PASS
    - User is visible and audible on the callers device
  + Information button displays page info
    - PASS
    - Information relevant to the page is displayed
* Testing *TabletCalendarPage* functionality
  + Page loads and displays the calendar
    - PASS
    - Pages loads and displays the users calendar. Only relevant information from the calendar is retrieved and displayed.
  + All required information must be provided
    - PASS
    - Not providing the necessary information results in error messages being displayed stating what is missing.
  + Creating a meeting in an available timeslot works
    - PASS
    - Meeting is created and displayed in the calendar
  + Creating a meeting that overlaps with another event is not possible
    - PASS
    - An error message displays telling the user that the time chosen is not available and to choose an available time slot.
  + Information button displays page info
    - PASS
    - Information relevant to the page is displayed

# 4. Project Planning

Jira Link: <https://cseegit.essex.ac.uk/ce301_21-22/CE301_blakkestad_knut_s_l>

GitLab link: <https://cseegit.essex.ac.uk/ce301_21-22/CE301_blakkestad_knut_s_l>

## 4.1 Planning and Execution

From the beginning of the project development cycle a clear plan was created. A list of functionalities that was to be implemented, with the details being discussed and agreed upon beforehand. A steady stream of new features was delivered every week by focusing on one feature at a time and moving on when deemed finished.

As the halfway mark of the project was closing in, and with only a single feature of the original plan missing, it was agreed that more features were necessary for the project to include enough hours of work. This was important for the project, as without this the project would have lost momentum to early and a lack of development time could have been questioned. Adapting this new feature into the application framework took many hours of work but did not feel out of place with the other feature, rather it elevated the usefulness of the application and has left it feeling more complete than if it did not have it.

## 4.2 Jira and GitLab Use

It would be a missed opportunity to not mention the negative effect the reliance on the *Horizon* webserver has had on the use and usefulness of *Jira* this year. It takes about 2 minutes to access Jira, something that would take 10 seconds or less in previous years. Not to mention the fact that Horizon has been inaccessible at several crucial points through the year, further showing how unreliable it is. This has had an impact on how much work was record on Jira, as many smaller tasks did not feel worth spending 3 minutes creating knowing that doing them would take at most 10 minutes. In turn this likely has had an impact on how many hours of the work done on the project have been logged and could reflect negatively on students just because accessing the system is slow and unreliable.’

Making excuses based on it not being safe otherwise when newer versions of *Jira* support two-factor authentication is simply not good enough. A more modern solution like a VPN to connect to the school’s network or simply upgrading Jira would have made Jira a more useful work management tool, and I sincerely hope for that this system is improved for future students.

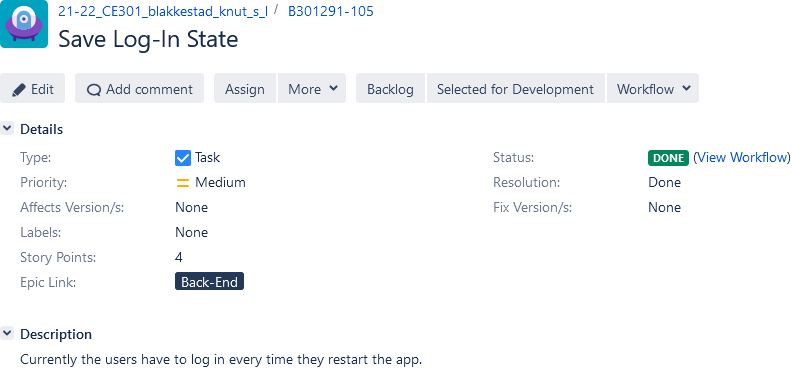
Most work undertaken were recorded in *tasks* with some work being recorded in *bugs*. A task would include a highly descriptive title to explain the task in just a couple of words. The description would provide more information regarding the *task* if necessary, but would often be left empty as the title gave a good enough description for me to remember what to do. All *tasks* would also be labelled with an *epic*, denoting what aspect of the project it belonged to and organising the tasks into groups.

