

Buer, Amund
Eriksen, Stig Hope
Aanstad, Tobias

Design and implementation of a mobile app for in-situ recommendations related to rig heave during offshore oil well drilling

Master's thesis in Cybernetics and Robotics
Supervisor: Ole Morten Aamo
June 2022

Buer, Amund
Eriksen, Stig Hope
Aanstad, Tobias

Design and implementation of a mobile app for in-situ recommendations related to rig heave during offshore oil well drilling

Master's thesis in Cybernetics and Robotics
Supervisor: Ole Morten Aamo
June 2022

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

Design and implementation of
a mobile app for in-situ recommendations
related to rig heave during offshore oil well drilling

Amund Buer
Stig Hope Eriksen
Tobias Aanstad

June 9, 2022

TTK4900 Master Thesis
Supervisor: Prof. Ole Morten Aamo



NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Engineering Cybernetics

Abstract

This MSc thesis documents the development process of improving the existing proof of concept app developed by the group during specialization projects for fall 2021. At the beginning of the project, a new app specification was formed in cooperation with Heavelock employees. Two main areas focused on were giving the app a more professional appearance and adding functionality for estimating operation progress.

The app, which is developed in React Native, implements functionality from the HeaveSIM website. It provides recommendations for drilling operations based on user inputs, where every recommendation is accompanied by new graphical representations. Additionally, the app has a new feature that estimates the operation progress based on a weather forecast. Multiple measures were taken to achieve an overall more professional appearance. The implementations and results are presented throughout this MSc thesis.

Requirements with the highest priority have been implemented. However, not all proposed requirements in the app specification were implemented for various reasons, mainly deprecated libraries and time limitations. At the end of this MSc thesis, ideas for future work that can improve the app are presented.

Sammendrag

Denne masteroppgaven dokumenterer utviklingsprosessen for å forbedre den eksisterende konseptbevis-appen utviklet av gruppen i fordypningsprosjekter høsten 2021. Under prosjektets begynnelse ble en ny appspesifikasjon formulert med hjelp fra ansatte i Heavelock Solutions. To hovedområder som har stått i fokus er å gi appen et mer profesjonelt utseende og å legge til funksjonalitet for å estimere progresjonen i en boreoperasjon.

Appen, utviklet i React Native, implementerer funksjonalitet som finnes på HeaveSIM sin nettside. Den gir anbefalinger for boreoperasjoner basert på brukerinput, hvor alle anbefalinger er fremvist grafisk. I tillegg har appen en ny funksjonalitet der den estimerer fremgang på operasjoner på grunnlag av værvarsel. Flere tiltak ble utført for å oppnå et mer profesjonelt utseende. Implementasjon i appen og hvordan appen til slutt så ut blir presentert i løpet av denne rapporten.

Spesifikasjonskravene med høyest prioritering har blitt implementert. Derimot har ikke alle foreslåtte spesifikasjonskrav blitt implementert av ulike årsaker, hovedsaklig utdaterte biblioteker og tidsbegrensning. Avslutningsvis blir det fremmet ideer for videre arbeid som kan forbedre appen.

Preface

We would like to thank our supervisor professor Ole Morten Aamo at the Norwegian University of Science and Technology (NTNU), for great support and guidance. We would also like to thank Dmitri Gorski and Martin Kvernland from Heavelock Solutions for great help and feedback during the thesis. Thanks to Heavelock Solutions in general for letting us develop this app for them, and thereby creating this thesis.

We want to thank Sverre Hendseth at the Norwegian University of Science and Technology (NTNU), for taking the time to give us some tips and recommendations regarding writing a thesis containing software development.

Thanks to the Norwegian University of Science and Technology (NTNU) for borrowing us computers used in the development, and providing us with a fixed place we could stay the whole project period.

Lastly we would like to thank each other for the cooperation throughout this MSc thesis. With our limited knowledge of app development when starting, it have been helpful to have each other for consultation and support.

Trondheim June 9, 2022

Amund Buer Stig Hope Eriksen Tobias Aanstad

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem Description	1
1.3	Background	2
1.3.1	The Heave Problem	2
1.3.2	HeaveSIM	3
1.3.3	Imagined Use Case	4
1.4	Report Scope	4
1.5	Report Outline	4
2	Theory	6
2.1	Development Tools	6
2.1.1	React Native	6
2.1.2	Expo	6
2.1.3	React Native Packages and Libraries	7
2.2	Code & Software Principles	9
2.2.1	Code Quality	9
2.2.2	Code Runtime	9
2.2.3	REST API	10
2.2.4	Cookies	10
2.2.5	State Persistence	10
2.2.6	AsyncStorage	10
2.3	Interpolation	10
2.4	Color Theory	11
3	Earlier Work	12
3.1	API Calls	12
3.2	Screens	12
3.3	Desired Functionality and Future Work	14
4	Method	15
4.1	Workflow	15
4.1.1	Learning App Development	15
4.1.2	Choice of Software Framework	15
4.2	App Specification	16
4.3	Early Design	18
4.4	Acquiring Weather Forecasts	19
4.5	Attempted Features	19
4.6	Deployment	21
5	Implementation	22
5.1	Code quality	22
5.2	Styles	22
5.3	Colors	23
5.4	Units & Conversions	24
5.5	Settings	25
5.5.1	Setting Units & Color Scheme	25
5.6	Indication of Loading Data	25
5.7	Dynamic Display	26

5.7.1	Dynamic Wells	26
5.7.2	Dynamic Analyses	27
5.8	Error Handling	29
5.9	Input Handling	30
5.10	Recommendation	31
5.11	Plots	33
5.11.1	Recommended Heave per Depth Plot	33
5.11.2	Pressure per Depth Plot	36
5.11.3	Forecast Connection Plot	38
6	Result	42
6.1	Infrastructure	42
6.2	Login Screen	43
6.3	Home Screen	45
6.4	Settings Screen	46
6.5	Wells & Analyses Screens	46
6.6	Connection Recommendations Screen	47
6.6.1	Tab 1: Heave	48
6.6.2	Tab 2: Forecast	49
6.7	Forecast Plot Screen	50
6.8	Tripping Recommendations Screen	51
6.8.1	Tab 1: Heave	51
6.8.2	Tab 2: Trip Speed	52
6.9	The App on Different Platforms	53
7	Discussion	55
7.1	Infrastructure	55
7.2	Features Throughout the App	55
7.3	Login Screen	56
7.4	Home Screen	56
7.5	Settings Screen	56
7.6	Wells & Analyses Screens	57
7.7	Recommendation Screens & Tabs	57
7.8	Forecast	60
7.9	Cross Platform	61
7.10	Code Quality	61
7.11	Resolved App Specification	62
8	Conclusion	63
8.1	Further work	63

Bibliography

Appendices

A Problem description handed out

B Guide to download the app

C Considered Tasks with comments

List of Figures

1	Flowchart HeaveSIM	3
2	React Native Example	7
3	<i>Animated</i> Prefixes Added to <i>View</i> Component	8
4	Victory Native Example	9
5	Screens Made Previously (1/2)	13
7	Plots in HeaveSIM	19
8	Forecast Early Design	20
9	Old and New Status Bars	23
10	<i>ActivityIndicator</i> on iOS and Android	26
11	<i>Button</i> replaced by <i>ActivityIndicator</i>	26
12	Section Containing a Well	27
13	Section Containing an Analysis	28
14	Error Message Examples	29
15	Modal of Invalid Input	30
16	Numerical Keyboards on Different Platforms	31
17	Examples of Recommendations for Heave	33
18	Examples of Recommended Heave per Depth Plots	34
19	Unsegmented vs Segmented plots	35
20	Examples of Pressure per Depth Plots	37
21	Different Pressure Plots at Depth: 3700	38
22	Example of Generated Tide	39
23	Forecast Plot	40
24	Illustration of App's Infrastructure	43
25	Login screen	44
26	Home screen	45
27	Settings screen	46
28	Wells & Analyses screens	47
29	Connections Recommendations screen	48
30	Recommendation Connections Heave Tab	48
31	Forecast Connection Input Screen	50
32	Forecast Plot	51
33	Tripping Recommendations screen	51
34	Recommendation Tripping: Heave Tab	52
35	Recommendation Tripping: Tripping Speed Tab	53
36	The App on Different Platforms	54
37	Comparison of Old and New Input Fields	58
38	Example of Input Dot vs Line placement	59
39	Comparison of Pressure Plots	59
40	Casing plan from HeaveSIM	65

List of Tables

1	Specification: Improvement of existing app	16
2	Specification: New functionality	17
3	Units	24
4	Variables Used for Pressure Plots	36
5	Recommendation Strings for Connection	49
6	Recommendation Strings for Tripping	51

Listings

1	React Native Example code	7
2	Victory Native Example Code Excerpt	9
3	Pseudocode for API Call	12
4	StyleSheet Code Example	23
5	Colors Code Example	24
6	Example Response from the Second API Call	27
7	Code for Populating a Scrollable List with Unique Well Sections	28
8	Function for Sorting Analyses	28
9	Example of Recommendation Data for Depth	31
10	Algorithm for Computing Recommendation: Case 4	32
11	Interpolation Function	33
12	Function for Recommendation Heave Plot	34
13	Pseudocode for Generating Line Segments	36
14	Algorithm for Heave Pressure Plot	37
15	Pseudocode for Comparing Heaves in Forecast Plot	41

Abbreviations

API Application Programming Interface

AWS Amazon Web Services

CLI Command Line Interface

CSS Cascading Style Sheet

d-a Double amplitude

EC2 Elastic Compute Cloud

GUI Graphical User Interface

JSON JavaScript Object Notation

NPT Non-Productive Time

OS Operating System

PoC Proof of Concept

PPG Pounds per Gallon

RAO Response Amplitude Operator

REST API Representational State Transfer API

RN React Native

ROP Rate of Penetration

S3 Simple Storage Service

SG Specific gravity

SoC Separation of Concerns

UI User Interface

UX User Experience

Introduction

1.1 Motivation

Previously, the motivation for developing an app was the ease of use and easier accessibility to HeaveSIM. For this project, the focus will be on further developing the proof of concept (PoC) app documented in earlier specialization project reports [1, 2, 3]. The goal is to have an app that can be shown to potential clients, partners, and investors to better demonstrate the usefulness of HeaveSIM.

The main direction taken in this project is to add new functionality to the existing app that aims to reduce Non-Productive Time (NPT) and connected costs by providing recommendations. One particularly interesting functionality is giving recommendations based on weather forecasts. These recommendations target to improve drill work planning, which can lead to a more efficient and safer working environment on floating oil rigs. The use case presented in section 1.3.3 captures more precisely what the new functionality aims to do.

On the market today, there are no similar or equivalent simulators to HeaveSIM. While its market value currently is the simulator functionality available through the HeaveSIM website, Heavelock employees expect the app to have an increased value as the HeaveSIM product matures.

1.2 Problem Description

Following is the problem description handed out to the group, also found in appendix A:

When drilling from a floater, the heaving motion of the floater causes major pressure fluctuations in the well. An important tool for assessing whether the operation can continue or must wait for better weather is a drilling simulator that is tailored to the heave problem. The computational engine and a web-based user interface (HeaveSIM) that is tailored for well construction planning have been developed but are not suitable for providing in-situ recommendations during operation. In this MSc thesis work, the objective is to design and implement a mobile application (iOS and Android) that provides such recommendations based on the current weather and stage of operation. In addition, the application should offer a short-term plan of recommendations based on weather forecasts – for instance, expected operability in terms of predicted heave for the next 24 hours. The following tasks should be performed by the student:

1. Background: Brief general description of the heave problem in offshore drilling from a floating (this can be brief and point to references since it has been described many times before).
2. Background: Brief description of HeaveSIM functionality (this can be brief and taken from project reports from fall 2021).
3. Improve existing and specify new requirements of the app.
 - Improve functionality/user interface implemented in fall 2021 to “production” quality.
 - Specify new functionality and design the graphical appearance of it (the short-term planning part of the app).

- Specify graphical appearance of background information explaining the reason for recommendations (which margin was violated and at which depth?).
4. Implement the app according to specifications in point 3. The app should have “production” quality in terms of appearance and user friendliness.
 5. Write a comprehensive report, describing the technology used, the software structure/design, and functionality of the app. The report should contain a user’s guide.

1.3 Background

This section is meant to give the reader an improved understanding of the problem description. Sections 1.3.1 and 1.3.2 are rewritings from corresponding sections from the specialization project reports [1, 2, 3].

1.3.1 The Heave Problem

As the heave problem is already well-described by Kvernland et. al., this section is kept short and based on their work in [4, 5].

Floating rigs are prone to movement caused by external factors such as weather and ocean waves. Vertical movement of the rig due to such factors is called heave. When drilling from a floating rig, the drill string will normally be disconnected from the rig and therefore be independent of the heave motion. During some stages of drilling operations, there are situations that require the drill string to be attached to the rig itself. This thesis will focus on two types of drilling operation; *Connection* and *Tripping*. The former operation type is to extend the drill string by adding approximately 30 meters long drill string sections during drilling. The latter operation type is to pull the drill string in or out of the well, for example, when replacing a broken or worn-out drill bit.

Whenever the drill string moves with a floating rig, heave may be a serious problem. Heave compensation systems exist to mitigate some of the heave effects, but the motion may cause the drill string to act as a piston in the well. These movements, called “surge and swab”, cause oscillations in the well pressure. Such oscillations are potentially disastrous, as pressure margins in oil wells can be narrow. If the well pressure does not lie between upper and lower pressure margins, dangerous events such as a stuck drill pipe or blow-outs may occur. These events are not only expensive as they create NPT but also pose a serious threat to the safety of rig workers.

For this reason, having the ability to predict whether the well pressure is within pressure margins or not can be highly helpful for planning and decision-making during drill work. Kvernland et. al. propose a down-hole choking technology meant to prevent some of the aforementioned “surge and swab” effects on the well pressure. In order to verify the results from their study, full-scale simulations were done with a simulator designed specifically for this purpose. The simulator, today known as HeaveSIM, models and simulates well pressure based on a known heave value. Using the simulator, one may find the maximum tripping speed or heave for safely performing a drill operation.

The simulator is a result of an effort carried out by numerous people over a time period of ten years. A selection of additional articles written to cover their work are [6, 7, 8, 9, 10, 11].

1.3.2 HeaveSIM

HeaveSIM is a service provided by Heavelock Solutions that aims to be an important tool in well planning and real-time decision-making during drill work. The web-based user interface(UI) allows users to define wells that replicate the specifications of planned or existing wells. This specification includes a casing plan with information about casings, liners, and bottom-hole assemblies.

Figure 1 shows how HeaveSIM utilizes cloud services provided by Amazon Web Services (AWS) to compute simulations. AWS is a pay-per-use cloud-based server service that offers scalability and several services for specific areas of use. This allows a company like Heavelock Solutions to rent rather than buy server hardware [12].

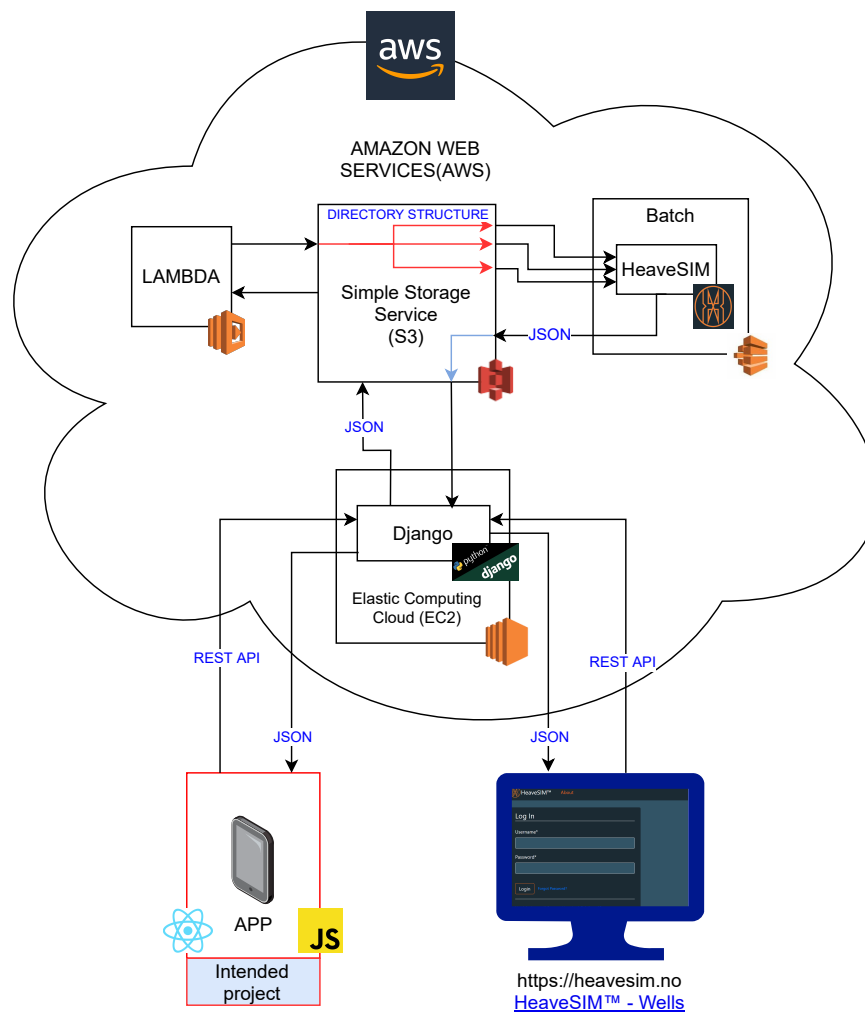


Figure 1: Flowchart HeaveSIM (from [1, 2, 3])
(logos from [13, 14, 15, 16, 17, 18, 19, 20, 21])

The main HeaveSIM application is developed using the Python web framework Django [22]. HeaveSIM runs on AWS Elastic Compute Cloud (EC2), a service for building virtual machines [23]. Upon a request from a client, a JSON file is transferred to AWS Simple Storage Service (S3) [24]. S3 provides a function called Lambda; a serverless, event-driven compute service that triggers batch computation of the requested simulation [25]. Clients like phones/tablets or web

users communicate with the simulator through REST API calls, described in section 2.2.3.

1.3.3 Imagined Use Case

The following use case is meant to capture the motivation behind developing additional features for explaining recommendations and providing forecasts.

Forecast Use Case

A drill operator starts a shift around 7 AM and enters the current drill bit depth and estimated drilling progress for the coming hours in the app. The heave forecast for the day shows that it will not be possible to carry out drill work between 11 AM and 2 PM. In those hours, the heave will cause a breach of a pressure margin. Instead of “detecting” this when it occurs, the operator can factor this pause in drilling into account when planning tasks that day.

The drill operator wonders exactly which pressure margin will be breached at 11 AM. They open the recommendation for Connection which is relevant for drilling. After entering the current drill string depth and heave, it becomes clear from the pressure plot that it is the upper pressure margin that will be breached.

1.4 Report Scope

This report will only briefly explain some of the app’s key features in the state that it was before the start of this project. Only background theory that is relevant for understanding the changes and new features developed during this project will be included. The report will document the work completed between January and June of 2022, and give some ideas for future work of the developed product.

Some assumptions are made about the reader’s knowledge, mainly that they have the same background knowledge as the group before the specialization projects (fall 2021). The group had no noteworthy experience with app development but some general experience with coding. Additionally, the reader is assumed to have a basic understanding of objects and classes, how instantiation works, and familiarity with data types such as dictionary, array, and stack.

1.5 Report Outline

The report outline is meant as a supplement for explaining of the structure in this report as in the table of contents:

- Chapter 2 - *Theory* describes development tools, coding principles and other miscellaneous aspects that are important for this report.
- Chapter 3 - *Earlier Work* is a short introduction to the app’s condition at the starting point of this project
- Chapter 4 - *Method* starts with an overview of the working habits during the project, followed by the process of specifying, designing and planning the app.

- Chapter 5 - *Implementation* goes into detail on how key elements in the app were implemented. This chapter focuses on explaining approaches and underlying logic of various solutions. The final results of these elements will be shown here. This was separated from section 4 to avoid too many subsections.
- Chapter 6 - *Result* gathers the elements shown and described in section 5 and presents them from the user's perspective. The chapter is meant to serve as the user guide specified in the problem description.
- Chapter 7 - *Discussion* includes critique and thoughts from the group about the app. The discussion revolves around requirements in the problem description and app specification.
- Chapter 8 - *Conclusion* summarizes the report and lists ideas for further work.

Throughout this thesis there are multiple pseudocodes. The pseudocodes use the following conventions, and generally mimic high-level code rather than using pure text descriptions:

- Block structure is indicated with indentation
- Comments are sometimes used to explain longer code sequences in brief
- Some operators are written in code, for example “i++”
- Use of common keywords such as **for**, **in** and **if/else**

Listings specified with “example code” or similar do not conform with the pseudocode conventions, but follow relevant syntaxes. A full specification list with every task the group considered is in appendix C.

Theory

This chapter will provide a necessary understanding of key aspects of the report. First is an introduction to the essential developer tools. Packages and libraries that are fundamental in the app are emphasized in their respective subsections. Afterward, some important coding and software principles will be explained as these were in focus while improving the existing app - finally, a brief presentation of linear interpolation and color theory.

2.1 Development Tools

While developing the HeaveSIM app, there were used multiple software tools and libraries. This section gives a brief introduction to what each of them is and their area of use. Section 2.1.1, 2.1.2, 2.1.3, 2.2.3 and 2.3 are based on or rewritten from similar sections in the specialization project reports [1, 2, 3].

2.1.1 React Native

React Native (henceforth RN) is the UI software framework used in this project. It was created and launched by Facebook, Inc in 2015 and is an open-source, cross-platform software framework for Android, iOS, and web apps [26]. RN lets developers create applications for various platforms by using a shared codebase. Since its launch in 2015, RN has grown to become one of the top solutions for developing mobile applications [27]. This is because RN is based on the JavaScript library React, which at the time already had a large developer community [28]. RN is also popular due to savings in time and resources by developing one shared codebase.

RN applications are based on the use of different components like *Text*, *Button* and many more. The large developer community leads to a vast selection of third-party libraries. Several of the components will map to native UI blocks, thus looking like the platform's native components [26]. A short RN code example is set up in fig. 2.

Lines 1-3 show how components are imported. Line 8 defines that the file exports a class and gives the defined component a name that can be used by other screens/components for instantiation. The `render()` function in line 9 is required for all class type components and is responsible for the initial rendering and updates to states used in the UI [29]. As can be seen between lines 14 - 33, each component must be initiated and closed with the logic of `<Component>` and `</Component>`. The components can also be styled within the initiation angle brackets, as shown for *View*, *Text* and *Button*. Lines 24 and 31 are where the texts for the two components are declared.

2.1.2 Expo

Running an RN app on an Android device or emulator is simple, as there are no OS requirements. To run the same RN app on an iOS device or emulator, Apple's guidelines require a Mac running macOS. One way to circumvent this is by using Expo. Expo is a framework and platform for RN applications and consists of a wide number of libraries supporting different functionality. Most important for this project are the two applications Expo CLI (Command Line Interface) and Expo Go [30]. The former application supplies a web-based GUI for building and serving the RN application. When the app is served from the browser, it can be opened on Android and iOS

Listing 1: React Native Example code

```

1 import React, { Component } from 'react';
2 import { Button, Dimensions, Text, View }
3   from 'react-native';
4
5 const WIDTH = Dimensions.get('screen').width;
6 const HEIGHT = Dimensions.get('screen').height;
7
8 class Example extends Component {
9   render() {
10    const emptyFunction = () => {
11      console.log('Pressed!');
12    }
13    return (
14      <View
15        style={{
16          position: 'absolute',
17          top: HEIGHT/2-150,
18          left: WIDTH/2-150,
19          width: 300,
20          height: 300}} >
21        <Text
22          style={{fontSize: 20,
23            textAlign: 'center'}}>
24          This is some text //text field
25        </Text>
26        <Button
27          style={{width: 300, height: 75}}
28          innerStyle={{fontSize: 20}}
29          onPress={emptyFunction()}
30          inline rounded >
31          Press me //text in button
32        </Button>
33      </View>
34    );
35  }
36 }
37 export default Example;

```



(a) React Native example result

Figure 2: React Native Example

emulators or in Expo Go. Expo Go is installed on the developer's phone and allows for running apps that are served from Expo CLI. In this way, the app may be run on any phone from any computer with an internet connection. Expo also lets developers share an app with their team and organization without having to host it on servers themselves. The only requirement is to have an Expo account, which is linked to the organization that develops the application.

2.1.3 React Native Packages and Libraries

Some important packages and libraries that were significant for this project will be described here. Gaining a quick understanding of them is necessary for the following chapters.

Rapi UI

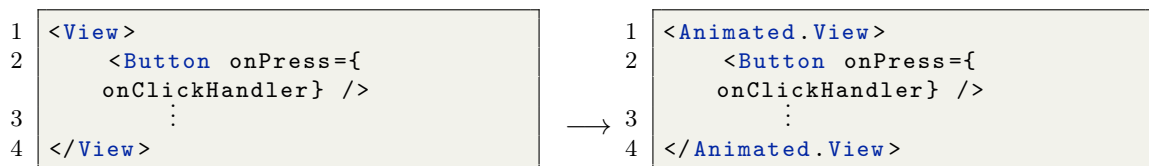
Rapi UI is a third-party library for RN that supplies a range of graphical components to simplify the development [31]. Components like *Layout*, *TopNav* and *Section* help structuring the content on the screen, while other components like *Button* and *Text* are directly interactive. All components are available and used like any standard RN component, which is shown in fig. 2.

