

Edvardsen, Joakim
Molnes, Petter
Picheta, Mateusz
Sætre, Håkon

Voice Pluck Application

A mobile application for improving the workflow
of plucking items in a warehouse

Bacheloroppgave i Dataingeniør
Veileder: Tollefsen, Mikael
Mai 2023

Edwardsen, Joakim
Molnes, Petter
Picheta, Mateusz
Sætre, Håkon

Voice Pluck Application

A mobile application for improving the workflow of plucking items in a warehouse

Bacheloroppgave i Dataingeniør
Veileder: Tollefsen, Mikael
Mai 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag



NTNU

Kunnskap for en bedre verden

ABSTRACT

Solwr is an Ålesund-based software company that creates systems to optimize trade logistics. They are responsible for the transportation-, sales-, and warehouse-logistics for major grocery chains in Norway. One of their customers, Gjørtz, uses a voice recognition system to fill pallets with the products ordered by customers. Solwr wants to explore solutions to provide this system with more flexibility, versatility, and without a paid third party for voice recognition. This thesis provides insight into the approach for the research and development of a mobile- and smart-watch application with both touch interface and voice recognition.

The main objective of the problem is to remove the need for third-party software to handle voice recognition. However, Solwr also wants to make the system easier to use, as there are certain tasks that cannot be easily done using voice commands. Therefore, we are developing a mobile application and exploring possibilities for a smartwatch application. We hope to make warehouse workers' workflow more efficient, and intuitive by combining the freedom of voice commands with a touch interface for tasks not suited by voice commands.

We utilized the SCRUM methodology, meaning bi-weekly sprint reviews with our employer and supervisor from NTNU, sprint planning, daily stand-ups, and sprint retrospectives. This helped streamline the process of development while getting useful feedback from our employer and supervisor. In addition, we visited H.I. Giørtz at their warehouse to get feedback directly from our target group.

By creating this application, Solwr gets to use our product and research to create their own product. Thus, Solwr can avoid pitfalls we uncover while developing the application, and they will have a streamlined development process where the necessary research has already been conducted.

Solwr er en Ålesund-basert programvare bedrift, som lager system for å optimalisere handleslogistikk. De er ansvarlige for transport-, salg-, og varehuse-logistikken for store dagligvarekjeder i Norge. En av deres kunder, H.I. Giørtz, bruker et stemme-gjenkjenning system for å stable produktene kunder har bestilt på paller. Solwr vil utforske løsninger for å gi dette systemet mer fleksibilitet, allsidighet, og uten å bruke en betalt tredje-part for stemme-gjenkjenning. Denne rapporten gir innsikt i tilnærmingen for forskningen og utviklingen av en mobil- og smartklokke applikasjon med både berøringsskjermgrensesnitt og stemmegjenkjenning.

Hovedmålet av problemet er å fjerne behovet for et tredje-part system for å håndtere stemmegjenkjenning. I tillegg, vil Solwr også gjøre systemet lettere å bruke, da noen av oppgavene er vanskelig å gjennomføre ved hjelp av stemmekommandoer. Derfor utvikler vi en mobil-applikasjon og utforsker mulighetene for en smartklokke-applikasjon. Vi håper å gjøre arbeidsflyten til lagerarbeidere mer effektiv, enklere, og intuitiv ved å kombinere friheten av stemmekommandoer med et berøringsskjermgrensesnitt for oppgaver som ikke egner seg for stemmekommandoer.

Vi har brukt SCRUM metodikken, noe som betyr sprintgjennomganger annenhver uke med arbeidsgiveren og veilederen fra NTNU, sprintplanlegging, daglige stand-ups og sprintretrospektiv. Dette hjalp med å effektivisere utviklingsprosessen vår, samtidig som vi fikk nyttig tilbakemelding fra arbeidsgiveren og veilederen vår. I tillegg, besøkte vi og hadde samtaler med Gjørtz, for å få tilbakemelding direkte fra målgruppen.

Ved å lage denne applikasjonen, får Solwr bruke produktet og forskningen vår for å utvikle deres eget produkt. Som følge av dette, kan de unngå feilsteg som vi har avdekt mens vi har utviklet applikasjonen, og de vil ha en effektiv utviklingprosess hvor den nødvendige forskningen allerede har blitt gjort.

PREFACE

The project presented in this paper was made in close collaboration with Solwr Software AS, especially our employer from Solwr, Reinhard Dietzel.

We chose this project, because we have a great relationship with the staff at Solwr, and we enjoyed the course Mobile Applications from our fifth semester. Additionally, Solwr wanted to research alternative approaches to their current system, and we got to explore different technologies and develop for nontraditional devices such as smartwatches.

We would like to express our gratitude to Reinhard for his involvement in this project. His guidance and constructive feedback have been extremely valuable for the outcome of this project. We would also like to thank our supervisor from the Norwegian University of Science and Technology, Mikael Tollefsen. He has played an important role in the completion of the project, by guiding us in the correct direction.

Additionally, we strongly appreciate the staff at Gjørtz, who welcomed us with open arms and let us test their current system and ask them for feedback.

