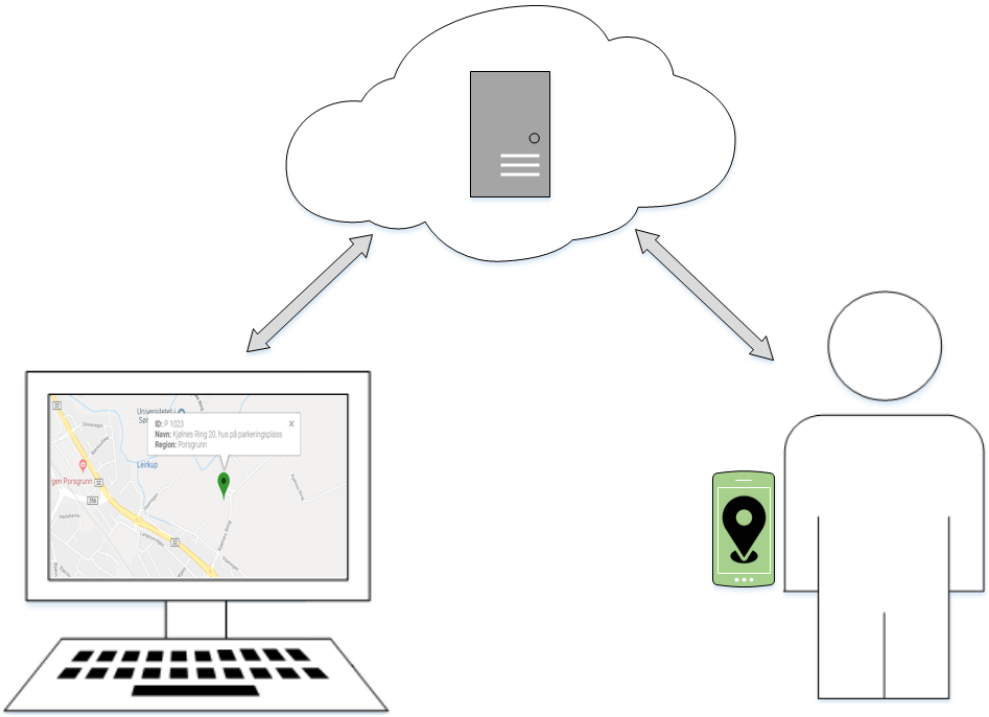
University of South-Eastern Norway

www.usn.no

PRH612-1 18V Bachelor thesis

Real-Time Notification System for Electrical Substations



IA6-6-18

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: PRH612-1 18V Bachelor thesis, 2018

Title: Real-time notification system for electrical substations

This report forms part of the basis for assessing the student's performance in the course.

Project group: IA6-6-18 **Availability:** Open

Group participants: Shing Wai Chan
Sander Heldal
Casper Nilsen
Lars Kittang Remme

Supervisor: Hans-Petter Halvorsen

Project partner: Skagerak Energi

Approved for archiving: _____

Summary:

The aim of the project was to develop a software solution for Skagerak Energi. The new solution is meant to be a replacement of an existing system that is used by technicians to notify the operating centre about which electrical substations they are working on. It is there to ensure the safety of the workers by giving the operating centre an overview of the ongoing jobs.

The system primarily consists of three parts: database, website and mobile application. The mobile and web application were developed in Visual Studio using Xamarin and ASP.NET Web Forms. MS SQL Server was used for database management. The cloud database and services for the applications were hosted on Microsoft Azure. As well as a connection to Skagerak's AD, allowing the applications to authenticate users.

The project resulted in a complete system named GridNote. The user-friendly mobile application lets technicians notify the operators directly by using the app. The web application for the operators show all active work in a well-presented manner. The database stores data from both applications using cloud services which also acts as the communication medium.

Preface

This bachelor thesis is written by four students at the University of South-Eastern Norway during the last semester of the engineering course Computer Science and Industrial Automation. The thesis is documentation for a product which will be used by the electrical power company Skagerak Energi. The product is called GridNote and consists of a web application, a mobile application and a database.

Each phase in the development process focused on creating a deployable product. Having basic knowledge about system development and programming is an advantage for the reader. Code elements such as methods and classes are written in italic text in this report. The language of the mobile and the web applications is in Norwegian, as requested by the project partner.

The system was developed in close collaboration with Skagerak Energi, mostly through our contact person, Marius Dolven, but also the company's IT department. All source code for the GridNote system can be found as a compressed file in the following link: https://www.dropbox.com/sh/pjfusyjyyhzpy0t/AACWmqn-FdKf5UxLBaO6f4tDa?dl=0

We would like to express our gratitude towards the project supervisor, Hans-Petter Halvorsen, for all the guidance and support. We would also like to thank our project partner Skagerak Energi, and especially Marius Dolven, for guidance and feedback during work on the thesis.

Tools used:

- Android Emulator
- Azure Cloud Service
- Google Chrome – version 66.0.3359.139 64-bit
- Microsoft Excel
- Microsoft Project
- Microsoft Visio
- Microsoft Word
- SQL Server 2014
- Visual Studio 2017
- Visual Studio Team Services

Porsgrunn, 16.05.2018

Shing Wai Chan	Sander Heldal
Casper Nilsen	Lars Kittang Remme

Nomenclature

1NF	–	First Normal Form
2NF	–	Second Normal Form
3NF	–	Third Normal Form
AAD	–	Azure Active Directory.
AD	–	Active Directory, (Directory Service for Windows Domain networks).
ADAL	–	Active Directory Authentication Libraries
AJAX	–	Asynchronous JavaScript And XML
API	–	Application Programming Interface
App	–	Application
ASP.NET		Server-side web application framework.
BCNF	–	Boyce and Codd Normal Form
CIL	–	Common Intermediate Language
CLI	–	Common Language Infrastructure
CLR	–	Common Language Runtime
CSS	–	Cascading Style Sheet
ECMA	–	European Computer Manufacturers Association
ETRS89	–	European Terrestrial Reference System 1989 (also called Euref 89)
GPS	–	Global Positioning System
GUI	–	Graphical User Interface
HSE	–	Health, Safety and Environment
HTML	–	Hypertext Mark-up Language
HTTP	–	Hypertext Transfer Protocol
IDE	–	Integrated Development Environment
IP	–	Internet Protocol
JSON	–	JavaScript Object Notation
MIT	–	Massachusetts Institute of Technology
MS	–	Microsoft
MVC	–	Model View Controller
MVP	–	Minimum Viable Product
NuGet	–	Microsoft supported mechanism for sharing code in packages.

OS	–	Operating System
PCL	–	Portable Class Libraries
SLA	–	Service-Level Agreement
SMS	–	Short Message Service
SQL	–	Structured Query Language
TCP	–	Transmission Control Protocol
UI	–	User Interface
UML	–	Unified Model Language
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
UTM	–	Universal Transvers Mercator (Coordinate system)
UWP	–	Universal Windows Platform
UX	–	User Experience
VS	–	Visual Studio
VSTS	–	Visual Studio Team Services
WBS	–	Work Breakdown Structure
WGS84	–	World Geodetic System 1984

Contents

Part I. Planning & Research

This part introduces the project and gives insight into the planning, design and research of the project.

1 Introduction

Today's industry is strongly characterized by work efficiency and strict HSE regulations surrounding the work area. An effort to meet these challenges has been the increasing implementation of technological solutions to improve the work environment for employees. In this development, the smartphone has established itself as a primary platform for digital communication and information.

This project was initiated to replace the existing SMS system currently in use by Skagerak Energi. This system is used to register the technicians' work on substations. The technicians send their call-in to a third-party company that displays it on screens at the operating centre.

Skagerak Energi requested an improved solution based on the use of modern technology. Such as a mobile application and a website supported by cloud services. In addition to a system that is easier to use and that take up less of the technicians' time, Skagerak Energi also wanted to present the information at the operating centre in an organised way.

The system was created to fulfil the following goals:

- To improve upon the present communication situation where technicians are required to notify the operating central by SMS.
- To increase efficiency of communication, while maintaining HSE routines.
- To use modern development tools to create an operational communication system for both technicians and operators.
- To create a routine tool for technicians to use when they are working in an electrical substation.

1.1 Scope

The assigned task for this bachelor thesis, which is attached as Appendix A, describes Skagerak Energi's present communication system and their requirements for the new system developed in this project.

Technicians can register a task and automatically notify both operating central and customer support, as mentioned in the goals. Considering the number of project members and the timeframe for this project, the focus has been to make a working product with at least basic functionalities and then continue expanding during the development process. To help define the scope, the following limitations were established:

- Develop a complete communication system for automatic data retrieval of registered jobs, submitted by technicians, and presentation of this data in an informative and graphical manner.
- The system should contain a website for displaying active registered jobs, a database for storing data, a mobile application for registering jobs, and cloud service for hosting applications.
- The mobile application should be able to support multiple platforms, but first be developed for Android.
- It should be a real-time system where information should be updated within a specified timeframe.

1.2 Project Workflow

Several development tools were used in this project. To develop the web and mobile applications, Visual Studio is used with respectively, ASP.NET and Xamarin Studio. MS SQL server is used to create, modify and maintain the database. Azure Cloud is used to host both the website and the database, as well as provide cloud services for mobile application.

The project's goals, WBS and Gantt are all added as attachments for further reading:

- Appendix B Project goals
- Appendix C WBS
- Appendix D Gantt

Project management tools used in this project are:

- Blog page created by Hans-Petter Halvorsen is a project management system for developers.
- Visual Studio Team Services is used as a source control for code.
- Dropbox is used as a collective document storage.

1.3 Reading Guidelines

Chapter 2 gives a brief description of how the system will work and a short overview of the planned solution.

Chapter 3 discusses planning and concept design for the website.

Chapter 4 discusses planning and concept design for the mobile application.

Chapter 5 is about the development tool for mobile application, what programming language it uses and how it can provide cross-platform coding.

Chapter 6 is about Microsoft Azure and how their services applies to this project.

Chapter 7 presents the system's database and how it communicates with Microsoft Azure.

Chapter 8 presents the current solution for the website and describes the development process.

Chapter 9 presents the current solution for the mobile application and describes the development process.

Chapter 10 gives a brief description of how testing of the system was performed.

Chapter 11 discusses different aspects of the system.

Chapter 12 provides suggestions for future improvements.

Chapter 13 gives a summary of the project.

2 System Overview

This chapter gives an insight into GridNote, the communication system developed in this project, and how individual parts of the system are connected. GridNote is a system consisting of an application for mobile devices, Azure Cloud Service and a website as mentioned in section 1.1. The main parts of the system are shown in Figure 2-1.

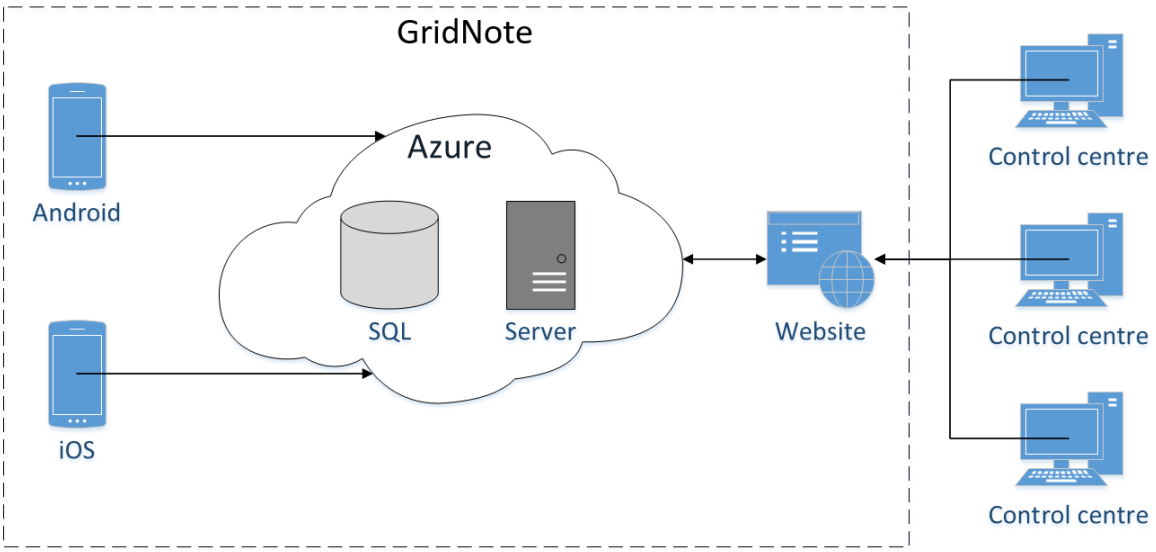


Figure 2-1 GridNote Overview

2.1 Planned Solution

Skagerak Energi has provided guidelines on how they expect the system should work. In addition to the given scope, other requirements were made to improve the outcome of the project such as it should be scalable, maintainable and have backward compatibility for the mobile application. A list of requirements for both applications is attached as Appendix E.

2.1.1 Web Application

A web application is developed to display jobs submitted by the technicians, in an informative and graphical manner. The approach for this is to use a map to indicate which electrical substations are active. It will also focus on being responsive, adjusting the objects on the website proportional to operator's screen. Further details on planning and concept design of the web application will be given in chapter 3.

2.1.2 Mobile Application

A mobile application is developed for technicians, where they can submit a task that will notify the operating centre about which electrical substation they are working on. This system should have a simple interface with limited options to ensure a straightforward interaction for the users of mobile application. It should eventually support multiple OS platforms because users have a combination of Android and iOS devices. Further details on planning and concept design of the mobile application will be given in chapter 4.

3 Website – Planning and Design

This chapter describes the planning and design process for the web application. It contains the diagrams created from the requirements together with User Experience (UX) and User Interface (UI) specifications.

3.1 Requirements

The requirements for the web application were divided into non-functional and functional requirements. The complete list of requirements will not be detailed in this section but is attached as Appendix E.

3.1.1 Diagrams

This subsection covers the different diagrams that were developed during the planning phase of the project. From the requirements found in Appendix E, a use case diagram was created. The use case classify as Unified Modelling Language (UML) diagrams. UML diagrams serve as a standard for data related modelling.

The use case diagram was created to display the relationship between user actions and the response from background functions. Actions required from user are: Login and change display information. The Use Case Diagram is shown in Figure 3-1.

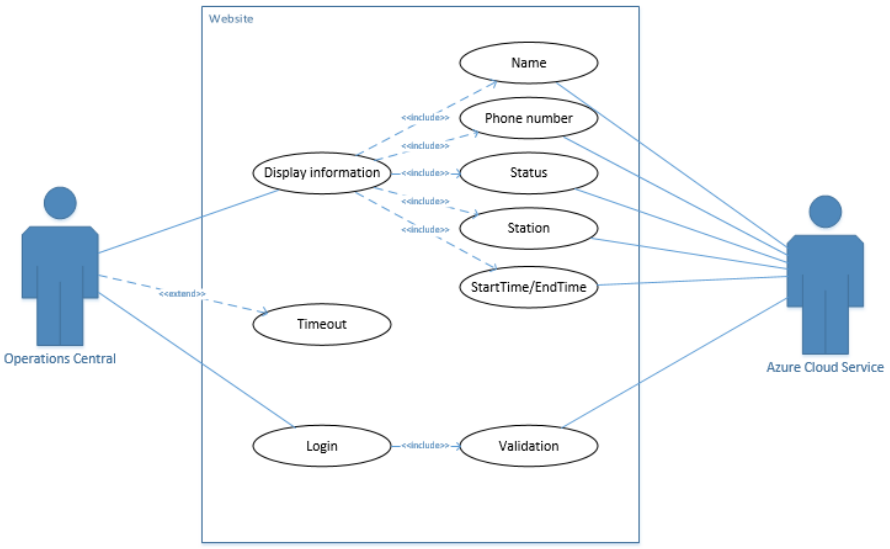


Figure 3-1 Use case diagram for web application

3.2 User Experience

The UX is the underlying functionality that makes the behaviour of the application more fluent for users. A list is derived from the requirements for the web application in Appendix E. The following list elaborates on how the UX should be for different functionalities.

- Sign in:
  The application should implement a sign in method, where the user only needs to provide email address and password. The user should stay logged in on the website, until the user explicitly logs out.
- Table:
  The table showing all the active workers should come with capabilities to sort for different categories. It should cover a large part of the screen so that it is more easily read. It should also be possible to search for names or stations, displaying only rows with those values. The status of the job should be indicated with an icon. One should be able to scroll the table if the number of active jobs is larger than the fixed table size.
- Map:
  The map should be zoomable and its view adjustable. If the view of the map comes out of bounds it can be brought back to the correct position. It should be able to open in full screen for better view. The map should indicate clearly where the different active jobs are taking place.
- Display information:
  Historic data should be accessible. The logged in user's name should be displayed. A trendline should be displayed with the number of jobs that have been completed since a given time.

3.3 User Interface

A UI was designed for the website during the design phase. A set of concept drawings were made from an interpretation of the UML Diagram, and the requirements in Appendix E. The concept drawings for the website contain three pages. These pages are the Login page, Status page and History page. This section covers the different elements on these pages.

3.3.1 Login Page

The login page has only one important element and that is the login field. The user types in email and password to login to the application. The concept drawing of the login page is shown in Figure 3-2.

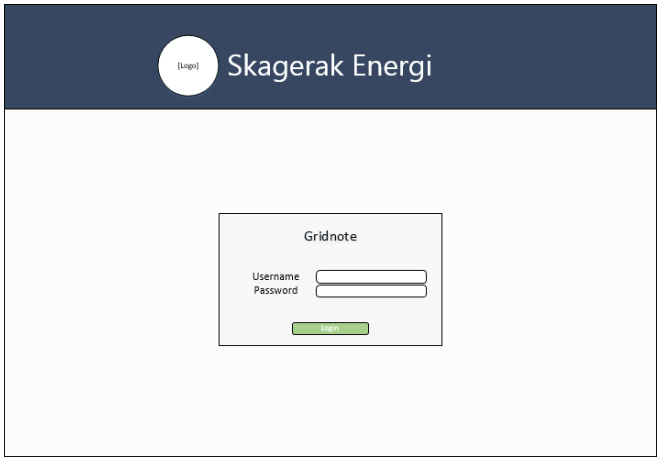


Figure 3-2 Concept drawing of login page

3.3.2 Status Page

After logging in with correct authentication and authorization the user is greeted to the status page. The status page contains most of the elements of the website. Elements such as a trendline, map, table, buttons and labels. By clicking the preference button, a popup menu appears that gives the user a selection of preferences to choose from. The trendline shows how many active jobs there have been in the last seven days. The map displays indicators of each location where a substation is being worked on. The table contains all the active jobs and useful information about the job such as start time, worker info and status. The concept drawing of the status page is shown in Figure 3-3.

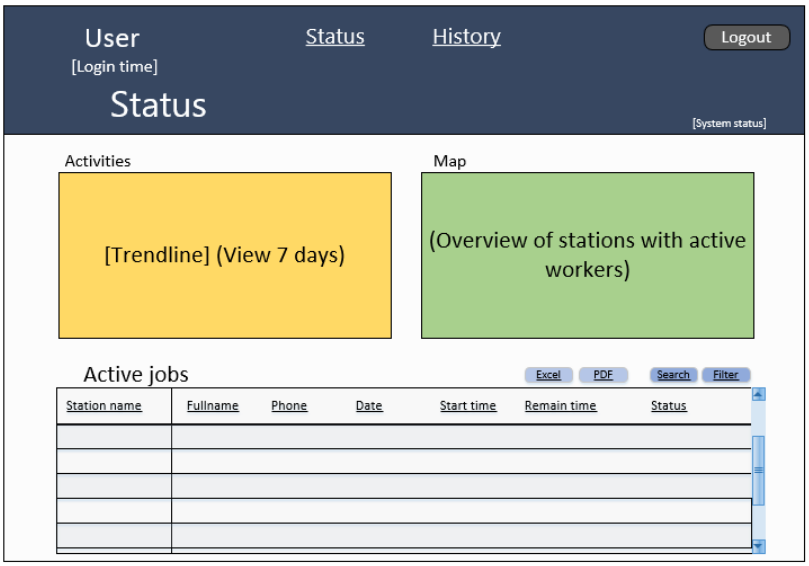


Figure 3-3 Concept drawing of status page

3.3.3 History page

By clicking on the history button on the status page, the user is directed to the history page. The elements on this page are identical to the status page except for the map with active jobs, which is absent. The table contains the full history of completed work. The concept drawing for the history page is shown in Figure 3-4.

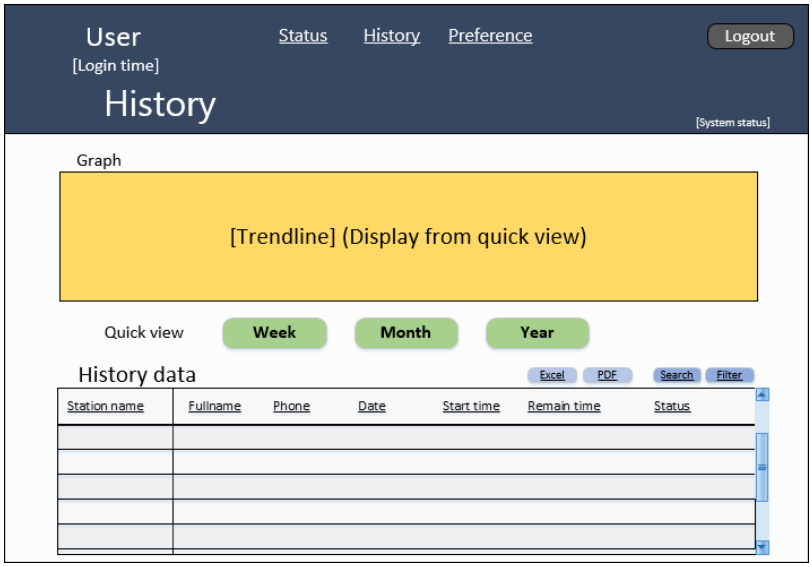


Figure 3-4 Concept drawing of history page

4 Mobile Application – Planning and Design

This chapter describes the planning and development processes for the mobile application. It describes which choices were made and how they affect the mobile application.

4.1 Use Case Diagram

This section describes how the application was designed based on the requirements. This includes concept drawings. The requirements for this application can be found chapter 1.1 with further details in Appendix E. From these requirements, a use case diagram was created to display the relations of the user's actions, and responses from the background functions of the system. The use case, shown in Figure 4-1, serves as a basis document for creating User Interface (UI) and User Experience (UX). To elicit a response from Azure Cloud Services, the user must make one of the following actions: call-in, set work time, choose electrical substation, Preferences and Login.

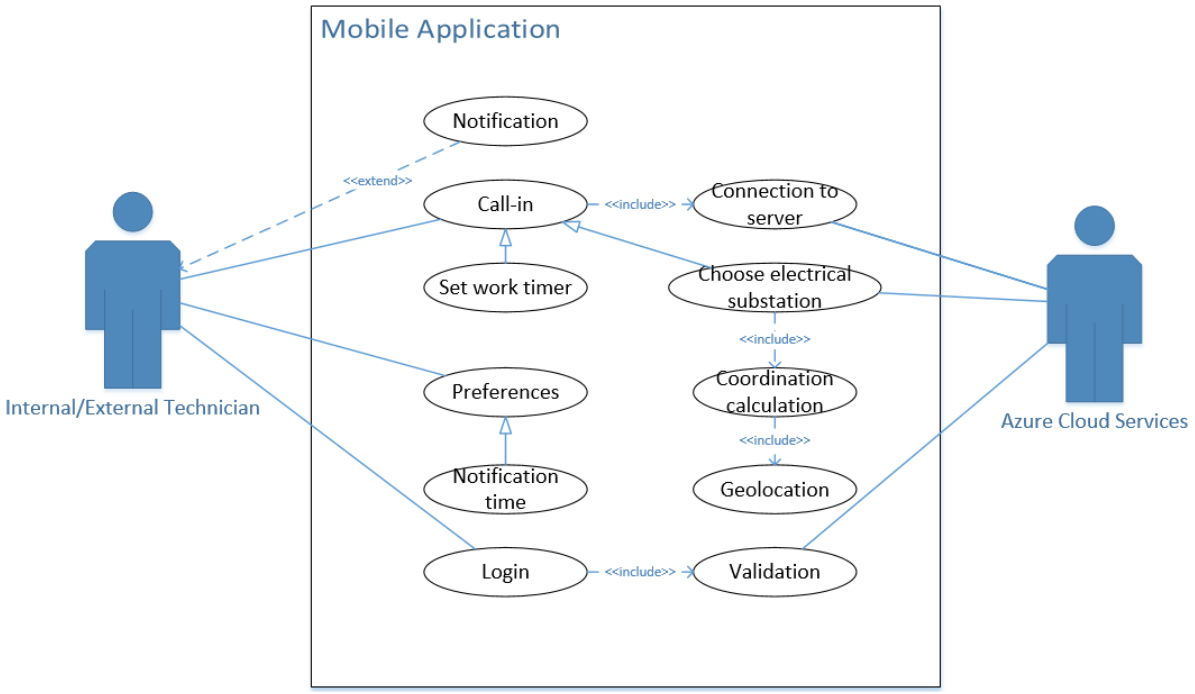


Figure 4-1 Use case diagram for mobile

4.2 User Experience

Focus on user experience has been important when developing the underlying functionalities to make the application behave fluent. The following list is, in addition to the requirements, functions the application should implement:

- Auto sign in:
  The application should implement a sign in method, where the user only need to provide email address and password. When the user has logged in once, it should not be necessary to log in the next time the application is opened.
- Map update:
  The map should focus on the user's position when it is opened and automatically update its substation information as the position changes.
- Check for work:
  The user should not be able to register more than one job at any time. The application should check for existing jobs in Azure and redirect the user to the status page. This is also intended for the user to get the same job, when signing into the application with a different device.
- Full text search:
  A few electrical substations could be outside of cell tower range. The application should include a search option for the user to find specific substations.
- Current time:
  The application should automatically get the users current time when submitting a job. The user should only have the possibility to select an end time.
- Display remaining work time:
  The application should calculate and display the remaining work time.
- Notifications:
  Notifications are used to inform the user when the registered end time is about to expire. The user is then allowed to either extend the work time or select job completed.

4.3 Application Design

A set of concept drawings was made to illustrate the functionalities of the mobile application based on the use case diagram in Figure 4-1. A recommendation from project partner was to have a guided application interface, where the user has limited actions and must follow a certain routine. The interface of the mobile application must, therefore, be simple and understandable. The following figures show the intended UI design during the development process. All the concept designs for this project is attached as Appendix F.

4.3.1 Login

The concept design of the login page is shown in Figure 4-2. This is what the application will be developed from. In this page, the functionality is to validate the user against the server, and also include the auto sign-in as mentioned in section 4.2.