Navigation in React Native Apps

For navigation in RN, the library React Navigation supplies a range of useful tools [32]. One type of navigation is stack navigation, where each screen that is navigated to is pushed to a stack. The navigator displays the screen at the top of the stack at all times. All previously visited screens stay mounted on the stack. Whenever navigation happens, the navigator checks if the next screen exists on the stack navigator. If it does, the screen is moved to the top of the stack. If not, a new screen is instantiated and pushed to the top of the stack. The library also supplies support for gestures to control navigation (e.g. swipe right to go back etc.).

Animated

Animated is an RN-supplied library that allows for animations to happen on-screen [33]. Animated is part of the official RN documentation and supports existing components such as *View* and *Text*, ensuring ease of use for developers familiar with RN. Only the components subject to change, i.e. to be moved or have changes to stylistic attributes, need the *Animated* prefix (fig. 3). This means that an animated *View* container supports third-party components and is not limited to RN components only.



```
1 <View>
2   <Button onPress={
3     onClickHandler} />
4 </View>
```

→

```
1 <Animated.View>
2   <Button onPress={
3     onClickHandler} />
4 </Animated.View>
```

Figure 3: *Animated* Prefixes Added to *View* Component

Animations can happen in several ways, and a common function is transforming the position of components. *Animated.Value* is the class that realizes these movements, but it can also be used to change stylistic attributes such as opacity or size [33].

Victory Native

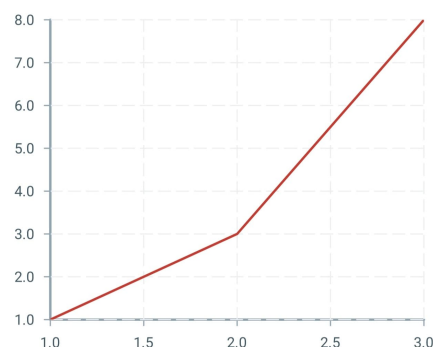
Victory Native is derived from the modular charting library Victory made for React and adapted for RN [34]. The library is composed of React components and is one of the most popular charting libraries for RN [35, 36]. To demonstrate how seamlessly Victory Native blends in with other RN components, fig. 4 shows a code excerpt of an example chart. To differentiate between RN and Victory Native components, they have been colored in blue and red, respectively.

Listing 2: Victory Native Example Code Excerpt

```

1 <View>
2   <VictoryChart      //Define plot-area
3     theme={VictoryTheme.material}
4   >
5     <VictoryLine    //Define type
6                       //of plot-component
7       style = {{
8         data: {stroke: '#c43a31' },
9         parent: {border: '1px solid #ccc'}
10      }}
11     data = [[
12       {x: 1, y: 1},
13       {x: 2, y: 3},
14       {x: 3, y: 8} ]]
15   />
16   </VictoryChart >
17 </View>

```



(a) Victory Native Example Result

Figure 4: Victory Native Example

2.2 Code & Software Principles

This section describes some of the code and software principles that were used in this project and are relevant for the implementation and discussion of results.

2.2.1 Code Quality

Having a set of code standards and conventions is useful for producing readable and maintainable code. A well-known principle within development is Separation of Concerns (SoC). It is a principle that describes how code may best be separated into separate files depending on what purpose it serves [37]. Keeping code that is referenced and imported by other files by itself in a separate file allows for much easier maintenance as there is only one place where code needs to be modified [37]. For app development, the codebase could be separated into three categories: logical code, stylistic code, and UI code. SoC does not only apply on a macro-level across a codebase but also to individual functions. If a function becomes too complex, it is wise to refactor it into smaller functions [37].

2.2.2 Code Runtime

The runtime of a code may translate to the actual time it takes for it to run, but more interesting is to study how the runtime is affected by the problem size. Many problems can be solved in several ways, and the time complexity of the code can be a measure of how efficiently the code solves them. Notations from the Bachmann-Landau family are frequently used to express time complexity [38, 39].

Big O is such a notation and can be written like $\mathcal{O}(n)$, $\mathcal{O}(n \log n)$, $\mathcal{O}(2^n)$, $\mathcal{O}(m * n)$, etc. Here, n denotes in some way the size of a problem. Using two variables indicates that there are two factors that affect the time complexity. The argument in the Big O notation refers to how the runtime changes, depending on the problem size n . This means that an algorithm that has

$\mathcal{O}(n^2)$ complexity will have a quadratic increase in runtime as n increases. Using sorting a list of numbers as an example, n would denote the length of the list. If two sorting algorithms have time complexities $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$, they may perform similarly on small data sets. However, if n is increased by a factor of 1000, one could start seeing significant differences in runtime [38, 39].

2.2.3 REST API

REST API is, as mentioned in section 1.3.2, the communication protocol between clients and the simulator. It is described in [1] as “REST API stands for Representational State Transfer Application Programming Interface, and it defines the process of requesting and sending a representation of the state of a resource [40, 41]. The response comes as a JavaScript Object Notation (JSON) object, which despite its name, is platform-independent. Regardless of the software framework, one wishes to use, as long as it is somewhat popular, it is guaranteed to come with a library for JSON support.”

2.2.4 Cookies

A cookie is, in this context, a small packet of data that a client receives from a website/server. The cookie contains information about the client’s visits and activity to improve the user experience. For example, a cookie may contain authentication information or user preferences, and settings [42]. Cookies may have expiration dates, e.g. 30 days, which can be updated when revisiting the website.

2.2.5 State Persistence

“Persistence refers to object and process characteristics that continue to exist even after the process that created it ceases or the machine it is running on is powered off” [43]. State persistence makes it possible to save the user’s location in the app, so they immediately return to the same location as before the app was closed [44]. When closing and opening the app, state persistence remembers the state from the past and relaunches the app at this state. This means one does not have to log in again and go through all the steps necessary to reach the previous state.

2.2.6 AsyncStorage

“AsyncStorage is an unencrypted, asynchronous, persistent, key-value storage system that is global to the app” [45]. It allows the app state to persist, i.e. save data offline in an app. This is useful for cookies and state persistence.

2.3 Interpolation

Interpolation is to make an approximation within a set of data [46]. Linear interpolation (eq. 1 with $n = 2$) is an interpolation method where a straight line is drawn between two points to approximate a value between them. It is based on the Lagrange Interpolating Polynomial formula [47], which is given in eq. (1):

$$P(x) = \sum_{j=1}^n P_j(x) , \text{ where } P_j(x) = y_j \prod_{k=1, k \neq j}^n \frac{(x - x_k)}{(x_j - x_k)} , x_j \neq x_k \quad (1)$$

2.4 Color Theory

The choice of colors is a key aspect of the UX, and to ensure accessibility [48, 49]. In this context, accessibility generally describes how well software provides an equivalent UX for people with disabilities. Color vision deficiency is a particularly normal visual impairment that a UX designer should take into consideration [50]. Satisfactory accessibility can be achieved by making sure components rendered on top of one another have suitable contrast. The contrast ratio quantifies two colors' contrast, and a contrast of 4.5 : 1 is recommended as a minimum [48]. A more strict ratio of 7 : 1 is required to meet the maximum accessibility standards [48]. More on accessibility, contrast, and color vision deficiency can be read about in [48, 49, 50].

Some colors have subconscious meanings for many people [51]. Subconscious meanings may have roots in other domains than UI and UX design. For example, red and green have subconscious meanings from traffic signals. Mixing these colors up, e.g. displaying an error message in green text, can cause misunderstandings. Such colors should therefore be used carefully [52]. Colors may also assist in showing the user which parts of the app are interactive. A UI hierarchy can be strengthened by keeping background elements in toned-down colors that do not seek the user's attention [52].

Earlier Work

The work that precedes this project is documented in the specialization project reports [1, 2, 3], and this chapter gives a short presentation of the earlier work. The purpose of this is to give a clear view of the starting point of this project and to show the differences between the “old” and “new” app. A visualization of the infrastructure, which displays how the app was structured during the specialization projects, can be found in section 3.3.1 in [2].

3.1 API Calls

In the earlier work, there were used three predefined API calls for communicating with the simulator:

1. Fetching a login token
2. Fetching a user’s predefined analyses
3. Fetching simulation results for a given analysis

These API calls allow the app to follow the principles of REST API described in section 2.2.3. Listing 3 shows a pseudocode for how the API calls generally work (based on [1]). The URL of the API call is called using a method `GET` or `POST` with necessary parameters, and the response data is stored if it is defined.

Listing 3: Pseudocode for API Call

```

1 function makeAPICall() {
2   try
3     fetch(<URL of API call>, {
4       method: 'GET'/'POST',
5       body: [parameters]})
6     if (response is undefined)
7       throw 'Error: undefined results'
8     else
9       //Store response data
10    catch (error)
11      console.error(error.message)
12 }

```

3.2 Screens

The screens in fig. 5 and fig. 6 were the finished product after the specialization projects. The functionality of each screen is described below.

- Login screen [5a]: The first screen the user meets. It requires a username and password, and then requests a token with the first API call to HeaveSIM. If it receives a token, the app navigates to the Home screen. Otherwise, an error message is displayed on the screen.
- Home screen [5b]: Support for a “Logout” button which navigates back to the Login screen. Another button which requests analysis data with the second API call and navigates to the Analyses screen.

- Analyses screen [6a]: This screen displays a list of analyses the user can get recommendations for. Each analysis contains a button which navigates to the relevant Recommendation screen.
- Recommendation screens [6b]: Fetches simulation results with the third API call. This screen requests inputs relevant to an operation type and displays a suitable recommendation to the user.

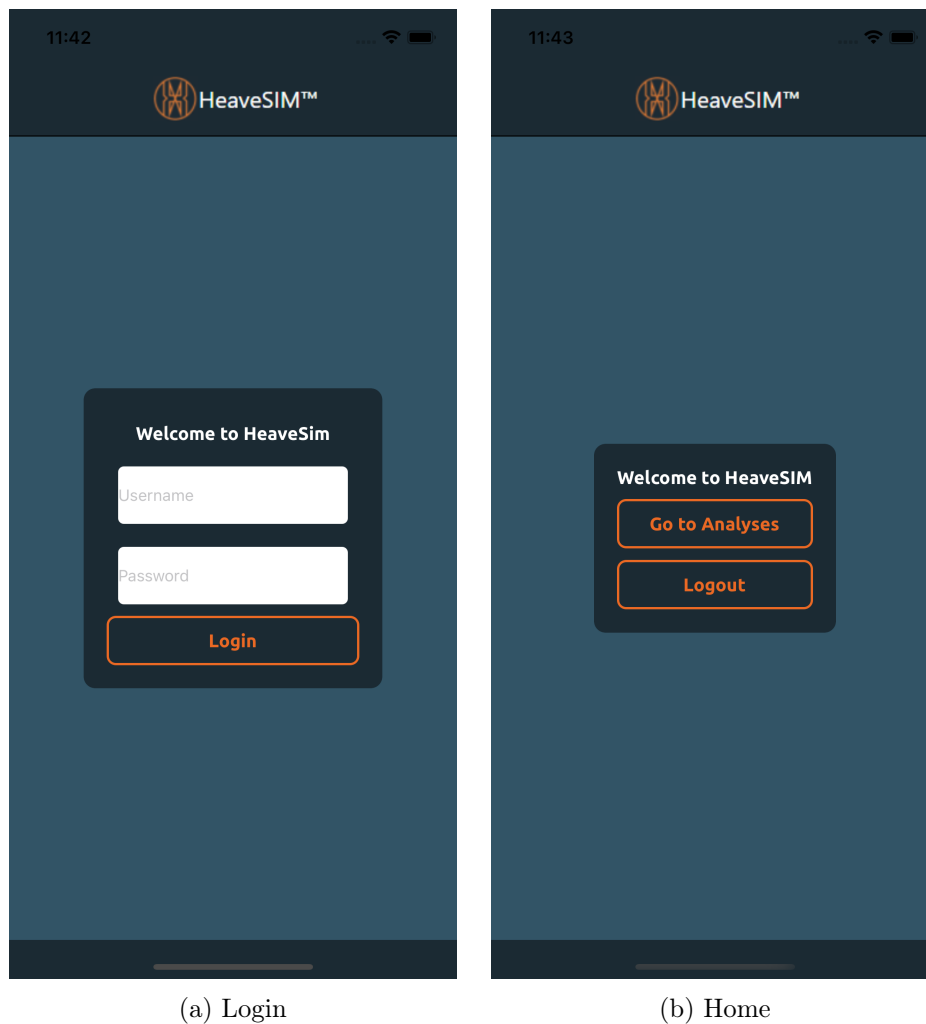


Figure 5: Screens Made Previously (1/2)

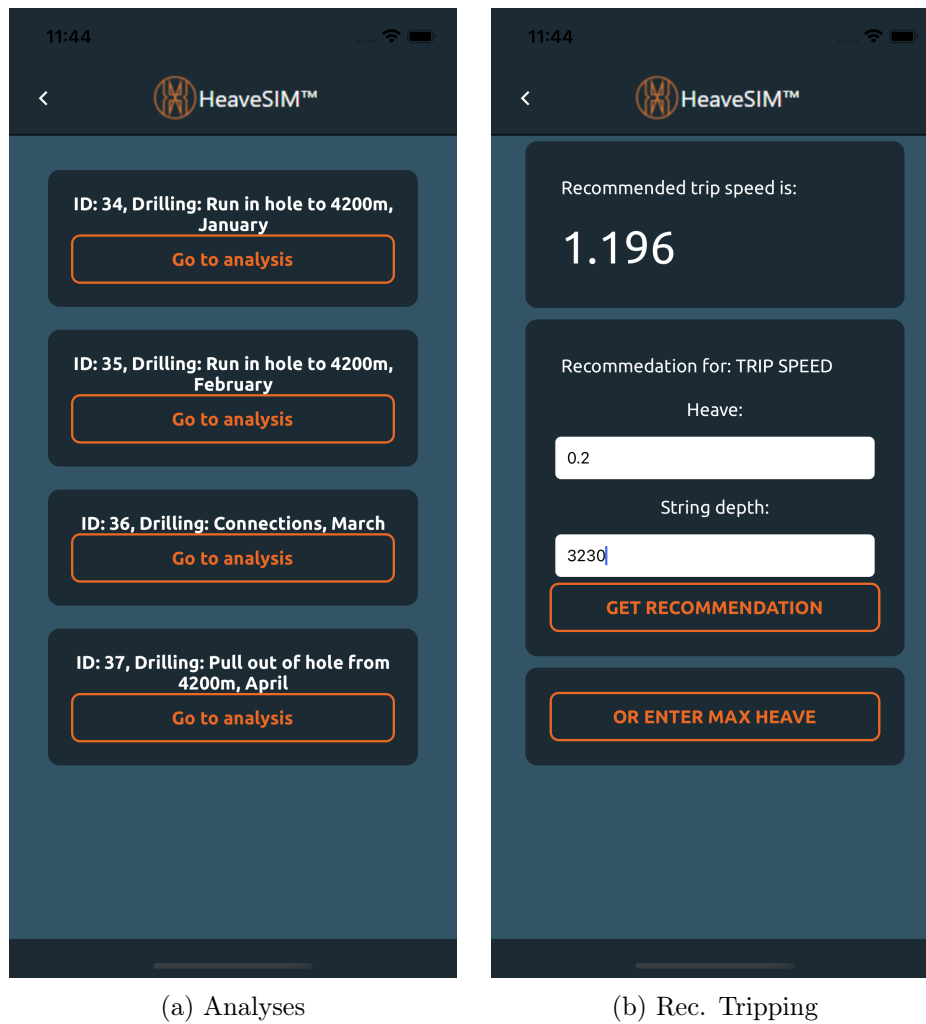


Figure 6: Screens Made Previously (2/2)

3.3 Desired Functionality and Future Work

The app had several issues that needed addressing as it was at an early PoC stage. There was support for two types of drilling operations, namely Connection and Tripping.

The app was built using stack navigation and class components. Class components are stateful and keep the UI updated whenever a state change occurs [53]. Parameters like the login token and analysis results were passed from one screen to the next as properties.

Suggested future work for the app at the point of delivery was:

- Restructure the code to better follow the SoC principles
- Improve the design to achieve a more professional look
- Include more of the functionality provided by the HeaveSIM website
- Implement input handling to avoid wrong inputs
- Remove unnecessary hard coding

These points were the baseline for some of the changes the group decided to make as the initial part of the project, presented later in section 4.2.

Method

This chapter describes the work routines during this project. It starts with a description of the workflow and utilized tools developed to streamline a development process. How the group read up on app development and decided on a software framework is briefly mentioned. Then, tables containing the app specification for this project are presented. Afterward, areas of improvement regarding the app's design are specified. An early design section outlines what the graphical presentation of data results was planned to look like. The process of getting access to weather data for forecasts is described before some attempted features in the app are presented.

4.1 Workflow

The workflow among the group members during the specialization projects was efficient and had left a positive experience. Deciding to keep a similar workflow was a natural choice. Weekly meetings with the supervisor were de facto deadlines for work in progress. Demonstrations for Heavelock employees were also a source of feedback and guidance.

Efficient workflow was assisted by the use of software developed for this exact purpose. The use of Git [54] and GitHub [55] allowed for changes to be implemented sequentially. Git is the industry standard version control system, which is why it was chosen in the first place [56, 57]. Performing code reviews on GitHub meant that bugs that slipped past the code's author were discovered by another group member before merging into the codebase.

To have an overview of the process, the online service Trello was used to keep track of remaining and completed tasks. Other miscellaneous documents were kept in a shared folder on Google Drive.

4.1.1 Learning App Development

Before the specialization projects, the group had no noteworthy experience with app development. To get a basic understanding, the preferred approach was to watch tutorials of RN fundamentals on YouTube. Coding examples on the websites W3 Schools [58] and the coding forum Stack Overflow [59] were highly useful throughout the development process. During this project, the skill set acquired during the specialization projects was further enhanced using the same method.

4.1.2 Choice of Software Framework

At the beginning of the specialization projects, the group considered two different options: native development for iOS or cross-platform development for both iOS and Android. Initially, the former option was favorable due to better performance and integration [60]. However, this option would require all group members to develop on Apple computers with macOS. With the former option ruled out due to high costs of Apple computers, the group chose to develop on both platforms. Different frameworks were considered, and the group decided to use RN. The group saw no need to choose a new development environment for this project, as the positive experience and knowledge from the specialization projects were advantageous.

4.2 App Specification

A set of requirements were determined during the first weeks of the project, and it was quickly determined that functionality for forecasting should have the highest priority (No. 8 in table 2). In combination with a revamped design, the end goal is to have the app presentable as a prototype. A meeting with Heavelock employees confirmed that these two points would have the greatest effect on the app’s usefulness. The problem description was formulated based on these requirements. The following two tables categorize all requirements into “Improvement of existing app” and “New functionality”. A more extensive table of all considered tasks is in appendix C.

Table 1: Specification: Improvement of existing app

No.	Functionality	Explanation	Priority
1.	Change from static to dynamic navigation	Navigation to different drilling operations’ recommendation screens was based on strings consisting of names and depths which varied in different analyses. To avoid errors for other analyses, this must be changed.	5
2.	Design / UI	The design needs a more professional look. This can be achieved by prettifying the current design, and also: <ul style="list-style-type: none"> • Add animations for navigation between screens • Implement a vertical navigation menu for quicker navigation back and forth • Include statusbar into the design to utilize the whole screen 	5
3.	Improve code quality/ Clean up code	<ul style="list-style-type: none"> • Follow SoC principles by creating script files • Set up a stylesheet file to remove repeated style definitions • Merge similar functions to reduce the number of functions 	5
4.	Write an extensive README.md	Complete the README to give an introduction to the code for better readability and so that “everyone” can try the app without difficulty.	5
5.	Better descriptions of inputs and outputs	Include suffixes and units of all inputs and outputs to reduce the chance of misunderstandings and errors between users and app.	5

Table 2: Specification: New functionality

No.	Functionality	Explanation	Priority
6.	Deployment of app	The app must be deployed such that it can easily be showcased to customers.	4
7.	Settings screen	Give the user the ability to change units and color scheme, with the option to add more at a later point of time.	2
8.	Inputs for multiple points in time/ Forecast	Try to either let users input multiple situations at the same time and then compare results, or design a predictor of how future weather affects operations. Other things to implement could be: <ul style="list-style-type: none"> • Show estimated progress based on input Rate of Penetration (ROP)* • Display predictions graphically and not only return numbers • Recommendations for future operations based on future weather conditions 	5
9.	Remember previous location in app	Implement a feature that saves the user’s last location in the app so that they do not have to log in and navigate back after the app is closed. In this feature, a solution for saving the user’s credentials must also be implemented.	4
10.	Automatic logout/ Demand credentials after given time	For safety reasons, the app should automatically log users out after a given time.	2
11.	Error handling	Find a solution that handles errors. The app should not crash whenever it encounters an error.	3
12.	Deny wrong inputs	Inputs that does not fit the simulation (e.g asking for string depths outside the scope of analysis) or wrong inputs (e.g. negative tripping speed) should not be allowed. Disabling buttons and/or giving warnings are possible solutions.	5
13.	Show data/ results graphically	It is important to show data graphically as this is a much more intuitive way to understand data for users, such as a plot.	5
14.	Expand to other devices	According to Heavelock, some companies use iPads offshore, so enabling the app to work on other devices can increase companies’ desire to use this service.	4

* ROP: “The speed at which the drill bit can break the rock under it and thus deepen the wellbore.” [61]

4.3 Early Design

Multiple tasks in the specification revolve around adding graphical elements to the app or improving the visual appearance, such as “Showing data/results graphically” (No. 13) and “Design / UI” (No. 2). These tasks require planning before the solutions can be developed, and this section presents ideas regarding that.

Measures for a More Professional Appearance

The app presented in section 3 was a PoC, and therefore little to no effort was put in to make it visually appealing. Additionally, the app was lacking features that are often associated with a professional app. Such features may be a settings screen or animations indicating that the app is loading data.

Animations on the Login and Home screens would be suitable as they are the first screens the user interacts with. It only takes seconds to create a first impression, therefore it is important to get it right [62, 63, 64]. Including background images on these two screens will improve the appearance as they are the screens that users first meet. Login and Home screens are very common in apps today, so mainly putting effort into these designs might have the greatest trade-off in terms of a more professional look. This does not rule out improving the designs on the recommendation screens, and avoiding attention-seeking design elements here lets the plots be in focus.

Previously there were two recommendation screens for Tripping operations that had awkward navigation between them. Gathering these screens’ content in different tabs on one screen would easily categorize the different recommendation types and avoid unnecessary navigation. A sliding animation when changing tabs suits the more professional look.

The HeaveSIM website allows users to see results in different units. Implementing this functionality makes the app more applicable to a larger share of the market. Allowing units to be set from a predefined list in a settings screen would be natural. The units appear on multiple screens, and one might only want to set the preferred units once.

Phones and tablets often allow users to select between a light or dark theme on the device. Implementing a selection of color schemes and integrating the app with the device’s theme could improve the user experience. As the app displays information that can be critical, ensuring accessibility to users with color vision deficiency can be emphasized with a color scheme following principles from section 2.4.

Plots of Analysis Data

To give users a more clear and intuitive understanding of analysis data, it was requested to show results graphically. Seeing that HeaveSIM uses two plots to show results, it was determined that the app should do the same. Since these plots already have been designed by HeaveSIM, the group decided to use them as inspiration. The plots used on the HeaveSIM website are presented in fig. 7. The implementation of these plots are shown later in the report in section 5.11.1 and section 5.11.2.

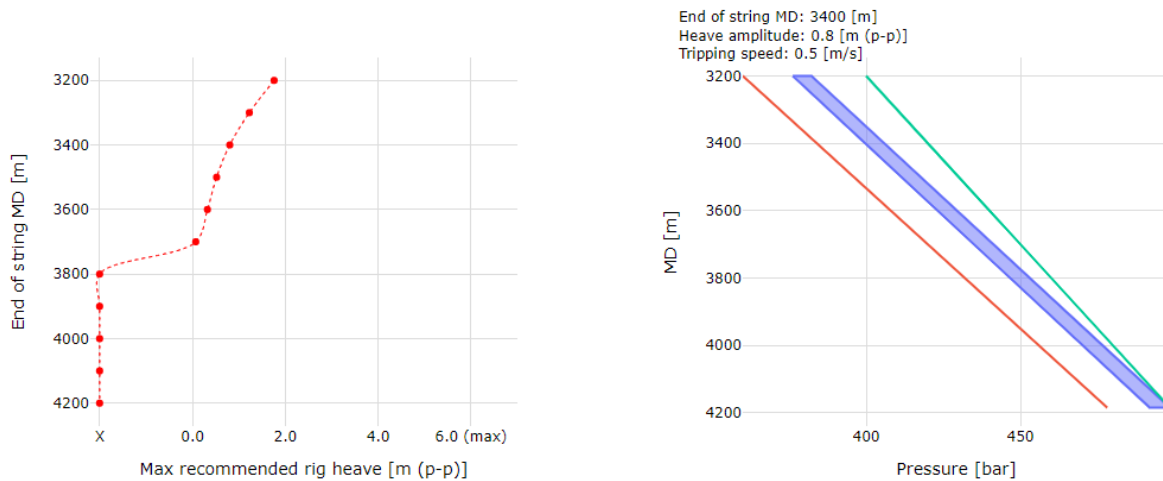


Figure 7: Plots in HeaveSIM [65]

The left plot is for showing recommended heave or tripping speed at each depth in an analysis. To the right is a plot over well pressure where the left line (red) is the lower pressure margin, right line (green) is the upper margin and the middle area (blue) is for well pressure.

Forecast Plot

Providing users with a forecast for future work is a new feature to HeaveSIM altogether. One of the main concerns for this functionality was presenting results to users in an intuitive and descriptive way. Multiple ideas and sketches were made, and the final sketch is shown in fig. 8.