CONTENTS

Abstract	i
Preface	iii
Contents	viii
List of Figures	viii
Definitions	xi
Abbreviations	xii
1 Introduction	1
1.1 Motivation	1
1.2 Project Description	1
1.2.1 Existing Solution	1
1.2.2 Problem Statement	1
1.2.3 Scope	3
1.2.4 Requirements	3
1.2.5 Boundaries	3
1.2.6 Long Term Effects	3
1.2.7 Stakeholders	4
2 Theory	5
2.1 Project Development Tools	5
2.2 Work Methodology	5
2.2.1 SCRUM	5
2.2.2 Top Down	7
2.2.3 Bottom Up	8
2.3 Version Control	8
2.3.1 Git Strategy	8
2.4 Client-Server Communication	8
2.4.1 REST API	8
2.4.2 WebSocket API	8
2.4.3 SMTP	8
2.5 Authentication and Authorization	8
2.5.1 Role-Based Access	9

2.5.2	JSON Web Token	9
2.5.3	Keycloak	9
2.6	Frameworks and Tools	9
2.6.1	Spring Boot	9
2.6.2	Swift	10
2.6.3	MidJourney	10
2.7	Development Concepts	10
2.7.1	Mapper	10
2.7.2	Debounce	10
2.7.3	Voice Recognition	10
2.7.4	Text-to-Speech	10
2.7.5	Feature Toggling	10
2.8	Database Concepts	11
2.8.1	Relational Database	11
2.8.2	Database Migration	11
2.8.3	Migration Tools	11
2.8.4	Migration Strategies	11
2.9	Testing Concepts	11
2.9.1	Test Driven Development	11
2.10	Deployment Concepts	11
2.10.1	Ubuntu	11
2.10.2	Terraform	12
2.10.3	Containers	12
2.10.4	Docker	12
2.10.5	Kubernetes	12
2.10.6	GitLab Runner	12
2.10.7	Reverse proxy	12
3	Methods	13
3.1	Planning	13
3.1.1	Research	13
3.1.2	Understanding the Problem Domain	13
3.2	Project Structure	15
3.2.1	Repository Structure	15
3.2.2	Application Structure	15
3.2.3	Design	16
3.3	Workflow	18
3.3.1	Top Down	18
3.3.2	Git Strategy	18
3.3.3	Priority	18
3.3.4	Time log	19
3.3.5	Branching	19
3.3.6	Merge request	19
3.3.7	Pipelines	20
3.4	Testing	20
3.4.1	Test Driven Development	22
3.4.2	Postman	22
3.4.3	Testing Persistence Data	22

3.4.4	User Testing	22
3.5	Technology Stack	22
3.5.1	SwiftUI	22
3.5.2	Spring Boot	23
3.5.3	Keycloak	24
3.5.4	SMTP	24
3.5.5	Endpoint Documentation	24
4	Results	25
4.1	Authentication Flow	25
4.1.1	Architecture	25
4.1.2	Features	27
4.2	Spring Boot API	27
4.2.1	Security Configuration	28
4.2.2	Entity Relations	28
4.2.3	SMTP	28
4.2.4	Leaking Internals	28
4.2.5	Error handling	28
4.2.6	Documentation	28
4.3	Deployment	31
4.3.1	Diagram	31
4.4	SwiftUI Application	31
4.4.1	Application layout	31
4.4.2	Speech Recognition	33
4.4.3	Text to Speech	36
4.4.4	Touch and Voice Concurrency	38
4.4.5	Warehouse Configuration	39
4.4.6	Additional pages	40
4.4.7	Keychain	41
4.4.8	Communicating with API	41
4.5	Real Life Environment	44
4.5.1	Pluck-flow Example	44
4.5.2	Efficiency	46
4.5.3	Removed Redundancy	46
4.5.4	Bluetooth devices	46
4.6	Android Implementation	46
4.6.1	Implementing Voice-Pluck	46
4.7	Smart Watch Companion App	47
4.7.1	Apple Smart Watch App	47
4.7.2	Android Smart Watch App	47
5	Discussion	49
5.1	Project Structure And Architecture	49
5.1.1	Project Plan and Roadmap	49
5.1.2	Reflection on Project Structure Choice	49
5.1.3	Matching DTOs Between Applications	49
5.1.4	Shared Services	50
5.2	Authentication	50

5.2.1	5.2.1	51
5.2.2	5.2.2	51
5.3	Choice of Front-end Framework	51
5.3.1	5.3.1	51
5.3.2	5.3.2	52
5.4	Text To Speech Alternatives	52
5.4.1	5.4.1	52
5.5	Voice and Touch Concurrency Challenges	53
5.5.1	5.5.1	53
5.6	Back-end	53
5.6.1	5.6.1	53
5.6.2	5.6.2	53
5.7	Database Challenges	55
5.7.1	5.7.1	55
5.7.2	5.7.2	55
5.7.3	5.7.3	56
6	Conclusions	57
6.1	6.1	57
6.2	6.2	58
6.2.1	6.2.1	58
6.2.2	6.2.2	58
6.2.3	6.2.3	58
6.2.4	6.2.4	58
6.2.5	6.2.5	58
6.2.6	6.2.6	58
7	Social Impact	59
	References	63
	Appendices:	67
A	Preliminary Project Plan	68
B	GitLab repository	73
B.0.1	B.0.1	73
B.0.2	B.0.2	73
B.0.3	B.0.3	73
C	Pluck flow Example	74
D	Research	75
	Mobile Application Frameworks	75
D.1	D.1	75
D.1.1	D.1.1	75
D.1.2	D.1.2	75
D.1.3	D.1.3	75
D.1.4	D.1.4	76