Figure 4-2 Design of the login page

4.3.2 Select Station

After having logged in, the user should be able to select a substation, either by picking one from the map or by searching for one with substation name or ID, as shown in Figure 4-3. There should not be possible to select and register work on a station that is too far away from the user.

The map should immediately display substations in near proximity to the user. This will require the use of the mobile device's GPS functionality to find the user's position, and the appropriate substations. The user should then be able to press the desired station on the map to select it.

When searching, all available substations matching the search word should appear in a dropdown menu. Clicking a station from this menu should then make it the selected choice.

Figure 4-3 Design of the select station page

4.3.3 Select Time

The select time page handles the amount of worktime the user chooses. Figure 4-4 shows the concept for that page. As seen, this page allows the user to decide the worktime with a time wheel, a feature the project partner requested. It should calculate the time differences between start and end time and present it in the next page. The remaining time function is mentioned in section 4.2.

Figure 4-4 Design of the select time page

4.3.4 Status

The status page, shown in Figure 4-5, displays the user's selected worktime as well as the status of the job. It can also display other information such as substation Id and substation address. This page will have buttons, one for completing the job and the one to edit the worktime.

If the user has completed the job on a substation, but forgets to register it as completed, the application should give an alert asking if the job is finished or if it should be extended. This should happen when the allotted time has expired or if the user has left the substation site.

Figure 4-5 Design of the status page

5 Xamarin

This chapter explains how Xamarin uses one programming language to create a mobile application that can target multiple platforms, which usually require their own specific languages.

5.1 Native

A native application is per definition: an application that only targets one specific operating system (OS). Which means applications must be built in their own programming language, such as Swift for iOS and Java for Android. A setback is that their code cannot be shared because of the language barrier. This causes developers to recreate identical applications, in different languages.

The application should have the possibility of supporting multiple platforms. An approach to this is to have a shared code base, where different native applications can use the same defined functionality without them being recreated for each platform.

Xamarin is an open source extension tool for Visual Studio. It allows developers to build cross-platform applications with its implementation of Common Language Infrastructure (CLI), which is explained in section 5.2. Xamarin is a part of the Microsoft.NET Framework and uses the programming language C#. This strong-type language covers almost all the functionalities a common language needs for developing a mobile application. These are: Swift, Java and Objective-C. [1][2]

Figure 5-1 Architecture of Xamarin [3]

The shared code base is showed at the bottom of Figure 5-1. Creating the logic once, with only one language, is an advantage when developing cross-platform applications.

An alternative option is to develop the mobile application as a web application that run on the device's native browser. This way the user is not required to download any application. Subsequently the user must then configure the browser's permissions for using local hardware, and the developer must build the application to support backward compatibility for browsers and OS versions.

5.2 Common Language Infrastructure

This part is outside of the project's scope; therefore, it is given as a general description on how the CLI operates, and how it permits one language to run on multiple OS by converting to machine language. [4][5].

CLI is a part of Microsoft.NET's strategy for supporting cross-platform coding. This permits the same programming language to run on different OS devices. CLI compiles the source code to Common Intermediate Language (CIL), which is a bytecode language. CIL has been accepted as an open standard by European Computer Manufactures Association (ECMA), an international organization for the promotion of technology standards. [4][5][6]

For the code to be understood by specific systems, CIL will further be compiled into Machine code. It is also known as machine language and operates in series of binary code. This is illustrated in Figure 5-2. The binary code is then stored inside a metadata. "*Metadata is stored with the code; every loadable common language runtime portable executable (PE) file contains metadata. The runtime uses metadata to locate and load classes, lay out instances in memory, resolve method invocations, generate native code, enforce security, and set run-time context boundaries."* [7]. [8]

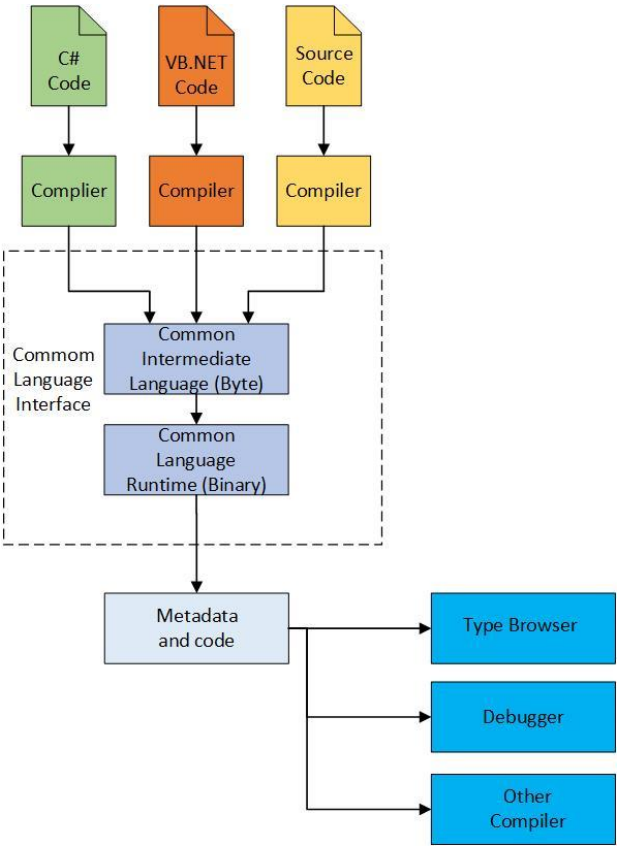


Figure 5-2 Common language interface

5.3 Cross-Platform

Sharing code logic, as shown in Figure 5-1, permits developers to spend more time with one programming language and comfortably scale their code to another platform.

Xamarin offers two code sharing architectures: Shared Project and Portable Class Libraries (PCL) also known as .NET. Both architectures have the same principle for sharing code, but their differences are how they handle the code behind. [9][10]

5.3.1 Shared Project

Figure 5-3 illustrates the Shared Project architecture. It permits developers to create platform specific code and compile after the target device. Consequently, when using this as the project architecture, it becomes more complex as the development process progresses and it becomes harder to refactor, since most of the code are platform explicit. This architecture creates a copy of the shared project code when compiling, and this would mean that each platform application could include others platform code specification as seen in Figure 5-3. [10][11]

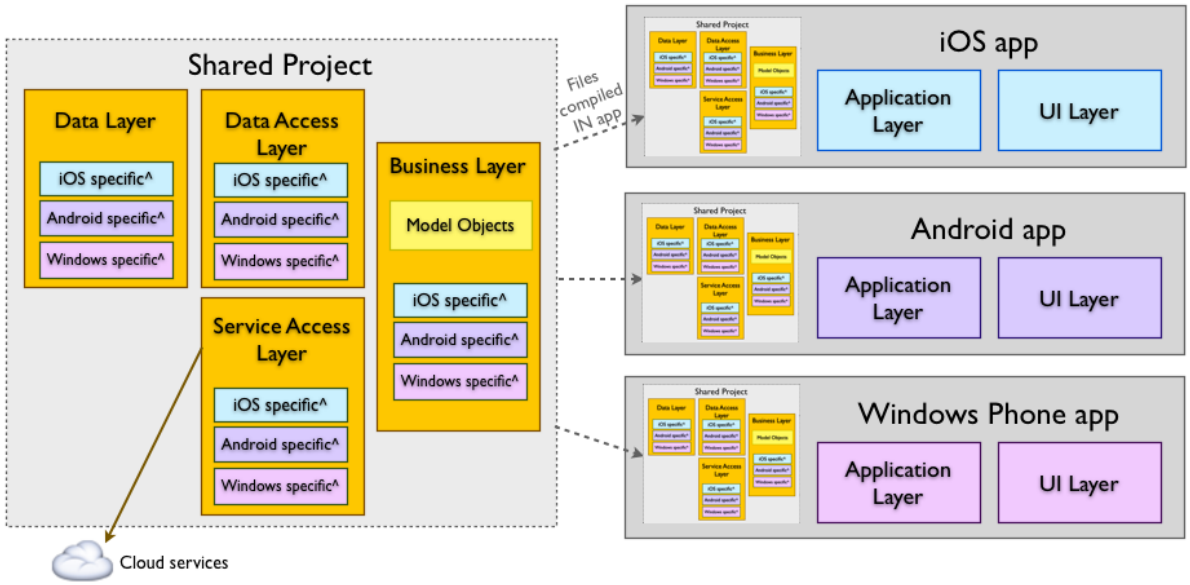


Figure 5-3 Shared code architecture [12]

5.3.2 Portable Class Libraries

The PCL lets developers create platform specific code, but only through each platform's specific projects. Almost every code inside of the PCL project is shared to platform specific projects, this means maintaining or refactoring code is uncomplicated. Instead of creating a copy of the shared code to each platform, PCL creates a reference as shown in Figure 5-4. This allows the solution file for the application to have a centralized code sharing, where applying refactoring operations on the code will affects all platform solution. [10][11]

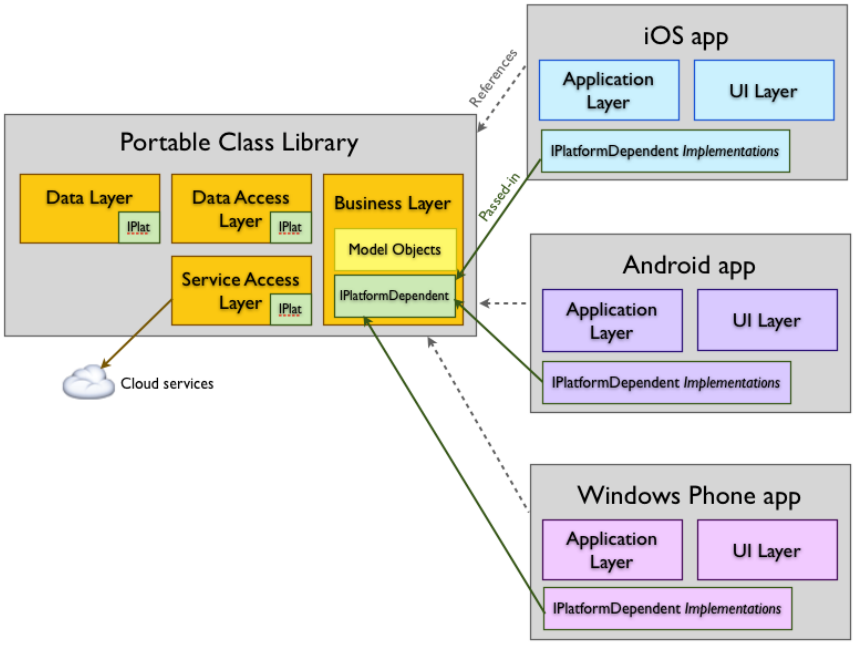


Figure 5-4 Portable code architecture [11]

Part II. Solution and Future Work

This part explains the present solution of the GridNote system, as well as code examples of functions. It also discusses the solution and how it can be improved with further development.

6 Microsoft Azure

The GridNote system uses Azure cloud services to accomplish different tasks. The resources that complete these tasks are managed using the Azure Portal platform. Azure Portal is used to build, deploy and administer the different applications and services. This chapter introduces Microsoft Azure and explains the different Azure cloud services that the GridNote system used. [13]

Some of the images used in this chapter were altered from Microsoft websites. These were used with permission from Microsoft. [14]

6.1 Introduction

Azure provides a set of cloud services on one single platform. Azure has several services, some of which are displayed in Figure 6-1. The figure only shows Platform Services and Security & Management services, which are the only categories of services that are relevant to the current GridNote system. The GridNote system primarily uses the services that are framed.

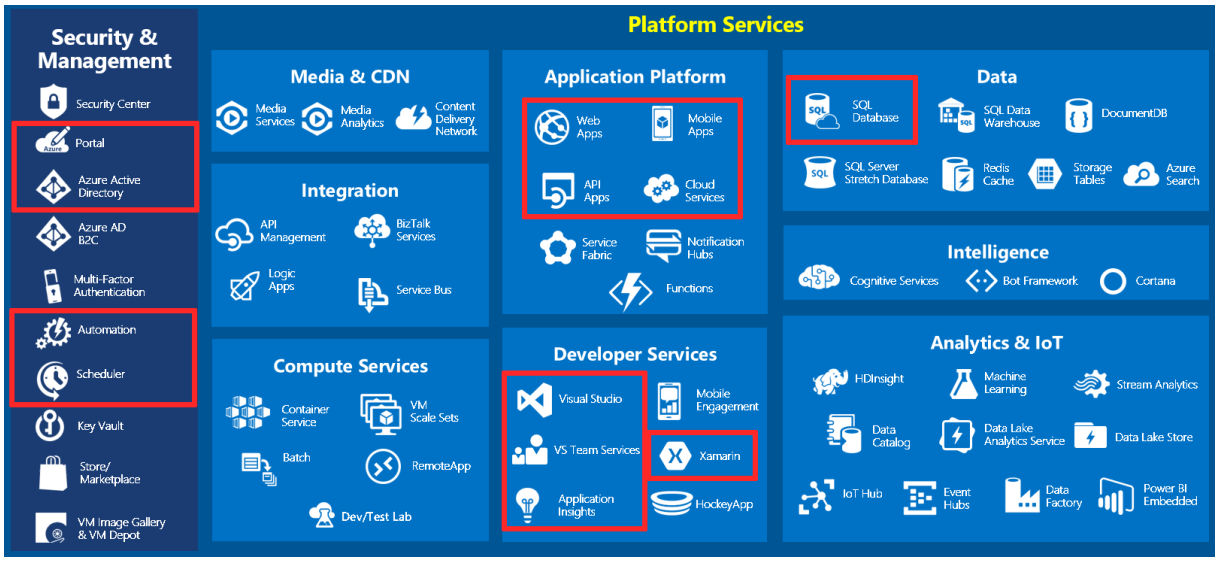


Figure 6-1 Part of Azure cloud services [15]

Another word for these services often used in Azure is resource. Different resources can be grouped in what is fittingly called a resource group. The system's resource group is called GridNote. It contains all the Azure resources used to support both the website and the mobile application. The main resources that will be explained in this section are the following:

- SQL Server and database
- Mobile application
- Web application
- Job scheduler
- Active Directory

6.2 SQL Server and Database

The SQL Server and Database hosted on Azure is stored on Microsoft Azure's servers. To connect to the server, it is required to have an internet connection as the databases are stored in the cloud. This means that any changes that are made without an internet connection will not go through and update the database.

When creating the server, a Server Admin with password must be appointed. The admin is the only user that can initially access the database in SQL Management Studio. Only the admin may add new users unless a user with sufficient privileges has already been added, like a user with the Login Manager role. The firewall on the SQL Server is configured in Azure Portal to only allow specific IPs to access it with a TCP connection. This is to restrict any unwelcome users from gaining access to the database. [16]

6.3 Web and Mobile Application Service

Two Web App Services were used to manage the mobile application and web application on Azure. The names given for the web app service is called GridNote and the Mobile App Service is called GridNoteApp.

Both app services have Authentication / Authorization configured to use the Azure Active Directory (AAD) from Skagerak Energi as authentication provider. The Azure Active Directory (AAD) is explained in further detail in section 6.5 later in the report. The app service configurations integrate the applications into the AAD. The configuration consists of a Client ID, Issuer Url and Client Secret as shown in Figure 6-2. These values were given by the project partner and were generated by registering the applications on their AAD. Figure 6-2 shows censored values as they can be used to obtain sensitive information. [17]

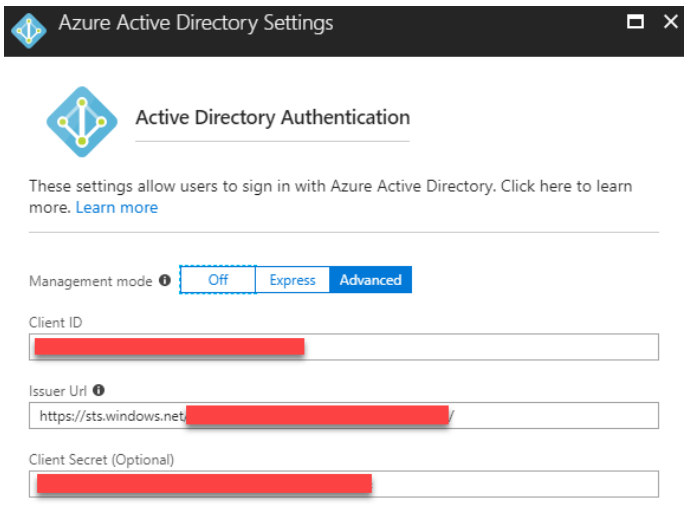


Figure 6-2 Authentication / Authorization settings

The Mobile application used Easy Tables as a means of communicating with the SQL Server Database. A connection to the existing Database Server was added and the tables were added individually to Easy Tables. This synced the SQL database with the Easy Tables database and when the mobile application makes changes to the Easy Tables it is automatically updating the

Azure SQL Server database. Other than adding the connection and tables there was no additional configuration needed on Azure portal. Figure 6-3 shows the current configuration of Easy Tables.

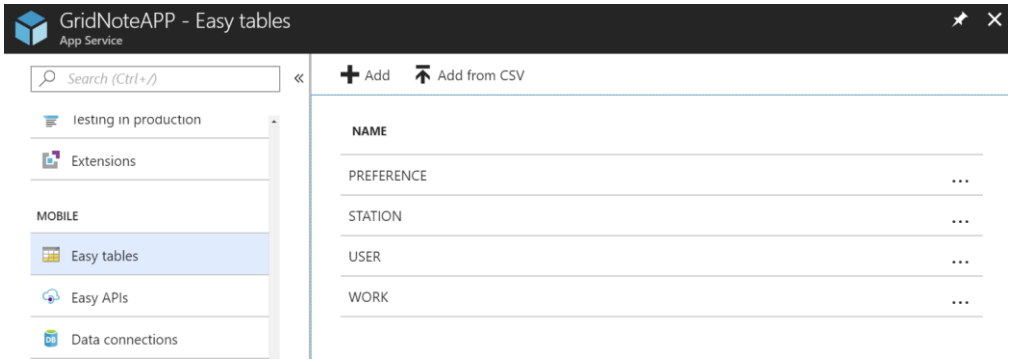


Figure 6-3 Mobile App Service Easy tables

6.4 Job Automation

When the time for a work specified by a technician runs out, the status of the work stored in the SQL Database needs to change its value. To make this repetitive task automated, several resources in Azure were used. This chapter explains how these resources were used to make a Stored procedure run in the SQL Database. Figure 6-4 shows the general concept of how this job was automated.

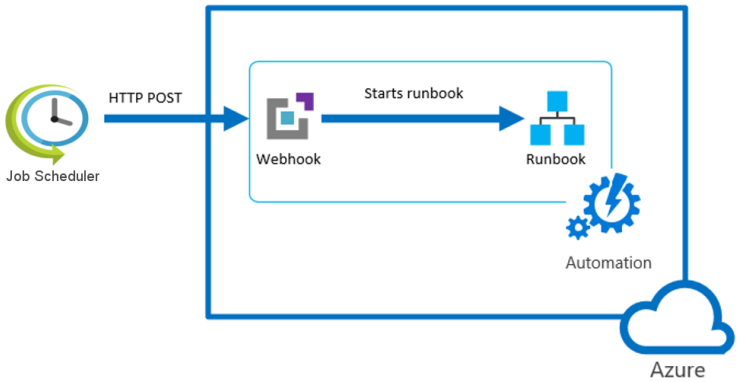


Figure 6-4 Concept of automated task [18][19]

6.4.1 Automation Account

The GridNote system uses an Automation Account to store and configure the runbook with the webhook that is used to complete the task. Runbooks and webhooks are explained in the next chapter. The Automation Account used for the GridNote system is called *GridNoteUpdateStatusAccount*.

6.4.2 Runbook

A runbook is the term used for an automated procedure that is run frequently. The runbook called *UpdateStatus* was added to *GridNoteUpdateStatusAccount* and configured to run a PowerShell Script. This PowerShell script that is shown in Figure 6-5 runs the *StatusUpdater* stored procedure. Stored procedures are explained later in section 7.3. [20][21]

```
1 $SqlConnection = New-Object System.Data.SqlClient.SqlConnection
2 $SqlConnection.ConnectionString = "Server=tcp:gridnote.database.windo
3 $SqlConnection.Open()
4 $SqlCmd = New-Object System.Data.SqlClient.SqlCommand
5 $SqlCmd.CommandText = "StatusUpdater"
6 $SqlCmd.Connection = $SqlConnection
7 $SqlCmd.ExecuteNonQuery()
8 $SqlConnection.Close()
```

Figure 6-5 PowerShell script

6.4.3 Webhook

To trigger the runbook, a webhook called *StatusUpdate* shown was created. A webhook is essentially a URL that triggers the runbook when someone sends a HTTP POST to it. The webhook was then added to the *UpdateStatus* runbook.

6.4.4 Job Scheduler

To run the runbook according to a specific schedule or interval, one must create a Job Schedule. But first one needs to create a *Scheduler Job Collection* which is a group of Job Schedulers. A Job Schedule called *getStations* was created and configured to repeatedly run every minute.

To trigger the *UpdateStatus* runbook, the URL for the *StatusUpdate* webhook associated with the *UpdateStatus* runbook was added to the action settings as shown in Figure 6-6.

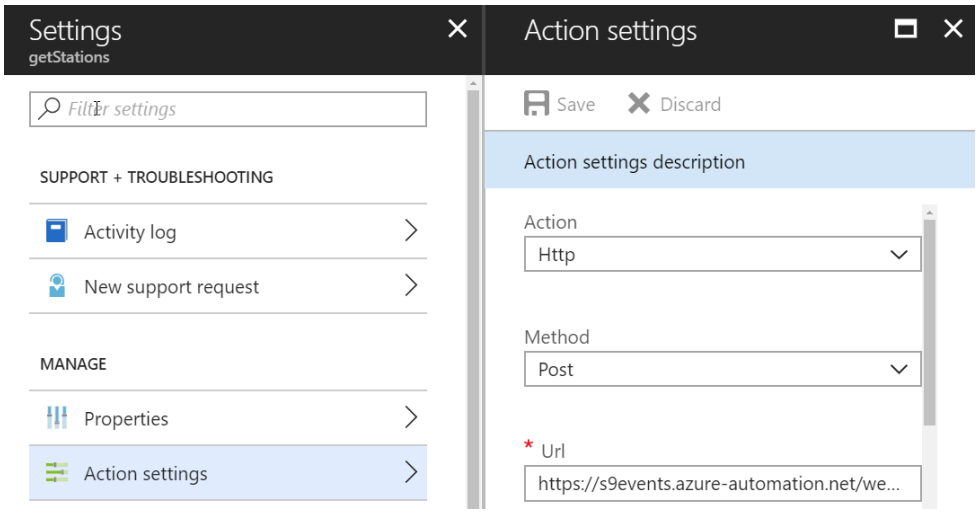


Figure 6-6 Settings for job scheduler

6.5 Active Directory

Azure Active Directory (AAD) works like a regular Active Directory (AD) but is managed through Azure Portal with an internet connection. Active Directory was made by Microsoft to manage users, computers and devices connected in a network. The purpose is to gain greater control over groupings by dividing them into groups or subgroups. These groups having various access privileges within the network. [22][23][24]

An AD containing personnel working at Skagerak Energi was added to the AAD in the GridNote resource group. For the values used in the authentication and authorization process described in section 6.3, both the Web and Mobile Application had to be registered within the AAD. To gain access to members of the AAD's user info, permissions from the two registered applications had to be configured. This configuration was done by the project partner.

7 Database

This chapter describes the database structure of the system, the stored procedures that are used and where the database is hosted.

7.1 Azure Connection

The database is hosted on Microsoft Azure Cloud Services. Using Azure will make it possible to save data from the mobile application to the database via internet. The Azure helps to get a secure database with a user and password login, as mentioned in section 6.2. To use database hosting in Azure, an SQL server resource is needed. This is also where the login possibilities are set. SQL server management studio is used to access the database created on the server. Since the database and the mobile application cannot communicate directly with each other, the Easy table API is used as an intermediary between the two.

7.2 Structure

In this section the structure of the database is described. Figure 7-1 shows an overview of the database structure. The database consists of three tables; *USER*, *WORK* and *STATION*. The first table is designed to save the necessary information about every user of the system. The second table is made for storing every submitted work, and the third one is for saving information about the substations. The second table has two foreign keys, which are the primary keys from the first and the third table. Both have a one-to-many relationship with the second table. When the database tables were inserted into Azure, Easy tables automatically generated more columns. The columns generated are: *id*, *createdAt*, *updatedAt*, *version* and *deleted*. These are default columns the mobile application's Easy table use to synchronize with the database. In Figure 7-1, the overview of the system's database structure is shown without these columns, except for the id columns which are used as primary keys for all three tables. These keys are needed when a user is creating a work. [25]

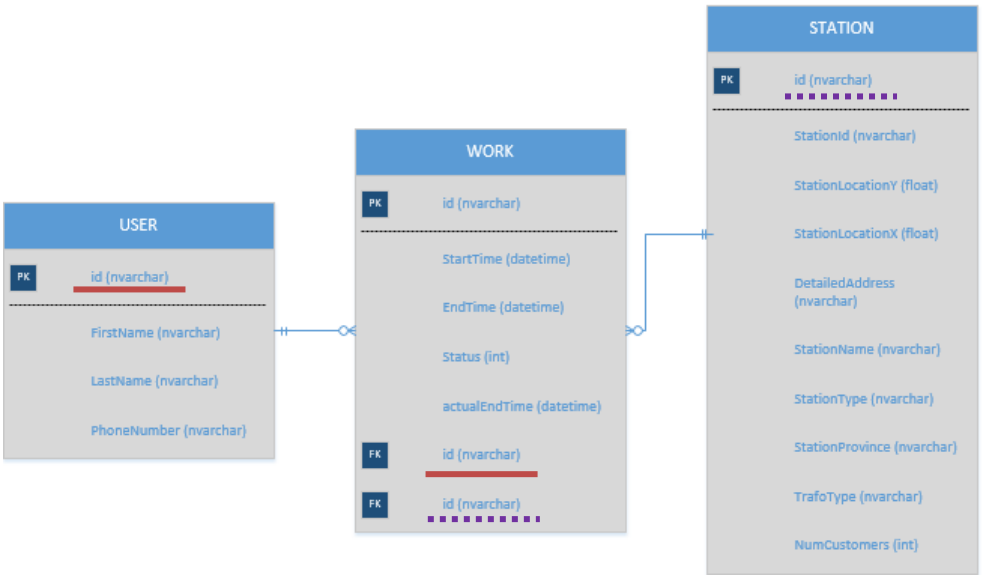


Figure 7-1 Database tables

7.3 Stored Procedures

"*A Stored procedure is a group of one or more database statements stored in the database's data dictionary and called from either a remote program, another stored procedure, or the command line."* [26]. The web application uses stored procedures to communicate with the database. The stored procedures update or retrieve information from the database to the website. Using stored procedures to communicate with the database, add a layer of security to the client side of the application. It also contributes to prevent unwanted access to the database tables by running stored procedures written by the developer, instead of giving the user full authority to insert, update and delete tables directly. An example of one of the stored procedures that is used in the web application is *TableRefreshHistory*, shown in Figure 7-2. This procedure makes it possible for the web application to filter a search from a formatted table in the database. As an example, if the user wants to search for a specific date, the stored procedure can extract all information from the database columns from this date. [27]

```
SELECT
[dbo].[STATION].DetailedAddress,
[dbo].[STATION].stationId,
[dbo].[STATION].stationProvince,
[dbo].[STATION].stationType,
[dbo].[USER].firstName + ' ' + [dbo].[USER].lastName,
[dbo].[USER].phoneNumber,
FORMAT([dbo].[WORK].startTime,'HH:mm') + ' - '  + FORMAT([dbo].[WORK].actualEndTime,'HH:mm') + '.  ' + FORMAT([dbo].[WORK].startTime,'dd/MM')
FROM
[dbo].[USER]
INNER JOIN [dbo].[WORK]
ON [dbo].[USER].Id=[dbo].[WORK].userId
INNER JOIN [dbo].[STATION]
ON [dbo].[WORK].stationId=[dbo].[STATION].[Id]
WHERE
(([DetailedAddress] LIKE '%' + @Filter + '%') OR ([STATION].[stationId] LIKE '%' + @Filter + '%')  OR
([stationProvince] LIKE '%' + @Filter + '%') OR ([stationType] LIKE '%' + @Filter + '%')
OR ([firstName] LIKE '%' + @Filter + '%') OR ([lastName] LIKE '%' + @Filter + '%') OR ([phoneNumber] LIKE '%' + @Filter + '%')
OR ([endTime] LIKE '%' + @Filter + '%')  OR ([startTime] LIKE '%' + @Filter + '%') OR
 FORMAT([dbo].[WORK].startTime,'dd/MM') LIKE '%' + @Filter + '%') AND [WORK].[Status] = 2
ORDER BY [dbo].[WORK].[endTime] ASC
```

Figure 7-2 The stored procedure *TableRefreshHistory*

8 Website – Solution

This chapter contains a description of how the present version of the web application works and the different choices made when developing the application. It also explains how certain functions operated and how they were implemented. Appendix G explains some of the terminologies used in this chapter. Appendix H contains the user manual for the website.