Figure 38: Task example

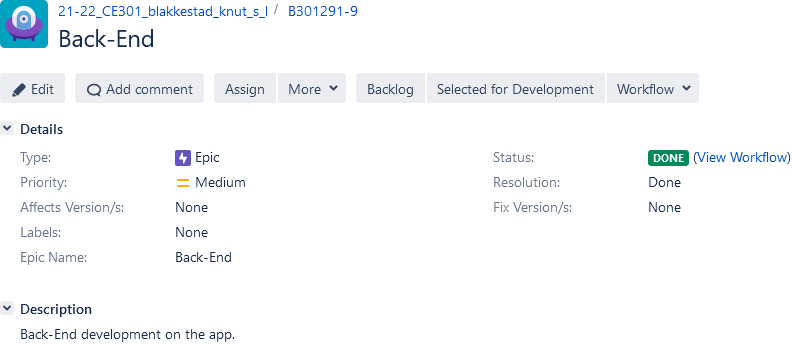
As mentioned several *epics* were used during the applications development, with the “Back-End” *epic* having the most *tasks* and *bugs* assigned to it as the development has been highly focused on the back-end of the app, providing it with all the functionalities it displays.

Figure 39: Epic example



Figure 40: Story example

Another helpful type of *issue* provided by *Jira* was the *story*. It was used to portray what features a potential user could want from the app, and was the end goals of many hours of development. After a feature was implemented, the *story* would be moved to “Done” column.

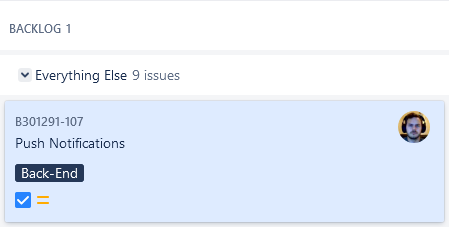
Moving onto the use of the Kanban board and how is set-up, this is how I have used it. The “Backlog” column is where all newly created *issues* are placed and where *issues* currently not being worked on are stored.

Figure 41: Backlog column

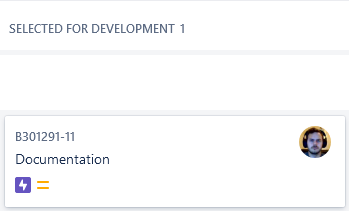
The next column on the Kanban board is “Selected For Development”, and was primarily used for storage of *epics* and *stories*. This is because in Jira older *issues* are displayed highest in the column, meaning new *issues* would be placed out of sight. Being the sole developer of this project, I felt this was a good use of a column that would be left without use otherwise.

Figure 42: Selected For Development column

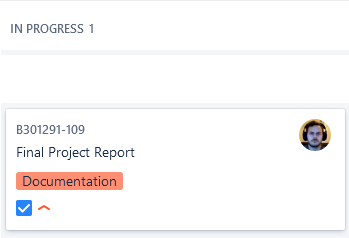
The third column is “In Progress” and was used for *issues* currently being worked on. Because Jira was less accessible than before, when an *issues* was created it was immediately moved from the “Backlog” into “In Progress” as keeping Horizon open without constantly activating it would log the user out.

Figure 43: In Progress column

The fourth and final column is “Done” where all *tasks*, *epics*, *stories,* and *bugs* were moved when deemed fixed or done. Moving *issues* to this column was often were the hours of work on the given *issue* were logged.

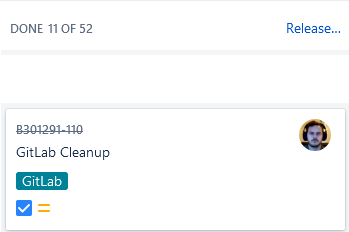
Together with *Jira, GitLab* has been heavily used to manage the project. It contains the project in its entirety and several README.md files have been added to explain the project, as well as smaller parts. *Commits* to *GitLab* has always contained a meaningful message detailing what updates and changes have been done, and when a *tasks* either have hours logged to it or are marked as done, a URL to the relevant *commit* is linked as a comment.

Figure 44: Done column

## 4.3 Risk Management and Bugs

Risks are a prevalent part of all development and during the development of this application risks has been a constant factor I have had to consider. Implementing all of the outside technologies poses a great risk, as with every update to them the risks of the app breaking increases without continued development. If *Firebase* suddenly changes how data is stored then large parts of the code would need to be rewritten to accommodate it. Services could change company owners recuring new accounts being created and more setup and changes to the code.