D.2	Flutter	77
D.2.1	Null Safety	77
D.2.2	Previous Experience	77
D.3	Android	77
D.3.1	Kotlin	77
D.3.2	Jetpack Compose	78
D.4	SwiftUI	78
D.4.1	Styling	78
D.4.2	Icons	78
D.4.3	Speech to Text	79

LIST OF FIGURES

1.2.1 The process flow that an employee must follow in order to fulfill a pluck order	2
2.2.1 Scrum workflow	6
3.1.1 Group member testing out current system	14
3.2.1 Design guidelines	17
3.3.1 Git strategies, Trunk (left), Flow (right)	18
3.3.2 Overview of time logged	19
3.3.3 Pipelines	20
3.3.4 SonarQube dashboard overview	21
3.5.1 Example of repository implementation	23
4.1.1 Login Sequence Diagram	26
4.1.2 Keycloak Admin Console	27
4.2.1 Database Diagram	29
4.2.2 Emails sent with SMTP	30
4.2.3 Swagger-UI	30
4.3.1 System Diagram	32
4.4.1 Swift app flow diagram	32
4.4.2 Authentication flow	33
4.4.3 Prompt when logging into the application	34
4.4.4 Speech filtering implementation	35
4.4.5 Code checking for connected devices	35
4.4.6 Mute options in the application header	36
4.4.7 Setting the Locale of the SpeechRecognizer	36
4.4.8 Speak function in TTSService	37
4.4.9 Pluck steps - Code snippet	38
4.4.10 Detailed employee view. A leader can change employee role	39
4.4.11 Warehouse config - Location Page - Update Location /w product on location	40
4.4.12 Barcode scanner in the application	41
4.4.13 Accountpage	42
4.4.14 Log page	42
4.4.15 Forgot password page	43
4.4.16 Post method exposed by the RequestService	43
4.4.17 Example usage of RequestService	43

4.5.1 Pluckflow Info/lobby	44
4.5.2 Pluckflow product list success/error	45
4.5.3 Pluckflow pluck finish/finish page	45
4.7.1 Smart Watch Sketches	48
5.5.1 Linked List Solution	54
D.1.1Expo Go output in the terminal when running the application . . .	76

TERMINOLOGY

List of all definitions in alphabetic order:

asynchronous Tasks are executed independently of each other.

authentication Provider Service that verifies the identity of a user. It is responsible for authenticating user credentials and providing a secure method for users to access the application or service.

back-end Software that runs on the server side.

cohesion Describes the degree to which elements within a module are related to one another. High cohesion means they are closely related and focused on a single task. Low cohesion means they are unrelated or have multiple tasks.

compile time The period of time when the program is converted to machine code.

coupling Describes the level of dependency between modules. High coupling means modules are closely connected, whilst low coupling means modules are more or less independent.

data structure A format for organizing and storing related data.

framework A system on top of a programming language that helps structure and reduce the complexity of an application.

front-end Software that runs on the client side.

HTTP request A request sent by a client using the HTTP protocol.

JSON Lightweight and readable way to store data in key-value pairs.

library A set of implemented functionalities.

pluck An action defining a product to be picked with associated metadata, like the amount to be picked

pluck list A list of plucks defining an order that is to be delivered by a warehouse.

runtime The state of the program while it's executing.

schema A definition that defines how data is organized in a relational database.

synchronous Code is executed in order, one after another.

ABBREVIATIONS

List of all abbreviations in alphabetic order:

API Application Programming Interface

CLI Command-line Interface

CRUD Create, Read, Update, and Delete

DBMS Database Management System

DLL Doubly Linked List

DTO Data Transfer Object

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IaC Infrastructure as Code

IoC Inversion of Control

iOS iPhone Operating System

IP Internet Protocol

JPA Jakarta Persistence API

JSON JavaScript Object Notation

JWT JSON Web Token

ORM Object Relational Mapper

RBA Role-Based Access

REST Representational State Transfer

SDK Software Development Kit

SMTP Simple Mail Transfer Protocol

SQL Structured Query Language

SSL Secure Socket Layer

TTS Text to Speech

VLAN Virtual Local Area Network

WMS Warehouse Management System

UI User Interface

UX User Experience

INTRODUCTION

The following chapter will introduce the project this report is based upon. To better understand our project, this chapter will include our motivation, description, requirements, and boundaries for the project.

1.1 Motivation

We applied for this project for several reasons. After taking the course IDATA2503 [1] we wanted to explore more about this topic. On top of that, we already had an understanding of the problem domain, because three of our group members already worked at Solwr part-time.

1.2 Project Description

This section describes the problem domain and why it needs to be solved. Additionally, we'll go over the requirements, boundaries, and stakeholders.

1.2.1 Existing Solution

H.I. Giørtz uses a WMS delivered by Solwr to pluck products ordered by customers onto Euro-pallets. This process is currently solved by wearing a headset that gives employees instructions. The headset is equipped with a microphone which is used to communicate with the system.