8.1 Framework and Software

The web application was developed in Microsoft Visual Studio (VS), a development tool for creating applications. All code was tested using Google Chrome. The framework used to develop the web application is ASP.NET Web Forms. In the VS environment, ASP.NET Web Forms is one of four programming models that are used in development of web applications. The others being MVC, Web Pages and Single Page Applications. Web Forms is a part of the ASP.NET framework that is included in Visual Studio.

Web Forms allows users to use drag-and-drop controls with event-driven properties. The webpage is first designed by adding controls such as textboxes, buttons, timers etc. Each of these controls have their own properties that change its behaviour and appearance. Each control also has events that can be used to trigger code. [28]

8.1.1 Bootstrap

Bootstrap is an open-source toolkit for HTML, CSS and JavaScript. Bootstrap is a front-end toolkit that is used by developers to build UI components like buttons, menus and grids. Bootstrap has a layout feature that allows for easy grouping of elements. This feature is the reason that it was added to the GridNote website. Components chosen and referenced in the web application are Bootstrap, Bootstrap Grid and Bootstrap Reboot. The version used in all Bootstrap components is version 4. [29]

8.2 Classes

The objective of the website is to display a table of active workers that is only accessible to staff in the operations central. As explained, Web Forms uses controls and events to trigger code. Classes are used to store back-end code that is triggered by the control events. [30]

This section describes the classes used in the GridNote web application. There are five classes called *Status*, *History*, *Map*, *Work* and *Database*. The *Database* class contains a global enumeration. Which is essentially a list of names of all the stored procedures in the database. The *Database* class will not be explained further in this section.

8.2.1 Status Class

The *Status* class is the main class of the whole application and is the Code-Behind for the status page. The *Status* class has seven methods called *Page_Load*, *GetMarkers*, *GetImage*, *timer1_Tick*, *TxtSearch_TextChanged*, *UTM2Deg* and *BtnMoreInfo_Click*. This subsection details the *Page_Load*, *GetMarkers* and *GetImage* methods.

The *Page_Load* method runs during loading of the status page. There are two functionalities that are addressed in the page load. The first is adding markers to the map and the second is getting the number of active work to display in a label. The first functionality of *Page_Load*, which is creating markers, will be shown in this subsection. Figure 8-1 shows how a marker on the map currently looks like on the website.

Figure 8-1 Map Marker

Creating markers to be shown on the map is done in three steps. The first step involves getting the station information from the database and storing it in lists. The information consists of the coordinates and information like address, id and region of the station. This is all retrieved from the database hosted on Azure mentioned in section 6.2. To store the information, a list of marker class objects called *myMarkers*, a map class object called *mapPoints*, a string from the search textbox called *searchText* and several lists for station information are initialized. The search text is sent into the method called *GetPoints* and the station information is sent back out, filling the lists initialized earlier. The list of marker class objects called *myMarkers* is used later in the third step. The *GetPoints* method is explained later in subsection 8.2.3. Figure 8-2 shows the first step of how the markers are created in the Page_Load method.

```
List<Marker> myMarkers = new List<Marker>();
Map mapPoints = new Map();
string searchText = txtSearch.Text;
List<double> stationLatList = new List<double>();
List<double> stationLngList = new List<double>();
List<string> stationAddressList = new List<string>();
List<string> stationRegionList = new List<string>();
List<string> stationIdList = new List<string>();
List<string> stationStatusList = new List<string>();
mapPoints.GetPoints(searchText, out stationLatList, out stationLngList,
    out stationAddressList, out stationRegionList,
    out stationIdList, out stationStatusList);
```

Figure 8-2 First step, initializing and populating variables

The second step involves validating the coordinates. A for-loop runs through both lists of co-ordinates to check if they are real numbers. If they are not, a bool variable named *isOk* is set to false. *isOk* is then used in the third step. Figure 8-3 shows the second step.

```
bool isOk = true;
double filler = 0;
for (int i = 0; i < stationLatList.Count; i++)
{
    if (double.TryParse(stationLatList[i].ToString(), out filler) == false)
    {
        isOk = false;
    }

    if (double.TryParse(stationLngList[i].ToString(), out filler) == false)
    {
        isOk = false;
    }
}
```

Figure 8-3 Second step, validation of coordinates

The third step involves converting the coordinates and adding the information into one big list before storing everything in a session state. Sessions are used to store and retrieve information that is only used for the period the user is on the website. The coordinates are converted because the map does not support their initial format. This conversion is detailed later in subsection 9.2.4. An if statement makes sure the validation from the second step was successful. The for-loop converts the coordinates and creates marker objects with the coordinates and station information. These marker objects were then added to *myMarkers* mentioned in the first step. Figure 8-4 shows the code for the third step. [31]

```
if (isOk == true)
{
    for (int i = 0; i < stationLatList.Count; i++)
    {
        double lati;
        double longi;
        UTM2Deg(stationLatList[i], stationLngList[i], out lati, out longi);
        myMarkers.Add(new Marker(lati, longi, stationAddressList[i],
            stationRegionList[i], stationIdList[i], stationStatusList[i]));
    }
    Session["markerData"] = myMarkers;
}
```

Figure 8-4 Third step, conversion and storing

The *GetMarkers* method is a static WebMethod that is called from the mark-up code (HTML) using the JavaScript library jQuery together with Asynchronous JavaScript And XML (Ajax). Ajax is used to get the data before the page loads by sending a *POST* request to the status page. A list of marker objects called *markerData* is initialized for the session that was created in the *Page_load* method described in subsection 8.2.1. This list of marker data is then returned to be used in the JavaScript code. Figure 8-5 shows the code for the *GetMarkers* method.

```
[WebMethod]
public static List<Marker> GetMarkers()
{
    List<Marker> markerData = (List<Marker>)HttpContext.Current.Session["markerData"];
    return markerData;
}
```

Figure 8-5 *GetMarkers* WebMethod

The *GetImage* method is called from the status page's mark-up code on load-up using JavaScript. It is used to display icons that indicate the status of the work. In the database, integers were used to indicate the work status. 1 means the work is within the time limits, 2 means the work has expired and 3 means the work has completed. The method receives the integer indication as a parameter. If the value is 1 it returns the "ok.png" image. If it is not 1 it returns the "notok.png" image. The icon is displayed inside the table, which will be detailed in subsection 8.3.2.

8.2.2 History Class

The *History* class contains three methods, *Page_load*, *Timer1_Tick* and *TxtSearch_TextChanged*. The *Page_Load* method uses the same code to get the number of work as the page load event described in subsection 8.3.2. The *Timer1_Tick* and *TxtSearch_TextChanged* methods are each triggered by different events (timer tick and text change) but have identical content. The only purpose of these two methods is to update the table.

8.2.3 Map Class

The *Map* class inherits from the interface *IMap*. An interface is used to control the content of a class by making sure the class has all the members as the interface. Meaning all methods, parameters and modifiers contained in the interface must be present in the class. There is only one method used in the *Map* class and it is called *GetPoints*. [32]

The *GetPoints* method is used by the *Status* class as mentioned in subsection 8.2.1. This method gets the coordinates and station information from the database. An SQL connection called *con* is initialized for the connection string that is found in the *Web.config* file. A connection string is a string that describes a data source and how to connect to it. Using the connection string, the search text is added as a parameter in the stored procedure called *getStationLocations*. The connection was opened, and a reader populates the initialized lists. This method uses a reader object to read the table generated in SQL row by row. The *GetPoints* method is shown in Figure 8-6.

```
SqlConnection con = new SqlConnection(WebConfigurationManager.ConnectionStrings["GridNote"].ConnectionString);

public void GetPoints(string searchText, out List<double> stationLatList, out List<double> stationLngList,
    out List<string> stationAddressList, out List<string> stationRegionList, out List<string> stationIdList,
    out List<string> stationStatusList)
{
    stationLatList = new List<double>();
    stationLngList = new List<double>();
    stationAddressList = new List<string>();
    stationRegionList = new List<string>();
    stationIdList = new List<string>();
    stationStatusList = new List<string>();

    using (SqlCommand cmd = new SqlCommand(StoredProcedure.getStationLocations.ToString(), con))
    {
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Parameters.AddWithValue("@Search", searchText);
        con.Open();
        var reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            stationLatList.Add(reader.GetDouble(0));
            stationLngList.Add(reader.GetDouble(1));
            stationAddressList.Add(reader.GetString(2));
            stationRegionList.Add(reader.GetString(3));
            stationIdList.Add(reader.GetString(4));
            stationStatusList.Add(reader.GetInt32(5).ToString());
        }
        con.Close();
    }
}
```

Figure 8-6 *GetPoints* method

8.2.4 Work Class

The *Work* class inherits from an interface called *IWork*. The first method was called *GetWorkQuantity* and the second *GetWorkQuantityHistory*. Both methods were used to fill the table with data, but the first was called from the *Status* page and the second was called from the *History* page. The methods were used to call different stored procedures. These methods use output parameters in stored procedures to get specific data. Figure 8-7 shows *GetWorkQuantity* method.

```
SqlConnection con = new SqlConnection(WebConfigurationManager.ConnectionStrings["GridNote"].ConnectionString);

public string GetWorkQuantity()
{
    string workQuantity;
    using (SqlCommand cmd = new SqlCommand(StoredProcedure.countWork.ToString(), con))
    {
        cmd.Parameters.Add("@Quantity", SqlDbType.NVarChar, 255).Direction = ParameterDirection.Output;
        cmd.CommandType = CommandType.StoredProcedure;
        con.Open();
        cmd.ExecuteNonQuery();
        workQuantity = cmd.Parameters["@Quantity"].Value.ToString();
        con.Close();
    }
    return workQuantity;
}
```

Figure 8-7 *GetWorkQuantity* method

8.2.5 Marker Class

The *marker* class contains a method called *Marker*. This *Marker* method shown in Figure 8-8 is the same method that was mentioned in subsection 8.2.1. This class was used to create marker objects. These marker objects contain the latitude, longitude, address, region, id and status of a station. When a new marker object was created, it was added to the list of marker objects that was declared in the first step.

```
public double stationlat;
public double stationlng;
public string stationAddress;
public string stationRegion;
public string stationId;
public string stationStatus;
public Marker(double latitude, double longitude, string address, string region, string Id, string Status)
{
    this.stationlat = latitude;
    this.stationlng = longitude;
    this.stationAddress = address;
    this.stationRegion = region;
    this.stationId = Id;
    this.stationStatus = Status;
}
```

Figure 8-8 *Marker* method

8.3 Architecture and Design

The website was based on the planning done for the UI and UX as well as the requirements and UML diagrams. When visiting the website, the first page prompted a login using a Microsoft account. The only configuration needed for this authentication process was adding the values explained in section 6.3 to Azure. Only operations central personnel with a Microsoft account registered in the AAD could access the website beyond this login page.

During loading of the *Status* page, the table used a web method to replace integers with icons that indicate the status of the work. Ajax was used to get the user's first name and display it in a label. The search box text was used to get work stored in the Azure database to fill the table. The number of work were also received from the database and displayed in a label.

Right before the *Status* page finished loading, the page would check if the user had clicked the more info button. If it was the case, the Google Maps API would be loaded. If there were any active work, the markers would be loaded and added to the map using ajax. After loading was finished, the table, map and number of work would refresh every five seconds or every time the user changed text in the search textbox.

The user had the option of clicking to the history page. The history page would also send the text in the search textbox to fill the table but would receive all completed work. The number of work shown in the label would reflect this. Each pages' design solution is shown in more detail in this section. Figure 8-9 shows the workflow of the web application.

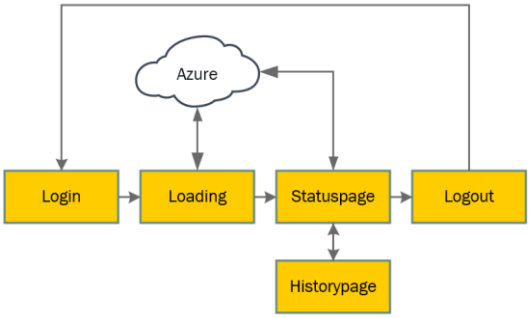


Figure 8-9 Web application flow

8.3.1 Login Page

Users visiting the website would be presented with the *Login* page shown in Figure 8-10. The user then typed in their email address and password for an account connected to Skagerak's Azure Active Directory. The page worked just as any other login page using Microsoft accounts for login, like the mobile application.

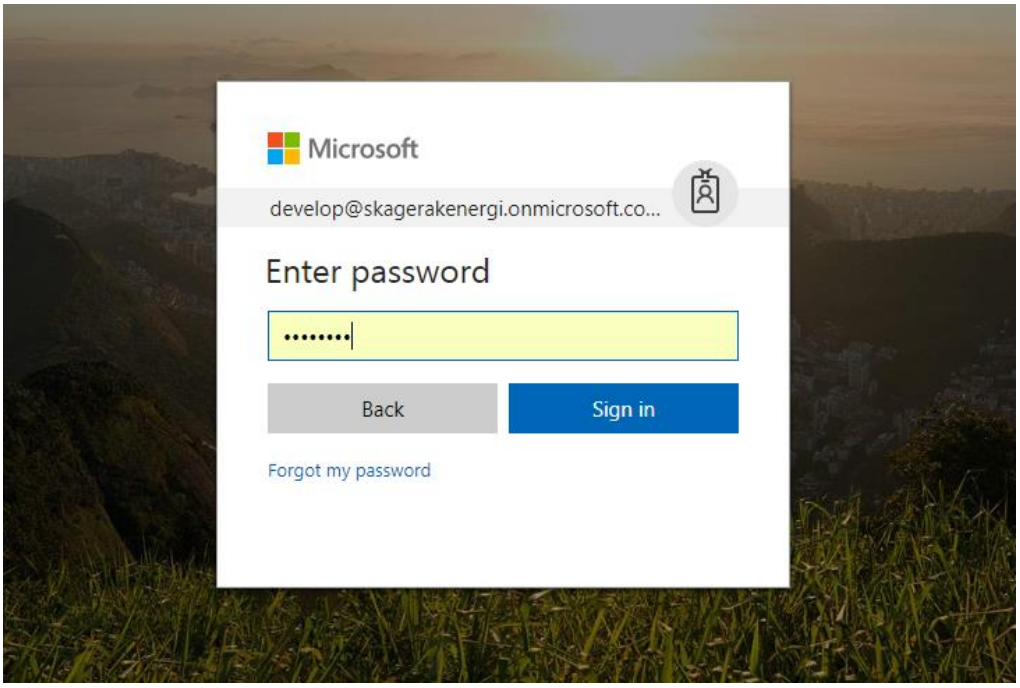


Figure 8-10 Web application Login page

8.3.2 Status Page

Figure 8-11 shows the status page for the web application. The table would only show rows that contained the text in the search box. The number of work was displayed beside the "Aktive:" label. Since the table only contains one work, only one marker is shown on the map. The status of the work is also indicated, both by the colour of the marker and the icon under the "Status" column.

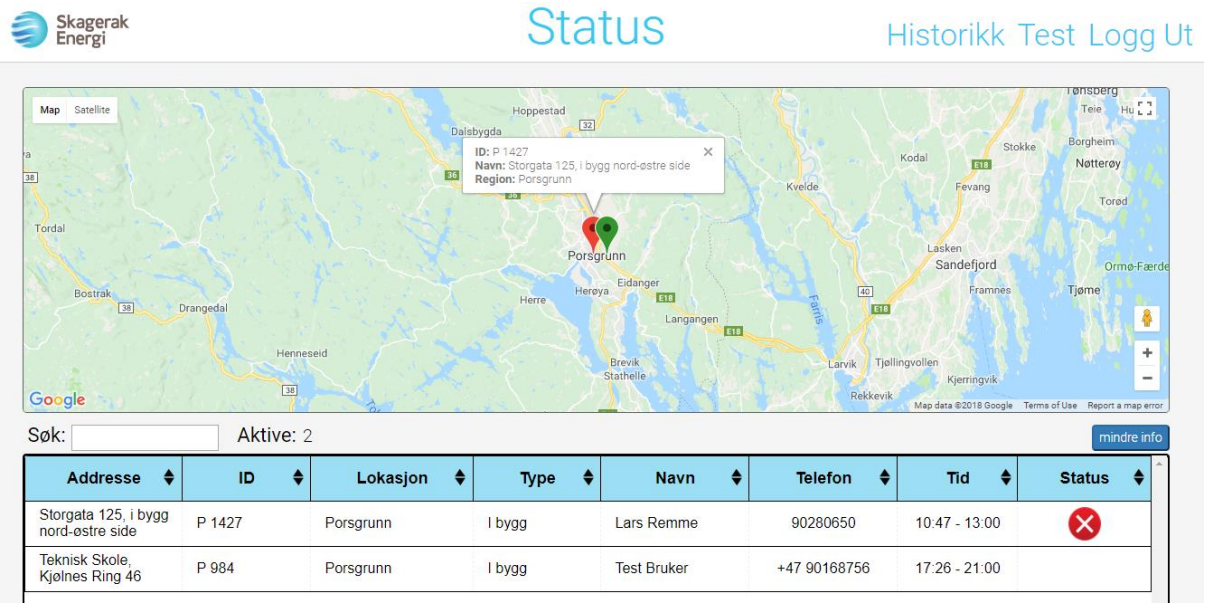


Figure 8-11 Web application status page

8.3.3 History Page

If the user clicked on the "Historikk" button shown in Figure 8-11, the user was directed to the history page. The history page shown in Figure 8-12 is identical to the status page when a map is not displayed. The user has the same functionalities as the status page. Clicking the column header sorted that column alphabetically, either ascending or descending. As Figure 8-12 shows, the number of completed work is 115.

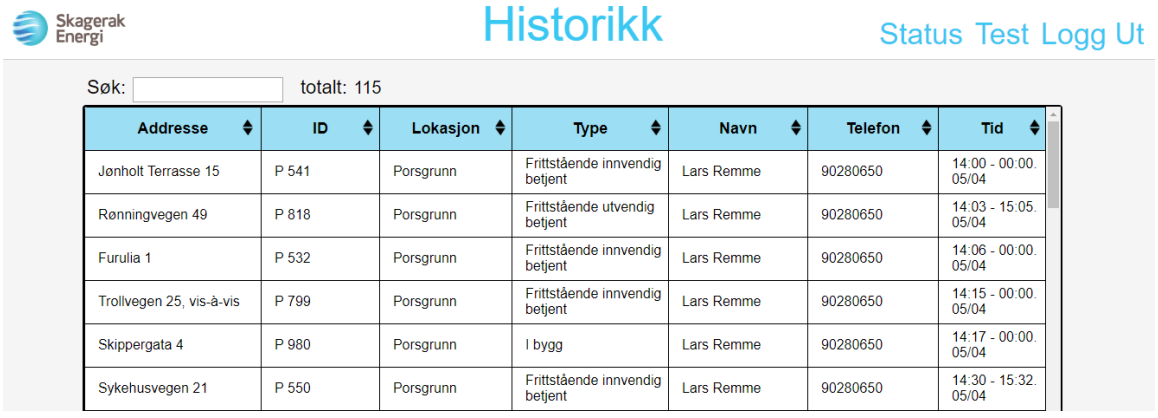


Figure 8-12 Web application history page

9 Mobile Application – Solution

This chapter will cover the present version of the mobile application, and which options were selected when developing the application. It explains how certain functions operate and how they were implemented. Appendix H contains the user manual for the mobile application.

9.1 Software and Code

The mobile application was developed in Microsoft Visual Studio (VS), an IDE for creating software applications. In VS, an extension was required for development of a cross-platform mobile application. This extension is called Xamarin. Background for Xamarin is covered in chapter 5, describing how it's able to build cross-platform application with only one programming language.

9.1.1 Platform

When developing a new cross-platform application some ground structure must be set before beginning. For example, choosing the UI, the architecture for code sharing and which platforms to cover. The UI was set to Xamarin.Forms, the architecture for code sharing was set to .NET standard, and the platform was set to cover both Android and iOS, as requested by the project partner. This also has the possibility to include UWP.

The reasons for the choices are explained in the next subsection. Figure 9-1 shows the options that were given during the creation of the application.

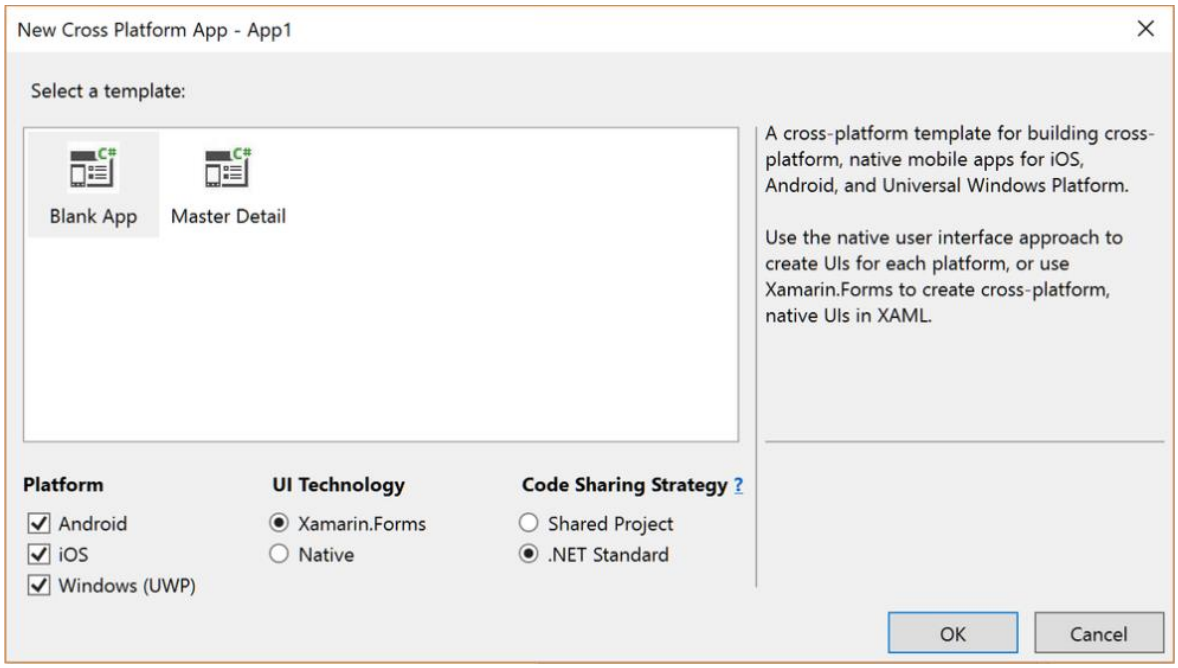


Figure 9-1 Creating a Xamarin project solution [33]

9.1.2 UI and Code Sharing

Figure 9-1 shows two options for the UI, which are Xamarin.Forms and native. Differences between them are how they shared the UI. For native option, each UI is developed independently, this allows the developers to have more freedom in designing each platform with its own custom looks. The other option is Xamarin.Forms, it creates a similar interface for all platforms in the same project.

These two options have their own strengths and weaknesses. The native option has more control over how the UI can be designed for each platform. The other reduces the extent of recreating similar code for the individual platform UIs. It compiles down and rebuilds the "same" interface across the chosen platforms. The UIs cannot be the same since they have their own code structures for how the interface will be presented. Based on the requirements, it is intended to be simple for the technicians to use in their daily work routine.

On the code sharing strategy, shown in Figure 9-1, the options were Shared Project and .NET standard. Both selections are covered in chapter 5, describing how their code sharing architecture is. The .NET standard's approach for a common code base was more straightforward than Shared Project, where most of the code is platform specific. A decision was made to choose the former approach. This allows the project to recycle a great portion of its code and use it on other platforms.

9.2 Classes

A good practice for developers is to create classes with their own set of functionalities and then reuse them. The code becomes easier to understand and maintain. The following subsection will explain the classes that are associated with Easy table, the map and substation information, as well as code examples of selected parts of the application.

9.2.1 Azure Table

For the application to communicate and exchange data with Easy table in Azure, it requires classes with the same property names as in the SQL tables. Figure 9-2 presents the classes for Easy table, where each have their own fields, properties and methods. These classes are primarily used to send and receive values in the correct SQL table, which is connected to Easy table. These tables are explained in section 6.3.