The design builds on the recommended heave plot shown to the left in fig. 7. It will have one axis displaying heave and one axis for depths. A new axis opposite of depths is for showing hours. Inside the plot itself, there will be two lines. The white line displays recommended heave for the different depths, and the blue line shows the predicted heave for each hour.

4.4 Acquiring Weather Forecasts

To design and implement recommendations for future operations, some form of weather forecast is necessary. Heavelock employees suggested the group to request access to weather services provided by StormGeo, a company specializing in weather intelligence [66]. With StormGeo declining the request, it was determined that dummy data would suffice as a replacement. To transform ocean waves into rig heave it is necessary to have the vessel's response amplitude operator (RAO) [67]. RAO is "ratio of a vessel's motion to the wave amplitude causing that motion and presented over a wide range of wave periods" [68]. The lack of the RAO necessary for transforming waves to heave requires this part to be redone at a later time regardless, thus justifying this temporary solution.

4.5 Attempted Features

This section describes some features that the group attempted to implement but were deemed not possible for different reasons. Some of these features are also mentioned in section 8.1, as

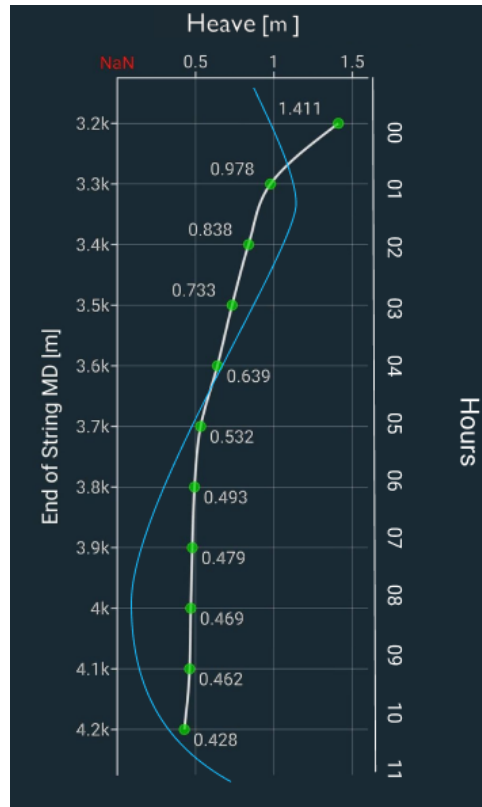


Figure 8: Forecast Early Design

ideas for future work.

State persistence

An attempt to implement state persistence was made to satisfy requirement 9 in table 2. This was done by trying to save the navigation state at each state change using AsyncStorage. All other state values would also need to be stored.

It was quickly discovered that AsyncStorage was deprecated, which meant that this method would no longer work [45]. Other attempts were made with different third-party packages instead, but the states could not be saved as desired. Therefore, state persistence became of lower priority and rather a task for further development.

Cookies

Requirements 9 and 10 in table 2 implied storing a valid login token. Storing it in a cookie allows users who log in frequently to not enter their credentials every time. The cookies would be stored locally on the device and have their expiration dates refreshed on every use of the app [69]. Implementation turned out to be problematic with Expo as there is no official support for cookies [70]. Alternatively, one could turn to AsyncStorage to create similar functionality, but for the previously mentioned reasons, the decision fell on suspend cookie functionality.

4.6 Deployment

In order for users to download and use the app, it needs to be deployed. This can be done in several different ways and on several different platforms. When the app is production-ready, it should be deployed to Google Play (Android) and App Store (iOS). That way, it will be available for everyone to download.

As stated in point 6 in table 2, the app should be deployed. Even though it is not available in the app stores yet, one way to showcase the app to potential customers is by taking advantage of Expo. In this case, an Expo organization and user were created for HeaveSIM, and then the app was deployed by using the command `expo publish` in the development terminal. Everyone with access to the username and password can download and test the app. See appendix B for a guide on how to download and test the app.

Implementation

This chapter describes how the specification and design in chapter 4 were implemented. The underlying logic of the results will be presented here in order to keep chapter 6 brief and solely review the final result as users experience it.

5.1 Code quality

It was decided to use capital letters at the start of every class name and write constant values such as the screen's width with capital letters only. As RN allows for both functions and variables to use the `const` declaration, the group decided to separate the two by using camelCase and underscores, respectively, to avoid mistaking one for the other. From the earlier work, giving functions and variables explanatory and intuitive names was already in focus, with no changes necessary.

At the beginning of this project, a significant overhaul of the codebase was done to better conform with the principles of SoC. The code for computing recommendations was previously separated into three files, each corresponding to one type of simulation. Generalizable functions were refactored and gathered in a new file, shortening the recommendation code by approximately 2/3. Hence the code has better readability and maintainability.

As mentioned in chapter 3.3, parts of the code contained hard coding. In the case of navigation from the Analyses screen to a Recommendation screen, a dictionary pairing the name of the analysis type and the next screen was used. The solution to avoid this was that HeaveSIM added an analysis type identifier (integer) to the analysis object in the second API response and navigating based on that instead. The analysis type names do not affect the navigation anymore, thus making the code more robust.

Other code that was changed or restructured to better conform with SoC and/or avoid hard coding will be mentioned in subsequent sections as their changes require a more in-depth explanation.

5.2 Styles

Stylistic code in RN is based on the use of StyleSheets [71]. A StyleSheet has the exact same purpose as Cascading Style Sheet (CSS) [72], and is a dictionary of stylistic attributes and their set values. Listing 4 is an example of how a StyleSheet can be generated, with two different styles declared lines 2 and 9.

To achieve improved SoC, the majority of the stylistic code was moved to a separate file. Earlier, multiple screens had reoccurring stylistic code. Gathering this code in a separate style file improved readability and maintainability. The stylistic code is split into two parts: `plotstyles.js` for styling the plots, and `styles.js` for styling the screens/components. In the case of `styles.js`, it exports a function `getStyles()` that returns a StyleSheet object with correctly colored stylistic attributes. Plot styles work similarly in `plotstyles.js`.

The status bar where time and battery information is displayed on phones has background colors matching the `main_dark` color of the app. This small detail improves the look and the

Listing 4: StyleSheet Code Example

```

1  const Styles = StyleSheet.create({
2    container: {
3      backgroundColor: 'blue',
4      :
5    },
6    :
7    logout_section: {
8      backgroundColor: '#FF0000',
9      :
10   },
11  },
12  });
13

```

accessibility. The font color of the content in the status bar is opposite the selected theme, i.e. white font for dark theme and vice versa. Fig. 9 shows the old and new statusbars.

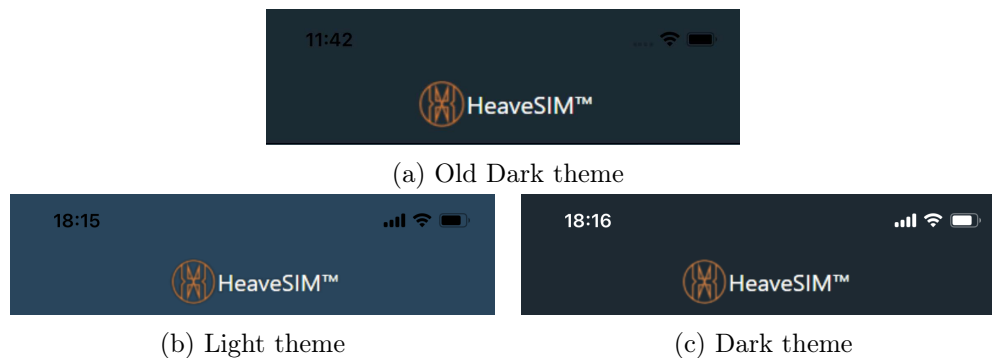


Figure 9: Old and New Status Bars

To cope with the vastly varying screen resolutions on phones and tablets, it was necessary to add an upper constraint on the section width. Sections that fill the screen width on phones look awkward on the iPad. Constraining the width of sections was implemented by adding two new variables to `styles.js`: `FULL_WIDTH` and `ADAPTED_WIDTH`. The former is set to the device's screen width, and the latter is set to either `FULL_WIDTH` or 800, whatever is less. As content did not fill the screen width on the iPad, items shown in a `ScrollView` were not centered. This was solved using the solution from [73].

5.3 Colors

Separating the colors from `styles.js` allows a developer to make global color changes/additions at a later stage. The file `colors.js` contains a dictionary with keys that are the available color schemes the user can choose from. Each key's value is a dictionary of colors. Shown in listing 5 is an example.

The name of the selected color scheme is stored in the user settings. Adding a brand new color scheme under a new key in the `COLORS` object will be selectable from the Settings screen in the app. The only requirement for a new color scheme is that it includes all existing keys (i.e.

Listing 5: Colors Code Example

```

1 export const COLORS = {
2   'LIGHT': {
3     'background': '#1f465e',
4     'text_color': 'white',
5     :
6     :
7     :
8     'statusbar': 'dark-content',}

```

background, text_color etc.). Colors can, in most cases, be specified with hexadecimal RGB values ('#ab01c3') or simply by a color name ('yellow').

5.4 Units & Conversions

The HeaveSIM website supports multiple different units, and although a settings screen had a low priority, it turned out to be useful anyway. Similar to the colors, the units and necessary conversion rates are stored in a separate file to ensure maintainability. Any unit that has a scalar conversion rate can be added at any time, and the rest of the app will support it.

Table 3 shows the units of the data received in the simulation results, as well as available units on both the HeaveSIM website and in the app. Pressure units specific gravity (SG) and pounds per gallon (PPG) were not implemented in the app as their conversion rates are not scalar with respect to bar. Note that simulation data is always given with metric units.

Table 3: Units

Measurement	Sim Data	Website	App
Heave	m	m(d-a)*, m, ft(d-a)*, ft	(same as website)
Depth	m	m	m, ft
Tripping speed	m/s	m/s, ft/s, ft/min	(same as website)
Pressure	bar	bar, psi, SG, PPG	bar, psi

*(d-a): Double amplitude, also known as Peak-to-Peak

As many screen and component files import functions from script files which again import functions from other script files, the logic was to apply conversions right before comparing inputs to simulation data. In most cases, this meant converting to the listed default units right before interpolation and then converting back to the selected units. To keep things consistent, inputs were divided by a conversion rate, and outputs were multiplied by them.

5.5 Settings

A script file `settings.js` was created to serve three purposes: storing global and user settings, storing cookies with login token or analyses/simulation data, and act as a hub for the styles, colors, and units script files. The screen files will mainly use *getters* and *setters* to access data in `settings.js`. Data may also be cleared with functions like `clearToken()`.

Global settings include upper limits for heave and tripping speed. Common for them is that they should apply to every user of the app. The idea with the upper limits is that an executive for a specific oil rig can customize them to suit a specific floating rig. User settings are kept separately. Stored in the user settings are the token, the username, and the selected units and color scheme.

By forcing the screens and components to access units, colors, and styles via the settings file only, global alterations are easy to implement. Removing the logic from the individual files for units, colors and styles makes them more maintainable and ensures a loose coupling [37].

5.5.1 Setting Units & Color Scheme

Units and the color scheme can be selected on the Settings screen. Styles are essentially self-updating in the current implementation. Whenever `getStyles()` is called, the color scheme name stored in the user settings is accessed. Then, the colors of this color scheme are loaded into `styles.js` with the function `getColorScheme()` in `settings.js`. A new `StyleSheet` object using the correct colors is returned to the screen.

Due to the new infrastructure presented later in section 6.1, the Login and Home screens are always underneath the Settings screen on the stack navigator. When a new color scheme is selected in the Settings screen, we want a color change to apply to all previously visited screens. Simply navigating back to the Home screen would not re-render the screen with new colors until somewhere on the screen was pressed. Several other solutions were tested: passing new colors as a parameter to Home, using a callback function, and adding a listener on the navigation state on the Home screen [74, 75]. The solution for the color update was to reset the stack navigator and pushing the Login and Home screens to it. An attempt to transfer the Login and Home screens was made but did not work out as wanted. When the user navigates back from Settings to Home, the new colors are in effect. A major downside of this solution is that all other previously visited screens are reset, removing any input data [76].

Once a unit is set in the Settings screen, the function `setUnit(measurement, unit)` will store the selected unit and corresponding conversion in the user settings in `settings.js`. For compound units (e.g. m/s to ft/min) the conversion rates for both individual units are multiplied to create the compound conversion rate. All screens displaying units as text will dynamically access the selected units from the user settings. The conversion rates are also accessed from the user settings, but the user never interacts with them directly. As simulation data is provided with fixed units seen in table 3, converting inputs of other units is necessary for the interpolation to work.

5.6 Indication of Loading Data

One of the measures mentioned in section 4.3 was to indicate to the user when the app is busy with loading data. RN supplies a component for this exact purpose: *ActivityIndicator* [77]. The

ActivityIndicator maps to the native UI, meaning that it looks different on iOS and Android. Figure 10 shows how the components look on these platforms.



Figure 10: *ActivityIndicator* on iOS and Android

We can implement the *ActivityIndicator* using states and conditional rendering of components. A boolean state `activity` is set to `true` before we make an asynchronous function call, which in the app are the three API calls. Then, after the API call is complete, we set the state `activity` to `false`. This logic allows us to render the *ActivityIndicator* whenever the app fetches data with one of the API calls. Surrounding *Button* components that make API calls, we can add an if-statement and use the state `activity` to either render the *Button* if false or the *ActivityIndicator* if true. Thus, the user can not repeatedly press buttons that make API calls.

Henceforth, components that implement this conditional rendering are said to “incorporate the *ActivityIndicator* functionality”. Figure 11 demonstrates such a component.

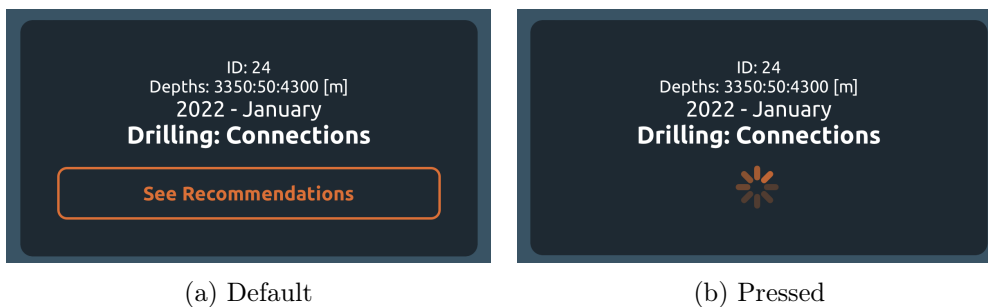


Figure 11: *Button* replaced by *ActivityIndicator*

5.7 Dynamic Display

According to requirement 15 in the specification, the app should display the predefined wells and analyses. Dynamic displaying of analyses were implemented in the specialization projects, but not the wells they belong to. Sections 5.7.1 and 5.7.2 describe how dynamic displaying of wells and analyses was implemented.

5.7.1 Dynamic Wells

Previously the app displayed a list of all analyses sorted by `analysis_id`. It was desired to display and sort by the different wells and then navigate to the respective analyses. A new screen Wells was created for this purpose. The screen accesses the response from the second API call in section 3.1. Listing 6 displays the structure of the API response, i.e. the list of a user’s analyses.

To extract a list of a user’s wells, we can search the list of analyses for well names. From listing 6, the relevant data is in line 8 and 16. We iterate over the list of analyses and add all unique well

Listing 6: Example Response from the Second API Call

```

1 Array [
2   Object {
3     "analysis_id": 34,
4     "analysis_type": 1,
5     "analysis_type_text": "Drilling: Run in hole",
6     :
7     "well_name": "Test1",
8     "well_owner": "Heavelock",
9   },
10  :
11  Object {
12    "analysis_id": 44,
13    :
14    "well_name": "Test2",
15  },
16 ]

```

names to an array `well_name_array` (a well may have several analyses). Applying this to the analyses list in listing 6 would result in the following `well_name_array`: `["Test1", "Test2"]`.

In order to display the wells on the Wells screen, we need to generate a section for each well to be put in a scrollable list. A component called `Well_list` takes a well name as a property and renders a section with the well name and a button “Go to Analyses”. Figure 12 shows what a section looks like.

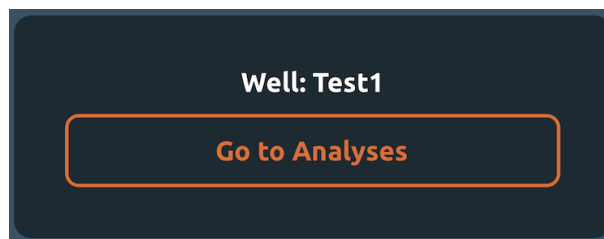


Figure 12: Section Containing a Well

The “Go to Analyses” button navigates to the subsequent Analyses screen, where all analyses belonging to the selected well are displayed similarly. To put all the well sections in a list on the Wells screen, we use the JavaScript function `Array.map()` on the array of well names. Inside a RN component `ScrollView` [78], each well is mapped to its own `Well_list` component according to listing 7.

5.7.2 Dynamic Analyses

After pressing a “Go to Analyses” button on a well on the Wells screen, the app navigates to the Analyses screen. The Analyses screen displays all analyses that belong to the well that was pressed. We can figure out which analyses belong to a well by iterating over the analyses list and comparing every well name to the selected one, adding only those who match to the array `analyses_map`.

Listing 7: Code for Populating a Scrollable List with Unique Well Sections

```

1 <ScrollView>
2   {well_name_array.map((well_name_array) =>
3     <Wells_list
4       key={well_name_array.Well_id}
5       well={well_name_array.Well_name}
6       well_name_array={well_name_array}
7       navigation={navigation}
8     /> }
9 </ScrollView>

```

The specification states in requirement 15 that the analyses need to be displayed in a specific order. First sort by month, then by operation in the order: *Run in hole* - *Connection* - *Pull out of hole*. *Run in hole* and *Pull out of hole* are types of Tripping operations. This order best matches how the operations work in real life. As seen in listing 6, each operation is defined by an analysis type. Connection has 0, Run in hole has 1 and Pull out of hole has 2. Since the wanted sorting was Run in hole(1) - Connection(0) - Pull out of hole(2), the indices of Run in hole and Connection need to be switched.

Now that the analysis contains the correct analysis type based on the operations, we can sort the analyses by month and operation using the comparison function in listing 8.

Listing 8: Function for Sorting Analyses

```

1 function compare(a, b) {
2   if (a.month_of_operation < b.month_of_operation)
3     return -1;
4   if (a.month_of_operation > b.month_of_operation)
5     return 1;
6   if (a.analysis_type < b.analysis_type)
7     return -1;
8   if (a.analysis_type > b.analysis_type)
9     return 1;
10  return 0; }

```

Displaying the analyses is done similarly to wells in listing 7, except we map each analysis to an `Analysis` component instead. Figure 13 shows what this component looks like.

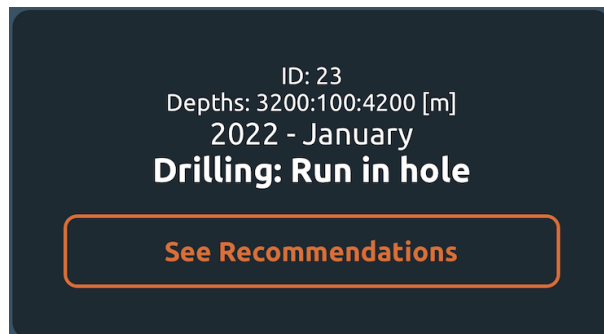


Figure 13: Section Containing an Analysis

5.8 Error Handling

To ensure a UX that lives up to the standards of a finished product, one has to avoid app crashes. All run-time errors are avoided through the code reviews mentioned in section 4.1. External communication is mainly the other source of errors, and some specific errors are:

1. No internet connection
2. Wrong username and/or password
3. The user has not defined any analyses
4. Attempt to fetch data from unknown analysis ID
5. No simulation results found
6. Simulation has incomplete results
7. Try to render a component that received invalid data

Each of the listed errors needs to be treated at a suitable time and place in the app. As the development of the app depends on an internet connection with the Expo server, testing and implementing error handling for a lacking internet connection has not been done.

If a wrong username and/or password is entered on the login screen, the first API call will return an object containing an error message. RN supplies an *Alert* component that displays the error message in a pop-up box, also known as a modal. A callback function may also be defined, which is run after pressing “OK” on the error modal. As a final check, no navigation to the Home screen happens unless there is a login token stored in the user settings.

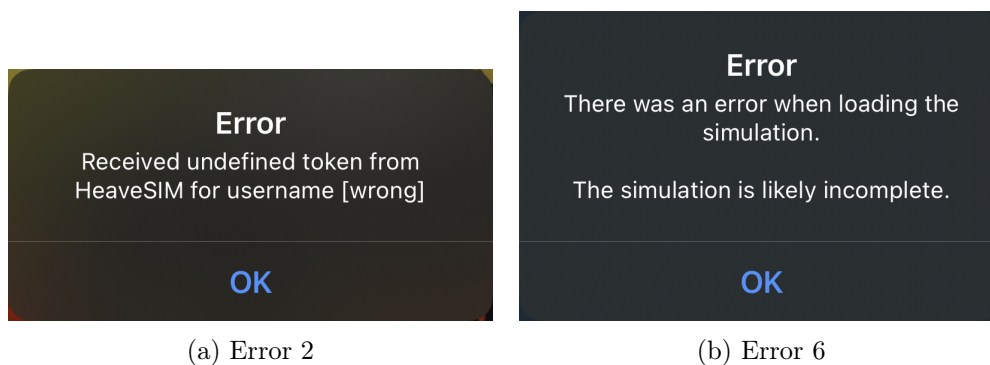


Figure 14: Error Message Examples

Errors when getting a user’s analyses (3 and 4) are dealt with on the Home screen before navigating to the Wells screen. The second API call will return different responses depending on which error occurred. An *Alert* modal shows a more understandable message for the user to describe exactly what caused the error.

Errors 5, 6, and 7 are similarly dealt with before navigation to a screen with recommendation results. Handling the 7th error also utilizes `navigation.goBack()` as the callback function, returning the user to the previous screen. This logic of navigating back to the previous screen allows the user to use the rest of the app’s functionality without crashing.

5.9 Input Handling

Every input field in the app has been equipped with some form of input handling to satisfy requirement 11 in table 2. This is a safety measure to ensure that correct recommendations are given to the user and also to avoid app crashes. Victory Native components can not operate when passed invalid data, such as NaN or empty strings as parameters where numeric values are expected.

The first input field the user meets is the login credentials. This is the input field with the simplest input handling. The “Login” button remains disabled until both username and password have been filled out with at least one character.

The remaining input fields in the app are for inserting numeric values and therefore use a numerical keyboard. If input values are outside the scope of their chosen analysis, e.g. users writing $depth = 2500m$ in an analysis with depths between $3200 - 4200m$, an *Alert* modal displays what input is wrong and its valid range. An example of what this looks like is shown in fig. 15. From this point forward, if nothing else is specified, all input fields expecting numeric values are restricted to only allow values between 0 - maximum or minimum - maximum in chosen analysis. These checks are important because giving recommendations with extrapolated result areas can be dangerous. Input checks are not prompted while the user types in an input field but when exited. To avoid the plotting library to crash when invalid inputs are entered, invalid variables are set to 0 until corrected.

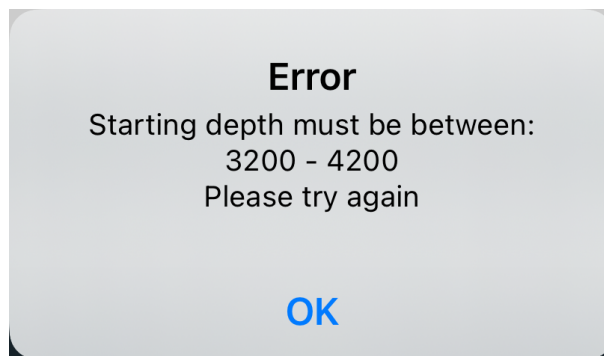


Figure 15: Modal of Invalid Input

On screens containing multiple input fields, submitting data in the first input field moves the text cursor to the next input field without dismissing (hiding) the keyboard. This feature was implemented according to solutions in [79, 80]. The feature works on all keyboard types for Android and iPad, but only the standard keyboard on iPhone. The numerical keyboard used is of the type 'numerical' and does not have an enter or submit button on iPhone.



Figure 16: Numerical Keyboards on Different Platforms

5.10 Recommendation

To give an explanation of how recommendations are computed, this section goes into detail about recommendations for Tripping operations as these are more advanced than Connection. Parts of the explanation is inspired by [2].

In the given dataset, there are recommendations for heave and tripping speed, where the user is required to specify depth and either heave or tripping speed. Focusing on recommendations for heave, there are four heave recommendations per depth. An example of how given data looks like for recommended heave can be seen in listing 9. The example shows four dictionaries with four variables where `hole_depth` is the maximum depth in the analysis, `string_depth` is the drill bit's measured depth, `tripping_speed` is the speed at which the drill bit is run in/out of hole. Lastly, `heave_amplitude` is the maximum heave amplitude the rig is recommended to have at the chosen string depth and tripping speed.

Listing 9: Example of Recommendation Data for Depth

```

1 {"hole_depth":4200.0,"string_depth":3500.0,
2  "tripping_speed":0.0,"heave_amplitude":1.3915855842029607, ...},
3
4 {"hole_depth":4200.0,"string_depth":3500.0,
5  "tripping_speed":0.5,"heave_amplitude":0.2601057971259974, ...},
6
7 {"hole_depth":4200.0,"string_depth":3500.0,
8  "tripping_speed":1.0,"heave_amplitude":-1.0, ...},
9
10 {"hole_depth":4200.0,"string_depth":3500.0,
11 "tripping_speed":1.5,"heave_amplitude":-1.0, ...},

```

Listing 9 contains the four most common tripping speeds in different analyses. Henceforth, when variables like `tripping_speed` are said to be included in an analysis, it means the numeric value the user is searching for is one of the four values stated in the analysis. In other words, tripping speed at 1.0 is included in an analysis, but tripping speed at 0.75 is not included. Additionally, referring to a single recommendation means referring to one entire dictionary where `heave_amplitude` is given after a specific pair of `string_depth` and `tripping_speed`. The latter is more relevant for explaining plots later in section 5.11.