In figure 1.2.1 you can see a flow chart of the plucking process. The employee is guided to a location and is asked to confirm the location with a set of numbers, then specify the amount to pluck. This process is repeated until the employer has plucked every product in an order. In the figure, squares represent the system's instructions, while the sloped boxes represent the action done by the user.

1.2.2 Problem Statement

The current system uses a third-party paid system; VoiceConnect, to handle voice recognition. As mentioned in the abstract, Solwr wants to know if this system can be replaced.

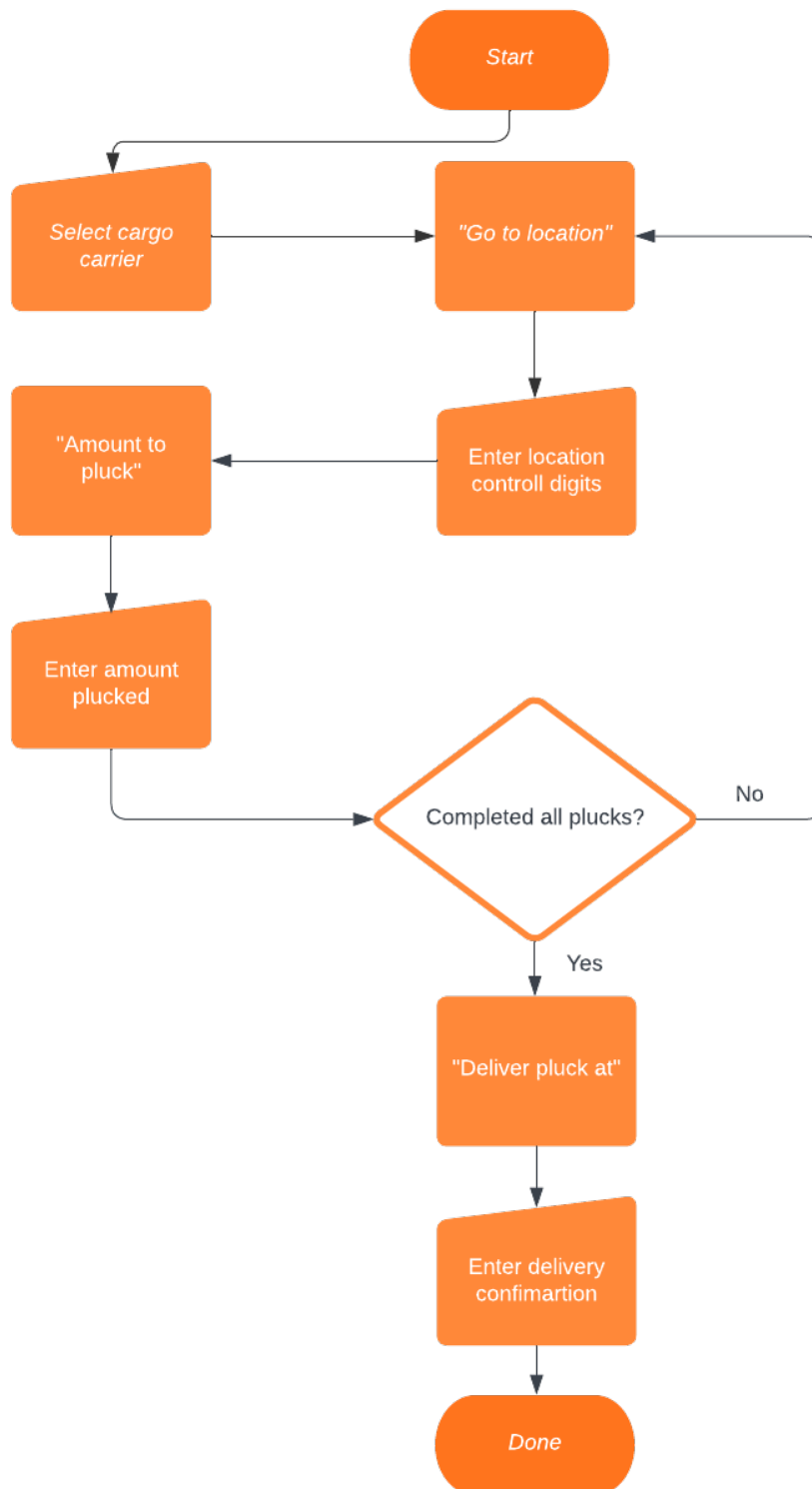


Figure 1.2.1: The process flow that an employee must follow in order to fulfill a pluck order

Additionally, some parts of the process aren't optimally solved using voice recognition. Therefore, Solwr wants a mobile application where the employees

can interact using their voice, and a touch interface for complex actions.

1.2.3 Scope

The scope of the project was somewhat vague. Solwr didn't have an exact description of what the solution should include or what we should prioritize. Therefore it was up to us to define the scope of the project and what direction we wanted to go. To research this further we visited H.I. Giørtz to test their current system and talk directly to the user group in order to establish the scope.

Originally, Solwr wanted us to create an application with the following requirements:

- A user should be able to sign up, join a warehouse or create a new one
- A warehouse should have a list of products and locations where the products are located
- A leader of a warehouse should be able to add, remove, and edit members, products, and locations of their warehouse
- **A user should be able to complete a pluck round using their voice and the touch interface concurrently**

During the research phase, the group discovered additional requirements which are further discussed in chapter 3.1.2.3.