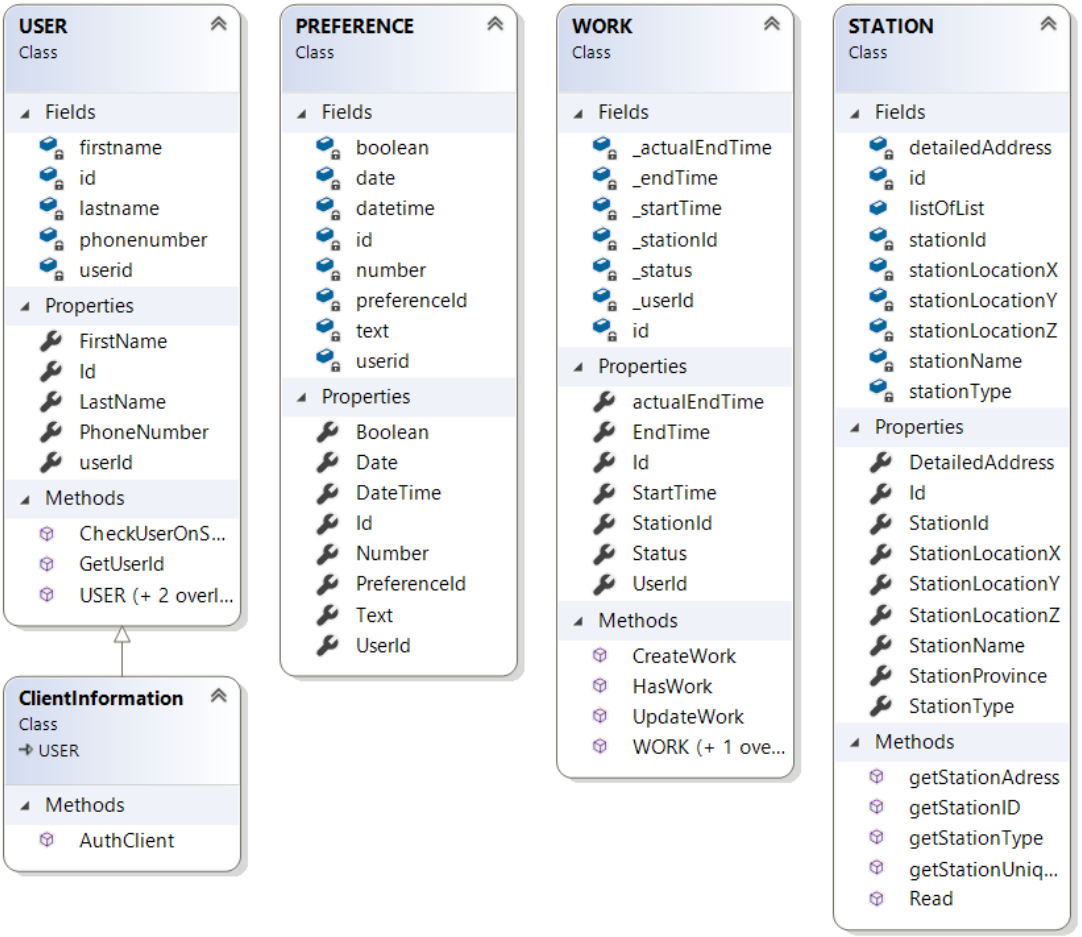


Figure 9-2 Azure classes

There are three methods that are used to communicate with Easy table: *ToListAsync*, *InsertAsync* and *UpdateAsync*. The first method sends a request for data, specific to the USER table, to Easy table, as seen in Figure 9-3 in frame. This method has a USER object as parameter and by calling the *App.MobileService*, located in the App class, targets the mobile applications website provided by Azure. This method checks if the user exists in Easy table after being authenticated at the login page. It returns the user's properties as a list. After receiving the list, it does a check if it contains any data. If the list is NULL, the second method is used to insert the same USER object into Easy table. How the authentication works is explained in more detail in section 9.4. [34]

```
public async Task CheckUserOnSql(USER usercheck)
    {
        var isExist = (await App.MobileService.GetTable<USER>().Where(u => u.FirstName == usercheck.FirstName &&
        u.LastName == usercheck.LastName).ToListAsync()).FirstOrDefault();
        if (isExist == null)
        {
            await App.MobileService.GetTable<USER>().InsertAsync(usercheck);
        }
    }
```

Figure 9-3 Get USER table method

To change specific data in Easy table the third method is used, *UpdateAsync*. This permits the application to change the data only related to the object parameter. A code snippet for this method is framed and shown in Figure 9-4.

```
public async Task UpdateWork(WORK updateWork)
    {
        try
        {
            await App.MobileService.GetTable<WORK>().UpdateAsync(updateWork);
        }
```

Figure 9-4 Update work table method

9.2.2 STATION – Read Method

This class contains properties for the different types of information stored about the substations in the database. It also contains a *Read* method, which is the focus of this subsection, used to load a list of substations onto the mobile device, as well as methods used when the user is searching for substations.

The *Read* method is needed because all the substations are stored in an Azure cloud database. A list of substations must be loaded onto the mobile device to be available when the user registers a job. The list is limited to a few thousand substations, so it must be updated if the user moves too far away from where it was originally loaded. The update of the list is not implemented at this point but will be covered as further improvements in subsection 12.3.4.

This is a static async task method. Static means that the method is not associated with an object of the class, but rather the class itself. This is done because the method shall load/update to the same static list each time it is called, even across classes. The async task part is added so that the method will be awaited from where it is called in the code, and at the same time can run in parallel with other code so that the application does not freeze while waiting for the method to finish.

```
double rxa = x - distance; // The user's x- and y- coordinate plus/minus a distance given in  meters.
double rxb = x + distance;
double rya = y - distance;
double ryb = y + distance;
try
{   List<STATION> stationList1 = await App.MobileService.GetTable<STATION>()
        .Where(p => p.StationLocationX <= rxb && p.StationLocationX >= rxa).Take(1000).ToListAsync();
    List<STATION> stationList2;
    List<STATION> stationList3;
    List<STATION> stationListX;

    if (stationList1.Count >= 1000)
    {       stationList2 = await App.MobileService.GetTable<STATION>()
            .Where(p => p.StationLocationX <= rxb && p.StationLocationX >= rxa).Skip(1000).Take(1000).ToListAsync();
        if (stationList2.Count >= 1000)
        {
            stationList3 = await App.MobileService.GetTable<STATION>()
                .Where(p => p.StationLocationX <= rxb && p.StationLocationX >= rxa).Skip(2000).Take(1000).ToListAsync();
            stationListX = stationList1.Concat(stationList2).Concat(stationList3).ToList();
        }
        else
        { stationListX = stationList1.Concat(stationList2).ToList(); }
    }
    else
    { stationListX = stationList1.ToList(); }

    listOfList = stationListX.Where(p => p.StationLocationY <= ryb && p.StationLocationY >= rya).ToList();
```

Figure 9-5 The main part of the Read method

The method is called with three parameters: the user's two coordinates and a distance in meters. It is important to note that the coordinates used in this method are in the UTM 32N format, and therefore the user's coordinates must be converted from latitude and longitude into easting and northing before the use of this method. This is also the reason why the distance is given in meters, as these types of coordinate represents approximate meters to the east and to the north of the set reference point given for UTM zone 32N. As Figure 9-5 shows, its function is to first request a list of substations from the database, where the substations have an x-coordinate that is within a certain range of the user's x-coordinate. Then use the received list to do the same for the y-coordinate. This range is given with the distance parameter as meters in both positive and negative x- and y-coordinate direction compared to the user's position. Consequently, the substations loaded to the list are within a rectangular area of the user.

The Azure mobile service does not return all rows matching a query, but instead is limited to a maximum of 1000 rows. Because of this the method also needs to check if the list contains 1000 items. In the case that it does, another request towards the database is made. This new request will skip the first 1000 matching rows and return up to 1000 more. This check is done twice, and the lists are concatenated making 3000 the maximum number of substations returned by the method. This method has an issue and can be improved as described in subsection 12.3.4. [35][36]

9.2.3 Location Class

The location class is used to handle information regarding the map, geographical positions, and finding which stations should be displayed on the map. The class contains two methods: *GetDistance* and *StationsWithinDistance*.

The first method, *GetDistance*, uses the Geolocator Plugin package, referred to in section 9.6.1, to calculate the distance between two geographical locations. This is done with both locations represented in latitude and longitude format, given as four parameters. The output represents a distance in kilometres, as a straight line between these locations.

The second method, *StationsWithinDistance*, loops through the list of substations that is loaded at start-up and finds the ones that are within a certain distance to the user. To do this, it uses the *GetDistance* method which means that the substation's coordinates must be converted from UTM 32N to Web Mercator. As the method loops through the substation list, it will create a list of map markers containing the appropriate substations. These are the markers that will be displayed on the map.

9.2.4 Coordinate Conversion - Background

Skagerak Energi provided a list of electrical substations with information such as name, address, types of station and location in the form of coordinates. This information was added to an Azure database which serves as a connection between the website and the mobile application.

To display the electrical substations on the map, the given coordinates must be in a specific format. Google maps, which is used by the android version of the mobile application, needs coordinates to be provided in latitude and longitude decimal degrees. The coordinates in the database however, are in the form of eastings and northings. Therefore, a conversion is needed

between WGS84 Web Mercator, which is what Google Maps uses, and ETRS89 with UTM zone 32N, which is the format of the provided coordinates. [37][38]

The two geodetic datums WGS84 and ETRS89, which are ellipsoid representations of Earth, are interchangeable as ETRS89 is derived from WGS84 and when converting between the two, the resulting position differs less than one meter from the original. Therefore, no conversion is needed between the datums. [39]

Web Mercator and UTM are both variants of the Mercator projection, which means that they show curved geographical areas of Earth projected onto a flat surface. Even though both are based on the same projection, there is a need for conversion between the two because of the format. Conversion between these formats are done with two different classes.

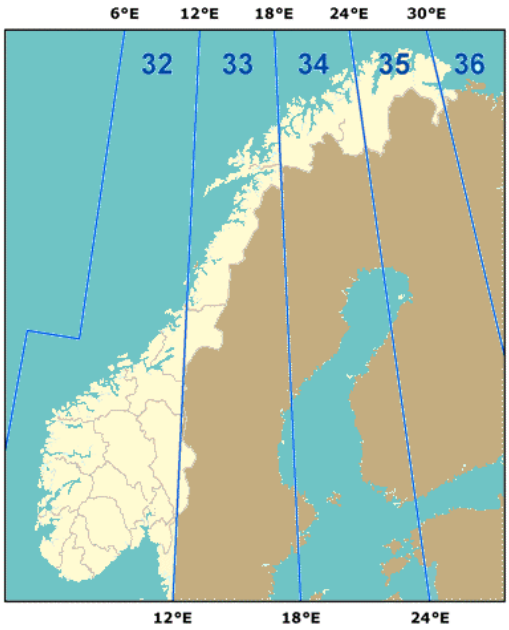


Figure 9-6 UTM zones [40]

Figure 9-6 shows how Norway is divided into UTM zones and what area is covered by zone 32 used in this application's code. These zones are necessary to minimize the distortion introduced when projecting the Earth's surface onto a flat plane. Each zone is a longitude band divided into northern and southern hemispheres, hence the N in UTM 32N, but can also be divided further into rectangles designated with other letters.

9.2.5 Coordinate Conversion - Implementation

The two classes *UTM2Deg* and *Deg2UTM* have been used to convert the coordinates back and forth between the two formats. [41]

UTM2Deg is the class used when converting from the coordinates stored in the database to what is needed when displaying the substations on the map. It is used when the markers on the map are updated, which happens when the user's location changes, and if the user find a substation through the search function. Figure 9-7 shows an example of how these classes are used.

```
UTM2Deg fromUTM = new UTM2Deg(App.listOfList[i].StationLocationX, App.listOfList[i].StationLocationY);
distance = GetDistance(userPos.Latitude, userPos.Longitude, fromUTM.Latitude, fromUTM.Longitude);
```

Figure 9-7 *UTM2Deg* class is used in the *StationWithinDistance* method

In addition, there is a class called *Deg2UTM* converting in the opposite direction. It is needed when a list of substations in the area of the user is extracted from the database; the user's coordinates are converted so they can be used to search for approximate matches in the database. See the *Read* method.

9.2.6 HomePage

This is a partial class inheriting from the *ContentPage* class in *Xamarin.Forms*. It contains methods for handling the user's position and updating information on the map, and event handlers for the various buttons on the page.

Two of the methods and one of the event handlers are based on the GeolocatorPlugin NuGet package and its documentation. The first method, *StartListeningForPosition*, is called when the Home Page is first opened. It is a task that runs asynchronously awaiting a *GeolocatorPlugin* method which takes arguments defining the update frequency of the user's location. The second method, *GetUserLocation*, awaits and returns the user's coordinates when called. The event handler, *Locator_PositionChanged*, calls for the *UpdatePosition* method described below when the location changes. [42]

The *UpdatePosition* method is called when Home Page is opened and when the location changes. It calls a method which puts together a list of map markers of the nearby electrical substations and then adds them to the map.

9.3 Architecture and Design

The UI for the application is based on the requirement for simplicity and guided behaviour. This is illustrated in Figure 9-8, which is an overview of the structure. When the application first starts up it will display the login page. This page authenticates the user of the application. After the user is verified, the loading page will obtain nearby stations from Easy table with the user's GPS location.

When it finishes acquiring the data from Easy table the home page will appear. This page displays a map with location indicators (pins), which marks substations nearby the user's present location.

When the user has selected a substation and pressed the time selection button, the application would then navigate to the time page where the user can choose the amount of worktime by rotating the clock.

A static work object is initialized when the application starts up. This object is created from the *WORK* class. Figure 9-2 shows the structure of this class. The object obtains data when the user has selected the worktime and pressed the confirm button. Before moving the user to the work page, the data from the work object is sent to Easy table with the method described in subsection 9.2.1.

After successfully inserting the data to Easy table, the application will then navigate the user to the work page and display general information about the current work and remaining work-time. Each page's functions are explained in more detail in the following subsections.

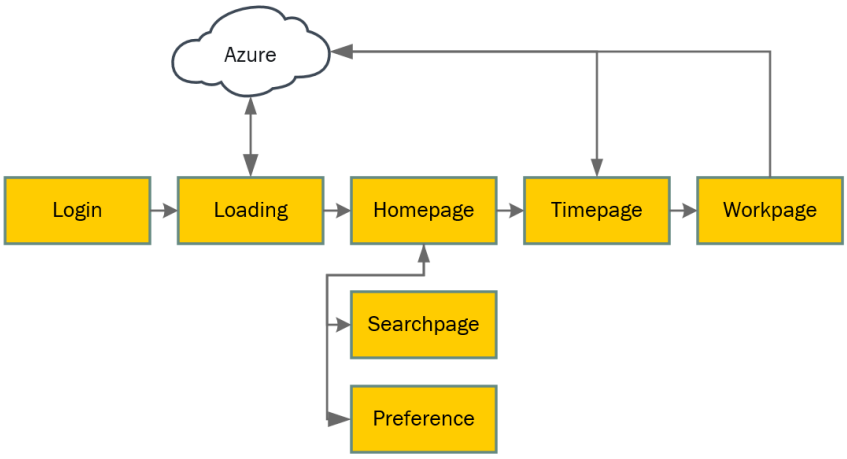


Figure 9-8 Mobile page flow

9.3.1 Login Page

Figure 9-9 shows the login page's UI appearance of the application's current solution. The UI present the user with a single login button. When pressed the application performs a call for authentication and opens the device's default web browser. Details on how the authenticator requests data is explained in the section 9.4. The web browser displays a Microsoft login page, where the user inputs email address and password for an account connected to Skagerak's AD.

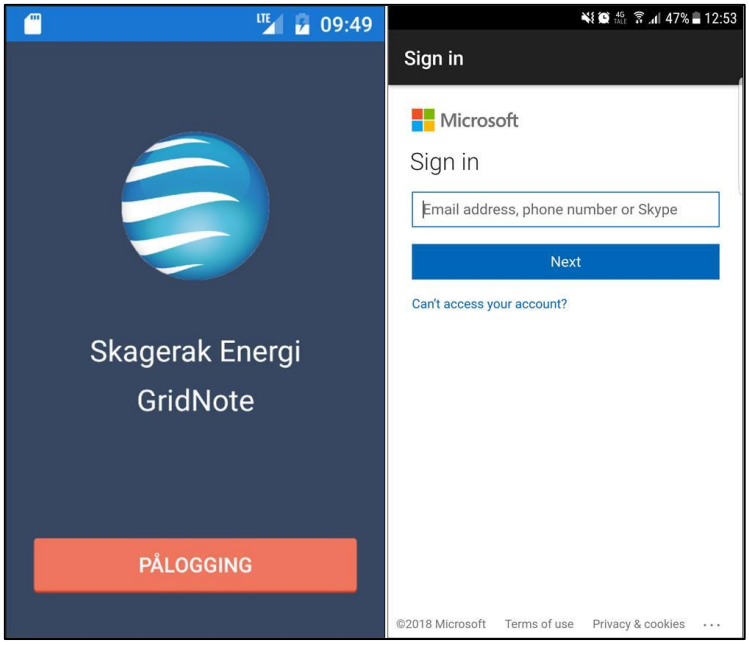


Figure 9-9 Login page

9.3.2 Loading Page

The loading page is shown while information regarding the substations are acquired from Easy table using the user's GPS location. Loading page invokes a method for loading stations which is mentioned in subsection 9.2.2. It's important to have search limit since the store data has information of more than 7000 substation spread across the country. Narrowing it down will potentially reduce the runtime for obtaining data. It also lessens the temporary store space in the application.

This page also checks if the user exists in the database, and then obtains the unique ID. If the user does not exist, a method is invoked which passes an object of the USER class. The structure of this class is shown in Figure 9-2. This object acquires its necessary data when the user gets authenticated in the Login page. This method is shown in Figure 9-3, it checks if the received data is NULL and inserts it if that is the case.

9.3.3 Home Page

Figure 9-10 shows the home page for the application. This is the page it returns to when the user has completed a job. This page has four touchable actions: "Preferanse", "Søk", Map and "Velg Arbeidstid". These interactions, respectively: navigates the user to a "Preferanse" page where it has a few settings for the user to change and this is further explained in the next subsection; navigates the user to a "Søk" page and allows the user to search for stations by typing letters, this is given in more detail in subsection 9.3.5; the map lets the user visually see the present position and nearby substations, the user can select a station by pressing the info box of the selected pin; the "Velg Arbeidstid" button navigates the user to the next page of the application which is the time page.

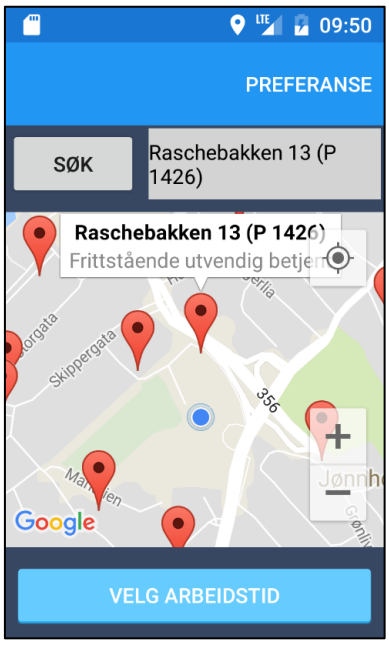


Figure 9-10 Home page

9.3.4 Preference Page

Features in the Preference page are for future development and currently has no available settings for the user to choose. Therefore, its only appropriate function is to have the sign out button, “Logg av”. When it is pressed, the application signs out the user by deleting the cookies and the page stacks, and then returns the user to the Login page. Figure 9-11 shows how the page looks like in the working application.

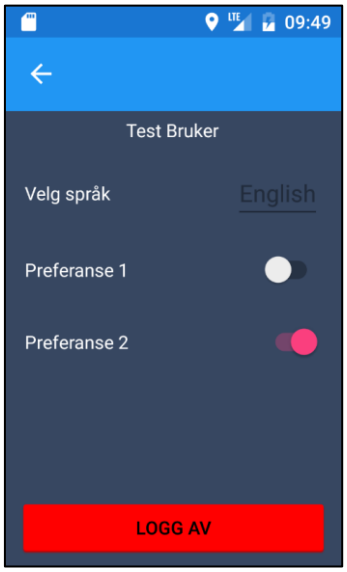


Figure 9-11 Preference page

9.3.5 Search Page

The approach for including a search page for the application is for cases where the substation is out of range of a network connection. This search feature lets the user input the station ID or location. This function currently requires the substations to already be loaded onto the device.

Each entered character in the search box triggers a refresh method which updates the shown list in Figure 9-12. This list provides suggestions of stations for the user to choose from and a feature for showing the total number of search results. The user can scroll through the list to view results of the search and by pressing one of the elements in the list, the application navigates the user back to Home page with the information of the selected substation.

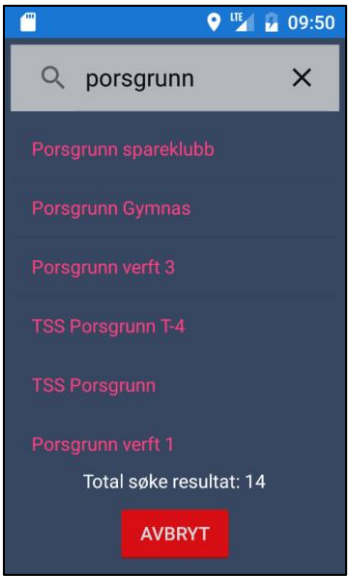


Figure 9-12 Search page

9.3.6 Time Page

The Time page asks the user to select their expected work time, as seen in Figure 9-13. A wheel displaying time selection appears when the user presses the time picker at the centre of the screen. When the user has selected a work time from the time wheel, the application disables the "Start Arbeid" button until the user has pressed the "Trykk her for å bekrefte valg av arbeidstid", as an explicit function for the user to confirm the work time. This page also features a value check for the selected time, which returns an alert message if the user's selected time is invalid. When an appropriate work time is confirmed, pressing the "Start Arbeid" button will navigate the user to the Work page, at the same time as the data for this work is sent up to Easy table with the method described in section 9.2.1.

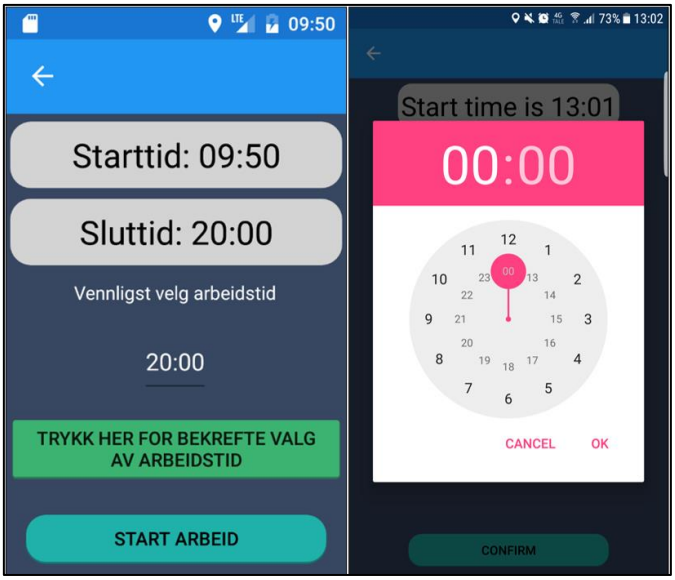


Figure 9-13 Time page

9.3.7 Work Page

The work page presents information of the current work in the grey area, shown in Figure 9-14. The grey region displays the substation's id, address, description, expected work time and remaining work time, in that order.

The remaining time invokes a function that counts down every second and displays the change. This feature provides a better UX because the user can just read from the information and does not need to calculate manually the remaining time each time they open the application.

This page only has two buttons for the user to select. "Arbeid Ferdig" ends the present work by sending a data package to Easy table, informing that the work is completed and navigates the user to Home page. "Endre" permits the user to change/extend the work time. It navigates the user back to the Time page where they can select a new work time.

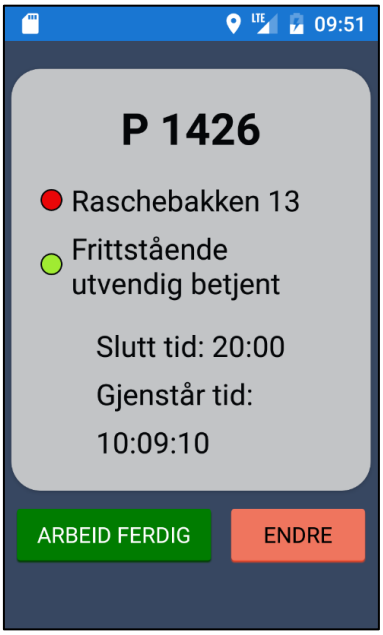


Figure 9-14 Work page

9.4 Implementing Active Directory

This section will explain the reason for implementing the Active Directory (AD) authentication and how this was done. Code examples from the application are included for the purpose of explaining how the data is sent and received.

Based on the requirements from the project partner the application must be secured and only permitted to use by specific workers at Skagerak Energi. This feature intends to have the flexibility of adding and removing people's access to the application while utilizing their existing AD.

Azure offers services, as mentioned in section 6.3, to connect the local AD with the Azure's AD. The application can, therefore, authenticate its users by asking the cloud AD and would

acquire the same information as if it were local. The Azure AD can create groups with permission to use this application, however, configuration for creating and assigning groups are not included in this report since it is outside of the project's scope. Necessary values for implementing the AD authentication service was provided by the project partner. This is shown in Figure 9-15. The following subsection will go in detail on what data is required for the application to obtain information from the cloud AD, using Active Directory Authentication Library (ADAL).

```
public static string clientId = "9cdaf42d-1e17-4819-ab85-3324dd459bef";
public static string authority =
"https://login.microsoftonline.com/070ea4ed-4fd0-4da1-872d-435bcf2f285b/oauth2/authorize?resource=https%3A%2F%2Fgraph.windows.net";
public static string returnUri = "https://gridnoteapp.azurewebsites.net/.auth/login/aad/callback";
private const string graphResourceUri = "https://graph.windows.net";
```

Figure 9-15 Active directory connection

9.4.1 Active Directory Authentication Library

The ADAL enables the application to have authentication to the cloud AD and obtains tokens for securing API calls. The library used in this application is downloaded from a NuGet manager:

- Microsoft.IdentityModel.Clients.ActiveDirectory (3.19.2). [43]

The login button code shown in Figure 9-16, invokes an await method called *DependencyService* when pressed. This allows the application to call platform-specific functions from the shared code, it finds the correct implementation of the interface from various platforms. [44][45][46]

A new interface is created for the authentication method, which returns the result from ADAL and contains an access token. "*An access token is an object that describes the security context of a process thread.*" [47]

```
private async void loginButton_Clicked(object sender, EventArgs e)
{
    try
    {
        var data = await DependencyService.Get<IAuthenticator>().Authenticate(authority,
        graphResourceUri, clientId, returnUri);
        ClientInformation clientInformation = new ClientInformation();
        token = data.AccessToken;
        var authConfirme = clientInformation.AuthClient(token);
```

Figure 9-16 Retrieves access token

The access token is store as a static string value named *token*, this value will then get passed into a method called *clientInformation.AuthClient(token).* The method is defined in *ClientInformation* class, Figure 9-16 is a code snippet from the application, where the method handles a request for obtaining the users information with Hypertext Transfer Protocol (HTTP).