When computing a recommendation there are four cases to be aware of. They are as following:

- Case 1: Wanted depth and tripping speed are included in the analysis
- Case 2: Wanted depth is included in the analysis, but tripping speed is not
- Case 3: Wanted tripping speed is included in the analysis, but depth is not
- Case 4: Wanted depth and tripping speed are not included in the analysis

Case 4 is the most complex one and will be explained here. The former cases skip some steps but start likewise. The outcomes after completing each step of this explanation are given in listing 10.

Listing 10: Algorithm for Computing Recommendation: Case 4

```

1 wanted_depth = 3250
2 wanted_trip = 0.25
3
4 //Step 1 - define arrays with unique values
5 depth_array = [3200, 3300, ... , 4200]
6 trip_array = [0, 0.5, 1.0, 1.5]
7
8 //Step 2 - find closest tripping speeds
9 closest are 0.0 and 0.5
10
11 //Step 3 - find heaves for depth and closest tripping speeds
12 heaves(0.0) = [6, 5.8, ... , 2]
13 heaves(0.5) = [4, 3.8, ... , 1]
14
15 //Step 4 - interpolate heaves
16 heaves(0.25) = [5, 4.8, ... , 1.5]
17
18 //Step 5 - find closest depths and corresponding heaves
19 closest are 3200 and 3300
20 heaves are 5 and 4.8 (from heaves(0.25))
21
22 //Step 6 - find heaves at 3200 and 3300 in heaves(0.25)
23 return interpolateFunction([5, 4.8], 3250, [3200,3300])

```

The first step is to set up two arrays containing unique values of depths and tripping speeds in ascending order. Since wanted tripping speed is not included in `trip_array`, we must find the closest tripping speeds using Amit Diwan's closest number function (step 2) [81]. Then we need to set up heave arrays for both tripping speeds at every depth (step 3). Afterward, in step 4, these arrays are linearly interpolated elementwise using listing 11. With a complete heave array for wanted tripping speed, we must compute recommended heave for wanted depth. We start by finding closest depths(step 5) and then interpolate corresponding heaves(step 6).

Not shown in listing 11 is that if either values in `Y` are below 0, then the interpolation returns 0 as having a recommended value at -1 indicates that operations are not possible at those

Listing 11: Interpolation Function

```

1 function interpolateFunction = (Y, desired_x, X ) => {
2   return (Y[0] * ( (desired_x - X[1]) / (X[0] - X[1]) )
3         + Y[1] * ( (desired_x - X[0]) / (X[1] - X[0]) ) );
4 };

```

circumstances. Figure 17 shows how the app returns different recommendations for various inputs.



Figure 17: Examples of Recommendations for Heave

It is worth noticing that the algorithm for getting a recommendation is simpler for the Connection operation. In analyses for Connection, trip speed is not a factor meaning there is only one heave recommendation for each depth. As a result, we can skip steps 2-3 and directly compile a recommended heave array for every depth.

5.11 Plots

To satisfy requirement 13 in table 2, “Showing data/results graphically”, three different plots were created using Victory Native. One plot depicts recommended heave or tripping speed for each depth in an analysis. The second plot illustrates upper and lower pressure margins and well pressure throughout the well. The last plot presents the user with an illustration of estimated drilling progress. The next sections give a walk-through of how to set up the logic behind each plot. As with section 5.10, the logic behind calculating a heave recommendation for Tripping operations is the main focus. All three plots have had their planned design presented in section 4.3.

5.11.1 Recommended Heave per Depth Plot

The logic for recommended heave plot is similar to what is gone through in the recommendation section 5.10. It starts similarly but stops at step 4 in listing 10 as we need heaves at every depth. Components in Victory Native require plotting data to be given in an array of dictionaries, such

IMPLEMENTATION

as; $\{\{x:1,y:1\}, \{x:2,y:2\}\}$. Therefore it is necessary to combine the two arrays `string_depth` and `rec_heave_array` into said structure.

Listing 12: Function for Recommendation Heave Plot

```
1 function getPlottingDataHeave =
2   (analysis_data, wanted_trip_speed, wanted_string_depth) => {
3   // Line 2-25 in Pseudocode for recommended heave for getting
4   // string_depth_array and rec_heave_array
5
6   for (every depth)
7     heave_plotting_data.push(
8     {x:rec_heave_array[depth_index], y:string_depth[depth_index]})
9   };
```

Not shown in listing 12 are various small additions, like making ticks for the x-axis in the plot. It is not a necessary step for the plot to function, but it gives developers more control of the plot's final design. For setting up and adjusting plots used in the app, see the documentation pages [34]. By filling in computed data into various Victory Native components, the recommendation plot for heave looks like what is shown in fig. 18. The figure displays three cases where there is no interpolation and interpolation after either trip speed or depth.

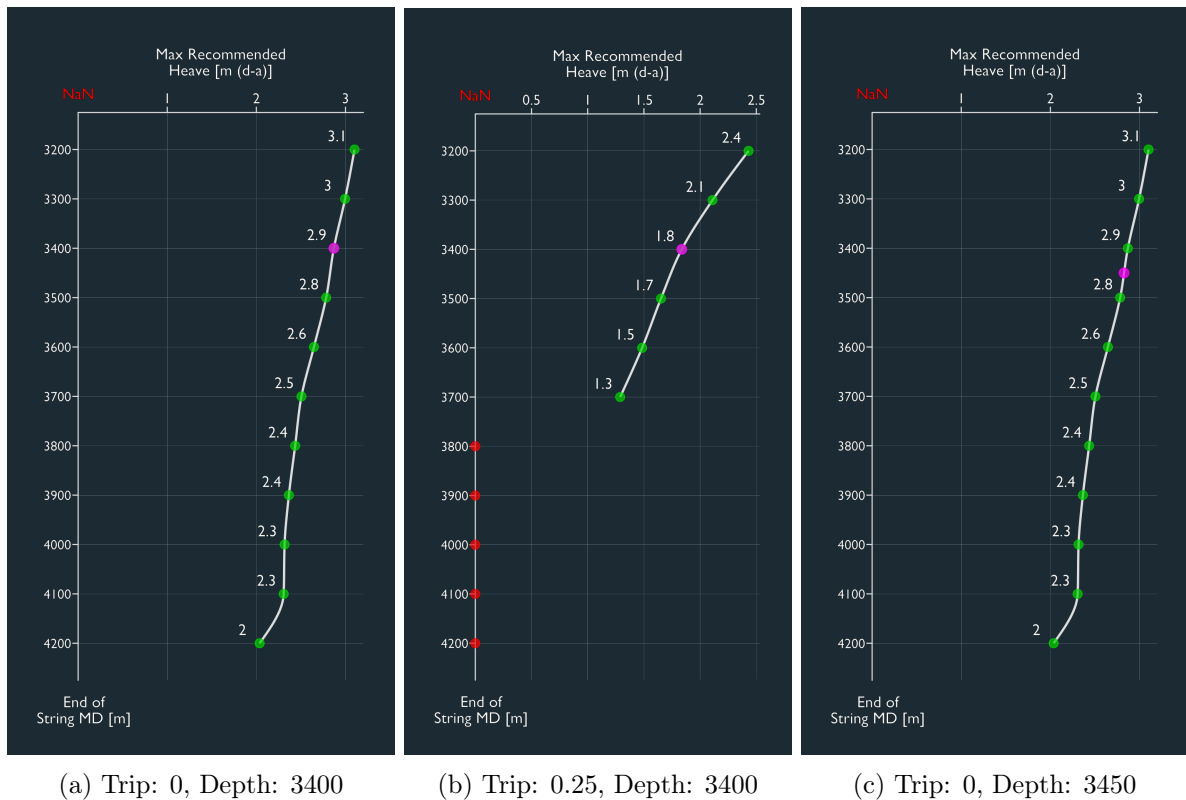


Figure 18: Examples of Recommended Heave per Depth Plots

There are two things worth mentioning in the setup of these plots. The first is that values displayed for each point are rounded down to have one decimal, but their actual values are not. This means that different dots can be labeled with the same number (e.g. 0.9), but are not placed at the same place. Secondly, the lines are not linearly interpolated but fitted to the data using monotone cubic interpolation [82]. This is implemented by specifying it as the wanted

interpolation method within the *VictoryLine* component and is done to give the line a more rounded look.

Creating Line Segments

In the case of a recommendation plot switching between valid and invalid data, it was sought to have a clear break between these points. Figure 19a shows simulation results where data at depths 3500m and 3900m are invalid. The plot draws a line between the valid and invalid points. This is problematic because it may seem to the user that it is possible to perform drill string connections at approximately 3550m with a heave amplitude of 3m(d-a) when it in reality may be ill-advised. Segmenting the plot line solves this problem, as can be seen in figure 19b, avoiding an “invalid” recommendation.

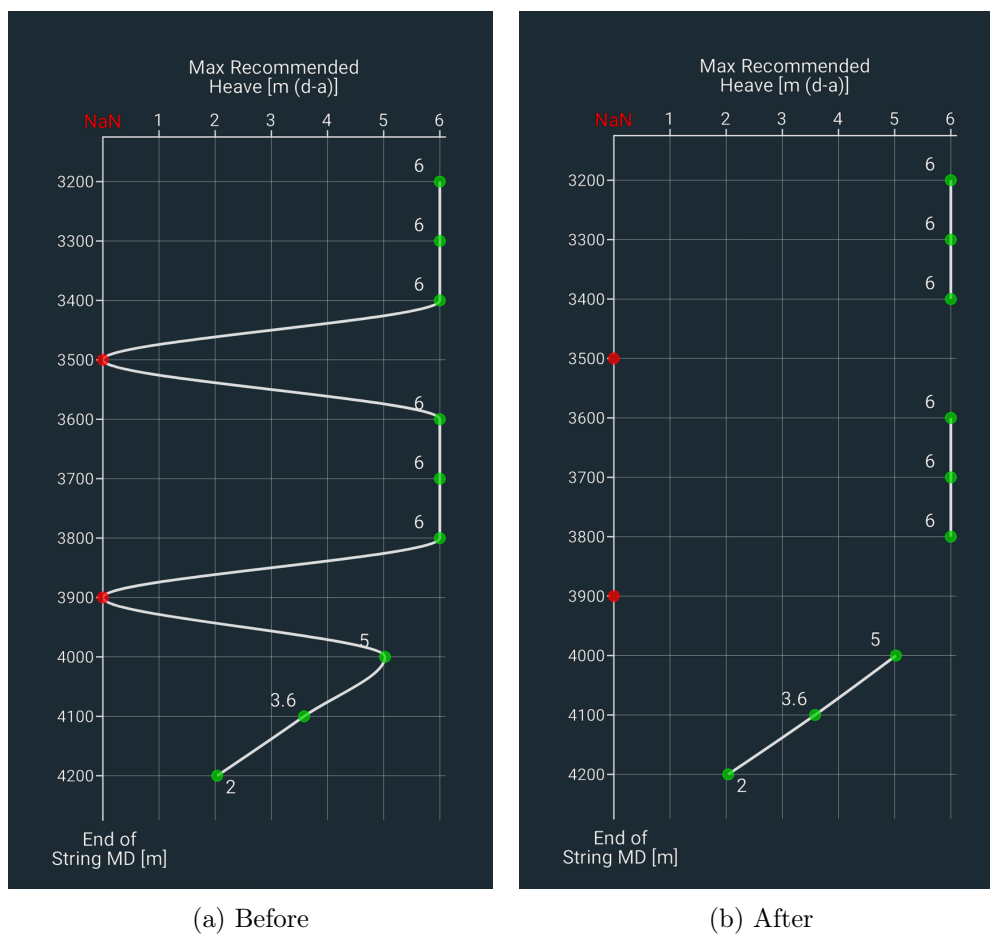


Figure 19: Unsegmented vs Segmented plots

To make a segmented plot line, we can iterate over the list of all data points and detect whenever an invalid point (recommended heave or tripping speed ≤ 0) appears in the list. Valid points are stored in a list `buffer`. Whenever an invalid point is found, the buffer is pushed to the list of segments and reset if it contains any points. After iterating over all data points, we make sure to add any remaining points in the buffer to the list of segments if there are any.

To plot these line segments, all that is necessary is to map each line segment to a *VictoryLine*

Listing 13: Pseudocode for Generating Line Segments

```

1 function lineSegments(all_data_points){
2   for (point in all_data_points)
3     if (point.x > 0)
4       //Valid point detected -> push it to the buffer
5       buffer.push(point)
6     else
7       //Invalid point detected -> push buffer to segments if not empty
8       if (buffer.length > 0)
9         segments.push(buffer)
10      buffer = [] //The buffer is reset
11 //Push remaining valid points in the buffer to the valid points array
12 if (buffer.length > 0)
13   segments.push(buffer)
14 return segments
15 }

```

component with the function `Array.map()`. This process is, in essence, the same as shown in listing 7.

5.11.2 Pressure per Depth Plot

In the problem description (1.2), task 3 was that pressure margin violations are to be graphically displayed. For the given data, every recommendation has an underlying set of pressure data with depth intervals set around approximately every 30 meters. To be able to display the pressure margin as correctly as possible, one must be aware of the four situations presented in section 5.10.

Here, the cases require more work to compute and case 4 will again be explained since the former cases are just simpler versions of this. Five of the variables in the underlying dataset are used for the pressure plot and, they are referred to as **pressures**. Table 4 presents the pressure variables.

Table 4: Variables Used for Pressure Plots

Variable	Description
md	Measured depth
margin_min	Lower pressure margin of well per md
margin_max	Upper pressure margin of well per md
min_pressure	Lower pressure in well per md
max_pressure	Upper pressure in well per md

Listing 14 is the algorithm for how case 4 is computed. The goal is to find and interpolate each pressure variable corresponding to wanted tripping speed and depth. As done earlier, we find the closest depths and tripping speeds (step 1). In step 2 we extract needed **pressures** for every combination of closest depths and tripping speeds. Then we interpolate each **pressure** after wanted data. Step 3 interpolate after wanted tripping speed while step 4 is after wanted depth.

To plot each pressure variable separately, we pair them with `md`. This creates four list of coordinates, which are plotted as the lines in fig. 20. Similar to the recommendation plot, there

Listing 14: Algorithm for Heave Pressure Plot

```

1 wanted_depth = 3250
2 wanted_trip = 0.25
3
4 //Step 1 - find closest depths and tripping speeds
5 closest depths are 3200 and 3300
6 closest tripping speeds are 0.0 and 0.5
7
8 //step 2 - extract needed dictionaries
9 pressures(T=0.0, D = 3200)
10 pressures(T=0.5, D = 3200)
11 pressures(T=0.0, D = 3300)
12 pressures(T=0.5, D = 3300)
13
14 //step 3 - interpolate to wanted tripping speed
15 pressures(T=0.25, D = 3200)
16 pressures(T=0.25, D = 3300)
17
18 //step 4 - interpolate to wanted depth
19 pressures(T= 0.25, D = 3250)

```

are some more steps done in the actual function for prettifying the plot, but those will not be explained because of their trivial nature. How the pressure plots end up looking can be seen in fig. 20.

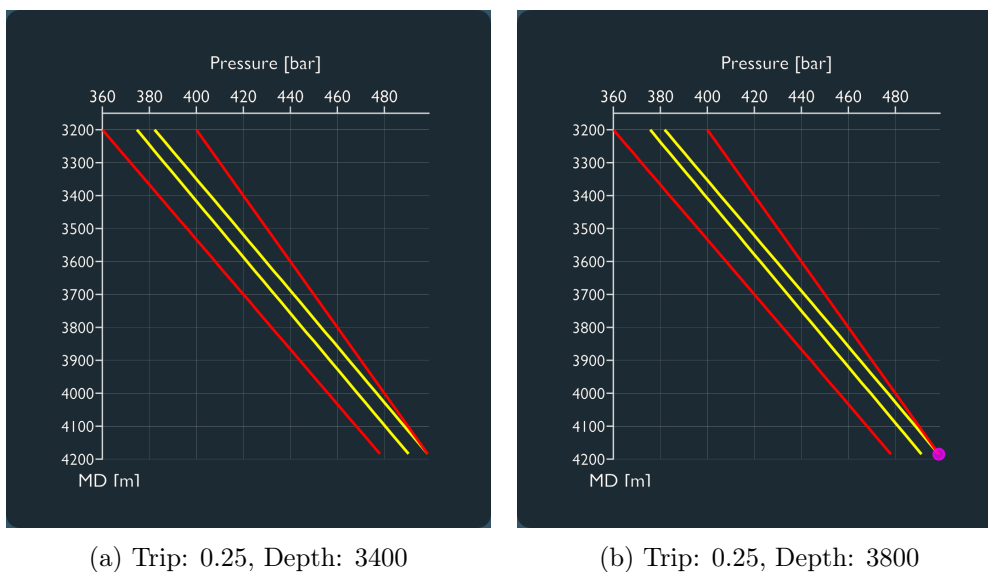


Figure 20: Examples of Pressure per Depth Plots

From fig. 20 we see that 20b has a magenta dot in it while 20a does not. The combination of tripping speed= 0.25 m/s and depth= 3800m is not possible in this analysis resulting in the pressure breach marked with a magenta dot. For both upper and lower margins, it is only the first breach for each margin that is marked.

Differences in Operations

It is worth mentioning that the drilling operation Connection has some differences in measured depth for pressure data that need to be addressed. Connection is the operation where the rig is drilling into the Earth's crust and the drill bit has yet to reach the analysis' deepest depth. Tripping is pulling or pushing the string bit through the already drilled well. This means that pressure plots in Tripping operations cover the entire depth range of the analysis, while Connection pressure plots cover the start of the analysis to the specified depth. Figure 21 displays the differences for each operations.

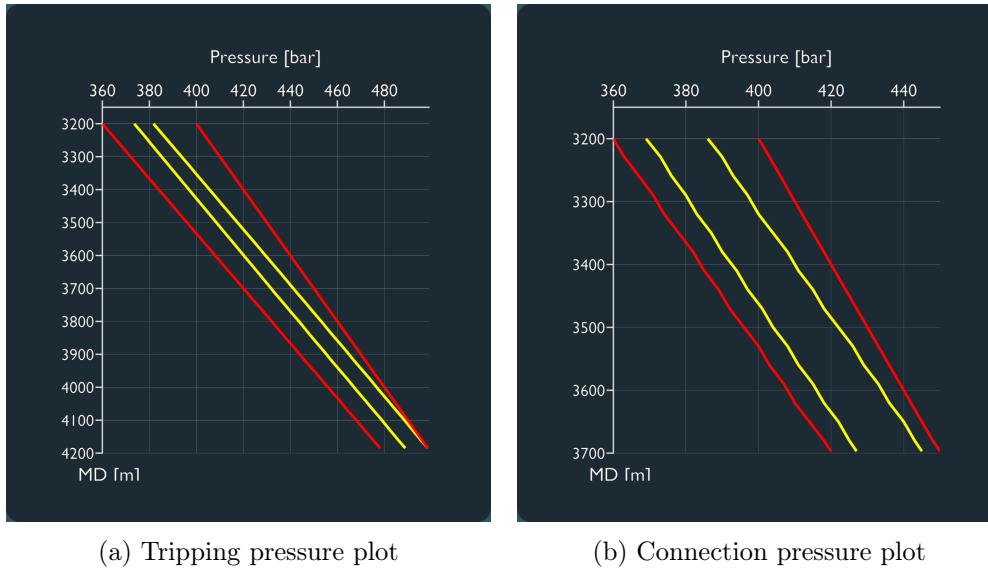


Figure 21: Different Pressure Plots at Depth: 3700

5.11.3 Forecast Connection Plot

Implementing a feature to estimate progress in operations was of high priority, as mentioned in requirement 8 for table 2. Without enough time to develop solutions for both drilling operations, HeaveSIM expressed that it was the operation Connection that should be prioritized. The different elements that will be reviewed in this section are; generating a weather/heave forecast and the logic behind the plot itself.

Weather/Heave Forecast

Without access to real-time weather forecasts, dummy data had to be created. That is done by simplifying tide behavior into a sine wave with a period of two in the course of one day. In reality, tides have no effect on floating rigs, but their characteristic oscillation was thought to make great example data for how different wave heights affect the rig. To give a vertical shift, 2.2 is added into the sine wave. Noise is implemented in the dummy data for variance. To summarize, eq. (2) shows the mathematical expression used for computing waves, and fig. 22 shows a plot of its result with and without noise.

$$\text{tide} = \text{Amplitude } A \sin\left(\frac{\text{period per day}}{2} \frac{24 \text{ hours}}{24} 2\pi i\right) + \text{vertical shift } 2.2 \pm \text{noise } 0.375, \quad A \in [0, 1), \quad i \in [0, 23] \quad (2a)$$

$$\text{tide} = A \sin\left(\frac{1}{6}\pi i\right) + 2.2 \pm 0.375, \quad A \in [0, 1), \quad i \in [0, 23] \quad (2b)$$

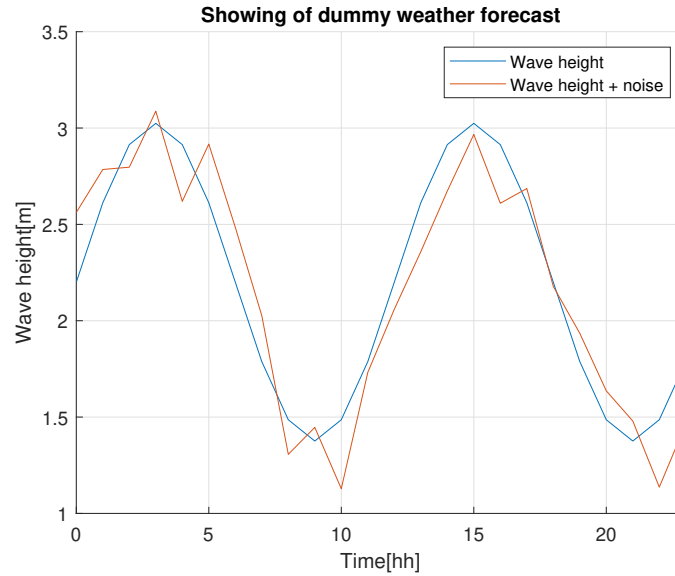


Figure 22: Example of Generated Tide

The reason A is inclusive of 0, but exclusive of 1, is because that is how JavaScript's `Math.random()` works [83]. The same applies to the noise of 0.375, meaning that all values in ± 0.375 is possible, except $+0.375$. The final step is to transform the dummy tide data to rig heave. As mentioned in section 4.4, it was impossible to compute this transformation due to lacking RAO [67]. Instead, the dummy data is multiplied with a gain of 0.85 to pretend there is some form of transformation happening.

Plot Setup

Now that we have a heave forecast ready to use, it is time to set up the logic for showing estimated progress. In section 6.6.2 an input screen is presented where the user will have to input three variables: `ROP`, `starting depth` and `hours ahead`. The default unit for `ROP` is m/h, and its value can vary a lot. `Starting depth` is where the drill bit is currently, while `hours ahead` denotes how many hours into the future the user wants a forecast for. When all variables are input, we find and compute recommended heaves from `starting depth` to `starting depth + ROP · hours ahead` at hourly intervals.

With forecasted and recommended heaves ready, we compare them to each other to estimate drilling progress. Listing 15 presents a pseudocode for how it is done. In essence, the point is to check if the heave forecast is greater than the recommended heave at computed depths. When doing this, it is important to be aware that the forecasted heave is paired with the hour we are checking, while recommended heave is paired with depth. To better explain each step,

the final result of this plot is now shown in fig. 23 with given inputs. In the figure, we see elements explained in early design (4.3), with a magenta line for recommended heave and yellow for forecasted heave. Added to the early design is the red and green dots on the hours axis, which represent if it is recommended to operate or not.

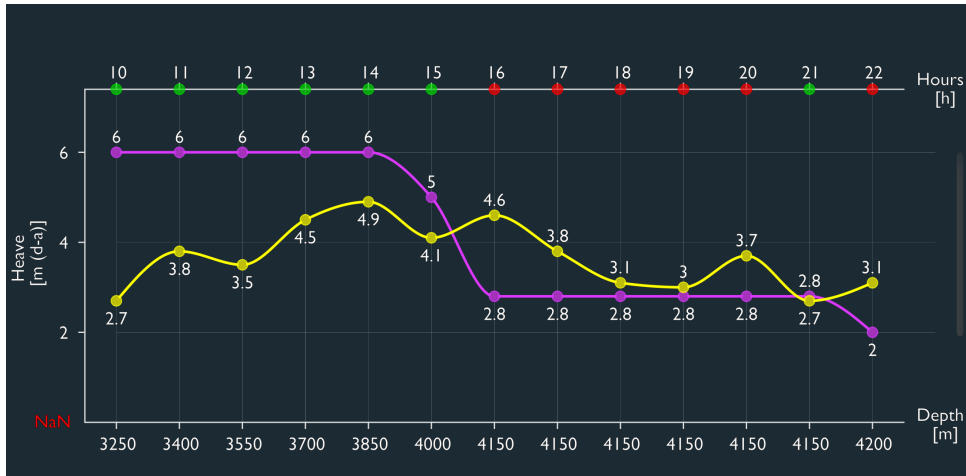


Figure 23: Forecast Plot
 ROP: 150, starting depth: 3250, hours ahead: 23

To draw dots and lines in the plot, there is used a custom-made array of dictionaries called `all_data`. This consist of following variables; `hours`, `depths`, `recommended_heave`, `forecasted_heave`, `index`, and `npt`. By now, all but the last two variables should be self-explanatory. `index` is used for coordinating all variables inside the plot, and `npt` is for the green or red dots on the hour axis. The pseudocode in listing 15 starts by comparing the two heaves with each other at 10 o'clock with starting depth 3250. With the recommended heave (6m) being greater than forecast (2.7m), it is recommended to continue operation. As shown in the pseudocode, this means that all variables are pushed to `all_data` and depth is incremented by the ROP (150m). At hour 16, we see the first red dot because recommended heave is lower than the forecast. For those situations, all variables are pushed to `all_data`, but depth is not incremented. This means that recommended heave and depth stay the same for the next comparison, but hours are incremented, resulting in forecasted heave also changes. These comparisons continue until either hours ahead or max depth for the analysis is reached. If max depth is reached, we push all variables at that depth and stop comparing. This is shown in the figure example.