Alongside the mobile application, we would also research the possibility to create a smart-watch application.

1.2.4 Requirements

As the system will be used in a fast-paced and loud warehouse, there were minor requirements with respect to noise. The system should not pick up voice commands from other employees working in close proximity. Additionally, the system will be used in large warehouses with industrial freezers and other obstacles where the WiFi signal is easily lost. Because of that, it was important that the voice-recognition part could work without an internet connection.

1.2.5 Boundaries

Solwr already delivers a WMS to H.I. Giørtz, which we could have integrated our application with. However, there were a lot of database tables that we wouldn't need in our WMS. So, to avoid an unnecessary extension of privileges, and ensure that we could work more independently, we would develop our own WMS with a REST interface.

1.2.6 Long Term Effects

Our solution seeks to simplify and optimize the workflow of employees at H.I. Giørtz. By introducing a touch interface to their workflow, employees can complete longer and more complex tasks faster than using the voice-recognition interface on its own.

1.2.7 Stakeholders

Solwr will use the results and the research of our project to evaluate the solution, and potentially develop their own application based on our work. H.I. Giørtz has been a great collaborator and will make use of any further development of our solution.

However, our group owns the right to ownership of the results. This was decided so that we can use and demonstrate our results freely as we want.

In this chapter, we will provide knowledge and background theory of methods, technologies, and tools used throughout the project. Reading this will give a good understanding of the tools and technologies discussed later on in the report.

2.1 Project Development Tools

This section contains theory about project development tools and terminology used in our project.

2.2 Work Methodology

2.2.1 SCRUM

Scrum is an agile framework/methodology which enables teams and/or organizations to structure and govern their work through a set of values, principles, and practices [2]. The methodology is often practiced during software development as it is designed to help teams adapt and react as new problems occur with its iterative workflow. Each sprint feeds back to the next, where you can learn and adapt from previous experiences.

2.2.1.1 Sprints

When working with the SCRUM methodology, teams typically work in sprints, which are time-bound iterations ranging between one and four weeks. The purpose of sprints is to provide a defined time frame for the team to work towards a specific goal while enabling them to manage their workload more efficiently [3]. At the beginning of each sprint, a goal is set, which should be both challenging and achievable. The goal serves as a guiding principle for the team's effort during the sprint, providing focus and direction.

2.2.1.2 Sprint Meetings

A sprint consists of meetings that facilitate communication, collaboration, and transparency among team members and stakeholders. Each meeting serves a spe-

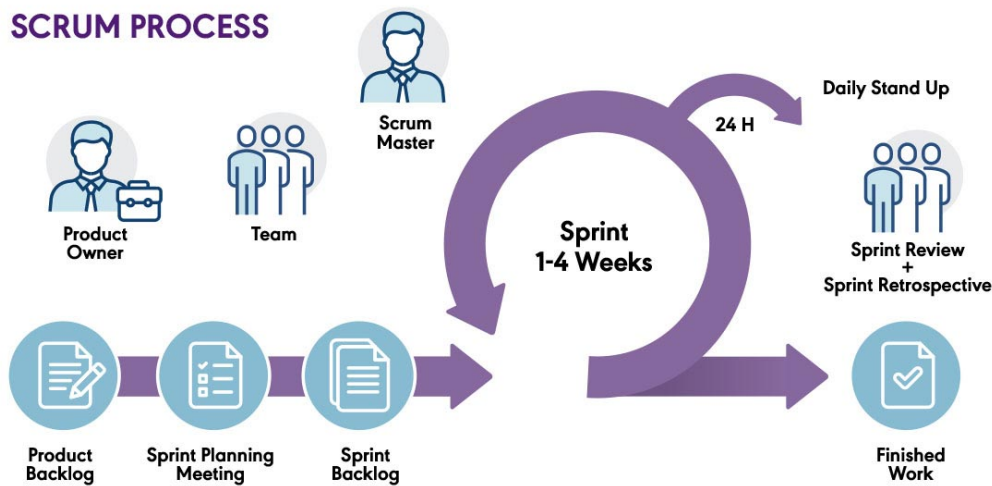


Figure 2.2.1: Scrum workflow

cific purpose in helping the team achieve its sprint goals.

2.2.1.3 Sprint Planning

Sprint planning is a meeting that occurs at the beginning of each sprint. Here the team together with the stakeholder discuss and plan the goal for the upcoming sprint. The participants of the meeting prioritize tasks from the backlog, taking into consideration various factors such as their complexity and dependencies. It is also recommended that each task should be assigned an estimated time to complete. The goal of sprint planning is to select tasks that will enable the team to achieve the sprint goal, which serves as a focal point for the team's efforts during the sprint.

2.2.1.4 Sprint Review

At the end of the sprint, the team meets up with the product owner to review the progress that was made during the sprint. This meeting also provides an opportunity for the team to demonstrate the features they've been working on.