HTTPRequestMessage is defined as a new object where the constructor takes a Uniform Resource Identifier (URI) and the GET property is set. A new object of http header for authentication is added to the request object. The http header is defined with a constructor consisting of a schema "Bearer" and the parameter *AccessToken*. Defining this schema means the server should give access for the bearer of this token. [48][49]

The variable *response* stores the responding data from the client object, shown in Figure 9-17, where *HTTPRequestMessage* object is sent as an input. The following if statement checks for the status code of the *response* variable to be 200. If true, the status code is referred to as a success and the *response* object contains the requested data. [50]

Data from the *response* is in a JavaScript Object Notation (JSON) format, a text format that is language independent. To extract the *response* data a NuGet packed was used:

- Newtonsoft.Json (11.0.2) [51]

The class inherits from the *USER* class and can, therefore, have the same properties from the base class. These properties are used for storing the values from the extracted *response* data. After obtaining all the necessary values, they are return as one object of type tuple. This is a data structure that has a specific number and sequence of elements. A benefit for using it, in this case, is to return multiple values without using out or ref parameters. [52][53]

```
public class ClientInformation : USER
{
public async Task<Tuple<string, string, string,bool>> AuthClient(string AccessToken)
{
    try
    {   var Client = new HttpClient();
        var userInfoRequest = new HttpRequestMessage
        {   RequestUri = new Uri("https://graph.windows.net/me?api-version=1.0"),
            Method = HttpMethod.Get);

        userInfoRequest.Headers.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", AccessToken);
        using (var response = await Client.SendAsync(userInfoRequest).ConfigureAwait(false))
        {
            if (response.IsSuccessStatusCode)
            {
                var responseString = await response.Content.ReadAsStringAsync().ConfigureAwait(false);
                var jObject = JObject.Parse(responseString);
               FirstName = (string) jObject["givenName"];
                if (String.IsNullOrWhiteSpace(FirstName))
                    throw new Exception("firstName was not set for authenticated user");
               LastName = (string) jObject["surname"];
                if (String.IsNullOrWhiteSpace(LastName))
                    throw new Exception("lastName was not set for authenticated user");
                PhoneNumber = (string) jObject["mobile"];
                if (String.IsNullOrWhiteSpace(PhoneNumber))
                    throw new Exception("Phone number was not set for authenticated user");
            }
        }
        var tuple = new Tuple<string, string, string, bool>(FirstName, LastName, PhoneNumber, true);
        return tuple;
    }
    catch (Exception e)
    {
        var tuple = new Tuple<string, string, string, bool>(FirstName, LastName, PhoneNumber, false);
        return tuple;
```

Figure 9-17 Retrieving information from active directory

9.5 Implementing Graphical Map

The mobile application uses map and location functionalities to display the user's position and the positions of electrical substations on a map. The map makers are shown only when the user

zooms in. Information is updated when the position of the user changes. It currently updates when the user moves about 20-30 meters.

Since the application uses “Google Maps for Android”, an API key connected to the package name of the project had to be added to the “Android Manifest” of project. Permissions such as “ACCESS_FINE_LOCATION” also had to be granted for the application to be able to access the geolocation feature of the mobile device. This is also added to the Android Manifest. The “Android Manifest” is shown in Figure 9-18.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0"
package="com.companyname.GridNote" android:installLocation="auto">
    <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="25" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
    <application android:label="GridNote.Android" android:icon="@drawable/ic_launcher">
        <meta-data android:name="com.google.android.geo.API_KEY" android:value="AIzaSyCwc-ffLHjEOKG4r0g2_DIM7osfrKDCWY4" />
    </application>
</manifest>
```

Figure 9-18 Android Manifest

9.6 NuGet Packages

This section contains information about the NuGet packages used map and geolocation functionalities.

9.6.1 GeolocatorPlugin

It is used to easily access the geographical location of the user and works on mobile platforms such as Android API version 14+, iOS 8+ and Windows 10 UWP 10+. The package is needed in the mobile application to find the users location, so that it will be possible to display electrical substations near the user. The package is developed and maintained by James Montemagno and is available under the MIT licence. [54]

Documentation with examples for the GeolocatorPlugin has been used when writing the code for the geolocation part of the project. [42]

9.6.2 TK.CustomMap

The NuGet package used to display the map is TK.CustomMap developed by Torben Sudau and is available under the MIT licence. This package is based on the Xamain.Forms.Maps package by Xamarin Inc. It provides advanced cross-platform map functionalities, some of which include customizing pins, adding overlays, calculate routes, search place predictions, cluster pins. [55][56]

10 Testing

This chapter describes several different test methods and gives examples of some of the tests performed. The results of tests for the mobile and web applications are attached as Appendix I and J.

10.1 Test Methods

This section describes the four different software testing methods. These are: unit test, integration test, system test and acceptance test.

10.1.1 Unit Test

The unit test is a software testing method where individual units of the code is tested. A unit can be viewed as the smallest part testable in the system such as a method, procedure or an event. The goal of unit testing is to validate that each unit of the software performs as designed. [57]

10.1.2 Integration Test

Integration testing is a software testing method where the individual software units are tested as a combined unit. The goal is to test the interface between the units. This testing method is performed after two or more units have been produced and unit tested separately. [58][59]

10.1.3 System Test

A system test is a software test method where the entire system is compared to a set of software requirements. This means that the system's design, user interface and overall behaviour is tested from the user's point of view. This type of testing is usually performed after the integration testing has been completed. [60][61]

10.1.4 Acceptance Test

For the system to be functional for the end-users, it must pass the acceptance test. The purpose of this test is to evaluate the systems requirements and to see if the program is acceptable for use. However, this test is not included in this project, because the product is under development. The acceptance test must be executed together with Skagerak Energi. [62]

10.2 Function Test

This section describes one example for each of the test types used during the project. The system was tested based on the requirements in Appendix E.

10.2.1 Unit Test

Case number 1 in Appendix I is used in the following example of a unit test. The method *UTM2Deg*. This method transforms UTM coordinates to latitude and longitude. To test this

method, a unit test was created in VS as shown in Figure 10-1. The unit test involves comparing a set of expected values against the resulting values from the tested method. In this example, the coordinates of substation Klokkerveien 12 was selected as the testing values. The expected values from this substation, were found by using a UTM to latitude and longitude converter on the internet. These values were then compared against the result values produced by the tested method, to check if they are equal. [39]

```
[TestMethod]
public void UnitTestUTM2Deg()
{
        double northing = 6566347.8203125; //Klokkerveien 12
        double easting = 580208.125;
        double expectedValueY = 59.228558;
        double expectedValueX = 10.405474;
        double latitude;
        double longitude;
        Status status = new Status();

        status.UTM2Deg(easting, northing, out latitude, out longitude);
        double actualY = latitude;
        double actualX = longitude;
        Assert.AreEqual(expectedValueY, actualY, 0.000001, "Not the same");
        Assert.AreEqual(expectedValueX, actualX, 0.000001, "Not the same");
```

Figure 10-1 Unit test method

10.2.2 Integration Test

The integration test in this section is test case number 20 in Appendix I: Search in table. It is used to test four parts of the system separately and as a group to check if they work together. The first part is the stored procedure *TableRefreshHistory* shown in Figure 10-2 (*TableRefresh* has also been tested, but in this subsection only *TableRefreshHistory* will be shown). The second part of the test is the search textbox on the website. This textbox is used to filter searches in the table. The third part of the test is to see if the right search will be shown in the table. The last part is the connection to the database.

The first test was used to check if the *TableRefreshHistory* is working, by searching for a specific date. The selected date for this were 08/04 as shown in Figure 10-2.