Listing 15: Pseudocode for Comparing Heaves in Forecast Plot

```
1 function compareForecastAndRec(...) {
2   all_data = []
3   depth = starting depth
4   for ( every hour from current hour to hours ahead )
5     if (depth of rec data > max depth analysis)
6       // max depth reached
7       push data at deepest depth to all_data
8       break
9     if (forecast_heave < rec_heave)
10      //Production can continue
11      push data at depth and hour to all_data
12      increment depth by ROP
13    else
14      //Production stop
15      push data at depth and hour to all_data
16    return all_data
17 }
```

Result

This chapter presents the resulting app from implementing the app specification, described earlier in chapters 4 and 5. The sections below describe and show the app in its final form, with discussion following in chapter 7. To answer part 5 of the problem description, this chapter serves as the user guide by showing the app from the user's perspective. A video user guide documenting all the features in the app can be found by using the link <https://youtu.be/RCgGqDRtBUQ> [84].

6.1 Infrastructure

Figure 24 shows an overview of the app infrastructure. The screens are connected using screen navigation, which also involves internal communication. However, internal communication does not mean navigation, and they are therefore separated in the figure. The external communication are the API calls from chapter 3. The screens are divided into different categories.

First is the Introduction category, which are the Login and Home screens. These screens communicate with HeaveSIM using the first and second API calls like previously in 3.

The next category is Settings. This category only contains the Settings screen and is connected with the Introduction category via screen navigation.

Then there is the Defined Wells & Analyses category, which consists of the Wells and Analyses screens. These screens use the data from the second API call, which is passed from the Introduction category using internal communication. The component Dynamic Analyses uses the third API call from section 3 in order to get the simulation data. It then passes this via internal communication to the next categories. From the Analyses screen, one can navigate to the final two categories, which are Recommendations for Connection and Tripping.

In the Recommendations for Connection category is the Connections Recommendations Screen. This screen uses tabs to display the two components and the internal communication between them. From here, one can navigate to the Forecast Plot screen, which gives the forecast plot for Connection. More on this in section 6.7.

The last category is Recommendations for Tripping, which gives recommendations for either heave or tripping speed. Simulation data from HeaveSIM is provided via internal communication from the previous category.

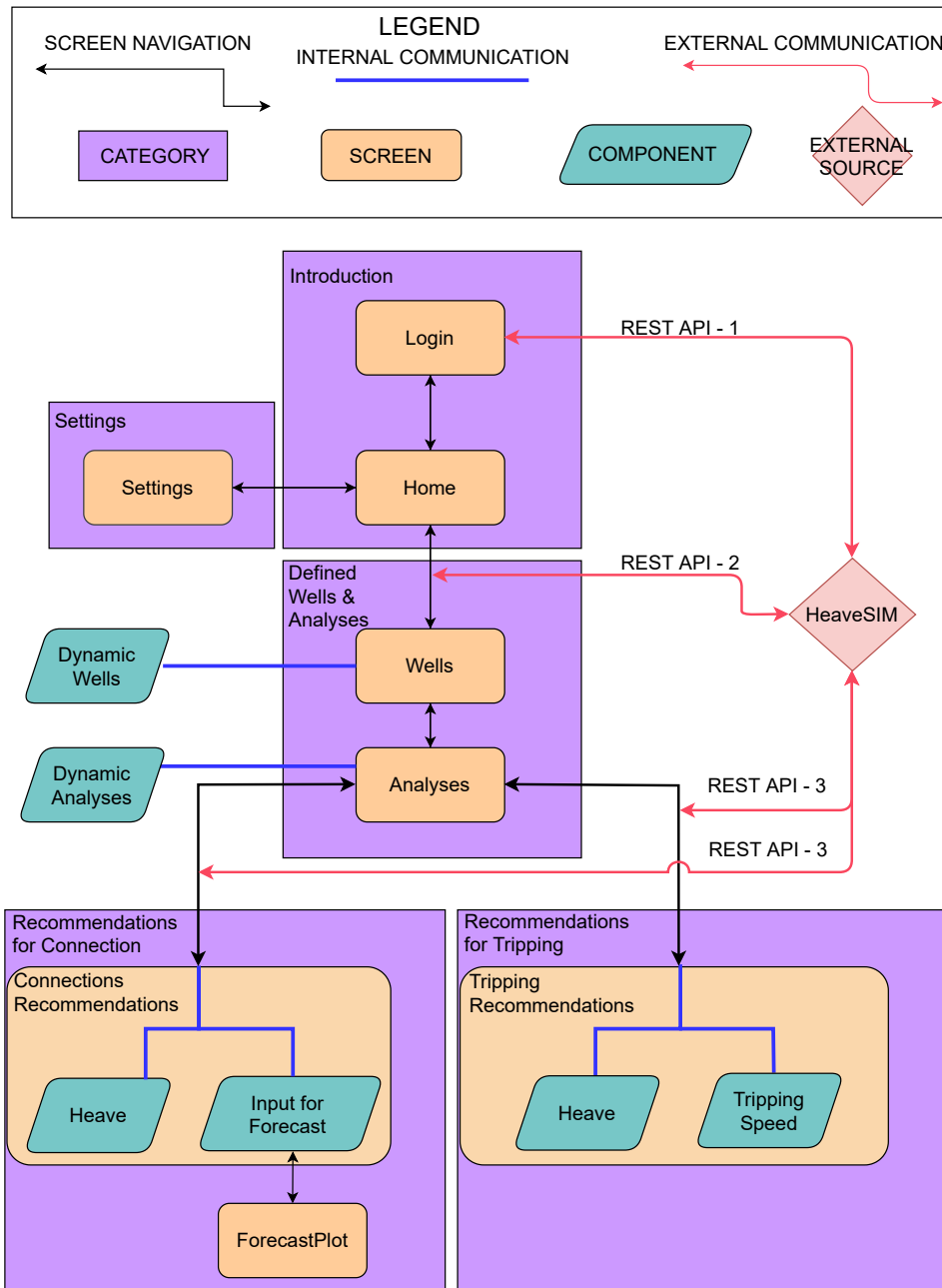


Figure 24: Illustration of App's Infrastructure

6.2 Login Screen

Shown in fig. 25 is the new Login screen. It is based on the design shown in chapter 3, but has several changes according to points made in section 4.3. The header at the top of the screen will scale to fill the screen width and will fade out when the “Login” button is pressed [85]. Simultaneously, the login input fields will fade into view and be placed above the keyboard. To do this, a component called *KeyboardAvoidingView* was used because it supplies the functionality to displace other components to never be covered by the keyboard [86]. As long as the input checks described in section 5.9 detect a missing username or password, the “Login” button has a gray color to indicate that it is disabled as seen in fig. 25b.

Pressing the “Cancel” button or hiding the keyboard will also fade out the input fields and fade in

the header. The “Login” button incorporates the *ActivityIndicator* functionality. When pressing “Login” after valid credentials are input, the app will fetch and store a login token with the username in user settings. The username and password are removed from the input fields such that the user sees clear input fields when logging out.

The background image is a stock photo from [87]. This is included in two resolutions to suit the different screen sizes on phones and tablets. The image is licensed for demonstrational purposes. Before the app is released to the market, the license must be extended to cover sales, etc.

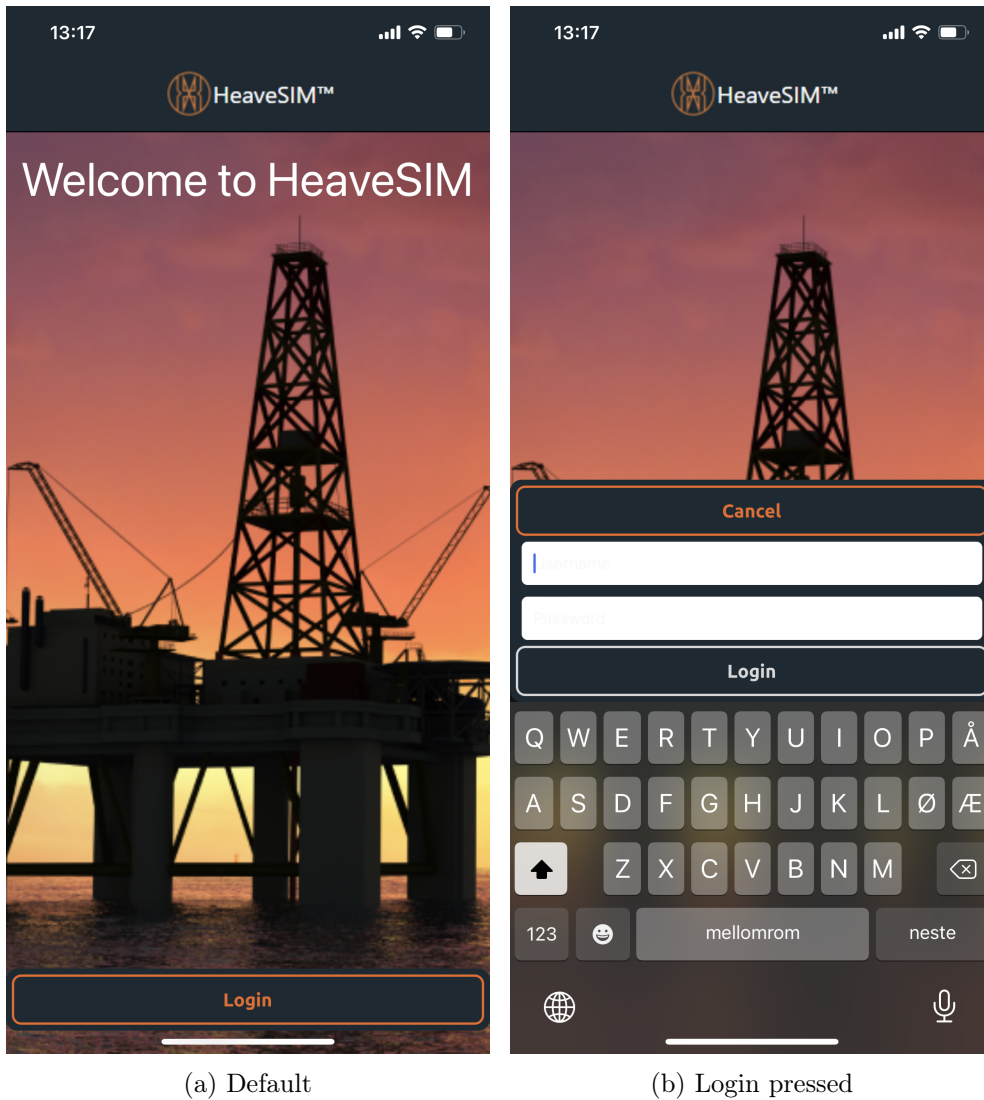


Figure 25: Login screen

6.3 Home Screen

Similar to the Login screen, a stock photo is used as the background image with the same license from [88]. As the welcome message is placed in the middle of the screen, a slight blur is added to the background to emphasize this content. The new infrastructure of the app means that the user is navigated to the Wells screen by pressing the “List of Wells” button. This button incorporates the *ActivityIndicator* functionality.

On the top left of the screen is a hamburger menu icon. Pressing it opens the sidebar. The sidebar uses Animated values to slide in and out of view. The sidebar contains an image of the HeaveSIM logo, the user’s username, and three buttons. The image is currently a placeholder for what could be a profile picture in the future. Buttons for navigating to Wells and Settings screens, as well as a “Logout” button, make up the rest of the sidebar content. Pressing the hamburger icon again or the background hides the sidebar.

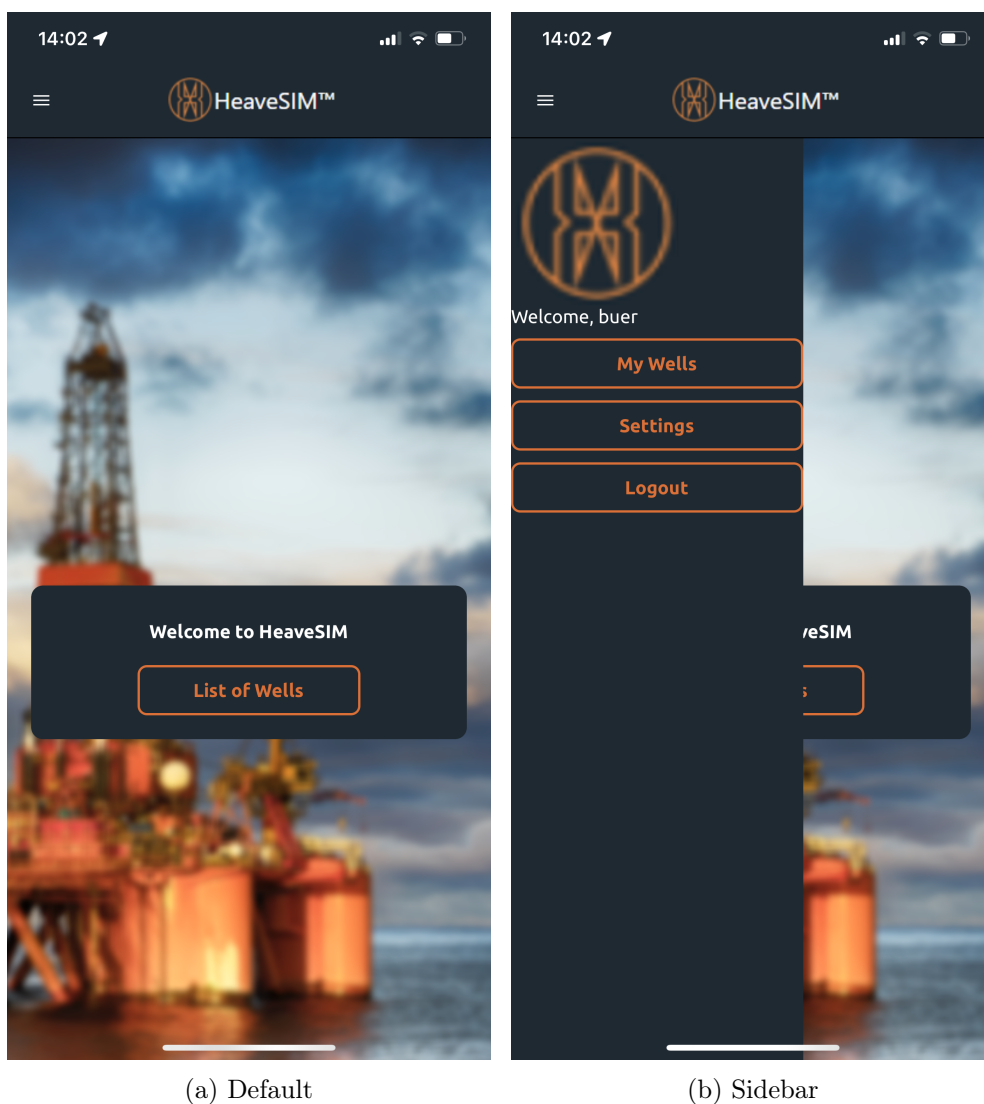


Figure 26: Home screen

6.4 Settings Screen

The Settings screen is new to the app and currently allows the user to change units and the color scheme. Figure 27a shows how the Settings screen looks by default. Units and colors can be set with the *Dropdown* menus [89], which contain all available units and color schemes. Additionally, two unit presets “Metric” and “USCS” can be selected. The data in each *Dropdown* menu is dynamically updated, as described in section 5.3 and section 5.4. The *Dropdown* component decides internally whether the alternatives appear above or below such that they never appear outside the screen.

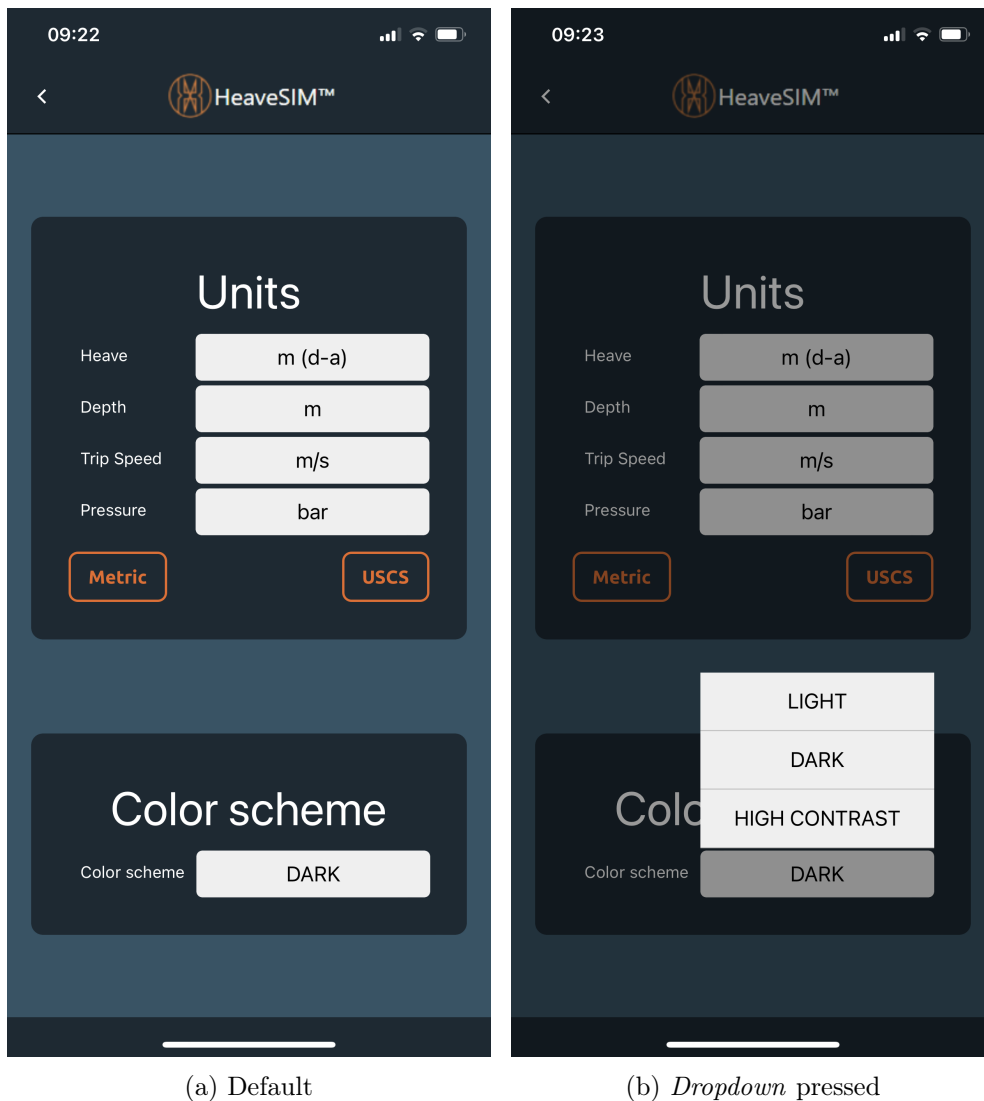


Figure 27: Settings screen

6.5 Wells & Analyses Screens

The results of the Wells and Analyses screens are displayed in fig. 28. Both screens dynamically display the wells and analyses in sections as explained in section 5.7. The Wells screen has a relatively simple design and only displays the wells in the order of when they were defined. By pressing the “Go to Analyses” button, the app will navigate to the Analyses screen.

The Analyses screen has the same design as the Wells screen, only more detailed. The header

shows the well name and depth. Each analysis is displayed inside of a *ScrollView*, allowing the user to scroll the analyses list if there are more than three analyses. Each analysis contains the ID, depth range, month and year, and operation type. The “See Recommendations” button incorporates the *ActivityIndicator* functionality and navigates the user to the recommendation screen for that analysis.

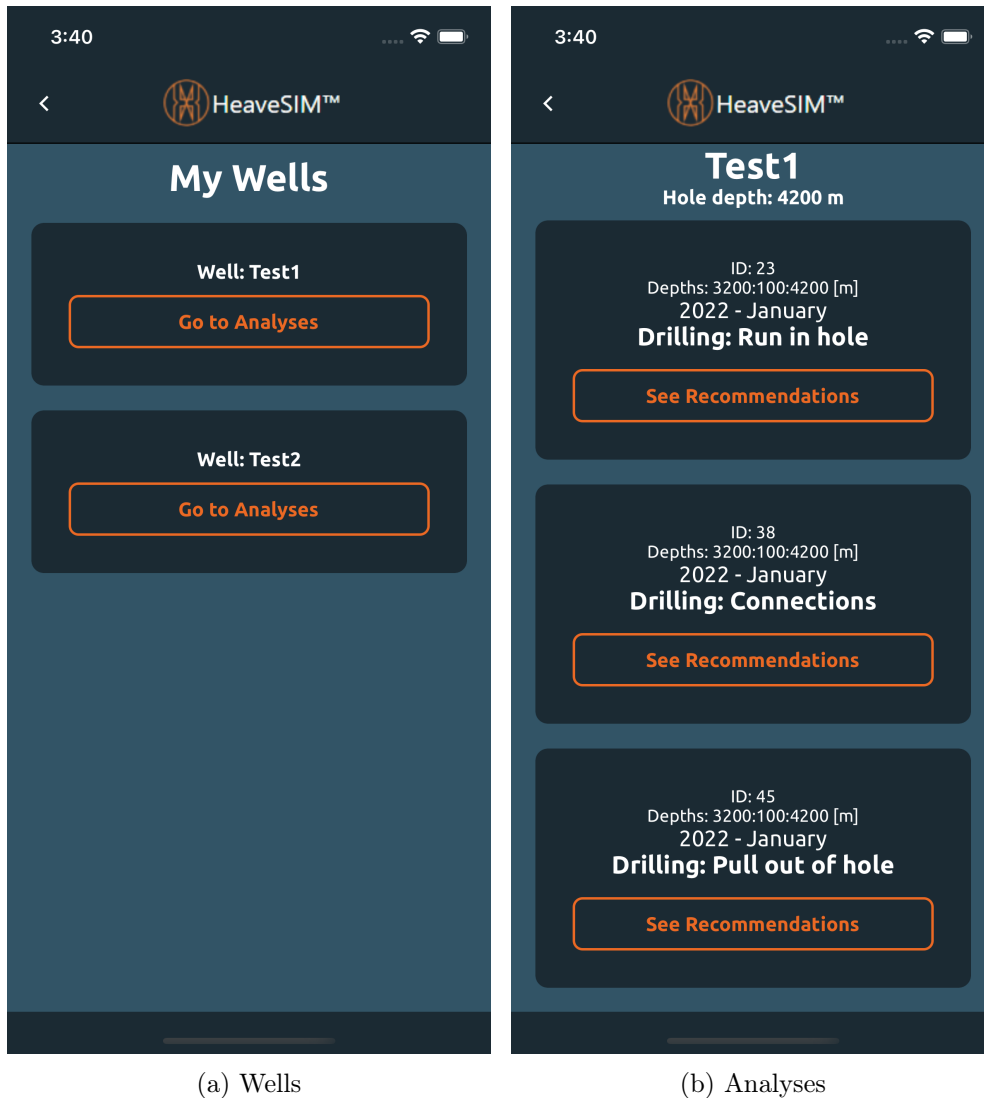


Figure 28: Wells & Analyses screens

6.6 Connection Recommendations Screen

Pressing “See Recommendations” for an analysis of the type *Drilling: Connection* navigates the user to the Connection Recommendations screen. This screen has adopted the use of tabs, *Heave* and *Forecast* (fig. 29). The contents of the tabs will be shown in the subsequent sections.

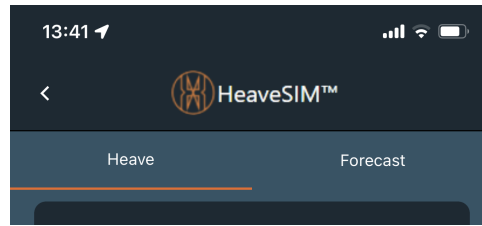


Figure 29: Connections Recommendations screen

Swapping tabs use the screen's states to keep track of which tab is in focus, i.e. currently shown. Clicking on an unfocused tab will set the orange bottom border to that tab and slide the content left or right to display the correct content using `Animated.Value` objects. The tab content is rendered side by side in a `View` container of width `2*screen_width`, as if they were on the same row. This implementation keeps both tabs rendered simultaneously.

6.6.1 Tab 1: Heave

The left tab on the Connection Recommendation screen lets the user specify situations to get recommendations for. When inputs are given, the analysis data is processed, and results are presented on the screen using elements reviewed in section 5. Figure 30 shows how the tab looks like with inputs given.