2.2.1.5 Retrospective

The retrospective is an internal meeting for the team that takes place at the end of each sprint. During this meeting, the team discuss and reflects on the sprint they just completed. The goal is to identify aspects that went well or wrong, what to continue or stop doing, and what to start doing differently. It's a valuable opportunity for the team to learn from previous experiences, enabling the team to grow and work more efficiently. The insights gathered from the retrospective inform the planning of the next sprint, ensuring the team can incorporate their learnings and improve their performance.

2.2.1.6 Daily Stand-up

The daily stand-up is a brief internal meeting that takes place once a day. The meeting should be kept short, lasting no more than 10 minutes. Its purpose is to keep every team member up to date and to discover any obstacles, known as blockers, that are hindering individual members. It's important to note that if any blockers are discovered during the daily stand-up, the team should not start discussing how to address the issue. Instead, the meeting serves as an opportunity to identify blockers and address them outside of the meeting.

2.2.1.7 Roles

There are three main roles; the product owner, the scrum master, and the development team member [4]. The product owner's responsibility is to prioritize the tasks in the backlog based on the customer and business requirements. The scrum master ensures that the team is following the SCRUM methodology and prompts them when they deviate from its guidelines. The developer role is a bit more complicated. With developers, it's meant all people that help build the project, not just the engineers. Their responsibility is to deliver the work they develop during the sprints [4].

2.2.1.8 Issue Tracking

In Scrum, every task is separated into issues. Normally, there are three to four different types of tasks depending on the scope; epics, stories, tasks, and subtasks. The intent is to group certain tasks together and keep the project organized.

Epics are the highest level of tasks. An epic is a group of stories that all belong together [5]. A story is an action a user should be able to do in the system. A story should describe, who, what, and how a client can do a certain action. A task is a single unit of work that a developer has to do in order to help complete a story [6].

All of the tasks are placed in a backlog. It's the product owner's responsibility to prioritize these tasks. When the development team is to start a new sprint, they simply select as many issues as they think they can complete from the backlog.

2.2.1.9 Planning Poker

Planning Poker is a way to accurately estimate issues by having each team member estimate it individually. When everyone has estimated the issue, they discuss the reasons for their estimate. As each team member has a different domain of knowledge, this will result in a more accurate estimation of issues.

2.2.2 Top Down

Top down strategy is an approach where development is started by creating UI and defining the layers required based on the UI. Front-end is implemented first, and the back-end with database relations is implemented as required.

2.2.3 Bottom Up

Bottom up is an approach where the database and underlying logic are typically implemented first. Front-end is implemented based on the underlying logic.

2.3 Version Control

Version control is a system responsible for tracking and administrating changes to a code base [7]. Git is one of the most used version control systems.

2.3.1 Git Strategy

Git offers various branching strategies that allow developers to work on different features and changes simultaneously without interfering with each other's work. These are repository-specific guidelines on how developers should submit new code to the codebase.

2.4 Client-Server Communication

2.4.1 REST API

REST API is an architecture for building web applications. It follows a stateless communication between the client and server, meaning all the information need to process a request is passed in the request [8]. This makes it easier for the application, as it has no responsibility for storing data between requests made by a client. Altogether, this allows the server to handle requests in a uniform and scalable way, without relying on any client-specific state.

2.4.2 WebSocket API

Web sockets are advanced and open up the possibility for two-way communication [9]. Whereas with REST API, the client has to "take the initiative" by sending a request, a WebSocket API can send event-driven responses without the need for the client to send a request.

2.4.3 SMTP

Simple mail transfer protocol is the most commonly used communication protocol for email transmission. Mail servers and other message transfer applications use SMTP to send and receive email messages. Many SMTP providers exist, and even prominent corporations like Apple offer their own SMTP service.

2.5 Authentication and Authorization

Authentication and authorization are two important terms that are often used in tandem, however, they have two separate meanings.

Authentication is the act of proving that something is true. In computer systems and applications, this often refers to proving a user's identity. This is important in case an application is storing sensitive user data or otherwise classified data.

Authorization refers to granting user access and permissions to different files and components in the system, based on their roles and responsibilities. An example of authorization is assigning different access levels, such as leader or admin. The authorization decides how much power and access a user has within the system.

2.5.1 Role-Based Access

RBA revolves around the idea of assigning permissions to users based on their roles. It's a simple yet manageable approach that is less error-prone than assigning permissions to each user individually. [10]

2.5.2 JSON Web Token

"JWT is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object" [11]. After the user is logged in, each subsequent request contains the token, and it can be decoded to retrieve information about the user.

2.5.3 Keycloak

Keycloak is an authentication provider that handles user authentication independently. Users authenticate with Keycloak instead of with individual applications. This lets you have features such as single-sign on.

2.6 Frameworks and Tools

2.6.1 Spring Boot

Spring boot is an open-source Java framework for micro-services and is often used to create web applications or APIs [12].

2.6.1.1 Beans

Spring Boot heavily uses beans to manage components throughout the application [12]. A bean is a component that is managed by the Spring Boot framework [13].

2.6.1.2 Swagger

Swagger is a tool for documenting API. Swagger is used to document endpoints in an API, explaining what the endpoint expects in the request, and what it returns [14]. This way, developers can read through the documentation to get an understanding of what the API does, without having to look at the source code.