```
DECLARE @Search nvarchar(255) = '08/04';
    SELECT
    [dbo].[STATION].DetailedAddress, [dbo].[STATION].stationId, [dbo].[STATION].stationProvince,
    [dbo].[STATION].stationType, [dbo].[USER].firstName + ' ' + [dbo].[USER].lastName,[dbo].[USER].phoneNumber,
    FORMAT([dbo].[WORK].startTime,'HH:mm') + ' - '
    + FORMAT([dbo].[WORK].actualEndTime,'HH:mm')
    + '.  ' + FORMAT([dbo].[WORK].startTime,'dd/MM')
    FROM
    [dbo].[USER]
    INNER JOIN [dbo].[WORK] ON [dbo].[USER].Id=[dbo].[WORK].userId
    INNER JOIN [dbo].[STATION] ON [dbo].[WORK].stationId=[dbo].[STATION].[Id]
    WHERE
    (([DetailedAddress] LIKE '%' + @Search + '%') OR ([STATION].[stationId] LIKE '%' + @Search + '%')
    OR ([stationProvince] LIKE '%' + @Search + '%') OR ([stationType] LIKE '%' + @Search + '%')
    OR ([firstName] LIKE '%' + @Search + '%') OR ([lastName] LIKE '%' + @Search + '%')
    OR ([phoneNumber] LIKE '%' + @Search + '%') OR ([endTime] LIKE '%' + @Search + '%')
    OR ([startTime] LIKE '%' + @Search + '%') OR FORMAT([dbo].[WORK].startTime,'dd/MM')
    LIKE '%' + @Search + '%') AND [WORK].[Status] = 2
    ORDER BY [dbo].[WORK].[startTime] ASC
```

Figure 10-2 The stored procedure *TableRefreshHistory*

The result for the stored procedure is shown in Figure 10-3. The same result was expected to appear on the website.

Results | Messages

	DetailedAddress	stationId	stationProvince	stationType	(No column name)	phoneNumber	(No column name)
1	Sverresgate 19	P 540	Porsgrunn	I bygg	Lars Remme	90280650	10:10 - 12:21. 08/04
2	VIC hotell, Skolegata 1	P 536	Porsgrunn	I bygg	Lars Remme	90280650	15:35 - 17:36. 08/04
3	Storgata 112	P 549	Porsgrunn	I bygg	Lars Remme	90280650	15:41 - 17:43. 08/04

Figure 10-3 Result from *TableRefreshHistory*

The next part tests if the textbox can filter the search from the SQL table according to the provided search word and display the result on the website, as shown in Figure 10-4.

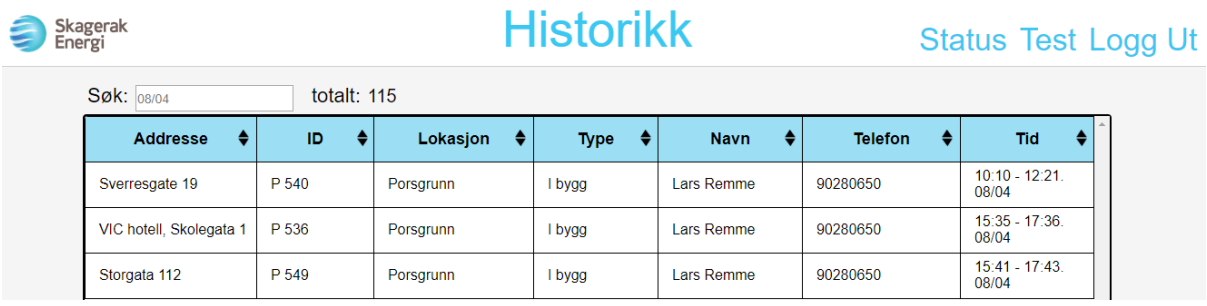


Figure 10-4 Result in the website

The results of this integrations test show that all the parts are working together. The filter search can change the table with the result from the stored procedure. This also means that the connection to the database works.

10.2.3 System Test

To get the system test approved, the system needs to pass the important requirements. This means that the mobile application needs to fulfil the requirements such as login/logout, create, edit and complete a job. The website needs to fulfil the requirements such as login/logout and the possibility to view ongoing jobs.

11 Discussion

This chapter discusses the project solution. It elaborates on each segment of the GridNote system and compares them to alternatives.

It was desired that the GridNote system should be a fully functional system at an early stage. Regarding the requirements in Appendix E, it was not feasible to fulfil all of these within the given timeframe. It was decided to first include the basic functionalities as soon as possible, and then improve on these as the project progressed. This approach ensured that the system always was in a deployable state. The overall system needs more testing to ensure the functions can perform as intended in a work environment and for a multitude of concurrent users.

The system's infrastructure was designed to be a cloud-based solution hosted on an Azure server. This has several benefits, such as a centralized data storage system which provides consistent and synchronized data to all devices. However, a discussion can be made regarding cloud-based services versus hosted or local services.

For cloud services and hosted services, software is stored on a remote server and accessed through an internet connection. One could argue that hosted services are not truly web-enabled as they might need infrastructure put in place on-site beforehand. Since the hosted service is owned by the business, the business is also responsible for maintaining the service. Contrary to cloud services that do not require any maintenance. Cloud services is the only option wholly accessible through the internet. [63][64][65]

Downsizing or upsizing any IT equipment on a local or self-hosted service requires additional capital costs. This cost exceeds the cost of upsizing a cloud service, as most offer the pay-as-you-go payment method. This method charges only based on the amount of usage, so one only pays for what is needed. Another argument for cloud services is the time it often takes to upsize or downsize a local service. This transition period often comes with some downtime and can be costly. [66]

The database was only developed to the extent that it would not affect system integrity. This was because the performance of the system would not be affected, as the numbers of users and thereby the processing power needed, would not exceed a certain limit. However, redundancy issues are present and the opportunity to optimize for this is clear. This optimization is detailed in the next chapter.

One request made from the personnel working at the operating centre was to increase the size of the table on their website. The table in the solution was therefore made to cover a large part of the webpage. The options for framework used for the website were between ASP.NET Web Forms and ASP.NET MVC. Web Forms was chosen because of previous experience with the framework, using it for projects developed prior to this thesis.

The mobile application was developed with opportunity to support multiple OS platforms. As explained in chapter 5, Xamarin was chosen because of the ability to share code between Android and iOS platform applications. Xamarin claims that, on average, 75% of the code that is created can be shared. [67]

A different development tool for the app could be Apache Cordova which uses the device's browser. This tool also allows for shared code and can utilize the same modules, but is written

in CSS, HTML and JavaScript. It would require knowledge in these languages that Xamarin does not use.

The mobile application's user interface should have been structured with the Model-View-ViewModel. This allows the code to become more testable, it uses data bindings, events and notifies property changes in the UI. This option was however not feasible due to the project's time constraint. [68]

12 Future Work

This chapter contains recommendations for future improvements to the GridNote system.

12.1 Database

The database structure, shown in Figure 7-1, is currently at 1NF and should be normalized to 2NF and eventually 3NF (or BCNF). An approach for this is to create new tables for repeating values such as *StationType*, *StationName*, *StationProvince* and *TrafoType*, where all values in these tables are uniquely identified by the table's primary keys. The planning for this normalization process has been started, but not yet implemented. An improved, but not complete, table structure can be seen in Appendix K. [69]

12.2 Web Application

This section describes future improvements of the web application such as tables, backup system and functions.

12.2.1 Interactive Table and Map

The data tables and map on the web application can be improved to be more interactive by including functions such as redirecting the map and displaying information from selected elements in table or map.

12.2.2 Real-Time Table

By implementing SignalR to the web application, the table and the other modules can be updated in real-time. SignalR involves making an open two-way connection between the database and the application so that information can flow freely and instantly. [70]

12.2.3 Subsystem

The web application should have a subsystem for operators to create, extend or complete tasks for technicians.

12.2.4 History

The history page in the web application should include features for operators to download historic data from the database in pdf, Excel and printing format. Operators should also have the opportunity to selected filtered data.

12.3 Mobile Application

This part describes future improvements to sections such as functionality, graphical representation, application behaviour, and also extending support to additional OS.

12.3.1 iOS

Adding support for iOS requires testing and development in a MacOS environment. This has not been done yet, but most of the functionalities in the current solution are shared between Android and iOS.

12.3.2 UI and UX

An adaptable UI can be implemented by auto adjusting objects to fit all device screens, including tablets, and for the text size to scale accordingly to objects' proportions. The language of the application should be adapted for the intended users, to avoid misinterpretations.

For the UX, a change would be to create a method allowing already signed in users to navigate automatically to the Home page, without the need of pressing the login button each time the application opens. This would eliminate a few unnecessary actions required from the user and offer a better experience.

Implement a more interactive time wheel instead of the current one shown in Figure 9-13. The time should continuously be displayed while the user adjusts it, using the wheel.

12.3.3 Location Class

The *Locationclass* has a method which is used to find substations within a radius of the user, and a method which shows a set number of the closest substations. Future improvements to the application may include making use of a combination of these two methods, so that the application always shows at least a few of the closest substations regardless of being within a specific range or not. This could be helpful in locations where there are few or no substations nearby. In the opposite situation, when there are many substations nearby, there might be convenient to have a maximum limit on the number of substations, showing only the closest.

12.3.4 Station Class

Loading substations from a large area, seems to cause an issue when displaying them on the map. This might have to do with loading duplicates from Azure, or the problem might be with trying to show too many map markers at once. This issue may be fixed by reducing the search area or showing map markers that are closer to the user. Both of these actions have been taken in the current version and the correct number of stations seems to be displayed. However, further testing is needed to know exactly what caused this issue. When further developing the application, this issue may be solved by changing the method altogether.

A better solution for the *Read* method might be to, instead of loading stations from Azure, have a local database on the mobile device containing a complete list of all the stations. It should then be possible to update this list by syncing with the Azure Cloud database. Changes in the list is assumed to happen rarely because electrical substations are not added or removed too frequently. Therefore, syncing the database may be done once a week or even rarer.

Another solution that is closer to the current one, might be to significantly reduce the area that the stations are loaded from during start-up. Then add a function which updates the resulting list when the user moves a certain distance away from the position where the original list was loaded.

13 Summary

The aim of the project was to develop a software solution for Skagerak Energi as well as documentation in the form of this thesis. The new solution is meant to be a replacement of an existing system that is used by technicians out in the field. This system is used to notify the operating centre about which electrical substations they are working on. It is there to ensure the safety of the workers by giving the operating centre an overview of the ongoing jobs.

The system that is currently in use employs a SMS solution to register the technicians' work. The information is handled by a third-party company before it is displayed on screens in the operating centre. Skagerak Energi requested an improved solution based on the use of modern technology. In addition to a solution that is easier to use and takes up less of the technicians' time, Skagerak Energi also wanted to present the information at the operating centre in an organised way.

In the starting phase of this project the system was determined to consist of a database, a mobile application and a web application. The foundation for the solution was laid by creating diagrams and design documentation. Simultaneously, relevant research to start implementing the system was conducted.

In the development phase, the focus was to make an intuitive and user-friendly interface that has the basic functionalities in place across the system. This way the system would always be in a useable state. Microsoft Azure was used to host the database and web services, and Xamarin to develop the cross platform mobile application.

Azure Cloud Services is used to centralize the database so that it is available for both the website and the mobile application. It is also connected to Skagerak's Active Directory, allowing them to create groups with permissions to access different applications.

The website was developed using ASP.NET Web Forms, a web development tool in Visual Studio. The website allows the operators to view registered jobs, using features such as a map to interpret the data content and portray them with clarity. It also includes a search function to filter active jobs according to a particular word. This can be helpful for the operators to see the visualize data without delay.

The mobile application is limited to Android, but it was developed using the Xamarin extension in Visual Studio to make it easier to eventually achieve cross-platform support. Development of the mobile application features AD verification of the user when logging in. This gives high security and permits Skagerak to have more control over the access to the app. The user has more information available, presented in a simple manner as a map with markers. The app grants the user a faster and simpler procedure for registering jobs, where the user only needs to select a station and a timeframe.

The whole system is functional; however, it has some issues that can be improved with further development.

References

[1] Shubham Jain, *Overview of Common Language Infrastructure*, 2017. Retrieved from: https://www.c-sharpcorner.com/blogs/overview-of-common-language-infrastructure , Downloaded: 09.05.2018.

[2] Xamarin, *Xamarin*, 2018. Retrieved from: https://www.xamarin.com/platform , Downloaded: 09.05.2018.

[3] Armut Kale, *Xamarin*, 2016. Retrieved from: http://varahitechnologies.com/xamarin/ , Downloaded: 09.05.2018.

[4] J.S.Miller, S.Ragsdal, *The Common Language Infrastructure Annotated Standard*, U.S: Pearson Education, 2004. [Online]. Retrieved from: https://books.google.no/books?hl=no&lr=&id=50PhgS8vjhwC&oi=fnd&pg=PR21&dq=common+language+infrastructure&ots=v-Gy_iraqU&sig=9zt_2qmTrP8uHIo5oFKtR6M_Jz4&redir_esc=y#v=onepage&q=common%20language%20infrastructure&f=false , Downloaded: 09.05.2018.

[5] *Common Language Infrastructure (CLI) Partitions I to VI*, ECMA-335, 2012. [Online]: https://www.ecma-international.org/publications/standards/Ecma-335.htm

[6] Ecma, *History of Ecma*, 2017. Retrieved from: https://www.ecma-international.org/memento/history.htm , Downloaded: 09.05.2018.

[7] R.Petrusha et al, *Common Language Runtime (CLR) overview*, 2018. Retrieved from: https://docs.microsoft.com/en-us/dotnet/standard/clr , Downloaded: 09.05.2018.

[8] S.Gibilisco and M.Doig, *Machine Code*, 2018. Retrieved from: https://whatis.techtarget.com/definition/machine-code-machine-language , Downloaded: 09.05.2018.

[9] P.Pedro, *Xamarin – PCL vs. Shared Project*, 2017. Retrieved from: https://medium.com/@daRochaPires/xamarin-pcl-vs-shared-project-a838806d5cc6 , Downloaded: 09.05.2018.

[10] J.Pepper, G.Taskos and C.Brilgin, *Xamarin: Cross-Platform Mobile Application Development*, UK: Packt Publishing, 2016. [Online]. Retrieved from: https://books.google.no/books?hl=no&lr=&id=SqfWDQAAQBAJ&oi=fnd&pg=PP1&dq=Xamarin+Portable+Class+library&ots=PqpD-8fjRL&sig=wG-EuEeQOKf0rZ6dReKz9XIIIKE&redir_esc=y#v=onepage&q=Xamarin%20Portable%20Class%20library&f=false , Downloaded: 09.05.2018.

[11] A.Bruns, B.Umbaugh and C.Dunn, *Introduction to Portable Class Libraries*, 2017. Retrieved from: https://docs.microsoft.com/nb-no/xamarin/cross-platform/app-fundamentals/pcl?tabs=vsmac , Downloaded: 09.05.2018.

[12] A.Bruns, B.Umbaugh and C.Dunn, *Shared Projects*, 2017. Retrieved from: https://docs.microsoft.com/nb-no/xamarin/cross-platform/app-fundamentals/shared-projects?tabs=vsmac , Downloaded: 09.05.2018.

[13] Cloud Direct, *An introduction to the Microsoft Azure portal*, 2018. Retrieved from: https://www.clouddirect.net/knowledge-base/a/KB0011450/an-introduction-to-the-microsoft-azure-portal, Downloaded: 08.05.2018.

[14] Microsoft, *Use of Microsoft Copyrighted Content*, 2018. Retrieved from: https://www.microsoft.com/en-us/legal/intellectualproperty/permissions, Downloaded: 08.05.2018.

[15] T. Ng, *What should I do to start a career in Cloud computing especially Azure?*, 2017. Retrieved from: https://www.quora.com/What-should-I-do-to-start-a-career-in-Cloud-computing-especially-Azure, Downloaded: 08.05.2018

[16] Carl Rabeler et al, *Controlling and granting database access*, 2018. Retrieved from: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-manage-logins, Downloaded: 06.05.2018.

[17] M. Henderson et al, *Configure your App Service app to use Azure Active Directory login*, 2018. Retrieved from: https://docs.microsoft.com/en-us/azure/app-service/app-service-mobile-how-to-configure-active-directory-authentication, Downloaded: 08.05.2018.

[18] G. Wallace et al., *Starting an Azure Automation runbook with a webhook*, 2018. Retrieved from: https://docs.microsoft.com/en-us/azure/automation/automation-webhooks, Downloaded: 08.05.2018.

[19] Microsoft, *Scheduler Documentation*, 2018. Retrieved from: https://docs.microsoft.com/en-us/azure/scheduler/, Downloaded: 08.05.2018.

[20] V. Beal, *run book*, 2018. Retrieved from: https://www.webopedia.com/TERM/R/run_book.html, Downloaded: 08.05.2018.

[21] G. Wallace et al., *Azure Automation runbook types*, 2018. Retrieved from: https://docs.microsoft.com/en-us/azure/automation/automation-runbook-types, Downloaded: 08.05.2018.

[22] C. Ng, *Active Directory: difference Between Windows and Azure AD*, 2018. Retrieved from: https://blog.varonis.com/windows-vs-azure-active-directory/, Downloaded: 08.05.2018.

[23] Techterms, *Active Directory*, 2017. Retrieved from: https://techterms.com/definition/active_directory, Downloaded: 08.05.2018.

[24] E. Ross et al., *What is Azure Active Directory?*, 2018. Retrieved from: https://docs.microsoft.com/en-us/azure/active-directory/active-directory-whatis, Downloaded: 08.05.2018.

[25] Adrian Hall, *Why is there a string ID in the data model of Azure Mobile Apps?*, 2016. Retrieved from: https://stackoverflow.com/questions/38231279/why-is-there-a-string-id-in-the-data-model-of-azure-mobile-apps, Downloaded: 09.05.2018.

[26] K. Wenzel, *Learn about Stored Procedures*, 2018. Retrieved from: https://www.essentialsql.com/what-is-a-stored-procedure/ , Downloaded: 10.05.2018.

[27] Tricalyx LLC, *Why use Stored Procedures?*, 2012. Retrieved from: http://mysql-storedprocedure.com/index.php?option=com_content&view=article&id=51&Itemid=40 , Downloaded: 10.05.2018.

[28] R. Anderson, A. Pasic and T. Dykstra, *What is Web Forms*, 2014. Retrieved from: https://docs.microsoft.com/en-us/aspnet/web-forms/what-is-web-forms, Downloaded: 09.05.2018.

[29] Ahmed, T. (2013, 31. Jul.). *What is Bootstrap* [Video file]. Retrieved from: https://www.youtube.com/watch?v=V7x_hosDoIo

[30] B. Wagner et al., *Classes (C# Programming Guide)*, 2018. Retrieved from: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/classes, Downloaded: 07.05.2018.

[31] Microsoft, *ASP.NET Session State Overview*, 2018. Retrieved from: https://msdn.microsoft.com/en-us/library/ms178581.aspx, Downloaded: 08.05.2018.

[32] B.Wagner et al, *interface (C# Reference)*, 2015. Retrieved from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface , Downloaded: 09.05.2018.

[33] E. Rosas, *Xamarin Forms – Selecting an Image form the Gallery*, 2018. Retrieved from: https://lalorosas.com/blog/xamarin-forms-selecting-image-from-the-gallery , Downloaded: 09.05.2018.

[34] D.Britch and C.Dunn , *Consuming an Azure Mobile App*, 2016. Retrieved from: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/consuming/azure , Downloaded: 09.05.2018.

[35] T. Myers and R. Shahan, *Query Entities*, 2016. Retrieved from: https://docs.microsoft.com/en-us/rest/api/storageservices/Query-Entities?redirectedfrom=MSDN , Downloaded: 06.05.18

[36] Microsoft, *MobileServiceTable.skip function*, 2013. Retrieved from: https://msdn.microsoft.com/en-us/library/azure/jj613355.aspx . Downloaded: 06.05.18.

[37] Spatial Reference, *EPSG:25832*, 2017. Retrieved from: http://spatialreference.org/ref/epsg/25832/ , Downloaded: 07.05.18.

[38] Spatial Reference, *SR-ORG:7483*, 2017. Retrieved from: http://spatialreference.org/ref/sr-org/epsg3857-wgs84-web-mercator-auxiliary-sphere/. Downloaded: 07.05.18.

[39] Engineering ToolBox, *UTM to Latitude and Longitude Converter*, 2008. Retrieved from: https://www.engineeringtoolbox.com/utm-latitude-longitude-d_1370.html , Downloaded: 06.05.18

[40] T. Tonning, *UTM to Latitude and Longitude Converter*, 2004. Retrieved from: http://www.mesterkart.no/Ordliste.htm#UTM-sone , Downloaded: 06.05.18

[41] user2548538, *Java, Convert lat/lon to UTM*, 2015. Retrieved from: https://stackoverflow.com/questions/176137/java-convert-lat-lon-to-utm , Downloaded: 27.03.18

[42] J. Montemagno, *GeolocatorPlugin*, 2017. Retrieved from: https://jamesmontemagno.github.io/GeolocatorPlugin/ , Downloaded: 07.03.18

[43] Microsoft, *Microsoft.IdentityModel.Clients.ActiveDirectory*, 2018. Retrieved from: https://www.nuget.org/packages/Microsoft.IdentityModel.Clients.ActiveDirectory/ , Downloaded: 09.05.2018.