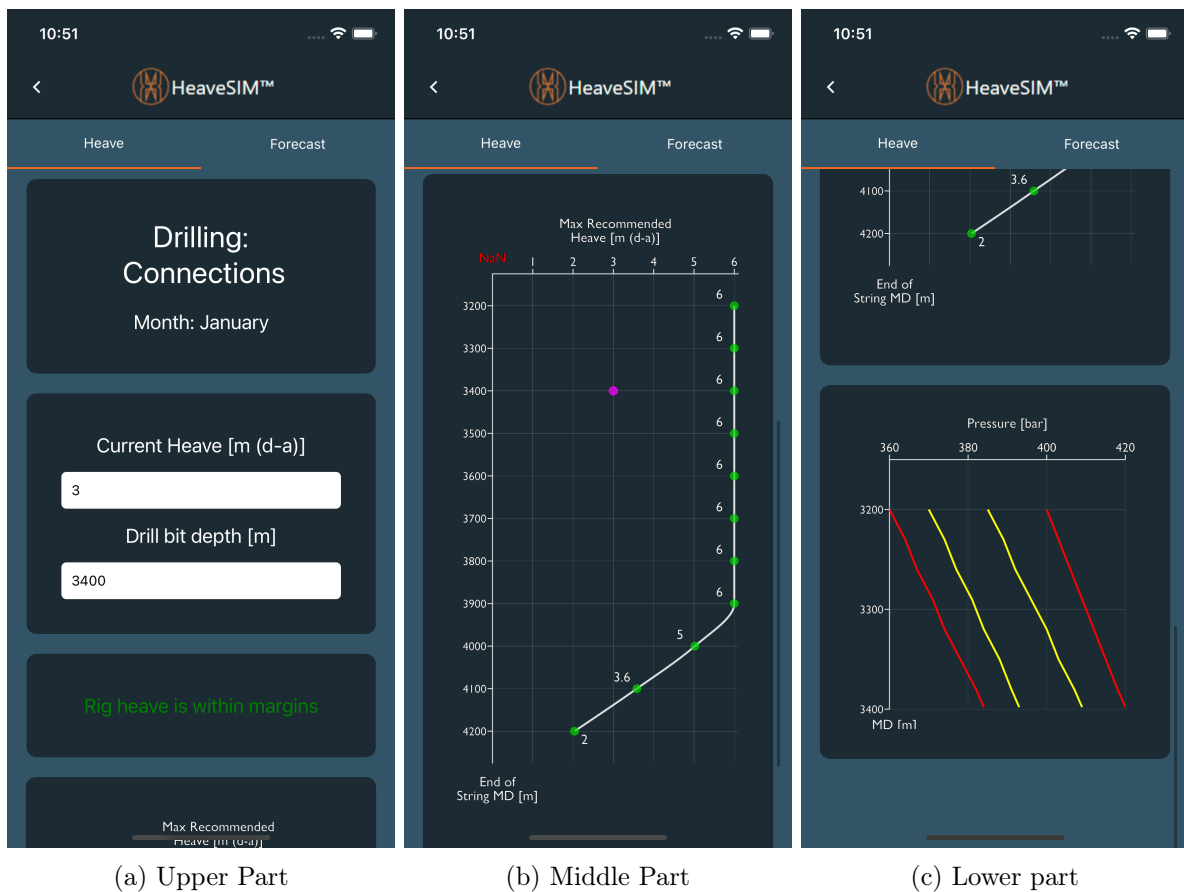


Figure 30: Recommendation Connections Heave Tab

The tab starts by displaying key information about a chosen analysis in the upper part (30a). Beneath it are input fields for heave and depth with descriptions of expected units. Recommendation for Connection is given based on whether the input heave is less or greater than recommended heave. Therefore, the recommendation is given qualitatively as a text without a numeric value. With that boolean condition as a baseline, the recommendation field displays one of three strings that are shown in table 5. In earlier work, the input section also contained a button for requesting recommendation, but this button is now replaced with automatic updates after an input field is exited.

Table 5: Recommendation Strings for Connection

Case	Recommendation	Color
No data	“Please input data”	White
Safe to operate	“Rig heave is within margin”	Green
Unsafe to operate	“Rig heave exceeds pressure margin”	Red

In the middle part (30b), a plot with recommendation for every depth is shown. As mentioned earlier, analyses for Connection only have one recommendation per depth. Thus the recommendation plot is not affected by input values. Input values are marked with a magenta dot on the corresponding coordinate. The remaining parts on the tab are presented in section 5.11.2.

6.6.2 Tab 2: Forecast

The second tab is for users to specify what they want to see in the forecast plot. The tab contains three input fields with restrictions for valid inputs. If not all fields have acceptable inputs, the “Continue” button is disabled. When correct, pressing the button will navigate the user to the Forecast Plot Screen. Figure 31 presents the input screen with different inputs given and how the tab responds.

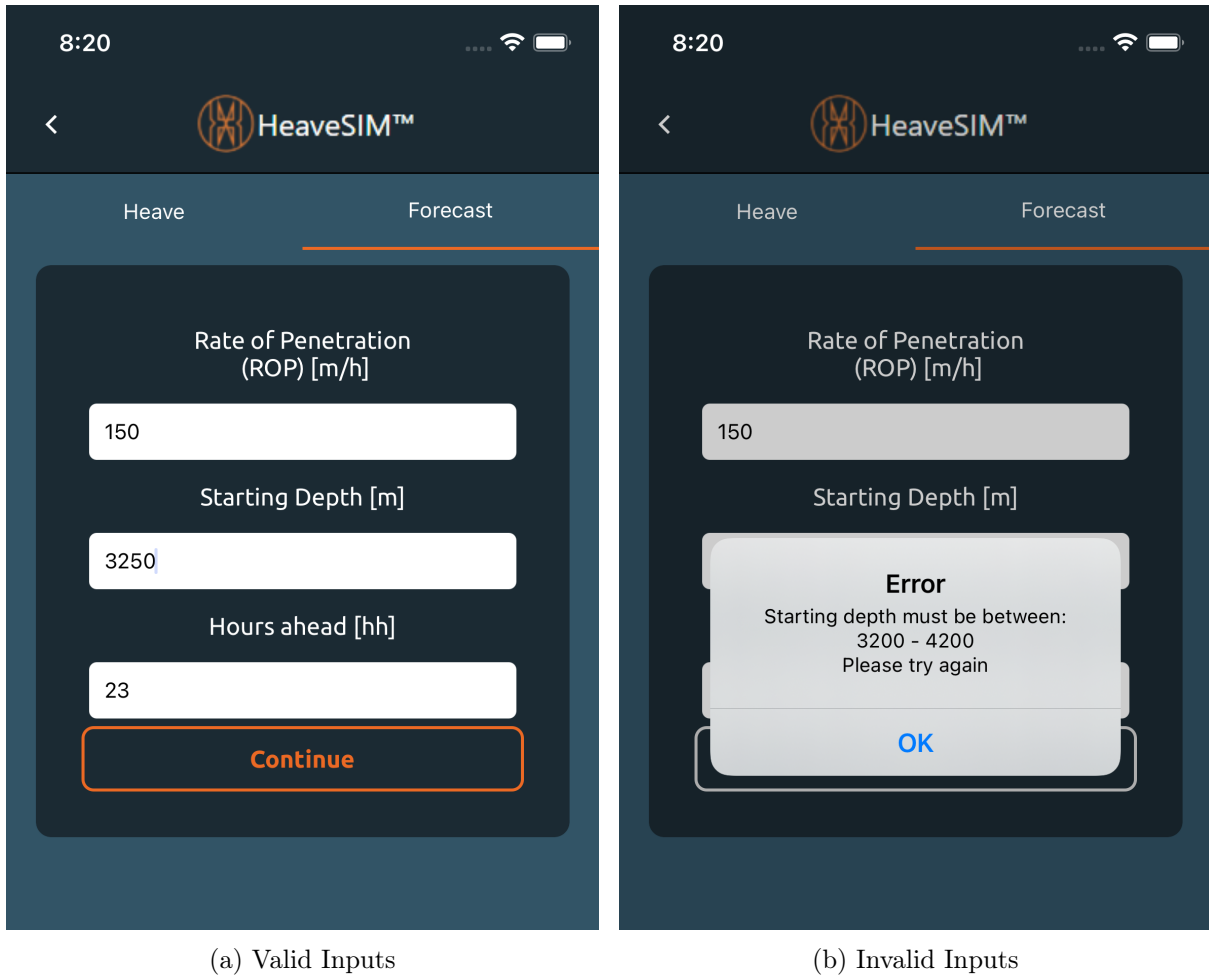


Figure 31: Forecast Connection Input Screen

Input for forecast has two input fields, “ROP” and “Hours ahead”, that are not restricted from analysis data. The latter accepts values between 0 – 23 because that is the range of the dummy data, and the former accepts values from 0 – 1000 in meters.

6.7 Forecast Plot Screen

The content on the Forecast plot screen is placed sideways to make the plot more readable. On the iPad, the screen lock must be switched on. As mentioned in section 5.11.3, the dots placed on the upper axis **Hours** signify whether drilling operations can continue or not. On the rightmost side, there is a legend with descriptions of the elements in the plot.

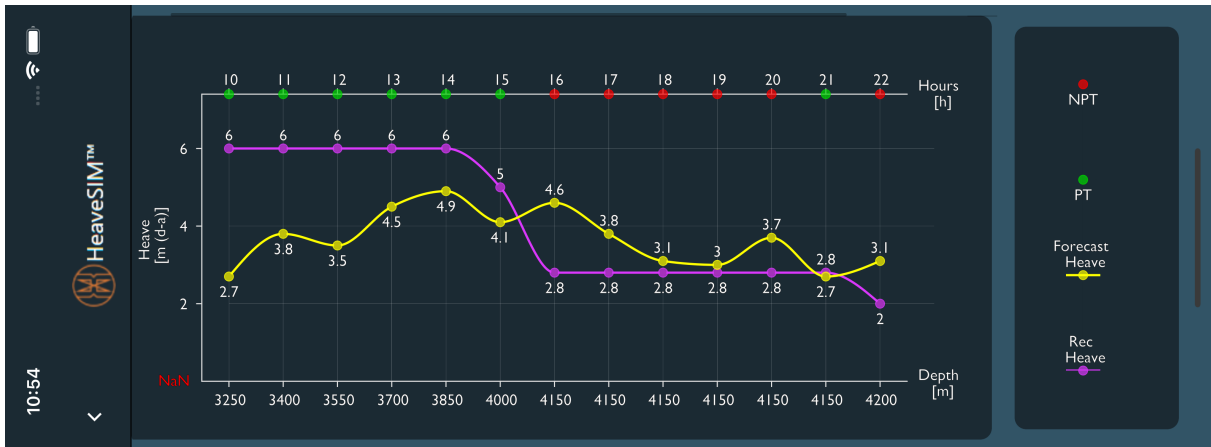


Figure 32: Forecast Plot
 ROP:150, Starting Depth: 3250, Hours ahead: 23

6.8 Tripping Recommendations Screen

This screen corresponds to the Connections Recommendations screen in section 6.6, but is relevant for Tripping operations (Run in Hole and Pull out of Hole). The screen functions exactly like the Connection Recommendations screen, the only differences being tab names and tab content. For this screen, the tabs are “Heave” and “Trip Speed” and their content is described in the subsequent sections, respectively. Both tabs are similar to the Heave tab on the Connection recommendations screen shown in section 6.6.1.

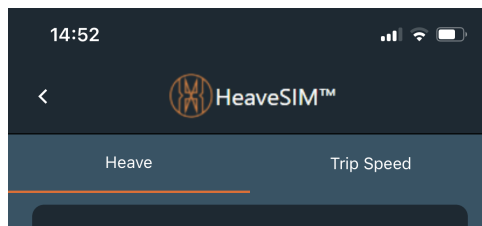


Figure 33: Tripping Recommendations screen

6.8.1 Tab 1: Heave

Much of what is seen on this tab is similar to elements already reviewed in section 6.6.1. Notable exceptions are the recommendation message and how input fields affect the plot. Recommendations that are returned to users for tripping operations are presented in table 6. Two different situations can be seen in fig. 34a and fig. 35a. In the former, the units are set to USCS to showcase what the app looks like after changing units in settings.

Table 6: Recommendation Strings for Tripping

Case	Recommendation	Color
No data	“Please input data”	Red
Safe to operate	Recommended number value + unit	White
Unsafe to operate	“Not possible”	Red

The recommendations for every depth are calculated based on what value of trip speed is specified by the user. Since the x-axis is designed to dynamically match the largest number of those recommendations, specifying a higher tripping speed might lower values in the axis because a higher tripping speed often requires lower heave. Moreover, the magenta dot representing the input data is now placed to give a recommendation for what is specified. This is unlike Connection, where the input values were just placed on the plot. The outcome of this change is that the magenta dot is now always placed on the recommended line instead of being freely placed in the plot.

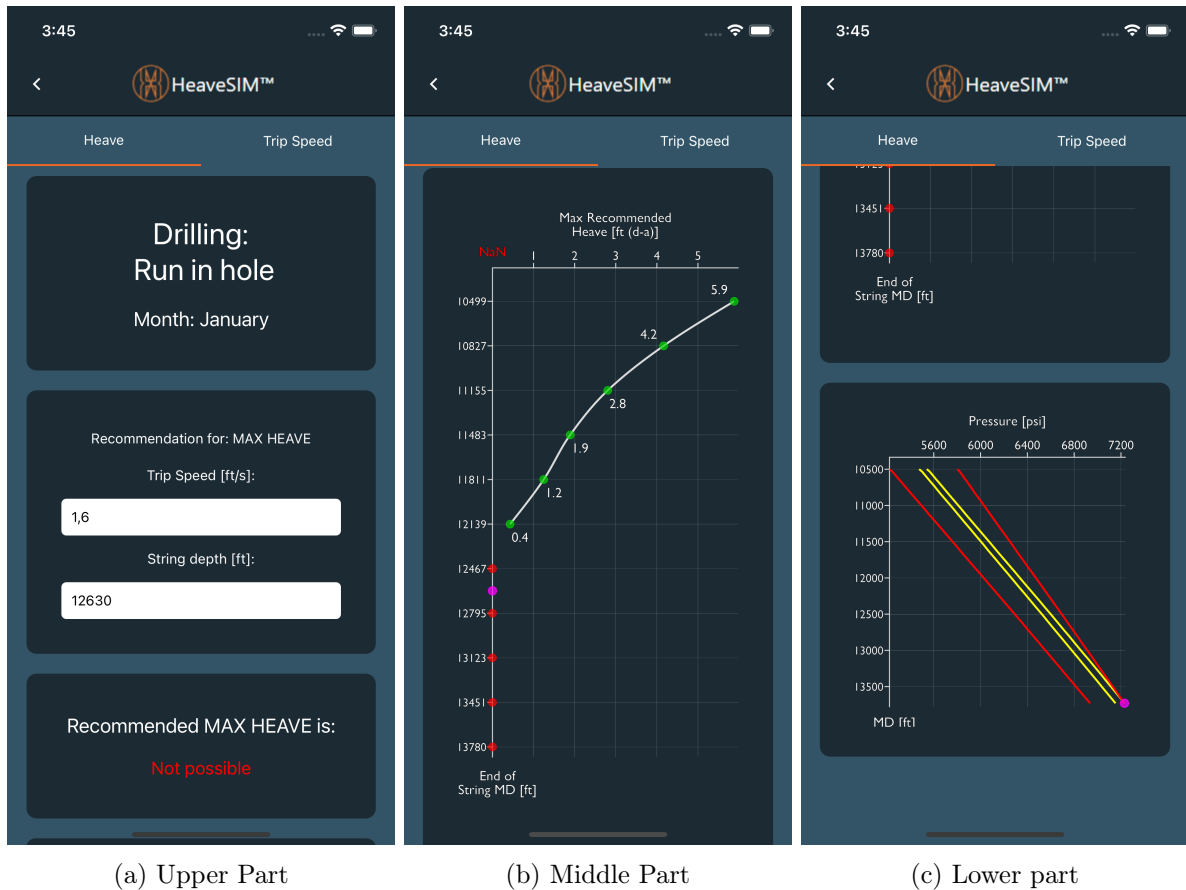


Figure 34: Recommendation Tripping: Heave Tab

6.8.2 Tab 2: Trip Speed

The same changes reviewed in previous section are applicable for this, but with one difference. Tripping speed and heave have swapped roles, meaning it is tripping speed that is recommended based on specified heave. Figure 35 shows how the tab looks like.

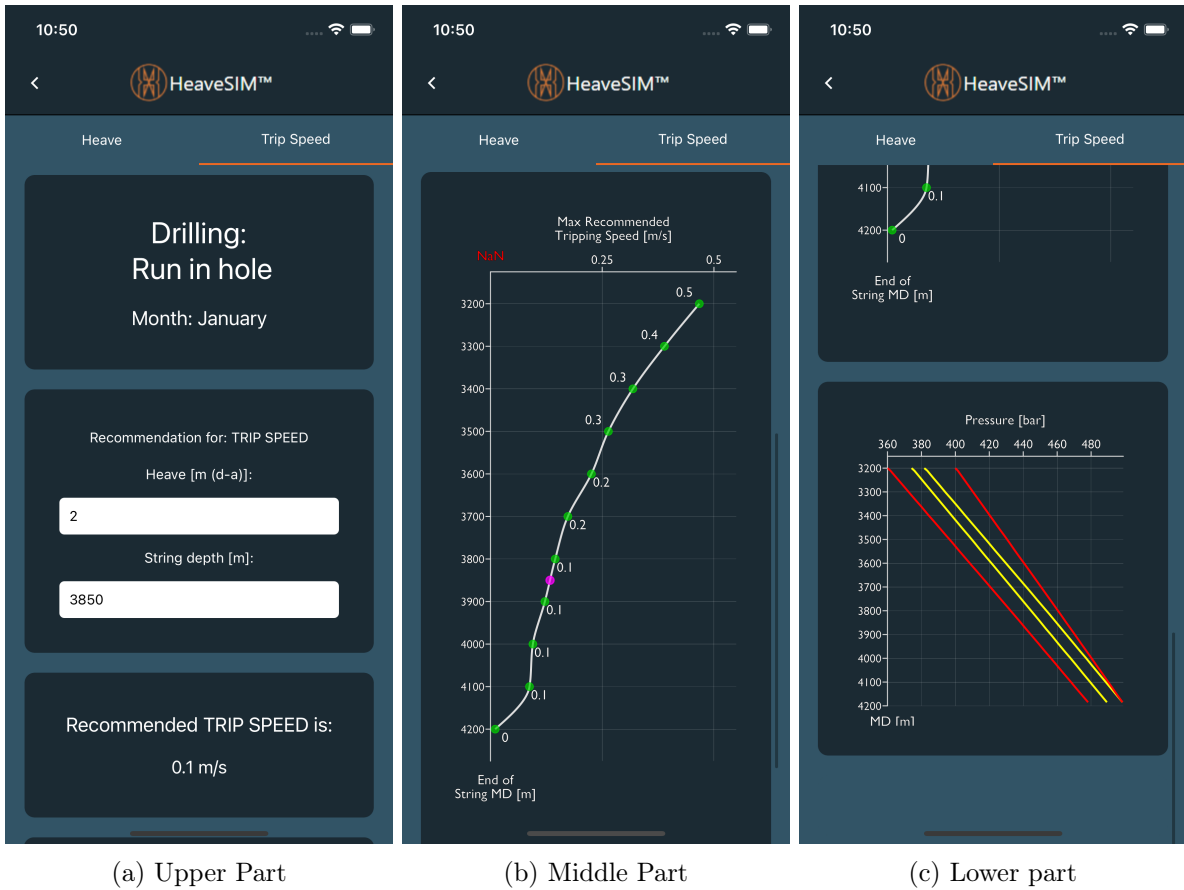
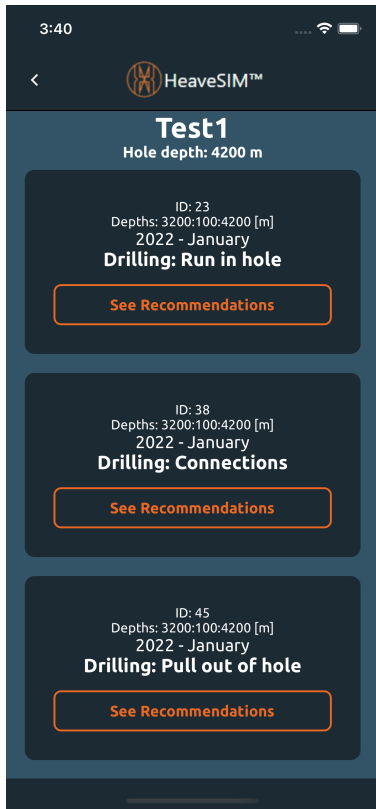


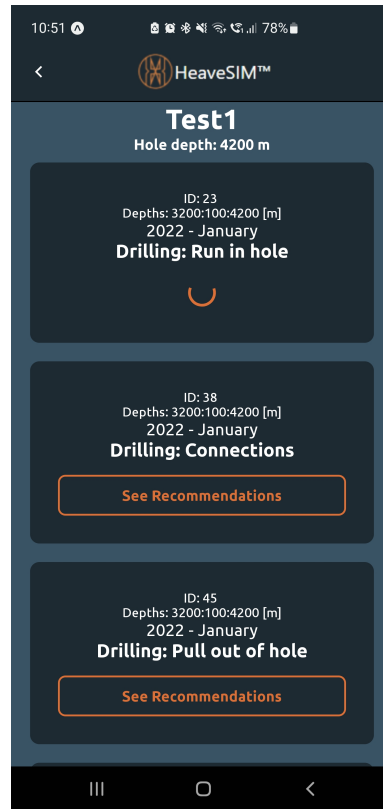
Figure 35: Recommendation Tripping: Tripping Speed Tab

6.9 The App on Different Platforms

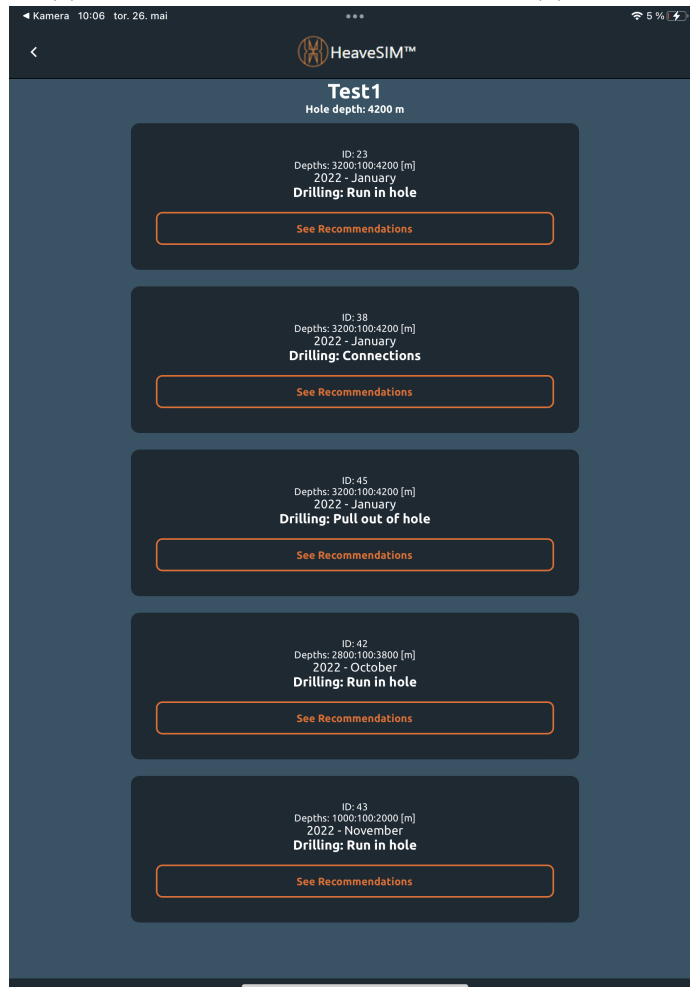
Although the app supports multiple platforms through RN, some adjustments were made in order to make it suit the iPad's larger display. As seen in fig. 36, the app is fully functional for iOS, Android and iPad.



(a) iPhone



(b) Android



(c) iPad

Discussion

This chapter contains discussion mainly of the results presented in chapters 5 and 6. Throughout this chapter, the app specification in section 4.2 will be referenced to clearly state to which degree each requirement is fulfilled.

7.1 Infrastructure

The infrastructure of the app is not tangible in the same way as the screens and is therefore not as simple to discuss. However, it is a vital part of the app as it defines how communication happens internally and externally.

As the infrastructure from the specialization project [2] worked well, the new infrastructure continues from the old with some differences. Examples of this are the new screens Wells and Forecast plot, tabs inside the Recommendation screens, and more internal communication between screens and scripts.

As intended, the infrastructure has not changed much. A less visible change in the infrastructure is dynamic navigation. Requirement 1 in table 1 states that the navigation needed to change from static to dynamic. As a result of this, more operations can now be added without having to rewrite the code for navigation.

7.2 Features Throughout the App

Figure 9 shows the different statusbars. The functionality is great to have as it makes the app look like it better integrates with the device's colors. Figure 9b displaying the statusbar with black text could do with greater contrast, achievable through a lighter shade of blue or simply using the statusbar with white text.

Regarding input fields that skip to the next as a feature throughout the app, it is a small detail that improves the UX. For iPhone, the numerical keyboard lacks an "ENTER" button, thus disabling this feature. The numerical keyboard lessens the chance of invalid inputs like letters, so it was preferable despite the missing input field function on the iPhone.

As mentioned in section 5.5.1, the final solution for applying a new color scheme to the app is far from optimal. Creating a new stack navigator removes all previously visited screens, resetting all plots and input fields. The solution is likely to implement a new global state management system and use listeners on all pages to detect a state change [90]. Storing the styles and color scheme in global states would mean all screens could detect whenever a change occurs.