2.6.2 Swift

SwiftUI is a modern framework developed by Apple for building applications across its operating systems, including iOS, iPadOS, macOS, watchOS, and tvOS [15].

2.6.3 MidJourney

MidJourney is an AI service that makes text-to-picture. You can make many types of art with it, from real-looking to more creative styles. MidJourney creates very high-quality, structured, and detailed images.

2.7 Development Concepts

2.7.1 Mapper

Mappers are utilized to transform objects from one type to another. In multilayered applications, these are commonly used to map database entities to DTOs. Thus minimizing the potential of disclosing sensitive information, such as database id.

2.7.2 Debounce

Debounce is a technique where you ensure that a function is not called too frequently [16]. For example, with a search input. Instead of searching for the value of the input every time it changes, you can debounce the input making it only search x milliseconds after the last key was pressed.

2.7.3 Voice Recognition

Voice recognition is a technology that allows machines to record and perform actions based on inputs through voice [17].

2.7.3.1 On-device recognition

On-device speech recognition refers to the ability of a device to perform speech recognition tasks locally, without requiring an internet connection.

2.7.4 Text-to-Speech

Text-to-speech is a technology that allows machines to output a string of text to a spoken sound [18]. Modern tools allow the sound to be more realistic and human-like.

2.7.5 Feature Toggling

Feature toggling is a concept that makes it possible to toggle features in runtime, without the need for a manual deployment [19].

2.8 Database Concepts

2.8.1 Relational Database

A relational database is a set of data stored in tables with rows and columns. A data point is stored as a row in a table with up to multiple columns representing attributes about that entry. It's called relational because tables can have relations to each other via pointers [20].

2.8.1.1 In-memory Database

A database that is running in memory, meaning it will be generated every time the application starts. On application exit, the database will be deleted and all data will be lost.

2.8.2 Database Migration

Migration is a set of instructions to update one or more databases to one or more target databases [21]. It can update the database schema, modify the data, and more.

2.8.3 Migration Tools

Flyway [22] and Liquibase [23] are two migration tools commonly used with spring boot. A migration tool is a service that performs migrations and helps version control your migrations.

2.8.4 Migration Strategies

Simple updates to the database schema like adding a table is straight forward. However, the more complex the change is the harder it will be to migrate the data in the database. Migrations strategies are predefined solutions to common problems related to migrating data.

2.9 Testing Concepts

2.9.1 Test Driven Development

Test-driven development involves implementing tests for new features and functionality before the features are implemented themselves. Because of this, the test will initially fail and the next step is to implement the actual features and make the tests pass [24].

2.10 Deployment Concepts

2.10.1 Ubuntu

Ubuntu Server is a version of the Ubuntu operating system that is designed specifically for servers. It provides a stable and secure platform for running applications,

hosting websites, and managing data.

2.10.2 Terraform

Terraform is an IaC tool that enables the deployment and configuration of servers with specific settings, such as the desired resources and IP address. This allows for fast server configuration and management and makes it possible to deploy an identical server anywhere in the cloud with one command.

2.10.3 Containers

Containers are lightweight virtual servers that can be easily deployed. They typically use a lightweight version of Ubuntu that comes with your application built into the file system. This allows it to entirely contain all of the dependencies used by the program and makes it deployable on any server without prior configuration.

2.10.4 Docker

Docker is a technology that enables containerization, allowing the creation of lightweight, reusable containers that can hold an application. This enables fast and efficient deployment. In addition, container deployment can be triggered from the pipeline, which eliminates the need for human involvement.

2.10.5 Kubernetes

Kubernetes is a container orchestration platform that provides a powerful and flexible way to manage containerized applications. It is a technology that builds upon the concept of containerization and allows for load balancing, monitoring, updating, and scaling up applications.

2.10.6 GitLab Runner

GitLab Runner is a feature provided by GitLab that involves installing a program on a server that is responsible for executing the code used in your pipelines. That server is then responsible for running code like building, testing, and deploying the application on the server.

2.10.7 Reverse proxy

A reverse proxy allows us to route multiple domains and subdomains to different hosts and ports. It also makes it possible to install SSL certificates to make the communication between the front-end and the back-end secure.

In this chapter, we will discuss the method and tools used in the project, both at the project management level and the development level.

3.1 Planning

We began our project work by creating a preliminary project plan. The plan included a list of main activities and a roadmap with milestones to keep the group on track. The details of this plan can be found as an attachment in Appendix A.

3.1.1 Research

After an extensive planning phase, the next step was to research the domain and what technologies we could use. Each team member looked into different mobile application frameworks with a key focus on native support provided by the framework and the developer experience.

A simple application was developed in each of the frameworks to get a feel of the core functionalities. Furthermore, more extensive research such as reading posts and articles about the frameworks was documented and later discussed, both in the team, but also with our supervisor and stakeholder.

For more detail about the research, read appendix D.

3.1.2 Understanding the Problem Domain

Before we could begin development it was crucial to get a better understanding of the problem domain. Solwr arranged for us to visit H.I. Giørtz to get first-hand experience with the current system. As well as getting a tour of the warehouse, we got the opportunity to be an employee for a day. This really helped us understand the system behind the process and further gave us a better understanding of what Solwr wanted to develop.