[44] D.Britch et al, *Introduction to DependencyService*, 2017. Retrieved from: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction , Downloaded: 09.05.2018.

[45] M. Tendulkar, *Put Some Azure Active Directory in Xamarin.Forms*, 2015. Retrieved from: https://blog.xamarin.com/put-adal-xamarin-forms/ , Downloaded: 09.05.2018.

[46] S.Akhter et al, *Azure Active Directory Authentication Libraries*, 2018. Retrieved from: https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-authentication-libraries , Downloaded: 09.05.2018.

[47] Microsoft, *Access Tokens*, 2018. Retrieved from: https://msdn.microsoft.com/en-us/library/windows/desktop/aa374909%28v=vs.85%29.aspx?f=255&MSPPError=-2147217396 , Downloaded: 09.05.2018.

[48] Microsoft, *Uri Class*, 2018. Retrieved from: https://msdn.microsoft.com/en-us/library/system.uri(v=vs.110).aspx , Downloaded: 09.05.2018.

[49] Swagger, *Bearer Authentication*, 2018. Retrieved from: https://swagger.io/docs/specification/authentication/bearer-authentication/ , Downloaded: 09.05.2018.

[50] Microsoft, *HttpStatusCode Enumeration*, 2018. Retrieved from: https://msdn.microsoft.com/en-us/library/system.net.httpstatuscode%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396 , Downloaded: 09.05.2018.

[51] J. Newton-King, *Newtonsoft.Json*, 2018. Retrieved from: https://www.nuget.org/packages/newtonsoft.json/ , Downloaded: 09.05.2018.

[52] Json, *Introducing JSON*, 2018. Retrieved from: https://www.json.org/ , Downloaded: 09.05.2018.

[53] Microsoft, *Tuple Class*, 2018. Retrieved from: https://msdn.microsoft.com/en-us/library/system.tuple(v=vs.110).aspx , Downloaded: 09.05.2018.

[54] J. Montemagno, *Geolocation plugin for Xamarin and Windows*, 2018. Hentet fra: https://github.com/jamesmontemagno/GeolocatorPlugin , Downloaded: 19.02.18

[55] T. Kruse, *TK.CustomMap.nuspec*, 2018. Retrieved from: https://github.com/TorbenK/TK.CustomMap/blob/Development/nuget/TK.CustomMap.nuspec , Downloaded: 20.04.18

[56] T. Kruse, *TK.CustomMap Extended Map Control*, 2018. Retrieved from: http://torbenk.github.io/TK.CustomMap/ , Downloaded: 20.03.18

[57] Software Testing Fundamentals, *Unit Testing*, 2018. Retrieved from: http://softwaretestingfundamentals.com/unit-testing/ , Downloaded: 10.05.2018.

[58] S. C. Roy, *What is Integration Testing and How It is Performed?*, 2018. Retrieved from: https://www.softwaretestinghelp.com/what-is-integration-testing/ , Downloaded: 10.05.2018.

[59] Software Testing Fundamentals, *Integration Testing*, 2018. Retrieved from: http://softwaretestingfundamentals.com/integration-testing/ , Downloaded: 10.05.2018.

[60] Software Testing Class., *System Testing: What? Why? & How?*, 2018. Retrieved from: http://www.softwaretestingclass.com/system-testing-what-why-how/, Downloaded: 07.05.2018.

[61] Software Testing Fundamentals, *System Testing*, 2018. Retrieved from: http://softwaretestingfundamentals.com/system-testing/ , Downloaded: 10.05.2018.

[62] Software Testing Fundamentals, *What is integration Testing and How It is Performed?*, 2018. Retrieved from: http://softwaretestingfundamentals.com/acceptance-testing/ , Downloaded: 10.05.2018.

[63] O. Swart, *Cloud vs. Hosted Services, what's the difference?*, 2011. Retrieved from: http://www.itnewsafrica.com/2011/04/cloud-vs-hosted-services/, Downloaded: 08.05.2018

[64] Custom Information Services, *The Difference Between Hosted and Cloud Computing for ERP Software*, 2012. Retrieved from: http://www.erpsoftwareblog.com/2012/08/the-difference-between-hosted-and-cloud-computing-for-erp-software/, Downloaded: 08.05.2018

[65] A. B. Newman, *Is a Cloud-based Solution the Same Thing as a Hosted Solution?*, 2013. Retrieved from: https://www.mitel.com/blog/cloud-based-solution-same-thing-hosted-solution, Downloaded: 08.05.2018

[66] M. Rouse, *pay-as-you-go cloud computing (PAYG cloud computing)*, 2015. Retrieved from: https://searchstorage.techtarget.com/definition/pay-as-you-go-cloud-computing-PAYG-cloud-computing, Downloaded: 08.05.2018

[67] Aleksandra Majkic, *Mobile Cross Platform Becoming Feasible*, 2016. Retrieved from: https://www.execom.eu/blog/post/mobile-cross-platform-becoming-feasible , Downloaded: 09.05.2018.

[68] Cordova, *Apache Cordova*, 2018. Retrieved from: https://cordova.apache.org/ , Downloaded: 09.05.2018.

[69] SQA, *The Normalisation Process*, 2008. Retrieved from: https://www.sqa.org.uk/e-learning/SoftDevRDS02CD/page_11.htm , Downloaded: 09.05.2018.

[70] Microsoft, *Learn About ASP.NET SignalR*, 2018. Retrieved from: https://www.asp.net/signalr , Downloaded: 09.05.2018.

Appendices

Appendix A Project Assignment

Appendix B Project Goals

Appendix C WBS

Appendix D Gantt

Appendix E Requirements

Appendix F Concept Design

Appendix G Web Fundamentals

Appendix H User Manual

Appendix I Test Case Web

Appendix J Test Case Mobile

Appendix K Improved Database

Appendix A Project Assignment

University College of Southeast Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

PRH612 Bachelor's Thesis

Title: **Innmelding i anlegg**

HSN supervisor: Hans-Petter Halvorsen

External partner: Marius Dolven, Skagerak Energi AS

Task background:
Det viktigste ansvaret til nettselskapet er å sikre montørenes sikkerhet i felt. Man ønsker derfor å se på digitale verktøy som montørene kan benytte for å sikre seg mot ulykker og få bedre informasjon ute i felt.

Task description:
I dag, når montører skal inn i en nettstasjon må de ringe inn til driftssentralen for å melde ifra, slik at man påser at anlegget er spenningsløst. Her ønsker vi en selvbetjent løsning hvor montør selv melder ifra om dette elektronisk. Løsningen må bygges slik at den sikrer at HMS krav er tilfredsstilt og at både kundesenter og driftssentral blir varslet.
Mulige løsninger vil være en app som varsler montør når han er i nærheten av en nettstasjon for å sikre at innmelding blir registret evnt. en SMS løsning. Det vil også være nødvendig å se på integrasjoner mot fagsystemer. Studentene vil måtte levere et løsningsforslag på mulige løsninger i samråd med Skagerak og utvikle en applikasjon for å utføre innmeldingen i anlegget.

Student category: IA

Practical arrangements:
Det er ønskelig at studentgruppa sitter regelmessig i Skagerak Energi sine lokaler, men ikke noe krav.

Signatures:

Students (date and signature):

Supervisor (date and signature):

Address: Kjølnes ring 56, NO-3918 Porsgrunn, Norway. **Phone:** 35 57 50 00. **Fax:** 35 55 75 47.

Project Goals

Primary goals

The main goals in this project are to reduce the communication errors and increase HSE for technician, who's working in an electrical substation. A modern solution that are faster, simpler and user-friendly. The system should contain a website for displaying information, a database to store data, and a mobile application for creating tasks that notifies the website. The system should also feature sign in method for authentication.

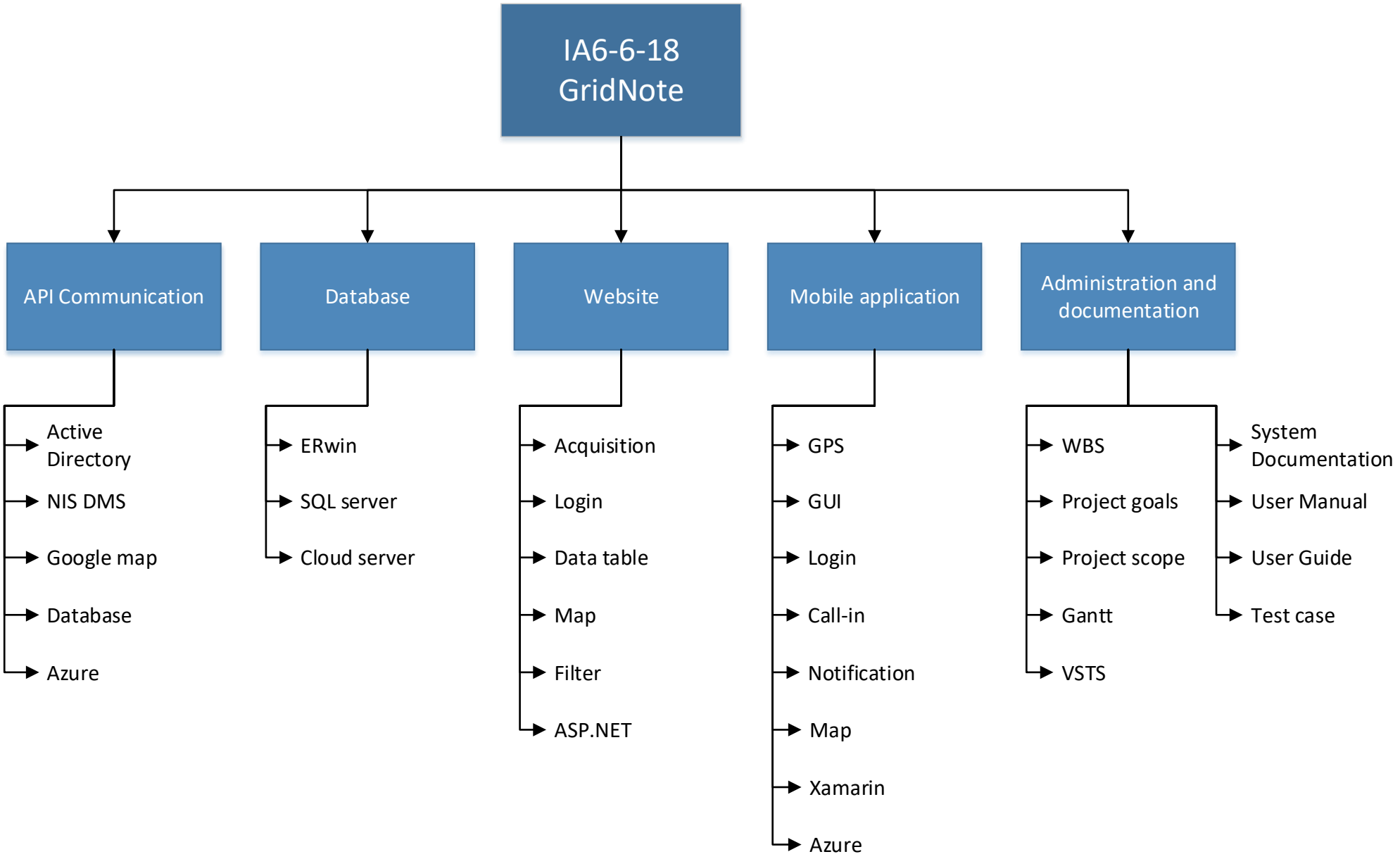
The mobile application should at least support one platform.

Competence goals

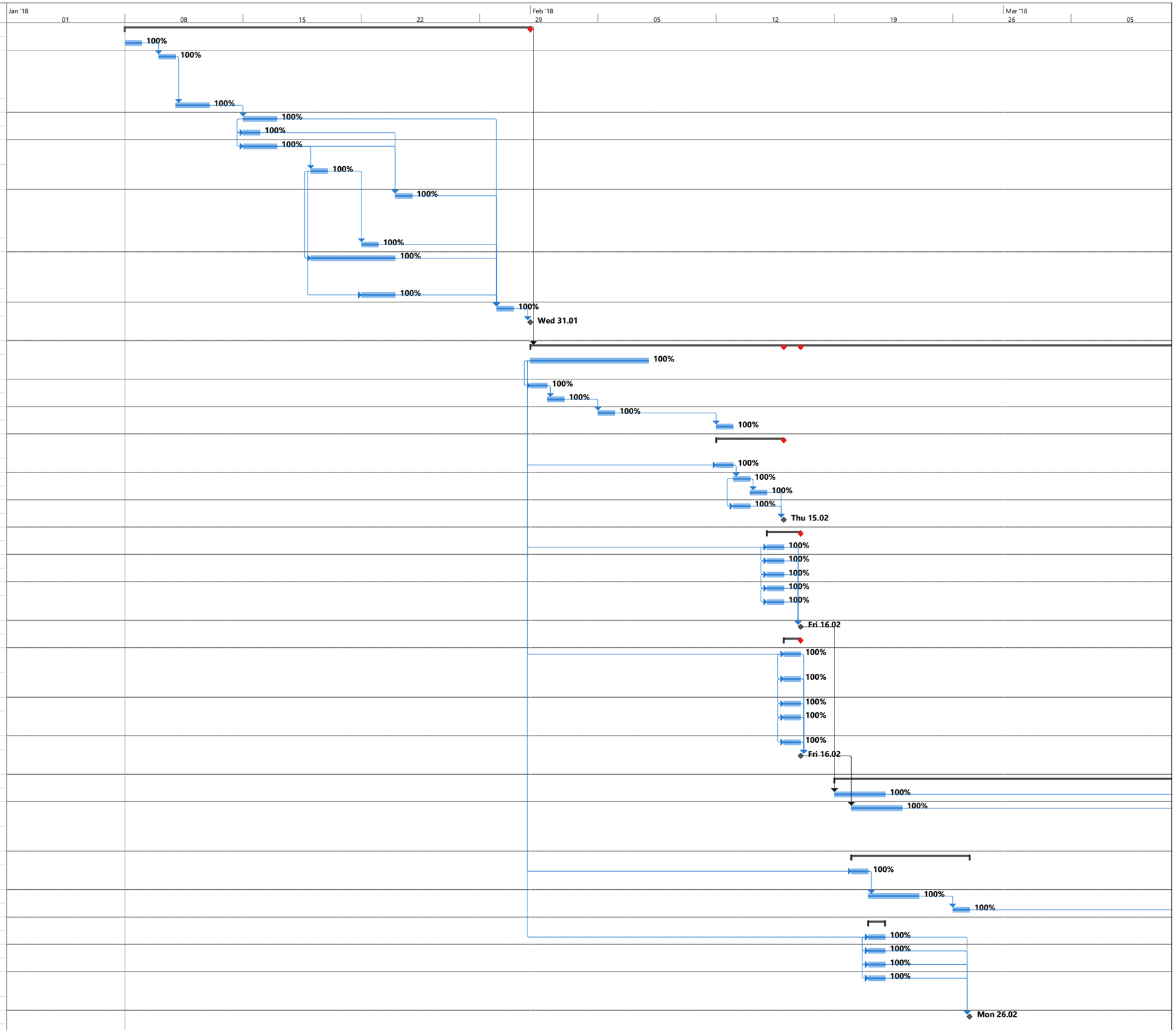
The experience this group should gain from this project, would be to increase their knowledge in both .NET framework and mobile application development. They would also get a better insight of the management tool scrum.

This group must also understand the existing system and provide a solution for potential improvement. This requires the group to have fundamental knowledge in current technological development and basic programming. They must also consider the database structure and its vulnerability and design a user-friendly graphical user interface for both operators and technicians.

Appendix C WBS

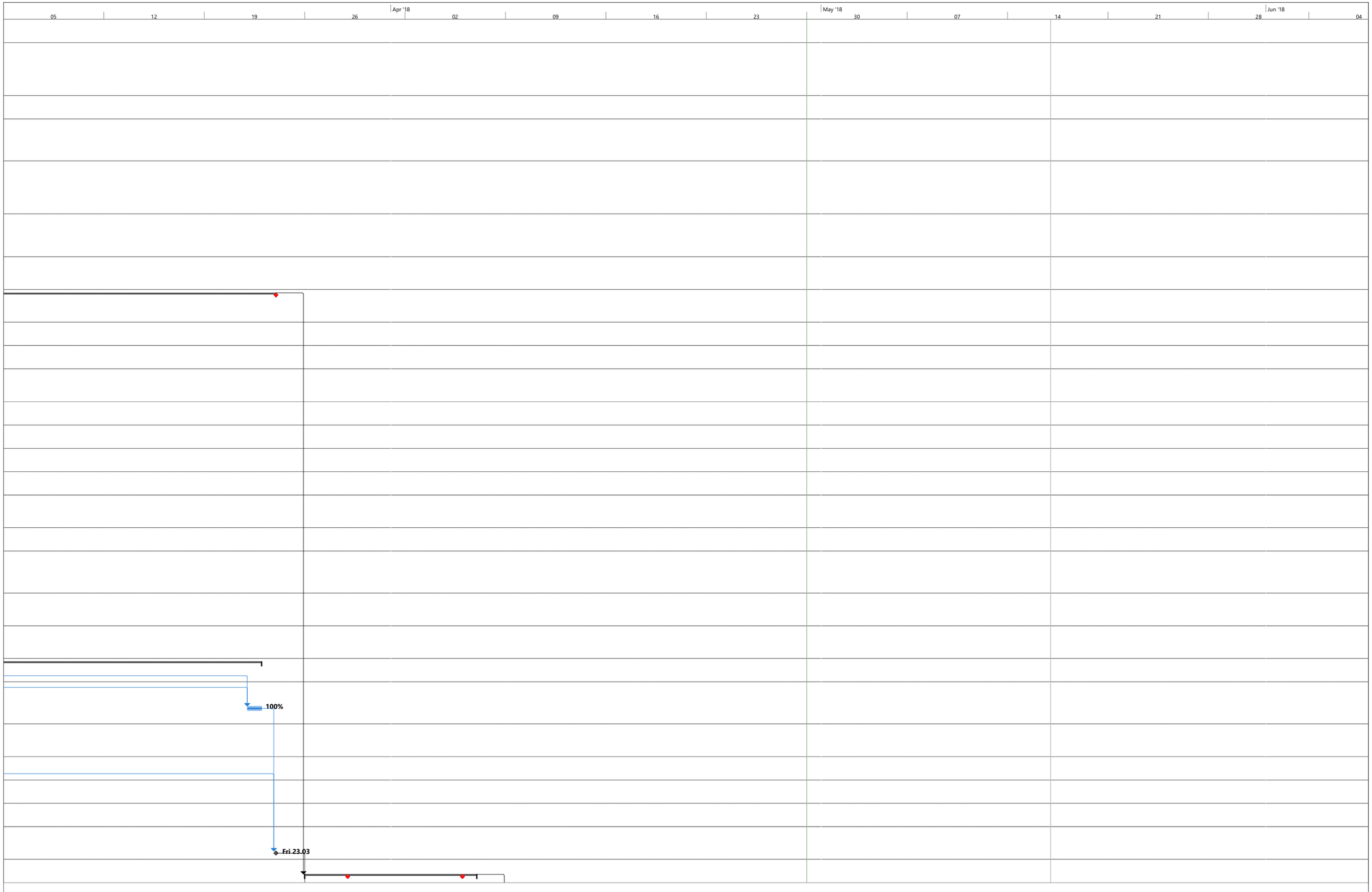


ID	Task Mode	Task Name	Duration	Start	Finish	% Complete
1		**Planning Phase**	**18 days**	**Mon 08.01.18**	**Wed 31.01.18**	**100%**
2		Group agreement	1 day	Mon 08.01.18	Mon 08.01.18	100%
3		Obtain requirements from Skagerak Energi	1 day	Wed 10.01.18	Wed 10.01.18	100%
4		Brainstorm	2 days	Thu 11.01.18	Fri 12.01.18	100%
5		WBS	2 days	Mon 15.01.18	Tue 16.01.18	100%
6		Project goals	1 day	Mon 15.01.18	Mon 15.01.18	100%
7		Research question	2 days	Mon 15.01.18	Tue 16.01.18	100%
8		Meeting at Skagerak Energi	1 day	Fri 19.01.18	Fri 19.01.18	100%
9		Approval of Project goal and Research question	1 day	Wed 24.01.18	Wed 24.01.18	100%
10		UML	1 day	Mon 22.01.18	Mon 22.01.18	100%
11		Software Development Plan	3 days	Fri 19.01.18	Tue 23.01.18	100%
12		Report draft	2 days	Mon 22.01.18	Tue 23.01.18	100%
13		1.Meeting	1 day	Tue 30.01.18	Tue 30.01.18	100%
14		Planning Phase completed	1 day	Wed 31.01.18	Wed 31.01.18	100%
15		**Alpha**	**37 days?**	**Thu 01.02.18**	**Fri 23.03.18**	**100%**
16		System requirement & design plan	5 days	Thu 01.02.18	Wed 07.02.18	100%
17		User manual	1 day	Thu 01.02.18	Thu 01.02.18	100%
18		Test Plan	1 day?	Fri 02.02.18	Fri 02.02.18	100%
19		Test documentation	1 day?	Mon 05.02.18	Mon 05.02.18	100%
20		API Requirements	1 day?	Mon 12.02.18	Mon 12.02.18	100%
21		**Database architecture**	**4 days?**	**Mon 12.02.18**	**Thu 15.02.18**	**100%**
22		Erwin	1 day?	Mon 12.02.18	Mon 12.02.18	100%
23		SQL	1 day?	Tue 13.02.18	Tue 13.02.18	100%
24		SQL query	1 day?	Wed 14.02.18	Wed 14.02.18	100%
25		Azure database	1 day?	Tue 13.02.18	Tue 13.02.18	100%
26		Database operating	1 day	Thu 15.02.18	Thu 15.02.18	100%
27		**UI design plan**	**2 days?**	**Thu 15.02.18**	**Fri 16.02.18**	**100%**
28		Login page	1 day?	Thu 15.02.18	Thu 15.02.18	100%
29		Home page	1 day?	Thu 15.02.18	Thu 15.02.18	100%
30		Map page	1 day?	Thu 15.02.18	Thu 15.02.18	100%
31		Call-in page	1 day?	Thu 15.02.18	Thu 15.02.18	100%
32		Preferences page	1 day?	Thu 15.02.18	Thu 15.02.18	100%
33		Plan for user interface	1 day	Fri 16.02.18	Fri 16.02.18	100%
34		**UX plan**	**1 day?**	**Fri 16.02.18**	**Fri 16.02.18**	**100%**
35		Information architecture	1 day?	Fri 16.02.18	Fri 16.02.18	100%
36		Functional analyse	1 day?	Fri 16.02.18	Fri 16.02.18	100%
37		Human factors	1 day?	Fri 16.02.18	Fri 16.02.18	100%
38		Human-computer interaction	1 day?	Fri 16.02.18	Fri 16.02.18	100%
39		Scenarios	1 day?	Fri 16.02.18	Fri 16.02.18	100%
40		Plan for User experience	0 days	Fri 16.02.18	Fri 16.02.18	100%
41		**Mobile application**	**24 days?**	**Mon 19.02.18**	**Thu 22.03.18**	**100%**
42		Create page from UI	3 days	Mon 19.02.18	Wed 21.02.18	100%
43		Create functions from UX	3 days	Tue 20.02.18	Thu 22.02.18	100%
44		Deploy alpha version of mobile app	1 day?	Thu 22.03.18	Thu 22.03.18	100%
45		**Website**	**5 days?**	**Tue 20.02.18**	**Mon 26.02.18**	**100%**
46		Host website from Azure	1 day?	Tue 20.02.18	Tue 20.02.18	100%
47		UI	3 days	Wed 21.02.18	Fri 23.02.18	100%
48		Deploy alpha version	1 day?	Mon 26.02.18	Mon 26.02.18	100%
49		**Update documents**	**1 day?**	**Wed 21.02.18**	**Wed 21.02.18**	**100%**
50		Report	1 day?	Wed 21.02.18	Wed 21.02.18	100%
51		WBS	1 day?	Wed 21.02.18	Wed 21.02.18	100%
52		UML	1 day?	Wed 21.02.18	Wed 21.02.18	100%
53		System development plan	1 day?	Wed 21.02.18	Wed 21.02.18	100%
54		Alpha version is deployed	1 day	Fri 23.03.18	Fri 23.03.18	100%
55		2.Meeting	1 day?	Mon 26.02.18	Mon 26.02.18	100%
56		**Beta**	**10 days**	**Mon 26.03.18**	**Fri 06.04.18**	**100%**



Project: Grid note
Date: Mon 30.04.18

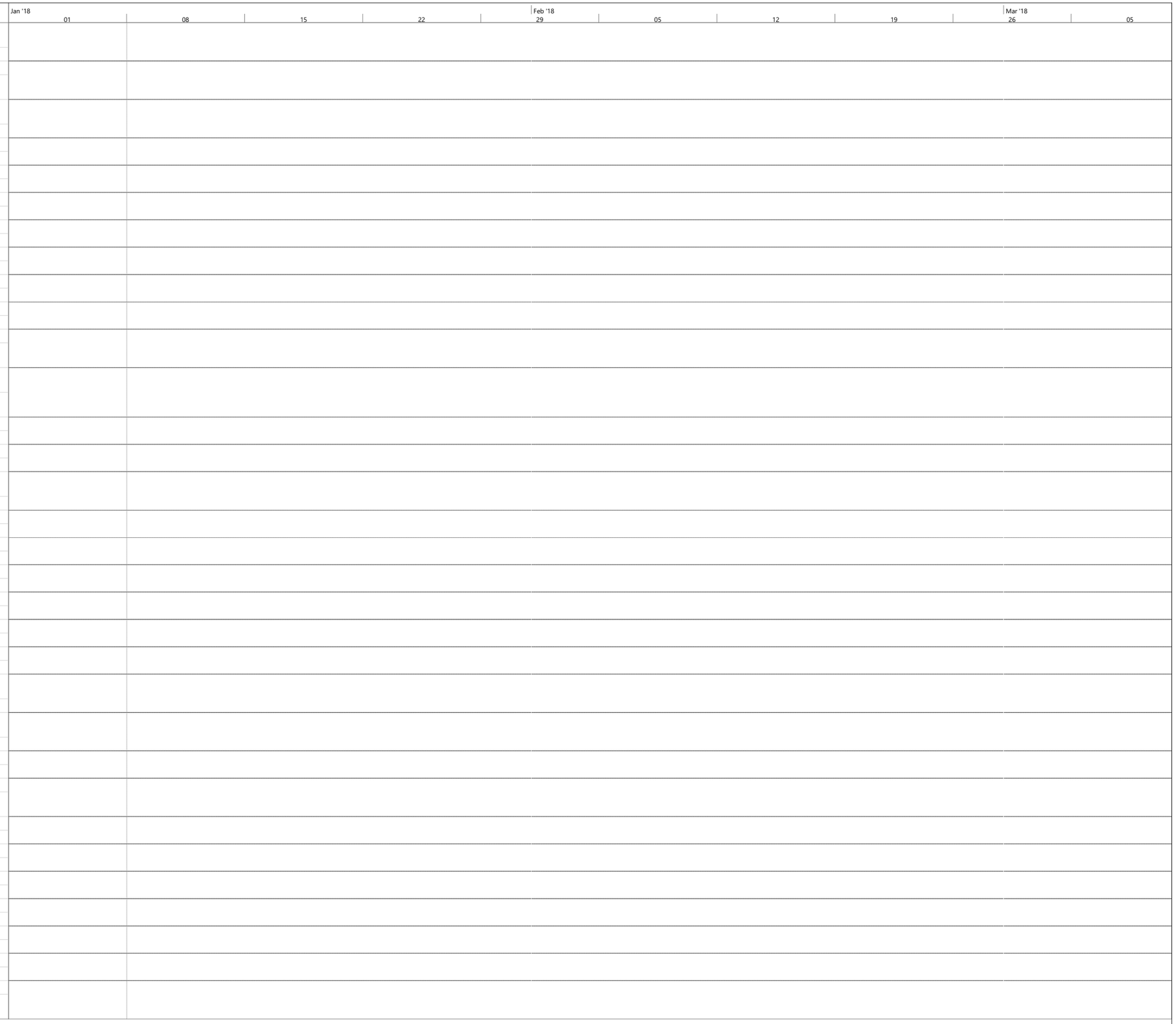
Task | Split | Milestone | Summary | Project Summary | Inactive Task | Inactive Milestone | Inactive Summary | Manual Task | Duration-only | Manual Summary Rollup | Manual Summary | Start-only | Finish-only | External Tasks | External Milestone | Deadline | Progress | Manual Progress



Project: Grid note
Date: Mon 30.04.18

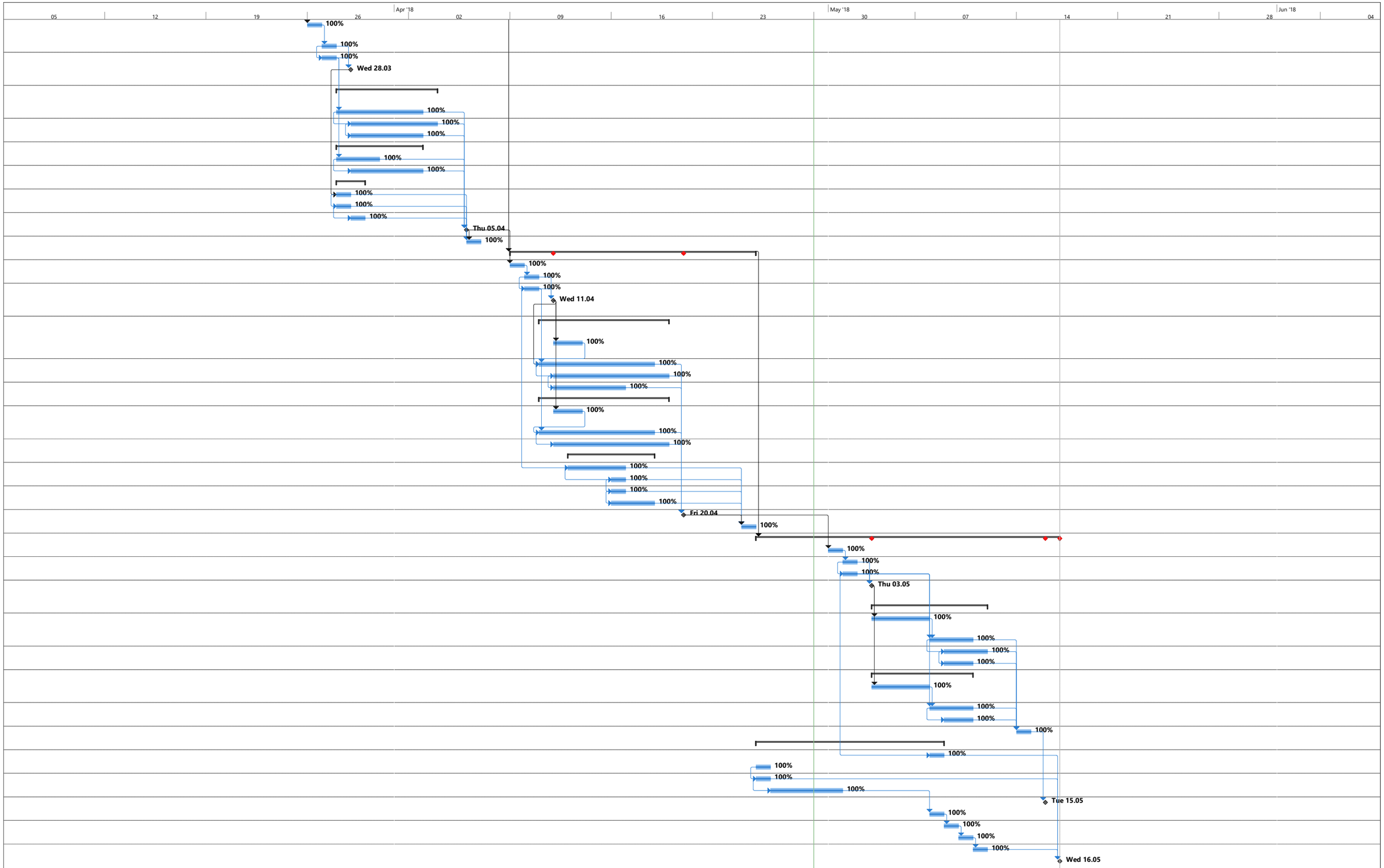
Task	Milestone	Project Summary	Inactive Milestone	Manual Task	Manual Summary Rollup	Start-only	External Tasks	Deadline	Manual Progress
Split	Summary	Inactive Task	Inactive Summary	Duration-only	Manual Summary	Finish-only	External Milestone	Progress	

ID	Task Mode	Task Name	Duration	Start	Finish	% Complete
57		Review from alpha release	1 day	Mon 26.03.18	Mon 26.03.18	100%
58		Test plan	1 day	Tue 27.03.18	Tue 27.03.18	100%
59		Test documentation	1 day	Tue 27.03.18	Tue 27.03.18	100%
60		Reevalueate and Improved UI and UX	1 day	Wed 28.03.18	Wed 28.03.18	100%
61		**Update mobile application**	**5 days**	**Wed 28.03.18**	**Tue 03.04.18**	**100%**
62		UI	4 days	Wed 28.03.18	Mon 02.04.18	100%
63		UX	4 days	Thu 29.03.18	Tue 03.04.18	100%
64		SQL	3 days	Thu 29.03.18	Mon 02.04.18	100%
65		**Update website**	**4 days**	**Wed 28.03.18**	**Mon 02.04.18**	**100%**
66		UI	3 days	Wed 28.03.18	Fri 30.03.18	100%
67		UX	3 days	Thu 29.03.18	Mon 02.04.18	100%
68		**Update documents**	**2 days**	**Wed 28.03.18**	**Thu 29.03.18**	**100%**
69		WBS	1 day	Wed 28.03.18	Wed 28.03.18	100%
70		UML	1 day	Wed 28.03.18	Wed 28.03.18	100%
71		Report	1 day	Thu 29.03.18	Thu 29.03.18	100%
72		Deploy beta version	1 day	Thu 05.04.18	Thu 05.04.18	100%
73		3.Meeting	1 day	Fri 06.04.18	Fri 06.04.18	100%
74		**RC**	**13 days?**	**Mon 09.04.18**	**Wed 25.04.18**	**100%**
75		Review from beta release	1 day	Mon 09.04.18	Mon 09.04.18	100%
76		Test plan	1 day	Tue 10.04.18	Tue 10.04.18	100%
77		Test documentation	1 day	Tue 10.04.18	Tue 10.04.18	100%
78		Reevalueate and Improved UI and UX	1 day	Wed 11.04.18	Wed 11.04.18	100%
79		**Update mobile application**	**7 days**	**Wed 11.04.18**	**Thu 19.04.18**	**100%**
80		Active Directory connection	2 days	Thu 12.04.18	Fri 13.04.18	100%
81		UI	6 days	Wed 11.04.18	Wed 18.04.18	100%
82		UX	6 days	Thu 12.04.18	Thu 19.04.18	100%
83		SQL	3 days	Thu 12.04.18	Mon 16.04.18	100%
84		**Update website**	**7 days**	**Wed 11.04.18**	**Thu 19.04.18**	**100%**
85		Active Directory connection	2 days	Thu 12.04.18	Fri 13.04.18	100%
86		UI	6 days	Wed 11.04.18	Wed 18.04.18	100%
87		UX	6 days	Thu 12.04.18	Thu 19.04.18	100%
88		**Update documents**	**4 days**	**Fri 13.04.18**	**Wed 18.04.18**	**100%**
89		User manual	2 days	Fri 13.04.18	Mon 16.04.18	100%
90		WBS	1 day	Mon 16.04.18	Mon 16.04.18	100%
91		UML	1 day	Mon 16.04.18	Mon 16.04.18	100%
92		Report	3 days	Mon 16.04.18	Wed 18.04.18	100%
93		Deploy RC version	1 day	Fri 20.04.18	Fri 20.04.18	100%
94		4.Meeting	1 day?	Wed 25.04.18	Wed 25.04.18	100%
95		**RTM**	**15 days?**	**Thu 26.04.18**	**Wed 16.05.18**	**100%**
96		Review from RTM release	1 day	Tue 01.05.18	Tue 01.05.18	100%
97		Test plan	1 day	Wed 02.05.18	Wed 02.05.18	100%
98		Test documentation	1 day	Wed 02.05.18	Wed 02.05.18	100%
99		Reevalueate and Improved UI and UX	1 day	Thu 03.05.18	Thu 03.05.18	100%
100		**Update mobile application**	**6 days**	**Fri 04.05.18**	**Fri 11.05.18**	**100%**
101		Active Directory connection	2 days	Fri 04.05.18	Mon 07.05.18	100%
102		UI	3 days	Tue 08.05.18	Thu 10.05.18	100%
103		UX	3 days	Wed 09.05.18	Fri 11.05.18	100%
104		SQL	2 days	Wed 09.05.18	Thu 10.05.18	100%
105		**Update website**	**5 days**	**Fri 04.05.18**	**Thu 10.05.18**	**100%**
106		Active Directory connection	2 days	Fri 04.05.18	Mon 07.05.18	100%
107		UI	3 days	Tue 08.05.18	Thu 10.05.18	100%
108		UX	2 days	Wed 09.05.18	Thu 10.05.18	100%
109		4.Meeting	1 day?	Mon 14.05.18	Mon 14.05.18	100%
110		**Complete documents**	**9 days**	**Thu 26.04.18**	**Tue 08.05.18**	**100%**
111		User manual	1 day	Tue 08.05.18	Tue 08.05.18	100%
112		WBS	1 day	Thu 26.04.18	Thu 26.04.18	100%
113		UML	1 day	Thu 26.04.18	Thu 26.04.18	100%
114		Report	3 days	Fri 27.04.18	Tue 01.05.18	100%
115		Deploy RTM version	1 day	Tue 15.05.18	Tue 15.05.18	100%
116		Report Correction nr.1	1 day?	Tue 08.05.18	Tue 08.05.18	100%
117		Review Report nr.1	1 day?	Wed 09.05.18	Wed 09.05.18	100%
118		Report Correction nr.2	1 day?	Thu 10.05.18	Thu 10.05.18	100%
119		Report Review nr.2	1 day?	Fri 11.05.18	Fri 11.05.18	100%
120		Deliver report with attachment	1 day	Wed 16.05.18	Wed 16.05.18	100%



Project: Grid note
Date: Mon 30.04.18

Task | Milestone | Project Summary | Inactive Milestone | Manual Task | Manual Summary Rollup | Start-only | External Tasks | Deadline | Manual Progress
Split | Summary | Inactive Task | Inactive Summary | Duration-only | Manual Summary | Finish-only | External Milestone | Progress



Project: Grid note
Date: Mon 30.04.18

Task, Split, Milestone, Summary, Project Summary, Inactive Task, Inactive Milestone, Inactive Summary, Manual Task, Duration-only, Manual Summary Rollup, Manual Summary, Start-only, Finish-only, External Tasks, External Milestone, Deadline, Progress, Manual Progress

F: Potential requirement in Future. Not present.

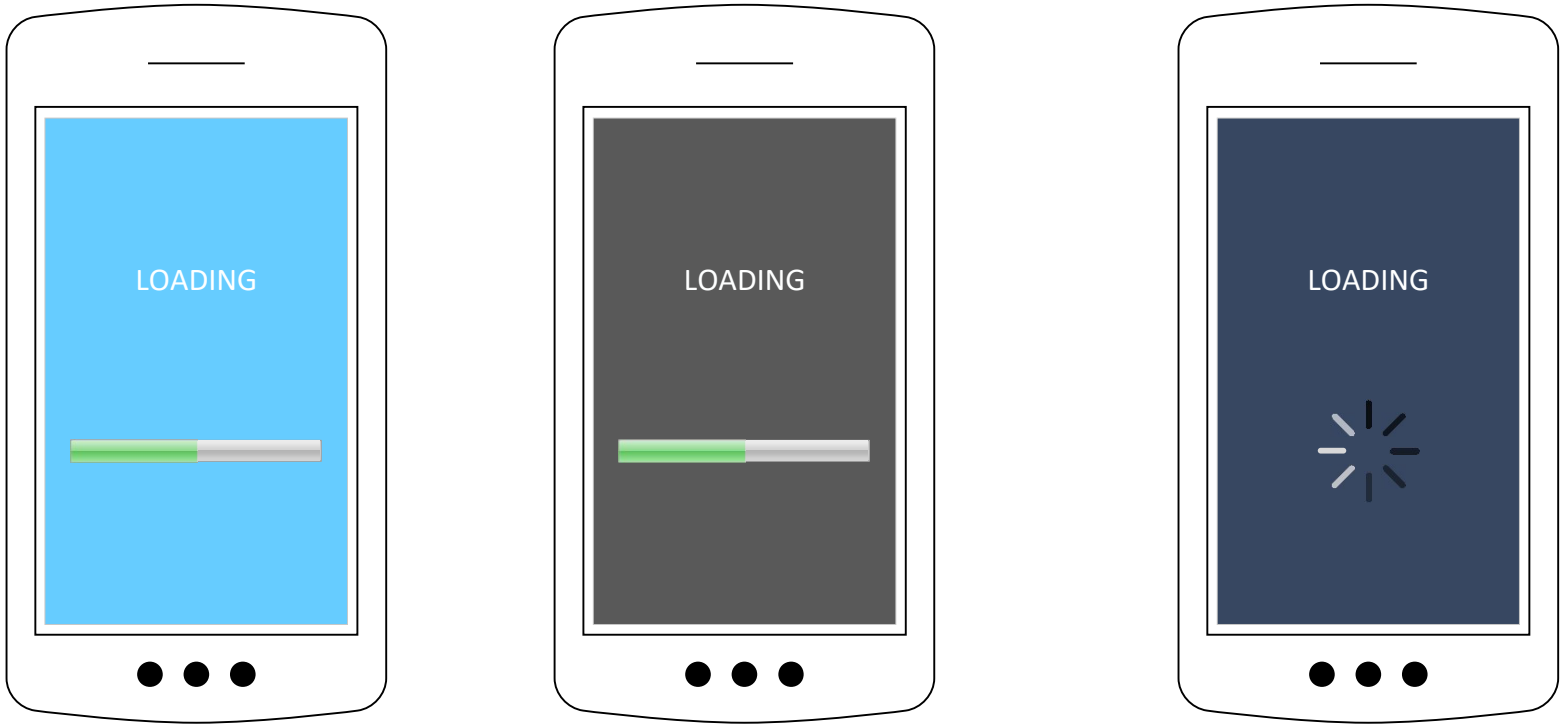
Mobile app			
Req #	**Functional requirement**	**Priority**	**Date Reviewed**
1	Working login/logout	1	26.01.2018
2	Can set start and end date for work	1	26.01.2018
3	Can search for substation	1	26.01.2018
4	Can logout from work	1	26.01.2018
5	Can only work on one substation at a time	1	26.01.2018
6	Can create new work	1	26.01.2018
7	Can communicate with database	1	26.01.2018
8	Working map with stations located	2	26.01.2018
9	Get notification when you are close to a substation	2	26.01.2018
10	Get notification 15 min before work runs out	2	26.01.2018
11	Get notification: Lock the door	2	26.01.2018
12	Can edit end date for a work	2	26.01.2018
13	Can set preferences	2	26.01.2018
14	Can edit substation for a work	2	26.01.2018
15	Get notification: No signal	2	26.01.2018
16	Get notification: No GPS (when creating work)	2	26.01.2018
17	Can add more users for work	F	26.01.2018
18	Can view notification list	F	26.01.2018
19	User can send feedback on mail	F	26.01.2018
Req #	**Non-functional requirement**	**Priority**	**Date Reviewed**
1	Can login in less than 15 seconds	1	26.01.2018
2	Safe encryption on login	1	26.01.2018
3	Less than 1 out of 20 crashes during startup	1	26.01.2018
4	Less than 1 out of 10 crashes during walkthrough	1	26.01.2018
5	Can startup in less than 20 seconds	1	26.01.2018
6	No unnecessary buttons	1	26.01.2018
7	Structured and user-friendly layout	1	26.01.2018
8	Can remember preferences	2	26.01.2018
9	Rotating button for selecting duration	2	26.01.2018
10	Works on Android and iOS	F	26.01.2018
11	English and Norwegian language	F	26.01.2018

Website			
Req #	**Functional requirement**	**Priority**	**Date Reviewed**
1	Working login/logout	1	26.01.2018
2	Table must contain: name, number, station, start time, end time, status	1	26.01.2018
3	Status will indicate if the endtime has run out	1	26.01.2018
4	Can work with popular web browsers (Chrome, Edge, Firefox, IE)	1	26.01.2018
5	Can see historical data	1	26.01.2018
6	Can add a work row manually	F	26.01.2018
7	Can export table to excel	F	26.01.2018
8	Can delete work row from table	F	26.01.2018
9	Can expand work row information	F	26.01.2018
10	Can chat with technicians	F	26.01.2018
Req #	**Non-functional requirement**	**Priority**	**Date Reviewed**
1	Can login in less than 15 seconds	1	26.01.2018
2	Safe encryption on login	1	26.01.2018
3	Less than 1 out of 20 crashes during startup	1	26.01.2018
4	Less than 1 out of 10 crashes during walkthrough	1	26.01.2018
5	Can startup in less than 20 seconds	1	26.01.2018
6	No unnecessary buttons	1	26.01.2018
7	Structured and user-friendly layout	1	26.01.2018
8	Can remember settings	2	26.01.2018
9	English and Norwegian language	F	26.01.2018

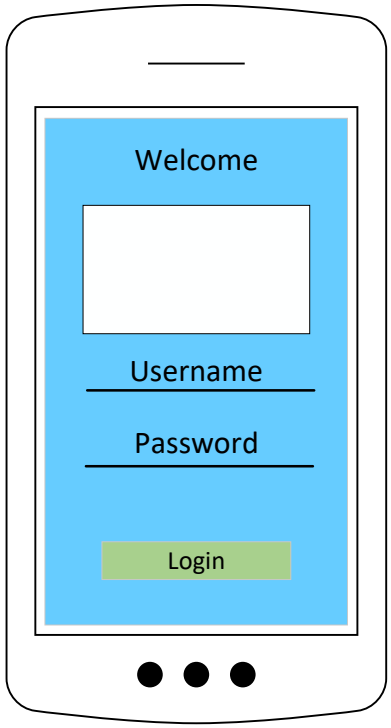
Appendix F Concept Design

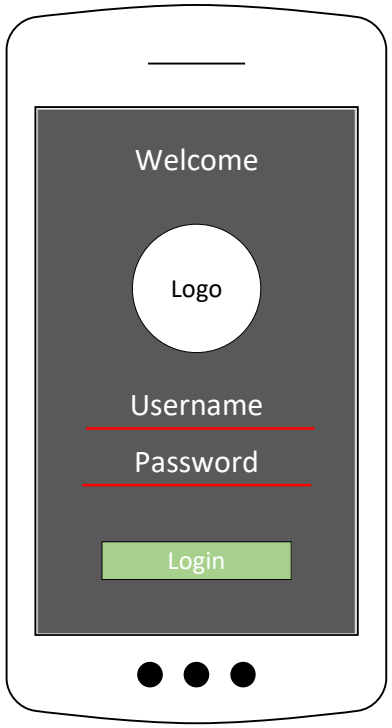
Concept UI design for mobile application

Loading page



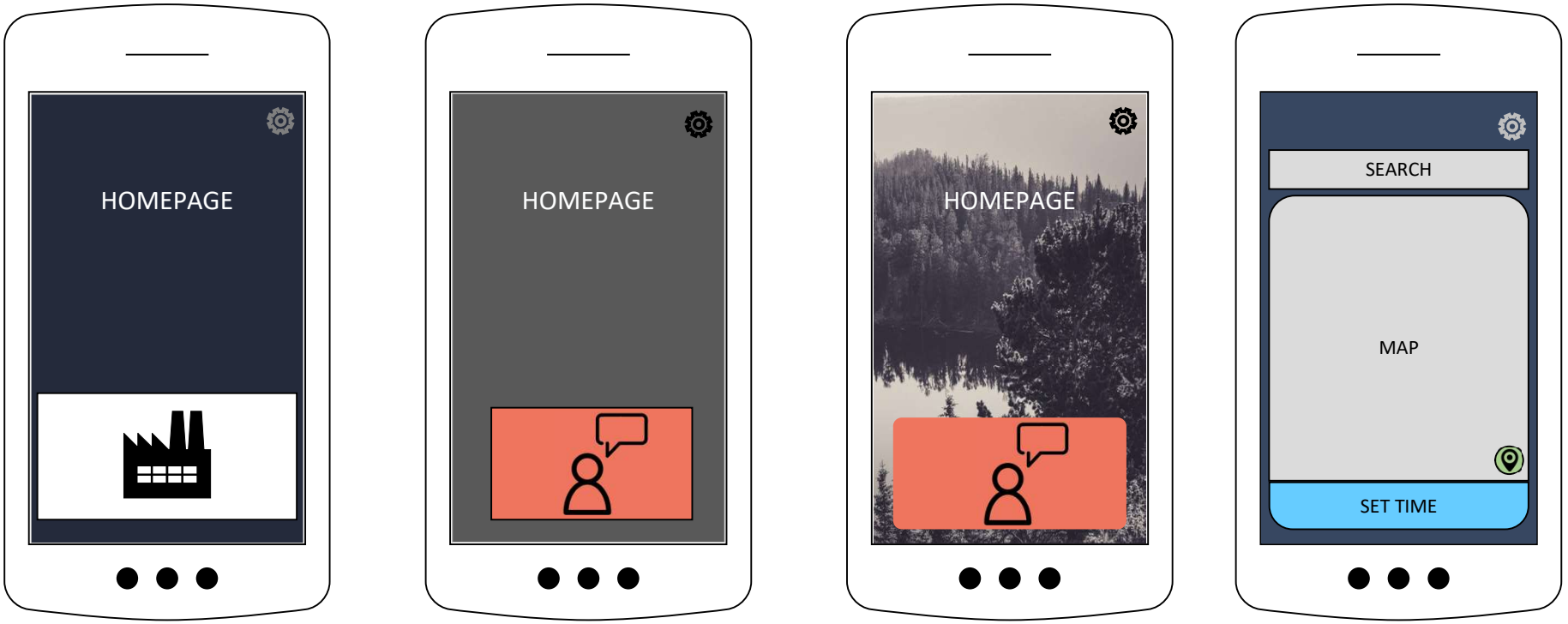
Login page


Welcome
Username
Password
Login


Welcome
Logo
Username
Password
Login


Logo
Username
Password
Login

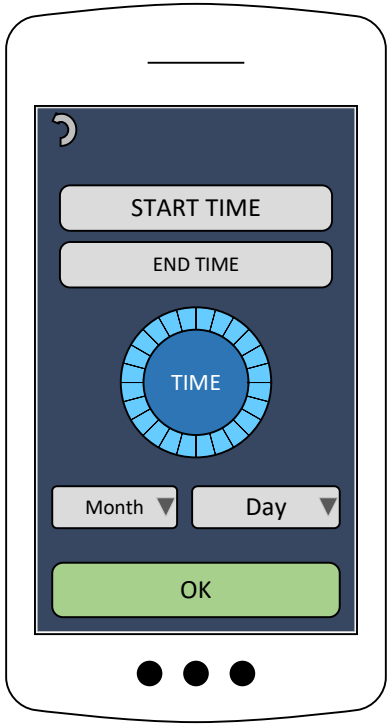
Home page



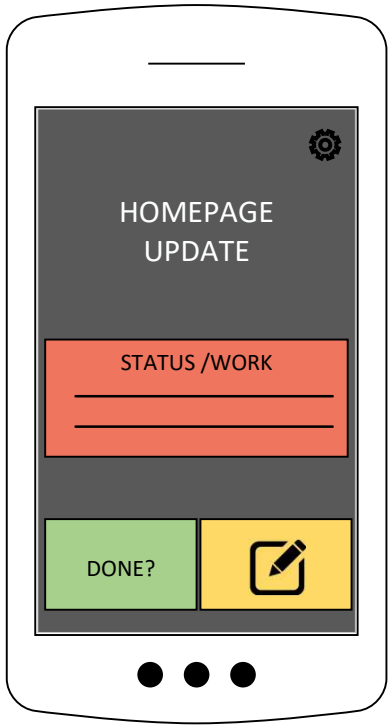
Time page

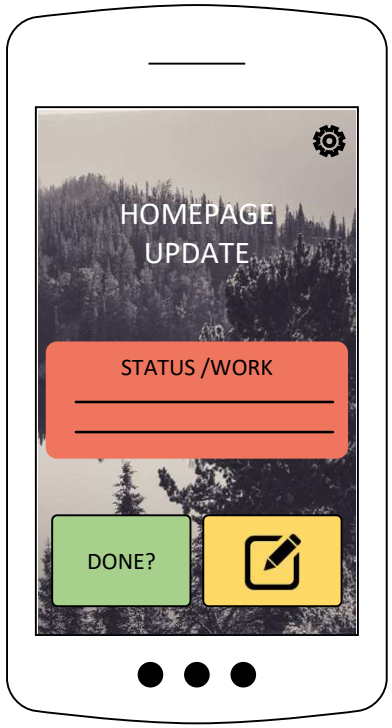
TIME PAGE

TIME PAGE



Home page w/ status

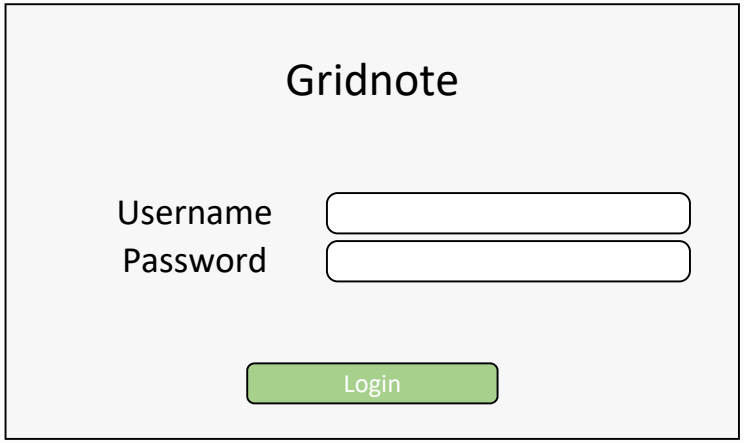




HOMEPAGE
UPDATE

[Logo]

Skagerak Energi



User

[Login time]

Dashboard History Preference

Logout

Dashboard

[System status]

Activities

[Trendline] (View 7 days)

Map

(Overview of stations with active workers)

Active workers

Excel PDF Search Filter

Station name	Fullname	Phone	Date	Start time	Remain time	Status

User

[Login time]

Dashboard History Preference

Logout

History

[System status]

Graph

[Trendline] (Display from quick view)

Quick view **Week** **Month** **Year**

History data

Excel PDF Search Filter

Station name	Fullname	Phone	Date	Start time	Remain time	Status

1 Web Fundamentals

This chapter gives a brief explanation of some of the fundamentals of web development. What will be detailed are the terms front-end and back-end, HTML, CSS and JavaScript.

1.1 Front-end and Back-end

Front-end and back-end are terms used in software engineering referring to two separate parts of a software application. Front-end being the presentation layer, which is what the user sees and interacts with. This is often referred to as the client side, as it is the part of the application that the client operates on. Prime examples being operations involving the user interface.

The back end is the data access layer, which contains all the operations working in the background. It is often called the server side because it is where all server operations take place. Examples being operations involving the database, automated functions and API handling. [1]

1.2 HTML

Hyper Text Mark-up Language (HTML) is a mark-up language that tells your browser how to structure web pages that one visits. HTML is the standard mark-up language for the internet. HTML uses tags that enclose, mark up or wrap letters to make them act a certain way. Each element has a start tag and an end tag, with the content in between. Different tags indicate different ways the content should be presented. In Figure 1-1 we see an example of how html is used to display a page with some text. The “body” start and end tag contains the content that will be shown on the page. The “p” tag is used for paragraphs and contains the tags “b” and “i” making the content **bold** or *italic*. [2]