A global state system would be time-consuming to learn and would require significant changes to the internal communication. Therefore, it was not implemented at the late stage of the project. The functionality to alter and add new color schemes was more important for HeaveSIM than perfecting existing color schemes. Details on how to edit color schemes are given in section 5.3.

State persistence and cookies were attempted features that would fulfill requirements 9 and 10 in the app specification. The missing possibility to store data locally and Expo lacking support for cookies put these features on hold. Both these requirements would make the app easier to

use and certainly lower the threshold for launching the app and continuing the previous session. One idea for implementing state persistence is described in section 7.4 and section 8.1.

7.3 Login Screen

Compared to the starting point, the revised Login screen gets rid of the PoC feel. Initially, the animations on the Login screen were implemented explicitly, i.e. functions that perform animations on the screen were written by the group. Later, the *KeyboardAvoidingView* component replaced the explicitly coded animations as it has the necessary moving (transform) animations integrated by default.

As mentioned, Hevelock must get an extended license if this background photo is put in the production-ready application. Changing the image is just a case of replacing the image file in the assets folder with a new one that has the same dimensions. The current background image has colors that suit the color scheme of blue and orange well. Should new color schemes be designed and implemented, one idea would be to add more background images that match them better.

7.4 Home Screen

The new Home screen looks more production-ready after the implementation of a sidebar and background image. The blurred background makes the welcoming section stand out. The sidebar would benefit from some more content; it looks somewhat empty at the moment. Exchanging the placeholder image with a profile picture would improve the look of the sidebar. New buttons could be added in due course as the app gets more functionality.

To close the sidebar, the user can press anywhere on the Home screen not covered by it. However, this does not work properly. If the user presses on the dark blue area below the “Logout” button, the sidebar closes. This behavior is unwanted and may cause frustration for the user. A solution to this could be to add an “Exit” button in the top right corner that is used for closing the sidebar, although some of the ease of use is taken away by removing the option to press the background. One solution that was attempted was to add a click handler on the background area of the sidebar but without success.

The Home screen itself also suffers from emptiness. There is currently only the possibility to navigate to the Wells screen or open the sidebar. Improvement to this problem could be to implement more functionality, and some ideas are given in section 8.1. If a solution for state persistence is implemented in the future, the Home screen may be a natural place to add a button for resuming the previous session from before closing the app.

7.5 Settings Screen

The Settings screen is easy to use due to *dropdown* menus and buttons for setting unit presets. Initially, a Settings screen was not of high priority in the app specification (requirement 7). When the necessity for such a screen was seen from only one perspective, e.g. units needed to be changed somewhere, it could seem like an overly complicated way to do this. As colors and units could both easily be implemented on a Settings screen, it seemed more useful to spend time on development.

Currently, units may only be changed from the Settings screen. The group assumed that the user might want to set the units once, but it can be interesting and relevant to change between

single and double amplitude more often. One solution would be to add a gear icon on the top bar of all recommendation screens and the Forecast Plot screen, which navigates to the Settings screen. Instead of showing the full Settings screen, a smaller modal with unit settings could be displayed. However, performing unit changes on a recommendation screen would lead to the same issues as changing colors; the new units should replace the old units as soon as the setting modal is closed. Hence, a global state system is likely a solution that allows for units to be changed from any screen in the app [90].

Similar to Home, the Settings screen may feel somewhat empty. It lacks some parts the user may expect to find on a settings screen, for example, information about the app version, the option to delete locally stored data, etc. Some options and features are intentionally left out from the app, for example requirement 16 in appendix C “Create and run new simulations”. A more general example is the possibility to reset the password, something that is possible on most apps with login. HeaveSIM wanted to reserve these features from the app for security reasons.

7.6 Wells & Analyses Screens

As described in section 5.7, the way the well names were extracted from the analyses list and then displayed was unnecessarily complicated. Changing the structure of the second API response or defining a new API call for fetching the well names directly would be more efficient. However, in discussion with Heavelock it was said that it was not in their interest to solve this problem through modifications to existing API calls. The current solution in section 5.7.1 circumvents the problem, and the wells and analyses are displayed in the correct order.

Because of the previously mentioned problem with the structure of the API response, displaying the analyses was not straightforward either. However, this was solved by passing the name of the well to the Analyses screen, and then comparing it with the well names in the analysis objects. There was also the task of sorting the analyses in a specific order. Analyses now include an analysis type identifier (integer). Changing the identifiers to match the wanted sorting order would simplify the sorting process, but this swap would require HeaveSIM to redefine all existing analyses. Therefore, the solution ended up as presented in section 5.7.2.

Making changes to the design of these screens was not of high priority and therefore remains similar to the specialization projects (3). The functionality was the main focus of these screens. However, some minor design features were implemented in order to make a more complete and professional app. This was, for example, the implementation of the *ActivityIndicator*, which played a role in not only design but also the functionality. Sometimes one could wait multiple seconds before navigating to the next screen because the API response can take some time. By having the *ActivityIndicator* displayed on screen, it becomes clear that the app is loading data.

7.7 Recommendation Screens & Tabs

The recommendation tabs for Connection and Tripping operations blend well with the more professional look through the use of animations and intuitive tabs. As mentioned in the results, the two tabs on either screen both stay rendered at the same time. This solution requires fairly uncomplicated animations, which is why it was chosen. The main advantages are that input fields are not reset when changing tabs, and it features a smooth transition. However, keeping more components rendered at once requires more processing power and affects battery life. A lagging UI will affect the overall UX. Detecting if the device has activated power saving mode could be used to deactivate the rendering of components outside the display.

Something that might seem less professional is the color and text differences that exist between the recommendation tabs of Connection and Tripping. Tables 5 and 6 shows how the tabs give different feedback. A case of no inputs has a white color in Connection while Tripping uses red. Under development, such differences were implemented with the intention to give experienced users another way to recognize which screen they are on. With having implemented a header at the top of each recommendation tab, these differences have become redundant.

Requirement 5 in the app specification was to give input fields better descriptions. Figure 37 shows how the input section has changed since the specialization projects. With explanatory text above the input field, there should be no confusion about what the input expects. The unit suffixes are dynamically changing after chosen unit.

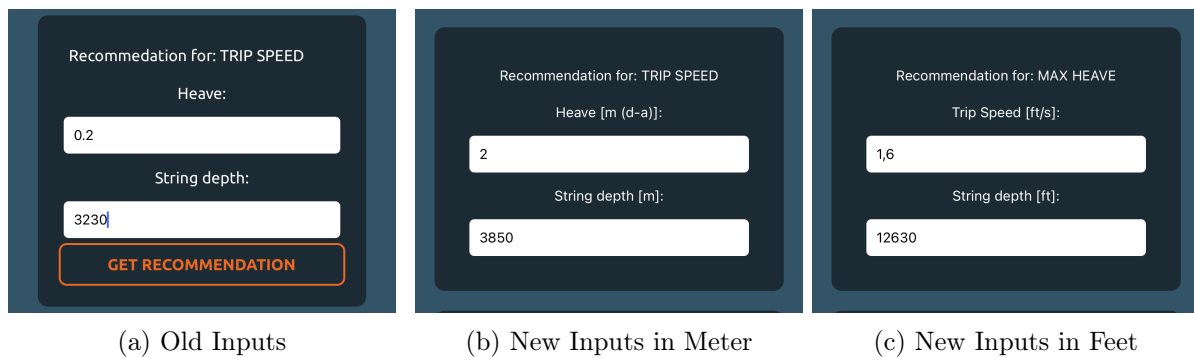


Figure 37: Comparison of Old and New Input Fields

In the app specification, requirement 13 is to display results graphically. As said in section 5.11, two plots were created with the purpose of fulfilling this task. Both of them have some minor issues that are worth mentioning. For heave/tripping speed plots in the Tripping recommendation screen, it was said in section 6.8.1 that the magenta dot is placed on the plotted line. This is not completely correct. The line is plotted using monotonic cubic interpolation, while the dot is calculated using linear interpolation. With different computation methods, they might end up with some variations. This is mostly relevant when choosing a depth in the middle of two known interest points and how it can end up looking is seen in fig. 38. A solution for correcting this is to include the input data into the dataset that *VictoryLine* uses for computation. This was not attempted in this project since it was said in consultation with Heavelock that the deviation was so small it was not deemed necessary.



Figure 38: Example of Input Dot vs Line placement

All pressure plots lack a visual feature that was desired by Heavelock. In the pressure plots on the HeaveSIM website, the well's pressure is intuitively displayed as an enclosed area. On the contrary, the app presents two lines with the same color without an explanation for what the lines represent. Figure 39b shows the disparities in the two designs. A component called *VictoryArea* could help imitate HeaveSIM's design, but setting it up proved to be challenging. With time running out, it was decided to be content with the two lines and rather prioritize other functionality. Either way, it would have been advantageous to have implemented a legend similar to the forecast plot to better convey the aspects of the pressure plot.

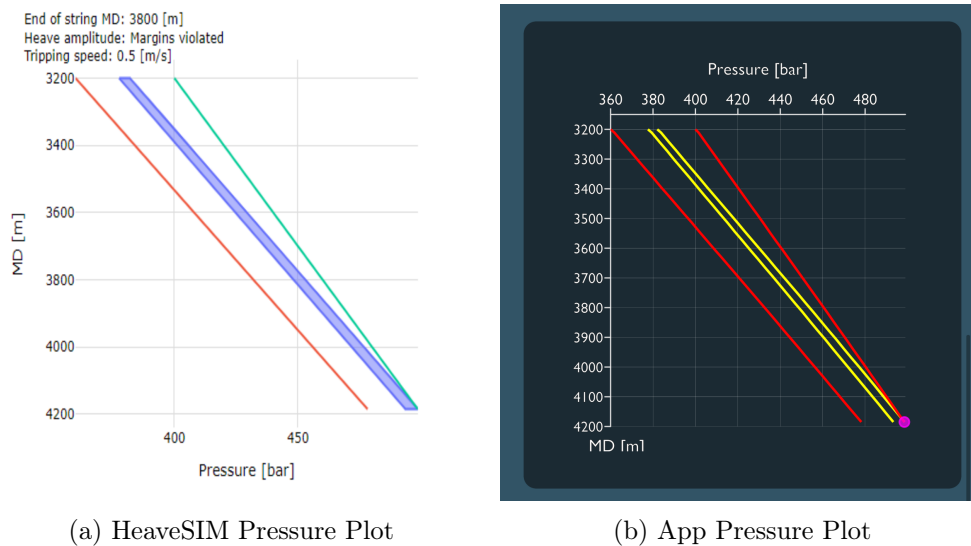


Figure 39: Comparison of Pressure Plots

Units and conversions were implemented at a late stage in the development process, which in hindsight required more tedious changes to implement compared to if it had been done earlier in the project. Optimally, the standard of dividing inputs and multiplying outputs by conversion rates should have been agreed on early. All new script files written afterward would satisfy this standard. Thus, the time-consuming process of adding these changes and verifying that the units and conversions produce the expected results would have been avoided.

7.8 Forecast

Input Tab

At this point in time, the forecast plot is limited to 23 hours ahead or less. This was done as the dummy data is only set up for that time range, and it was thought not necessary to expand further than 24 hours. When a real weather forecast service is implemented into the app, one could also consider increasing the valid time range if that is desired.

While setting up the input tab, there was an uncertainty of what would be best between having a singular input of hours ahead or inputs for start and stop times. With one variable, there are fewer inputs for users, and thus the forecast plot can be reached more effortlessly. However, letting users choose their own starting and stopping times is a more versatile solution. Another possibility could be to implement both concepts and let the user toggle between what is appropriate for their situation. Using hours ahead for situations where the ROP is consistent and two variables where it is not. By having inconsistent ROP, a user needs to reset the plot for every depth where the ROP changes. When doing so, it is crucial to be able to match the relevant hour to the depth by setting a starting time. If the solution of hours ahead is not sufficient when forecast is revised later, shifting to a more acceptable one could be realized then.

Forecast Screen

In requirement 8 of the app specification, one of the wanted features was to give recommendations for future operations. As of now, the app's forecast only supports the Connection operation. As said earlier, prioritizing Connection was requested from Heavelock when the group stated that there was only time to develop a forecast for one operation. At a later point, one could expand this feature to Tripping or other operations when they are implemented in HeaveSIM.

The final design of the Forecast screen deviates somewhat from what was intended. It was planned to have the plot set up in an established style that other plots in the app follow. Elements like labels for axes or data points reduced the plot's readability and had to be changed. The app is meant to be used with a portrait orientation lock on, as components on screens are not customized for both portrait and landscape mode. This means that when users rotate their phone to look at the forecast screen, they must rotate the phone the correct way. With that in mind, the group decided on rotating the plot to the left with the assumption that most users use their left hand to hold their phone [91] and thus, rotating this way would be easier.

The dummy data created for weather forecast can be criticized for having too much noise. With the amount of noise included in the equation, waves might have far too rapid changes from one hour to the next for them to be considered realistic. However, this was intentional as the app now gives more examples of how it handles different weather situations.

Moreover, it can be said that there exist more weather service companies than just StormGeo, which could be contacted. However, as Heavelock's customers use StormGeo, it was discussed and decided that it would be better to use dummy data instead of incorporating API calls from other companies, which would later be redone anyway.

Another element worth reviewing is the legend on the rightmost side. The plot could seem somewhat confusing, and the legend was added in an attempt to clarify what all the items are.

Having a legend was not in the original design but deemed necessary after the implementation of the plot. An idea that was considered was to utilize a pop-up modal with the legend. Users could then open this modal with the press of a button when needed. Displaying the legend at all times is more user-friendly and was thought to be preferable.

7.9 Cross Platform

Requirement 14 in the app specification (table 2) was of high priority because the iPad is commonly used on offshore rigs. As seen in section 6.9, the app is now supported on iOS, Android, and iPadOS. Supporting multiple platforms makes the app available to a larger user base.

The support for iOS and Android was already implemented in the earlier project. Although the iPadOS is supported by default, the app was not adapted stylistically for this platform. Necessary changes include scaling of on-screen components such as sections, plots, and background images. These had to be platform-specific in order to work as intended. Larger plots are easier to read, and therefore having the app support iPadOS is valuable.

Although RN simplifies cross-platform development, not all functionality and built-in components are supported on all platforms. Therefore, the group has been conscious when choosing components and solutions throughout this project. The main criteria was support on all three platforms.

7.10 Code Quality

The code quality has substantially improved from the specialization projects due to a greater focus on SoC, as stated in requirement 3 in the app specification. The codebase has increased in size and spending some time early in the project to clean up and restructure the existing codebase was advantageous. New script files, screens, and components are written with these principles in mind, keeping the codebase well-structured. The codebase is now categorized into the folders: Assets, Components, Navigation, Screens, Scripts, and Styles. In hindsight, some of the filenames could have been even more intuitive.

Still, there is code that could do with a generalization. Similar to how the interpolation code was generalized and reused, data preparation for the plots could receive the same treatment. Plotting code for the “Heave” and “Trip speed” tabs 6.8 utilizes generalized functions shared between them. With a better generalization of functions, there might be a decrease in required processing power to run the app.

The README.md file has been updated to include all the new packages that were added during the project. The README file is now divided into smaller sections that better describe different parts of the code repository. Requirement 4 in the app specification regards the README file, and the updated file now better reflects how to install and use the code repository for further development.

Another relevant element that can be enhanced is the search algorithm used when processing analysis data. The app uses *linear search* for finding closest numbers and data, which has a time complexity of $\mathcal{O}(n)$ [92]. Currently, the maximum number of interest points (depths) allowed in an analysis is 20, each with multiple parameters as seen in listing 9. Due to the occurrence of identical numeric values in the analysis, e.g. four tripping speeds per depth, *linear search* was a simple choice that guaranteed correct results. If HeaveSIM considerably expands either

the number of interest points or parameters per depth, replacing the search algorithm becomes relevant.

Testing of the app has not been done to industry standards. Industry standards are defined by ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission), and *ISO/IEC 25051* is one example of a standard that gives instructions for software testing [93]. Due to the workflow in this project, new code was tested by other group members before merging it with the codebase. Having other people test new features with a fresh mindset was very helpful in ruling out bugs and errors.

Error handling was specified in requirement 11 of the app specification. It has been implemented, and the cases possible to emulate, namely wrong login credentials and incomplete simulation results, have been tested. However, testing how the app behaves after a sudden loss of internet connection was not possible to do. The reason for this is that Expo requires an internet connection when running the app from the CLI or a complete build. As the app is dependent on an internet connection to function, this testing should have a high priority for a finished product.

Another form of handling included in the app specification is input handling (req 12). The main goal was to stop users from being able to enter wrong input relative to the defined analysis. The solution implemented was to return a pop-up modal with information about the valid range for the input variable. Such a solution was thought to be satisfactory, but there are other options worthy of consideration. Instead of giving warnings in modals, one could rather give a red outline around input fields with invalid input to signal where the error is. The group implemented the modal solution because having the possibility of explaining the error was thought to be the most precise way to give the user feedback. These solutions could also be combined for even better feedback. Another feature that could have been combined with either mentioned solutions is to hide all recommendation fields and plots if invalid input is entered. This would prevent the plotting library from crashing, as mentioned in section 5.9.

7.11 Resolved App Specification

As stated in the introduction of this chapter, the requirements of the app specification in section 4.2 have been important to complete during this project, at least to a degree of satisfaction. The sections above show that most of these requirements have been resolved, with the exception of a few. As previously mentioned, requirement 9 “remember the previous location in the app”, was not completed due to a lack of working technology. More on this in section 8.1.

Requirement 6 - “Deployment of the app” is not mentioned in the sections above. The solution today is not optimal, with deploying via Expo Go and not through proper app stores. However, this is done intentionally because the app is still a proof of concept and the maintenance it requires when having an app on app stores. Every time an update comes to the operating system, the app would also have to be updated. Since the point of deploying this app is to display it to potential customers, it would not be worth the effort of maintenance.

Conclusion

The motivation behind this project was to further develop the app so it could potentially be shown to clients, partners, and investors to better demonstrate the usefulness of HeaveSIM. Brief descriptions of the heave problem and HeaveSIM's functionality were given in the introduction. In the planning stage of the project, a new app specification was formed. Two main goals were to give the app a more professional appearance and to add functionality for forecasts. Additionally, each recommendation should present more background information to give the user greater insight into the operation.

The final result presented in section 6 shows that these goals have been accomplished. The app now has "production" quality in terms of professional appearance and user-friendliness. The new forecast feature is innovative, not only for the app but also for HeaveSIM. Lacking weather data and rig dynamics compromised the forecast feature to some degree, but the work in this project provides a solid fundament for further development.

During demonstrations of the app, Heavelock employees confirmed that the ease of use and availability the app provides has great value to the HeaveSIM product.

8.1 Further work

Although the app has been greatly improved during this project, there is still work that needs to be completed in order for it to be production ready. Summarized below are suggested aspects of the app which can benefit from further development.

Real Weather Data and Heave Transformation

Dummy data was the solution used in this project because StormGeo did not allow the group to access its services for free. The usefulness of the forecast feature depends on real weather data and using rig dynamics to transform it into heave. This task should have a high priority during potential further development. Implementing real heave data with the existing functionality in the app should be simple as the app will work with any heave data, real or simulated.

Local Data Storage

The implementation of state persistence would improve the UX, but at the time of this project, there were no suitable solutions for this. It is expected that necessary functionality similar to the deprecated AsyncStorage should appear in the future if it does not already exist. Similar functionality should not require a full restructuring of the app. If an alternative to AsyncStorage is implemented, the attempted cookie functionality can be added. Data such as the login token and analyses can then be stored locally for a set duration. Updating the cookies can be done in the background during use, but it may be critical to show the user when analyses and simulation data were last updated.

Home Screen Functionality

Some added functionality to the Home screen is also desired. Listed below are ideas for features that could be implemented:

- Ability to continue previous session, i.e. navigate to the state the app was in before it was exited. This could be done by adding a button on the Home screen, assuming there is a solution for local storage.
- Weather/heave widget, showing just the heave/wave forecast without a recommendation.
- A notification center under the “Welcome to Heavesim” section for displaying various push notifications, e.g. for changes in simulation data.
- A list of all defined analyses independently of any well, accessible from the sidebar.
- Profile picture and full name in the sidebar.
- Extend license or change background picture (replace stock photo).

Settings Screen Functionality

Features and functionality should also be added to the Settings screen. Some features that would be smart to implement in order to get an even more professional app are:

- Button for clearing local data/cache, assuming local storage becomes possible
- Info about the app version

Analyses Screen Functionality

During the final demonstration of the app, Heavelock employees expressed that a graphical overview of the casing plan could be useful to include in the app. The casing plan can be viewed on the HeaveSIM website, and is shown in fig. 40.

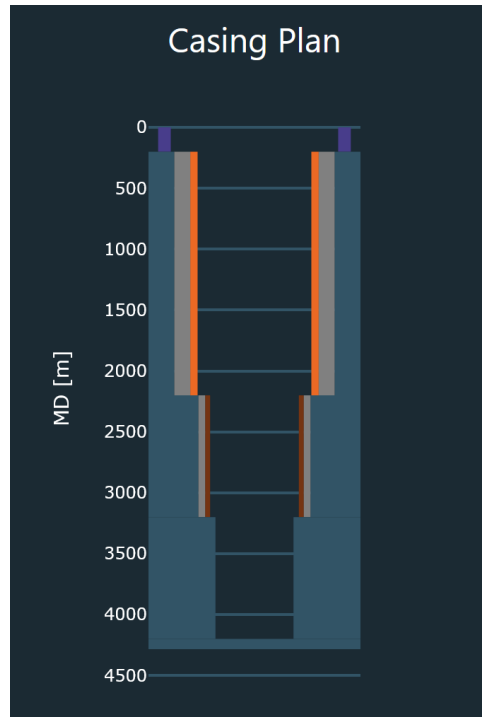


Figure 40: Casing Plan from HeaveSIM [65]

A natural place to present the casing plan is the Analyses screen. It can be included at the top of the screen before the analyses are listed or by utilizing the tab functionality used on the Recommendation screens. Modifications to existing or creating a new API call for this purpose are necessary. Displaying the casing plan was never a part of the app specification for the project due to the late suggestion from Heavelock employees.

The analyses are currently sorted after month, then analysis type. Choosing a different sorting order could be very useful, especially if the user has many analyses in the same month. Custom sorting could be implemented by using *dropdown* menus on the Analyses screen to support quick changes to the sorting order. Replacing the standard scrollable list with expandable vertical sections for each month could also structure the analyses better.

Miscellaneous Screens

- Extend license or change background image on Login screen (replace stock photo)
- Optimize tab functionality with respect to performance/battery life
- Design new color schemes “LIGHT” and “HIGH CONTRAST”, with basis in color theory
- Expand the forecast functionality to other operations
- Add support for other operations when they are added to HeaveSIM

Deployment to App Stores

Currently, the app is shared in Expo Go for everyone with access to the Expo HeaveSIM user. When the app eventually is production-ready, it should be deployed to the different app stores to make it available for the public. Read more about this in [94].