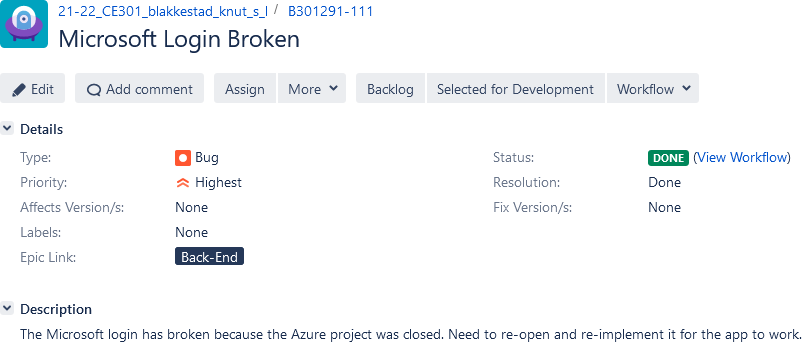
With all the risks of app development *bugs* can, are and will be prevalent. Luckily most bugs discovered during the development cycle were fixed as soon as they were discovered, and because of the unreliability of *Jira* most of these were not logged. The *bugs* discovered during testing were logged, and take on a similar structure to *tasks* as seen below.

Figure 45: Bug example

# 5. Conclusion

## 5.1 Achievements

The three points mentioned in the “Aims and Objectives” section of the “Project Context” chapter was the original project scope and was supposed to occupy the work for the entire project. By the interim presentation before the end of autumn term, the application had the following features implemented.

* The staff could set their status as available or busy, as well as indicate whether they were in or out of office. This was done using the app on their phone, and changes appeared instantly on the tablet on the door.
* Messaging was implemented and working.
* The staff could change their basic information easily through their app if office hours or related info changed.
* An account creation system was created so anyone who wanted to use the app could create their own account.

Looking at what had been achieved so far, and with the functionality of booking meetings being the only remaining feature to not have been implemented, it was agreed that another feature was needed to fill the hours required for the project. This feature was the ability for someone at the staff door to call the staff using video. With this, the final iteration of the project has the following features in addition to what has been mentioned before.

* A person can book a meeting if they are at the staff door. The app interacts with the staff’s calendar to display when they are busy or available. Meetings cannot be booked if they overlap with other events.
* Calling is implemented and working.
* Every page has a button that give some basic information about the given page of the app, to help users unfamiliar with it.
* Users with a Microsoft account have the option of linking their account with DoorSine, using their Microsoft login to access the app.
* Visual improvements to make the app more user friendly.
* Software improvements to make the app run smoother and more efficient.

With all these features, the app has completed the goal of being a assistant for a staff office door with more features than initially planned. It is ready for the integration testing stage of its lifecycle, in need of testing in a real environment for a longer period.

## 5.2 Future Development

Even though the application is ready for deployment I still have ideas for more features that would improve it further.

An earlier addition could be adding login options for other external sites like Facebook. Google, Apple, and others. This would open the app for a wider range of users and make setting up an account easier. In theory the application should run on IOs devices, but as I do not own any, I have been unable to test this. Getting an IOs device to get the application working on both Android and IOs would also be an important addition.

For the benefit of the lecturers, adding ways of disabling features when they are unavailable or outside of working hours would help them manage notifications better. This could be added to the settings page, allowing them to disable and enable features at will. This could then be further developed into allowing them to set time periods where they want certain features disabled, even linking it up with the calendar to automatically change their status based on it.

A big development even further in the future of the lifecycle of the app would focus on allowing the lecturers to modify the tablet screen to make it more personalised. Giving them the ability to move, resize, add, and remove objects via a web-based design tool. Several pre-set designs would also be made readily available for the lecturers to pick and choose from. This would then be further developed to expand beyond usage of lecturers only and make it something any company could have the use for.

## 5.3 Final Words

This project has been a learning experience from start to finish, and I have had a great time learning the intricacies of application development. Learning a new programming language, working with a totally unknown framework, and implementing several different technologies has been challenging but has taught me so much that I can take with me into the future. I once again want to thank Dr. Shoaib Jameel for supervising this project and thank whoever you are reading this for taking the time to do so.