```
1.  <!DOCTYPE html>
2.  <html>
3.  <body>
4.
5.  <p><b>bold</b></p>
6.
7.  <p><i>italic</i></p>
8.
9.  </body>
10. </html>
```

Figure 1-1 HTML example

1.3 CSS

Cascading Style Sheets (CSS) is used to style the content displayed on a web page. The style sheet contains selectors containing properties that describe how content should be displayed. For example, if a page inherits from a stylesheet that has properties set for the “h1” selector. Any content with a h1 tag on the HTML page will get the properties from that selector in the stylesheet. In Figure 1-2 we see how the text “Hello World!” will get the properties set for selector “h1” in a CSS document shown in Figure 1-3. [3]

```
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Hello World!</h1>
</body>
```

Figure 1-2 HTML referencing CSS

```
h1 {
  color: red;
  background-color: black;
  border: 1px solid black;
}
```

Figure 1-3 CSS Stylesheet

1.4 JavaScript

Client-side JavaScript is used to implement all kinds of features other than just static text. Features like dynamically updating content, multimedia (audio and video), maps, games, all sorts of APIs, running code and so on. For HTML to understand it's reading JavaScript code, one only needs to add the "script" tag. Figure 1-4 shows a typical example with a button that when clicked adds a new paragraph with the text "Button clicked!". [4]

```
<button onclick="createLine()">Click me!</button>

<script>
function createLine() {
  var newLine = document.createElement('p');
  newLine.textContent = 'Button clicked!';
  document.body.appendChild(newLine);
}
</script>
```

Figure 1-4 JavaScript example

[1] Pluralsight, *What's the difference between the Front-End and Back-End?*, 2015. Retrieved from: https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end, Downloaded: 08.05.2018.

[2] C. Mills et al., *Getting started with HTML*, 2018. Retrieved from: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started, Downloaded: 08.05.2018.

[3] Suterj et al., *Learn to style HTML using CSS*, 2017, Retrieved from: https://developer.mozilla.org/en-US/docs/Learn/CSS, Downloaded: 08.05.2018.

[4] C. Mills et al., *JavaScript*, 2017. Retrieved from: https://developer.mozilla.org/enUS/docs/Learn/JavaScript, Downloaded: 08.05.2018.

1 Mobile application – User Manual

This user manual will guide you through the application and show you how you can create, edit and complete a work.

Step 1.

When the app is open you need to sign in before using the application Figure 1-1. Press "PÅLOGGING" button and write your username and password.

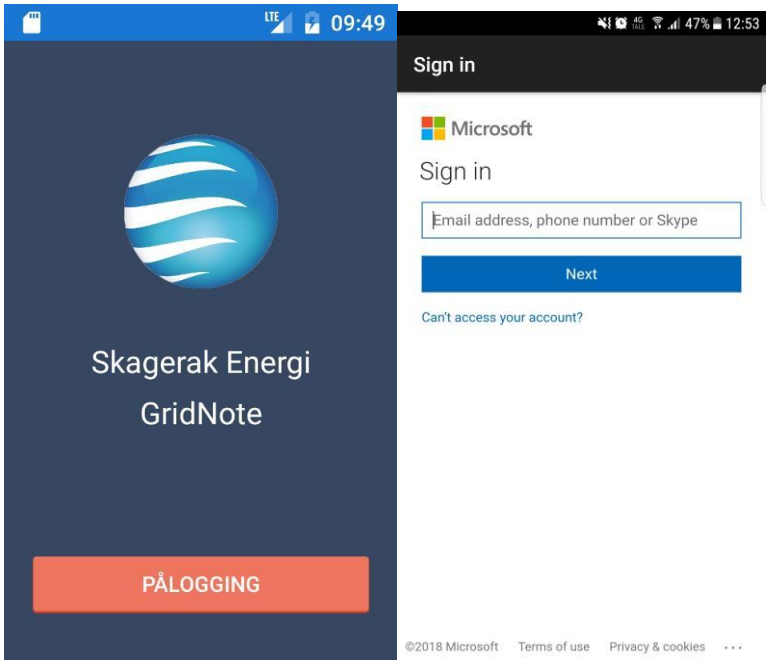
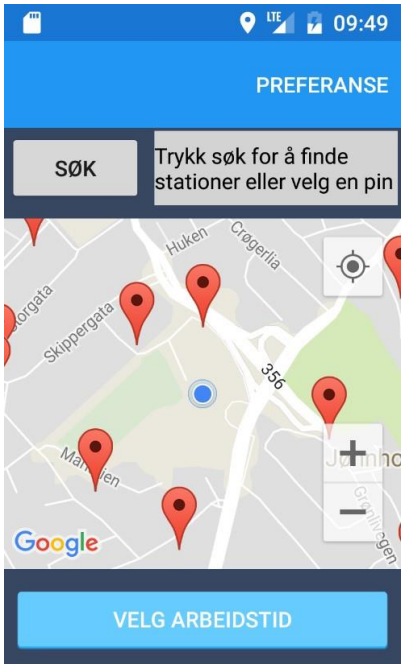


Figure 1-1 Login page

Step 2.

After you logged in you will come to the homepage, Figure 1-2. Here can you select the station you want to work at or you can search for it. Press the station you want to work at (red mark) and then select “VELG ARBEIDSTID” button to continue (shows in Figure 1-3). When a station is selected the information about it will be shown in a white popup box. If you don’t find the station you are looking for you can press the search button (top left corner).



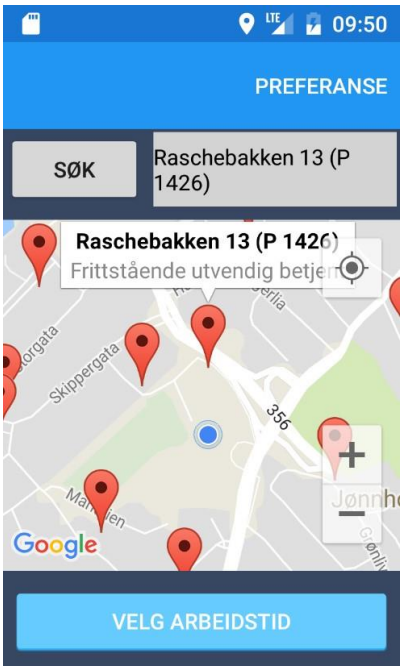


Figure Figure 11--32 HomepageHomepage Figure Figure1-3 1Home page marked-4 Search page

Step 3.

When the search button is pressed, you will be sent to the search page, Figure 1-4. You can search for the station you want using the name or the id of the station. In Figure 1-5 is an example of a search of a station in “Porsgrunn”. Several alternative choices will be listed below. Click the station you want to work at to continue to the time page.

Figure 1-4 Search page

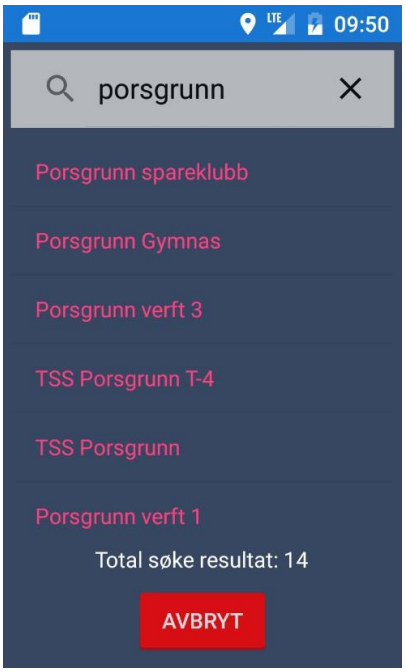


Figure 1-5Figure 1 Search for "Porsgrunn”-5

Step 4.

On this page you will get the option to choose how long you going to work at the station. Click the 00:00 button in the middle of the screen in Figure 1-6. You will get up an android time picker where you can choose your end time, shows in Figure 1-7. Press “TRYKK HER FOR BEKREFTE VALG AV ARBEIDSTID” button to confirm the time, shows in Figure 1-8. Press “START ARBIED” to create the work. After the button is pressed you will be sent to update page.

Figure Figure 11--66 Time pageTime page

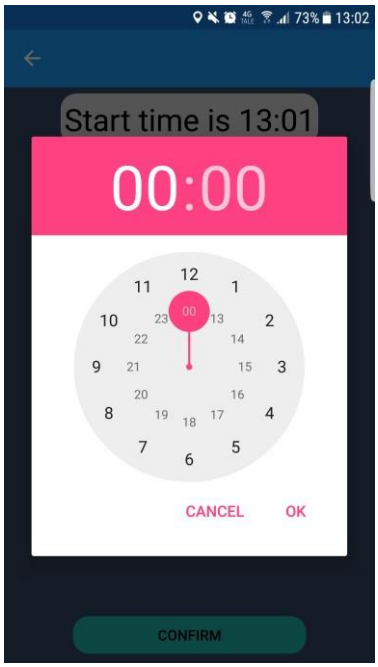


Figure 1-8 Time selected

Figure Figure 11--77 Time pickerTime picker

Step 5.

In the update page, Figure 1-9 you will get an overview about the work you have created. On this page you have two options. One is to complete the work by pressing the “ARBEID FERDIG” button. If you do that you will be sent to Homepage again, the work you have created will be ended. The second options are the “ENDRE” button. When this button is pressed you will be sent to the time page. Here can you edit your old end time, just the way you did last time.

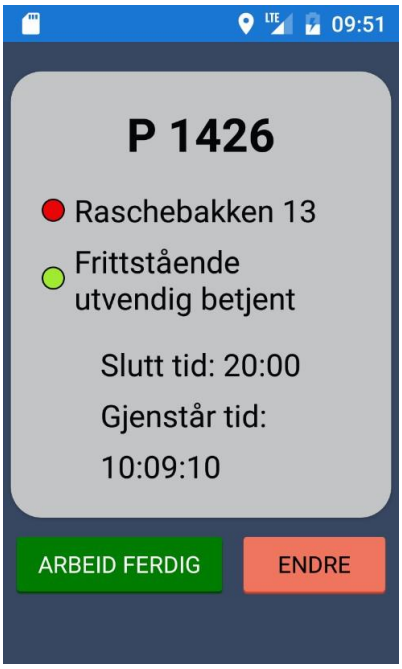


Figure 1Figure 1--99 Update pageUpdate page

2 Website – User Manual

When logged in the front page of the website, Figure 2-1. This is the page that will be shown in the operations centre. This page shows an overview over all technicians that are in a work right now. In the “Søk” textbox it is possible to search in the table to make it easier to find what you are looking for. The text “Aktive” shows how many technicians that are in a work right

now. The status in the table describes if the technicians work has run out of time or not. If status is marked with a cross it means that the technicians time has run out, then the technicians need to edit their time.

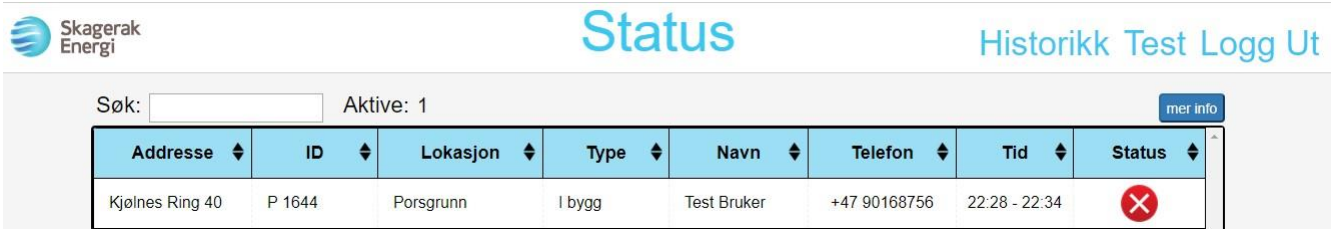


Figure 2-1 Status page website

If “mer info” button is pressed a map will be shown, Figure 2-2. In this map you can see where the technicians work. The green mark is where the time is not run out. Red marks are where the time has run out. On the map you can zoom out and in and move around. The same abilities you have on a google map.

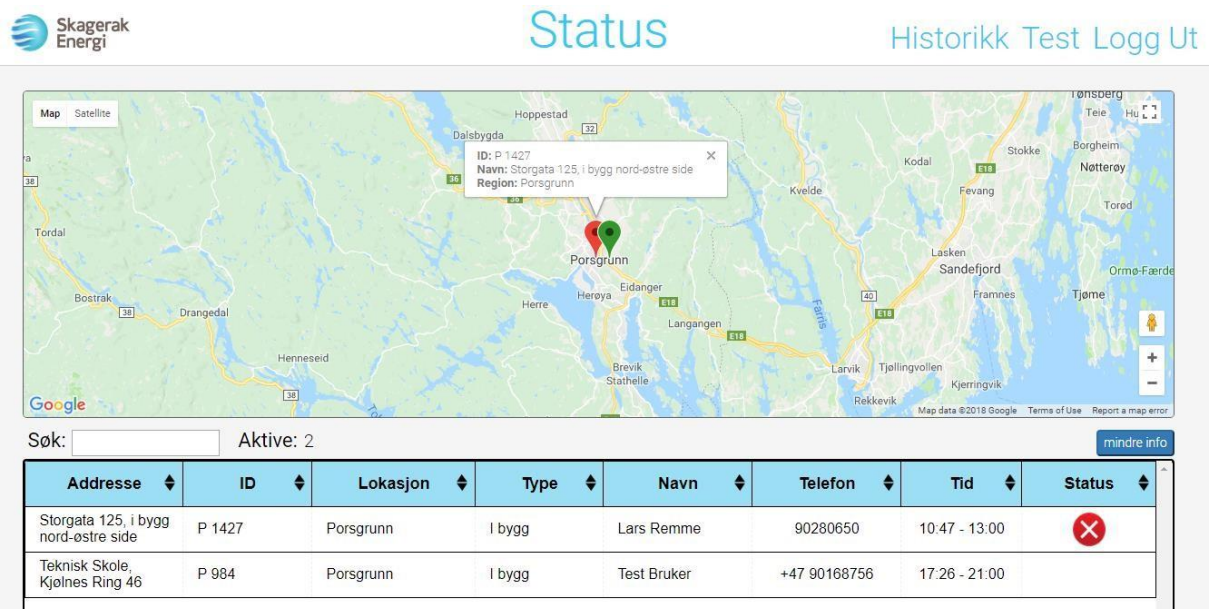


Figure 2-2 Status page with map

On the top right corner, you can press the "Historikk" button that will send you to our history page Figure 2-3. In the history page can you see every work created. You can search after example name, address or date.

Figure 2-3 History page

Test Case Document: Website.

Unit testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
1	void UTM2Deg(double easting, double northing, out double latitude, out double longitude)	19.03.2018	✓	Convert UTM to degrees.	1
2	string GetWorkQuantityHistory()	19.03.2018	✓	Count every work with status = 2.	1
3	void GetPoints(string, out List<double>, out List<double>, out List<string>, out List<string> , out List<string> , out List<string>)	19.03.2018	✓	Get map pins on what you have search on.	1
4	void btnMoreInfo_Click(object sender, EventArgs e)	19.03.2018	✓	Show/ hide map.	1

Integration testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
5	Loging from one user to another?	19.03.2018	✓	its possible to log from one user to another.	1
6	Can add a work row manually	19.03.2018	X	Potential requirement in future.	4
7	Can delete work row from table	19.03.2018	X	Potential requirement in future.	4
8	Can export table to excel	19.03.2018	X	Potential requirement in future.	4
9	Can expand work row information	20.03.2018	X	Potential requirement in future.	4
10	Logout	20.03.2018	✓	User will be forget and you will need to login again.	1
11	Login	19.03.2018	✓	Login using AD works.	1
12	Can you see historic data?	20.03.2018	✓	Historic data can be seen on Historic page. Status = 2.	2
13	Does the connection to the database work?	20.03.2018	✓	Connection with SQL works.	1
14	Does the connection to Azure works	20.03.2018	✓	Yes.	1
15	Working map with stations located?	20.03.2018	✓	Yes.	1
16	Are you able to see a work on the website?	20.03.2018	✓	Yes.	4
17	Can remember settings	20.03.2018	X	Potential requirement in future.	3
18	Table must contain: name, number, station, start time, end time, status	20.03.2018	✓	Needed more coloums: ID, location and type.	1
19	Status will indicate if the endtime has run out	20.03.2018	✓	Azure will change the status when endtime run out.	2
20	Search in table	20.03.2018	✓	Can search for name, station , Id, type, phonenumber, location and date.	1
21	Login on multiple device	21.03.2018	✓	No problem.	1
22	English and Norwegian language	21.03.2018	X	Potential requirement in future.	4
23	Can chat with technicians	20.03.2018	X	Potential requirement in future.	4
24	Is station pins seen on map	20.03.2018	✓	Works where status = 1 or 3 will be seen on the map.	1
25	Login twice?	20.03.2018	✓	you cant login twice.	1

System testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
26	Does the program work in different web browsers?	21.03.2018	X	Prefer Chrome. Requirement in future.	3
27	Does the application freeze	21.03.2018	✓	No.	1
28	No unnecessary buttons	21.03.2018	✓	All button have a function.	1
29	Structured and user-friendly layout	21.03.2018	✓	Yes.	1
30	does every tool work?	21.03.2018	✓	Every tool works.	1

Acceptance Testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
31	Can login in less than 15 seconds	21.03.2018	✓	Yes.	2
32	Less than 1 out of 20 crashes during startup	21.03.2018	✓	Tested 0/20 crashes.	1
33	Less than 1 out of 10 crashes during walkthrough	21.03.2018	✓	Tested 0/10 crashes.	2
34	Can startup in less than 20 seconds	21.03.2018	✓	No problem with startup.	1
35	Is the user data secure?	21.03.2018	✓	Yes AD login make sure of that.	1

Test Case Document mobile app

Unit testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
1	async Task<string> returnId()	22.03.2018	✓	Id from the work created will be returned.	1
2	void extractList(List<STATION> stations, ref List<string>localstation)	22.03.2018	✓	No comments	1
3	async void CompletedButtonOnClicked(object sender, EventArgs e)	22.03.2018	✓	Status in work table will be 2 when button clicked.	1
4	async Task GetListOfStations()	22.03.2018	✓	No comments	1
5	async void CheckUserHasWork()	22.03.2018	✓	No comments	1
6	void UserMap_PinSelected(object sender, TKGenericEventArgs<TKCustomMapPin> e)	22.03.2018	✓	No comments	1

Integration testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
7	Change work time	22.03.2018	✓	You can edit work time after created a work.	1
8	Loging from one user to another?	22.03.2018	✓	Works fine.	1
9	Logout, forgetting information correctly?	22.03.2018	✓	Information will be forgot.	1
10	Does map updates when user moves	22.03.2018	✓	The map will be updated whil the user moves.	2
11	Does the connection to Azure works	23.03.2018	✓	Yes.	1
12	Does the connection to the database work?	23.03.2018	✓	Yes.	1
13	Can create new work?	23.03.2018	✓	New work will be saved in the database.	1
14	Login	23.03.2018	✓	Login using AD from skagerak.	1
15	Logout	23.03.2018	✓	Logout works.	1
16	Get notification 15 min before work runs out	23.03.2018	✓	You will get notification 5 min before works runs out.	3
17	End work	23.03.2018	✓	The status will change when you end a work.	1
18	Search for station	23.03.2018	✓	Search for all station in the database.	1
19	Can set end date for work	23.03.2018	✓	After station is select. End date can med selected.	1
20	Get notification: Lock the door	23.03.2018	✓	When complete button is clicked a notification will pop up.	3
21	Are you able to see the map	23.03.2018	✓	The map is shown in homepage.	1
22	Is station pins seen on map	23.03.2018	✓	Yes.	1
23	Do you get notification when you are close to a substation?	23.03.2018	X	Potential requirement in future.	3
24	Can set preferences	23.03.2018	X	Potential requirement in future.	3
25	Can add more users for work	23.03.2018	X	Potential requirement in future.	4
26	Can view notification list	23.03.2018	X	Potential requirement in future.	4
27	User can send feedback on mail	23.03.2018	X	Potential requirement in future.	4
28	Get notification: No signal	23.03.2018	X	Potential requirement in future.	3
29	Get notification: No GPS (when creating work)	23.03.2018	X	Potential requirement in future.	3
30	Can only work on one substation at a time?	23.03.2018	✓	You need to end your work before you start a new one.	1
31	Working map with stations located?	23.03.2018	✓	Yes.	1
32	Can remember preferences	23.03.2018	X	Potential requirement in future.	3
33	Rotating button for selecting duration	23.03.2018	X	Antoher time selecting button has been used insted.	3
34	Login on multiple device	26.03.2018	✓	No problem.	1
35	English and Norwegian language	26.03.2018	X	Potential requirement in future.	4
36	Does it work on mobile device	26.03.2018	✓	Yes.	1
37	If there is work, can it be show on other device	26.03.2018	✓	Yes.	1
38	Login twice?	22.03.2018	✓	Can't login twice.	1
39	Start and end date works?	22.03.2018	✓	Right start and end date will be saved in the database.	1

System testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
40	does every tool work?	26.03.2018	X	Preference tools dosent work. Requirement in the future.	1
41	Does the application freeze	26.03.2018	✓	No.	1
42	Works on Android and iOS	26.03.2018	X	Only works on android. Works on IOS is a requirement in the future.	4
43	No unnecessary buttons	26.03.2018	X	Buttons on preference page dosent work. Requirement in the future.	1

Acceptance Testing					
No.	Test Case:	Date	Test Status (✓ , X)	Comment	Priority (1,2,3, 4)
44	Structured and user-friendly layout	26.03.2018	✓	Yes.	1
45	Can login in less than 15 seconds	26.03.2018	✓	Yes.	2
46	Less than 1 out of 20 crashes during startup	26.03.2018	✓	Need to accept wifi and location on app before start.	1
47	Less than 1 out of 10 crashes during walkthrough	26.03.2018	✓	No problems during walkthrough.	1
48	Can startup in less than 20 seconds	26.03.2018	✓	Yes.	2
49	Is the user data secure?	26.03.2018	✓	The app is secure with a login page.	1

Appendix K Improved Database