Bibliography

- [1] Amund Buer. *Design and implementation of a user interface for HeaveSIM*. URL: <https://drive.google.com/drive/folders/1d0Msg6W6rWxzhqMEpZ0SuTYxqAW8PjxY?usp=sharing>. (accessed: 25.04.2022).
 - [2] Stig Hope Eriksen. *Design and implementation of a user interface for HeaveSIM*. URL: <https://drive.google.com/drive/folders/1d0Msg6W6rWxzhqMEpZ0SuTYxqAW8PjxY?usp=sharing>. (accessed: 25.04.2022).
 - [3] Tobias Aanstad. *Design and implementation of a user interface for HeaveSIM*. URL: <https://drive.google.com/drive/folders/1d0Msg6W6rWxzhqMEpZ0SuTYxqAW8PjxY?usp=sharing>. (accessed: 25.04.2022).
 - [4] Kvernland, M. and Gorski, D. and Sant' Ana, M. and Godhavn, J. -M. and Aamo, O. M. and Sangesland, S. *Verification of Downhole Choke Technology in a Simulator Using Data from a North Sea Well*. D031S013R002. Mar. 2019. eprint: <https://onepetro.org/SPEDC/proceedings-pdf/19DC/3-19DC/D031S013R002/1166041/spe-194143-ms.pdf>. URL: <https://doi.org/10.2118/194143-MS>. (accessed: 22.05.2022).
 - [5] Kvernland, M. and Christensen, M. Ø. and Borgen, H. and Godhavn, J. -M. and Aamo, O. M. and Sangesland, S. *Attenuating Heave-Induced Pressure Oscillations using Automated Down-hole Choking*. D031S015R003. Mar. 2018. eprint: <https://onepetro.org/SPEDC/proceedings-pdf/18DC/3-18DC/D031S015R003/1228245/spe-189657-ms.pdf>. URL: <https://doi.org/10.2118/189657-MS>. (accessed: 22.05.2022).
 - [6] *Modeling for MPD Operations With Experimental Validation*. Vol. All Days. SPE/IADC Drilling Conference and Exhibition. SPE-150461-MS. Mar. 2012. eprint: <https://onepetro.org/SPEDC/proceedings-pdf/12DC/A11-12DC/SPE-150461-MS/1625603/spe-150461-ms.pdf>. URL: <https://doi.org/10.2118/150461-MS>. (accessed: 09.06.2022).
 - [7] Ingar Skyberg Landet, Alexey Pavlov, and Ole Morten Aamo. “Modeling and Control of Heave-Induced Pressure Fluctuations in Managed Pressure Drilling”. In: *IEEE Transactions on Control Systems Technology* 21.4 (2013), pp. 1340–1351. URL: <https://ieeexplore.ieee.org/document/6238317>. (accessed: 09.06.2022).
 - [8] Henrik Anfinen and Ole Morten Aamo. “Rejection of Heave-Induced Pressure Fluctuations at the Casing Shoe in Managed Pressure Drilling”. In: *IFAC Proceedings Volumes* 47.3 (2014). 19th IFAC World Congress, pp. 7717–7722. URL: <https://www.sciencedirect.com/science/article/pii/S1474667016428284>. (accessed: 09.06.2022).
 - [9] Hessam Mahdianfar et al. “Robust Regulation of Heave-Induced Pressure Oscillations in Offshore MPD - This work was supported by Statoil ASA and the Research Council of Norway.” In: *IFAC-PapersOnLine* 48.6 (2015). 2nd IFAC Workshop on Automatic Control in Offshore Oil and Gas Production OOGP 2015, pp. 112–119. URL: <https://www.sciencedirect.com/science/article/pii/S240589631500885X>. (accessed: 09.06.2022).
 - [10] Timm Strecker and Ole Morten Aamo. “Rejecting heave-induced pressure oscillations in a semilinear hyperbolic well model”. In: *2017 American Control Conference (ACC)*. 2017, pp. 1163–1168. (accessed: 09.06.2022).
 - [11] John-Morten Godhavn et al. “Significant Surge and Swab Offshore Brazil Induced by Rig Heave during Drillpipe Connections”. In: *SPE Drilling & Completion* 36.03 (Sept. 2021), pp. 552–559. eprint: <https://onepetro.org/DC/article-pdf/36/03/552/2483942/spe-200518-pa.pdf>. URL: <https://doi.org/10.2118/200518-PA>. (accessed: 09.06.2022).
 - [12] Amazon Web Services. *What is AWS?* URL: https://aws.amazon.com/what-is-aws/?nc2=h_q1_le_int. (accessed: 22.05.2022).
-

BIBLIOGRAPHY

- [13] AWS. *AWS Batch*. URL: <https://aws.amazon.com/>. (accessed: 20.11.2021).
 - [14] freebiesupply. *AWS Lambda Logo*. URL: <https://freebiesupply.com/logos/aws-lambda-logo/>. (accessed: 20.11.2021).
 - [15] Heavelock Solutions. *HeaveSIM*. URL: <https://heavesim.no/login/>. (accessed: 16.12.2021).
 - [16] Anna Chebanova. *When to Use Python/Django?* URL: <https://www.crowdbotics.com/blog/when-to-use-python-django-top-10-web-applications-build-with-python>. (accessed: 20.11.2021).
 - [17] Edpresso Team. *What is Amazon EC2?* URL: <https://www.educative.io/edpresso/what-is-amazon-ec2>. (accessed: 20.11.2021).
 - [18] React Native. *React Native Logo*. URL: https://reactnative.dev/img/header_logo.svg. (accessed: 25.04.2022).
 - [19] ZABBIX. *AWS S3*. URL: https://www.zabbix.com/integrations/aws_s3. (accessed: 20.11.2021).
 - [20] Stackshare. *AWS Batch*. URL: <https://stackshare.io/aws-batch>. (accessed: 20.11.2021).
 - [21] Chris Williams. *Javascript Logo*. URL: <https://github.com/voodootikigod/logo.js/blob/1544bdeed/js.svg>. (accessed: 20.11.2021).
 - [22] Django Software Foundation. *Why Django?* URL: <https://www.djangoproject.com/start/overview>. (accessed: 22.05.2022).
 - [23] Amazon Web Services. *What is Amazon EC2?* URL: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>. (accessed: 22.05.2022).
 - [24] Amazon Web Services. *What is Amazon S3?* URL: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>. (accessed: 22.05.2022).
 - [25] Amazon Web Services. *What is Amazon Lambda?* URL: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>. (accessed: 22.05.2022).
 - [26] Meta Platforms, Inc. *React native*. URL: <https://reactnative.dev/>. (accessed: 25.04.2022).
 - [27] Statista. *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021*. URL: <https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>. (accessed: 05.06.2022).
 - [28] W3Techs. *Usage statistics of JavaScript as client-side programming language on websites*. URL: <https://w3techs.com/technologies/details/cp-javascript>. (accessed: 05.06.2022).
 - [29] Meta Platforms, Inc. *Render, Commit, and Mount*. URL: <https://reactnative.dev/architecture/render-pipeline>. (accessed: 25.04.2022).
 - [30] Expo. *Expo*. URL: <https://docs.expo.dev/>. (accessed: 24.05.2022).
 - [31] Kiki Ding. *Rapi UI documentation*. URL: <https://rapi-ui.kikiding.space/docs/>. (accessed: 24.05.2022).
 - [32] React Navigation Org. *React Navigation. Routing and navigation for Expo and React Native apps*. URL: <https://reactnavigation.org/>. (accessed: 09.04.2022).
 - [33] React Native. *Animated*. URL: <https://reactnative.dev/docs/animated>. (accessed: 07.03.2022).
 - [34] Formidable. *VICTORY*. URL: <https://formidable.com/open-source/victory/>. (accessed: 23.03.2022).
-

- [35] John Potter. *react-native-charts-wrapper vs react-native-pure-chart vs react-native-svg-charts vs victory*. URL: <https://www.npmtrends.com/react-native-charts-wrapper-vs-react-native-pure-chart-vs-react-native-svg-charts-vs-victory>. (accessed: 25.04.2022).
 - [36] Website: Openbase. *10 Most Popular React Native Chart Libraries*. URL: <https://openbase.com/categories/js/most-popular-react-native-chart-libraries>. (accessed: 25.04.2022).
 - [37] Alexey Naumov. *Separation of Concerns in Software Design*. URL: <https://nalexn.github.io/separation-of-concerns/>. (accessed: 28.05.2022).
 - [38] Paul E. Black. *big-O notation*. URL: <https://xlinux.nist.gov/dads/HTML/bigOnotation.html>. (accessed: 26.05.2022).
 - [39] Donald E. Knuth. *The Art of Computer Programming*. Vol. 1. (3rd ed.) USA: Addison-Wesley, 1997, pp. 107–110.
 - [40] RedHat. *What is a REST API?* URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>. (accessed: 22.05.2022).
 - [41] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. (accessed: 22.05.2022).
 - [42] NortonLifeLock Inc. *What are cookies?* Aug. 2019. URL: <https://us.norton.com/internetsecurity-privacy-what-are-cookies.html>. (accessed: 22.05.2022).
 - [43] Techopedia. *Persistence*. URL: <https://www.techopedia.com/definition/8842/persistence-computing>. (accessed: 09.06.2022).
 - [44] React Navigation Org. *State persistence*. URL: <https://reactnavigation.org/docs/state-persistence/>. (accessed: 22.05.2022).
 - [45] React Native. *AsyncStorage*. URL: <https://reactnative.dev/docs/asyncstorage>. (accessed: 23.05.2022).
 - [46] T. Editors of Encyclopaedia Britannica. *interpolation*. URL: <https://www.britannica.com/science/interpolation>. (accessed: 01.06.2022).
 - [47] Wolfram MathWorld. *Lagrange Interpolating Polynomial*. URL: <https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html>. (accessed: 26.05.2022).
 - [48] Tamal Das. *How to Choose a Color Scheme for Your App: 10 Things to Consider*. URL: <https://www.makeuseof.com/choose-color-scheme-for-apps-things-to-consider/>. (accessed: 24.05.2022).
 - [49] World Wide Web Consortium. *Accessibility, Usability, and Inclusion*. URL: <https://www.w3.org/WAI/fundamentals/accessibility-usability-inclusion/>. (accessed: 24.05.2022).
 - [50] National Health Service. *Colour vision deficiency (colour blindness)*. URL: <https://www.nhs.uk/conditions/colour-vision-deficiency/>. (accessed: 24.05.2022).
 - [51] Kathy Lamancusa. *Emotional Reactions to Color*. URL: https://web.archive.org/web/20160311005142/http://www.creativelatitude.com/articles/articles_lamacusa_color.html. (accessed: 08.06.2022).
 - [52] Jill Morton. *Why Color Matters*. URL: <https://www.colorcom.com/research/why-color-matters>. (accessed: 05.06.2022).
 - [53] React Native. *React Fundamentals*. URL: <https://reactnative.dev/docs/intro-react>. (accessed: 05.06.2022).
-

BIBLIOGRAPHY

- [54] Git. *Git –fast-version-control*. URL: <https://git-scm.com/>. (accessed: 26.05.2022).
 - [55] GitHub. *Where the world build software*. URL: <https://github.com/>. (accessed: 26.05.2022).
 - [56] Atlassian. *What is Git*. URL: <https://www.atlassian.com/git/tutorials/what-is-git>. (accessed: 31.05.2022).
 - [57] Jacob Stopak. *Why Git is Still Relevant in 2021, and Will Be for a Long Time*. URL: <https://simpleprogrammer.com/git-relevant-in-2021/>. (accessed: 31.05.2022).
 - [58] W3Schools. *JavaScript Tutorial*. URL: <https://www.w3schools.com/js/default.asp>. (accessed: 26.05.2022).
 - [59] StackOverflow. *Stack Overflow*. URL: <https://stackoverflow.com/>. (accessed: 26.05.2022).
 - [60] Fabio Ferreira. *React Native vs Native for Mobile App Deelopment*. URL: <https://www.scalablepath.com/react-native/react-native-vs-native>. (accessed: 05.06.2022).
 - [61] Schlumberger Oilfield Glossary. *rate of penetration (ROP)*. URL: https://glossary.oilfield.slb.com/en/terms/r/rate_of_penetration. (accessed: 04.06.2022).
 - [62] Serenity Gibbons. *You And Your Business Have 7 Seconds To Make A First Impression: Here's How To Succeed*. URL: <https://uxplanet.org/mobile-app-ux-design-making-a-great-first-impression-bed2805b967d>. (accessed: 05.06.2022).
 - [63] Nick Babich. *Mobile App UX Design: Making a Great First Impression*. URL: <https://www.forbes.com/sites/serenitygibbons/2018/06/19/you-have-7-seconds-to-make-a-first-impression-heres-how-to-succeed/?sh=33416fe856c2>. (accessed: 05.06.2022).
 - [64] Laura Levy. *The most important 20 seconds of your mobile app onboarding*. URL: <https://www.invisionapp.com/inside-design/mobile-app-onboarding/>. (accessed: 05.06.2022).
 - [65] Heavelock Solutions. *HeaveSIM*. URL: <https://heavesim.no>. (accessed: 28.05.2022).
 - [66] StormGeo. *Who We Are*. URL: <https://www.stormgeo.com/company/who-we-are/>. (accessed: 27.05.2022).
 - [67] Thor Inge Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Vol. 1. (2nd ed.) Norway: Wiley-Blackwell, 2021. Chap. 10.2, pp. 274–275.
 - [68] Oil & Gas Drilling Glossary - IADCLeXicon.org. *Response Amplitude Operator (RAO)*. URL: <https://iadclexicon.org/response-amplitude-operator/>. (accessed: 08.06.2022).
 - [69] W3 Schools. *JavaScript Cookies*. URL: https://www.w3schools.com/js/js_cookies.asp. (accessed: 31.05.2022).
 - [70] Luís C Meireles. *Answer to: How to access cookies with React Native Expo*. URL: <https://stackoverflow.com/questions/67145010/how-to-access-cookies-with-react-native-expo>. (accessed: 05.06.2022).
 - [71] React Native. *StyleSheet. A StyleSheet is an abstraction similar to CSS StyleSheets*. URL: <https://reactnative.dev/docs/stylesheet>. (accessed: 23.05.2022).
 - [72] W3C Group. *CSS Snapshot 2021*. URL: <https://www.w3.org/TR/CSS/#css>. (accessed: 23.05.2022).
 - [73] Jake Coxon. *React Native: vertical centering when using ScrollView*. URL: <https://stackoverflow.com/questions/32664397/react-native-vertical-centering-when-using-scrollview>. (accessed: 20.05.2022).
 - [74] jave.web and David Scholz. *React Native - pass callback to another screen*. URL: <https://stackoverflow.com/questions/71449606/react-native-pass-callback-to-another-screen>. (accessed: 02.05.2022).
-

- [75] Lakshmikant Despande and Phú Quốc Đinh. *Refresh previous screen on goBack()*. URL: <https://stackoverflow.com/questions/46504660/refresh-previous-screen-on-goback>. (accessed: 28.04.2022).
 - [76] React Navigation Org. *StackActions reference*. URL: <https://reactnavigation.org/docs/stack-actions/>. (accessed: 29.04.2022).
 - [77] React Native. *ActivityIndicator*. URL: <https://reactnative.dev/docs/activityindicator>. (accessed: 07.06.2022).
 - [78] React Native. *ScrollView*. URL: <https://reactnative.dev/docs/scrollview>. (accessed: 27.05.2022).
 - [79] Penny Liu and Hamza Waleed. *React native TextInput Autofocus is not working*. URL: <https://stackoverflow.com/questions/59860865/react-native-textinput-autofocus-is-not-working>. (accessed: 17.03.2022).
 - [80] Andreas Wienes and Stephen Hanson. *React Native: How to select the next TextInput after pressing the "next" keyboard button?* URL: <https://stackoverflow.com/questions/32748718/react-native-how-to-select-the-next-textinput-after-pressing-the-next-keyboard>. (accessed: 17.03.2022).
 - [81] Amit Diwan. *Finding two closest elements to a specific number in an array using JavaScript*. URL: <https://www.tutorialspoint.com/finding-two-closest-elements-to-a-specific-number-in-an-array-using-javascript>. (accessed: 18.12.2021).
 - [82] James Bird. *Monotone Cubic Interpolation*. URL: <http://jbrd.github.io/2020/12/27/monotone-cubic-interpolation.html>. (accessed: 09.06.2022).
 - [83] MDN. *Math.random()*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random. (accessed: 25.05.2022).
 - [84] Tobias Aanstad, Amund Buer, Stig Hope Eriksen. *HeaveSIM App User Guide*. URL: <https://youtu.be/RCgGqDRtBUQ>. (accessed: 09.06.2022).
 - [85] Gundam Meister and Sydney Loteria. *React Native Responsive Font Size*. URL: <https://stackoverflow.com/questions/33628677/react-native-responsive-font-size>. (accessed: 20.05.2022).
 - [86] React Native. *KeyboardAvoidingView*. URL: <https://reactnative.dev/docs/keyboardavoidingview>. (accessed: 16.03.2022).
 - [87] TebNad. *Silhouette oil rig stock photo*. URL: <https://www.istockphoto.com/photo/silhouette-oil-rig-gm1161679863-318391262#>. (accessed: 08.06.2022).
 - [88] nielubieklonu. *HDR of Offshore drilling rig at day stock photo*. URL: <https://www.istockphoto.com/photo/hdr-of-offshore-drilling-rig-at-day-gm901140746-248612310>. (accessed: 08.06.2022).
 - [89] NPMJS. *react-native-select-dropdown*. URL: <https://www.npmjs.com/package/react-native-select-dropdown>. (accessed: 05.06.2022).
 - [90] A. Abramov et. al. *Getting Started with Redux*. URL: <https://redux.js.org/introduction/getting-started>. (accessed: 06.06.2022).
 - [91] Tristan Denyer. *User research: how do people hold and interact with their phone?* URL: <https://tristandeny.com/work/user-research-people-hold-interact-phone/>. (accessed: 29.05.2022).
 - [92] GeeksForGeeks. *Linear Search*. URL: <https://www.geeksforgeeks.org/linear-search/>. (accessed: 29.05.2022).
-

BIBLIOGRAPHY

- [93] ISO. *ISO/IEC 25051:2014(en) Software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Requirements for quality of Ready to Use Software Product (RUSP) and instructions for testing*. URL: <https://www.iso.org/obp/ui/#iso:std:iso-iec:25051:ed-2:v1:en>. (accessed: 08.06.2022).
- [94] Expo. *Deploying to App Stores*. URL: <https://docs.expo.dev/distribution/app-stores/>. (accessed: 28.05.2022).

Appendix

A Problem description handed out

NTNU
Norges teknisk-naturvitenskapelige
universitet

Fakultet for informasjonsteknologi
og elektroteknikk

Institutt for teknisk kybernetikk



MASTEROPPGAVE

Kandidatens navn: Amund Buer, Stig Hope Eriksen, Tobias Aanstad
Fag: Teknisk Kybernetikk
Oppgavens tittel: Design and implementation of a mobile app for in-situ recommendations related to rig heave during offshore oil well drilling

Bakgrunn

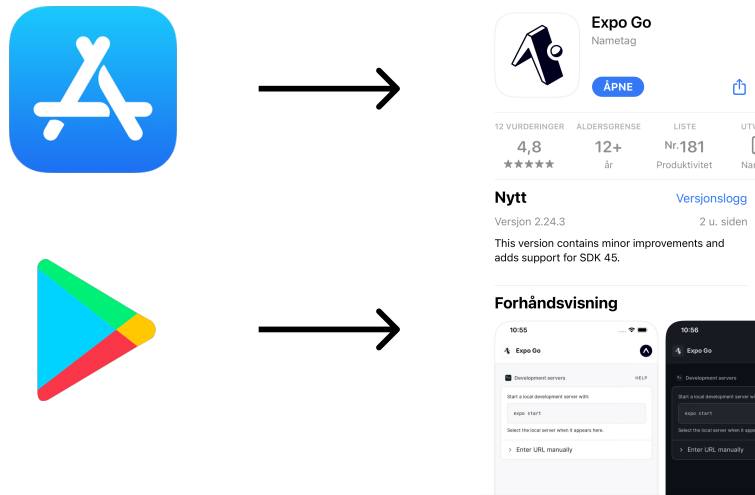
When drilling from a floater, the heaving motion of the floater causes major pressure fluctuations in the well. An important tool for assessing whether the operation can continue or must wait for better weather is a drilling simulator that is tailored to the heave problem. The computational engine and a web-based user interface (HeaveSIM) that is tailored for well construction planning has been developed but is not suitable for providing in-situ recommendations during operation. In this MSc thesis work, the objective is to design and implement a mobile application (iOS and Android) that provides such recommendations based on the current weather and stage of operation. In addition, the application should offer short-term plan of recommendations based on weather forecasts – for instance expected operability in terms of predicted heave for the next 24 hours. The following tasks should be performed by the student:

1. Background: Brief general description of the heave problem in offshore drilling from a floating (this can be brief and point to references since it has been described many times before).
2. Background: Brief description of HeaveSIM functionality (this can be brief and taken from project reports from fall 2021).
3. Improve existing and specify new requirements of the app.
 - Improve functionality/user interface implemented in fall 2021 to “production” quality.
 - Specify new functionality and design the graphical appearance of it (the short-term planning part of the app).
 - Specify graphical appearance of background information explaining the reason for recommendations (which margin was violated and at which depth?).
4. Implement the app according to specifications in point 3. The app should have “production” quality in terms of appearance and user friendliness.
5. Write a comprehensive report, describing the technology used, the software structure/design, and functionality of the app. The report should contain a user’s guide.

Faglærer/Veileder: Professor Ole Morten Aamo

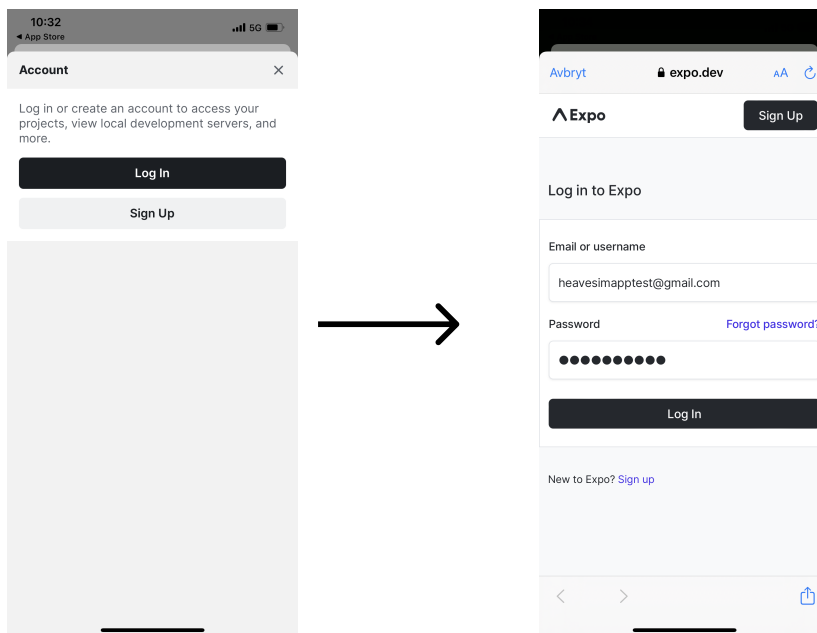
B Guide to download the app

Step 1: Open App Store or Google Play and download Expo Go.



Step 2: Open the Expo Go and log in using the credentials:

- Username: `heavesimapptest@gmail.com`
- Password: `testing321`



Step 3: Scan the QR-code



Step 4: Test the application

If you don't have an account, you can test with:

- Username: eriksen
- Password: testing321

C Considered Tasks with comments

Table 7: Specifications: Improvement of existing app

No.	Title	Functionality/Explanation	Priority
1.	Change from static to dynamic navigation	Navigation to different drilling operations' recommendation screens was based on strings consisting of names and depths which varied in different analyses. To avoid errors for other analyses, this must be changed. Comment: Completed.	5
2.	Design / UI	The design needs a more professional look. This can be achieved by prettifying the current design, and also: <ul style="list-style-type: none"> • Add animations for navigation between screens • Implement a vertical navigation menu for quicker navigation back and forth • Include statusbar into the design to utilize the whole screen Comment: Completed.	5
3.	Improve code quality/ Clean up code	<ul style="list-style-type: none"> • Follow SoC principles by creating script files • Set up a stylesheet file to remove repeated style definitions • Merge similar functions to reduce the number of functions Comment: Completed.	5
4.	Write an extensive README.md	Complete the README to give an introduction to the code for better readability and so that "everyone" can try the app without difficulty. Comment: Completed.	5
5.	Better descriptions of inputs and outputs	Include suffixes and units of all inputs and outputs to reduce the chance of misunderstandings and errors between users and app. Comment: Completed.	5

Table 8: Specifications: New functionality [part 1/2]

No.	Title	Functionality/Explanation	Priority
6.	Deployment of app	The app must be deployed such that it can easily be showcased to customers. Comment: Completed.	4
7.	Settings screen	Give the user the ability to change units and color scheme, with the option to add more at a later point of time. Comment: Completed.	2
8.	Inputs for multiple points in time/ Forecast	Try to either let users input multiple situations at the same time and then compare results, or design a predictor of how future weather affects operations. Other things to implement could be: <ul style="list-style-type: none"> • Show estimated progress based on input Rate of Penetration (ROP)* • Display predictions graphically and not only return numbers • Recommendations for future operations based on future weather conditions Comment: Forecast/Prediction for connection completed, although live weather data and heave transformation are done with dummy data and is set to be redone in future work. Forecast/prediction for tripping not done because of not enough time.	5
9.	Remember previous location in app	Implement a feature that saves the user’s last location in the app so that they do not have to log in and navigate back after the app is closed. In this feature, a solution for saving the user’s credentials must also be implemented. Comment: Attempted without success. All solutions found were deprecated.	4
10.	Automatic logout/ Demand credentials after given time	For safety reasons, the app should automatically log users out after a given time. Comment: Attempted without success. Lacking support for local storage and cookies.	2

* ROP: “The speed at which the drill bit can break the rock under it and thus deepen the wellbore.” [61]

Table 9: Specifications: New functionality [part 2/2]

No.	Title	Functionality/Explanation	Priority
11.	Error handling	Find a solution that handles errors. The app should not crash whenever it encounters an error. Comment: Completed, but needs to be properly tested. Particularly, errors caused by a missing internet connection need testing.	3
12.	Deny wrong inputs	Inputs that does not fit the simulation (e.g asking for string depths outside the scope of analysis) or wrong inputs (e.g. negative tripping speed) should not be allowed. Disabling buttons and/or giving warnings are possible solutions. Comment: Completed	5
13.	Show data/ results graphically	It is important to show data graphically as this is a much more intuitive way to understand data for users, such as a plot. Comment: Completed.	5
14.	Expand to other devices	According to Heavelock, some companies use iPads offshore, so enabling the app to work on other devices can increase companies' desire to use this service. Comment: Completed.	4

Table 10: Specifications: Functionality Not Attempted

No.	Title	Functionality/Explanation	Priority
15.	Adjust interpolation method	<p>Consider going away from linear interpolation to more advanced options</p> <p>Comment: Linear interpolation was good enough for HeaveSIM therefor not attempted</p>	1
16.	Create/modify analyses in app	<p>Give users the option to either create new ones or make adjustments to existing analyses</p> <p>Comment: HeaveSIM was not a fan of creating analyses, that features should be reserved for web-application. Modifying analyses was interesting, but HeaveSIM was afraid such a feature would be misused by users</p>	2
17.	Provide services based on geo-location and accelerometer	<p>Provide weather forecast by utilizing GPS-coordinates from cellphones or calculate on-site heave by using cellphone's own accelerometer</p> <p>Comment: No value for HeaveSIM. All platform locations are well know, making such a feature useless. The same arguments are applicable to on-site heave calculations</p>	1
18.	Support for other simulations	<p>Include analyses of other operations than Connection/Tripping.</p> <p>Comment: Not done as HeaveSIM services are not properly set up for other simulations</p>	2
19.	Save earlier searches	<p>Create a page that records and displays earlier searches and results</p> <p>Comment: Not interesting for HeaveSIM</p>	2
20.	Push notifications	<p>Thing that could send out push notifications:</p> <ul style="list-style-type: none"> • Changes in weather forecast • Changes in simulations • Simulation/Analysis completed <p>Comment: Not attempted as HeaveSIM already provides services for simulation changes and there was no weather data to give notifications for</p>	2