# 6. References

|  |  |
| --- | --- |
| [1] | S. O'Dea, “Number of smartphone subscriptions worldwide from 2016 to 2027,” Statista, 23 February 2022. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed 23 March 2022]. |
| [2] | S. O'Dea, “Do you personally use a smartphone?,” Statista, 15 June 2021. [Online]. Available: https://www.statista.com/statistics/300402/smartphone-usage-in-the-uk-by-age/. [Accessed 23 March 2022]. |
| [3] | A. M. a. P. K. M. E. Joorabchi, “Real Challenges in Mobile App Development,” 2013. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6681334. [Accessed 23 March 2022]. |
| [4] | D. D. K. T. K. M. I. Venkata N. Inukollu, “Factors Influencing Quality of Mobile Apps:Role of Mobile App Development Life Cycle,” 16 October 2014. [Online]. Available: https://arxiv.org/abs/1410.4537. [Accessed 2022 March 28]. |
| [5] | L. Ceci, “Number of mobile app downloads worldwide from 2016 to 2021,” Statista, 31 January 2022. [Online]. Available: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. [Accessed 28 March 2022]. |

1. <https://moodle.essex.ac.uk/mod/book/view.php?id=759993&chapterid=19559>, 18.03.2022 [↑](#footnote-ref-1)
2. <https://medium.com/javarevisited/top-5-programming-languages-for-mobile-app-development-in-2021-19a1778195b8>, 23.03.2022 [↑](#footnote-ref-2)
3. <https://www.spinxdigital.com/blog/mobile-app-development-languages/>, 23.03.2022 [↑](#footnote-ref-3)
4. <https://medium.com/javarevisited/top-5-programming-languages-for-mobile-app-development-in-2021-19a1778195b8>, 15.09.2021 [↑](#footnote-ref-4)
5. <https://dart.dev/>, 22.03.2022 [↑](#footnote-ref-5)
6. <https://dart.dev/overview>, 22.03.2022 [↑](#footnote-ref-6)
7. <https://flutter.dev/>, 23.04.2022 [↑](#footnote-ref-7)
8. <https://www.door-tablet.com/door-app/home>, 28.03.2022 [↑](#footnote-ref-8)
9. <https://www.meetio.com/customers>, 28.03.2022 [↑](#footnote-ref-9)
10. <https://www.condecosoftware.com/>, 28.03.2022 [↑](#footnote-ref-10)
11. <https://pronestor.com/uk>, 28.03.2022 [↑](#footnote-ref-11)
12. <https://dart.dev/overview>, 31.03.2022 [↑](#footnote-ref-12)
13. <https://flutter.dev/>, 31.03.2022 [↑](#footnote-ref-13)
14. <https://firebase.google.com/>, 31.03.2022 [↑](#footnote-ref-14)
15. <https://www.agora.io/en/>, 31.03.2022 [↑](#footnote-ref-15)
16. <https://www.heroku.com/>, 31.03.2022 [↑](#footnote-ref-16)
17. <https://azure.microsoft.com/en-us/>, 31.03.2022 [↑](#footnote-ref-17)
18. <https://firebase.google.com/downloads/gdpr/NOV2021_Firebase_ISO_27001.pdf>, 31.03.2022 [↑](#footnote-ref-18)
19. <https://cloud.google.com/security/compliance/iso-27017>, 31.03.2022 [↑](#footnote-ref-19)
20. <https://cloud.google.com/security/compliance/iso-27018>, 31.03.2022 [↑](#footnote-ref-20)
21. <https://us.aicpa.org/interestareas/frc/assuranceadvisoryservices/aicpasoc1report>, 31.03.2022 [↑](#footnote-ref-21)
22. <https://us.aicpa.org/interestareas/frc/assuranceadvisoryservices/aicpasoc2report>, 31.03.2022 [↑](#footnote-ref-22)
23. <https://firebase.google.com/downloads/gdpr/DEC2021_Firebase_SOC3_Report.pdf>, 31.03.2022 [↑](#footnote-ref-23)