Throughout the day, we worked alongside employees at the warehouse and interviewed them about the current system. A recurring theme was that the current voice-controlled system works really well for the most part. However, in some situations, it lacked some functionality resulting in inefficiency. Most of the



Figure 3.1.1: Group member testing out current system

employees looked optimistic at the application we discussed with them, however, some concerns were raised, which are discussed in chapter 3.1.2.2.

Some flaws and room for improvement were discovered with the current system. These were noted and discussed later in the development process, further improving our complete implementation of the system.

3.1.2.1 Problem With Current System

List of flaws with the current system:

- Heavy products should be placed at the bottom of a cargo carrier. Normally, this is not a problem as the pluck list should be optimized for this. However, on some occasions, it fails and the employees have to rearrange the pluck list themselves. With voice only, there is no fast way of doing this operation.
- The headset was uncomfortable and fell off when you were to bend down to pluck a product from the lower shelves.
- Some of the areas in the warehouse were loud resulting in a hard time understanding the commands given by the system.
- The current system had a hard time picking up the correct digits resulting in employees having to repeat a command multiple times.

3.1.2.2 Concerns For The Application

Some of the employees at Giørtz also raised some concerns about the application:

- The application has to implement at least the same level of voice control as the current system, as having both hands free for driving the truck and picking up products is crucial.

- There exist strict policies for using mobile devices while on the truck. The truck should be standing still if they were to use their personal phones.

3.1.2.3 Additional Requirements

With both the problems and the concerns in mind we extended the requirements list with the following functionality:

- A user should be able to rearrange the plucks in a pluck list using the touch interface
- A user should be able to use a connected headset with the application, preferably earbuds.
- A user should be able to cancel the commands uttered by the device

3.2 Project Structure

When developing a program or a system, it's important to keep your work organized. For developers, this means structuring your code base systematically and logically.

3.2.1 Repository Structure

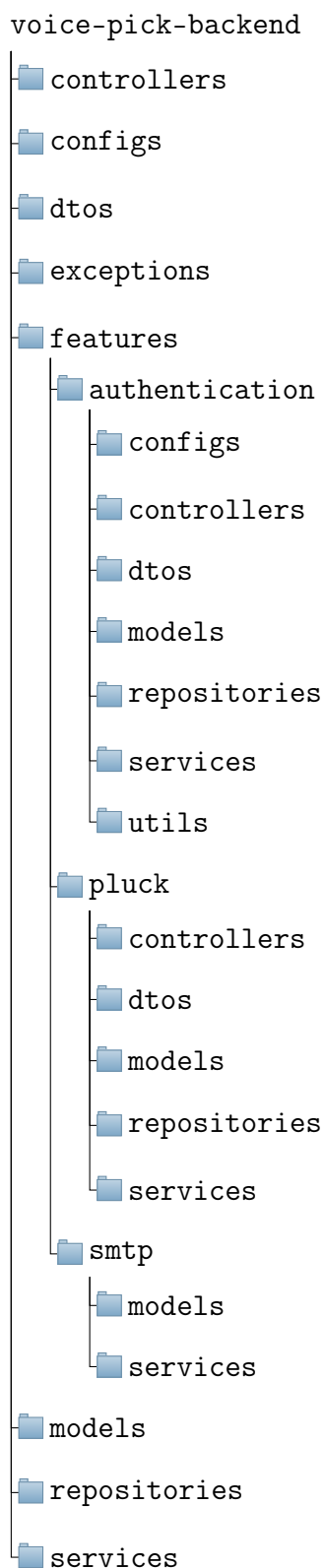
We have decided to implement a multi-repo approach for our project and have split it into three repositories: an iOS front-end, a Spring Boot back-end, and an Infrastructure as Code (IaC) repository for setting up our infrastructure. This approach was chosen because these repositories operate independently of each other, with the front-end relying solely on API calls to communicate with the back-end. Adopting a multi-repo structure offers several advantages, including increased flexibility, scalability, enhanced isolation, and simplified maintenance, making it a more suitable solution for the long term in projects that use independent technologies.

3.2.2 Application Structure

To help developers understand and maintain a code base, a well-organized and structured project is required [25]. A good project structure will help with coupling and cohesion which will further help developers refactor, improve, and add new features to the system. This is especially important for our project, as developers at Solwr might look into our code base if they decide to create a similar application in the future.

We decided to go for a featured strategy where we encapsulate each feature in its own package. Every component related to a feature is found in the respective features package [26].

Here's a snippet of the tree of structure in our API application:



3.2.3 Design

Figma is a sketching tool to develop user interfaces and was used as our primary design tool. The team's previous experience with Figma was the main reason for

the use of this tool, together with its ease of use and collaboration features.

Our initial approach was to create a design guideline that would define colors, fonts, and components used throughout the application. Since Solvr already has systems in production, we followed their theme as reflected in the finished design guidelines 3.2.1. The guideline was developed for both light and dark themes. With this, we developed sketches for the application that all were uniform.

3.2.3.1 Following Principles

Design principles are a set of rules describing how to develop an effective and attractive design. These were extensively used throughout the process, ensuring a visually pleasing and accessible design.

3.2.3.2 Design Feedback

As the design developed, we had close communication with both our employer and workers at H.I. Giørtz, ensuring that their adjustments and concerns were addressed.

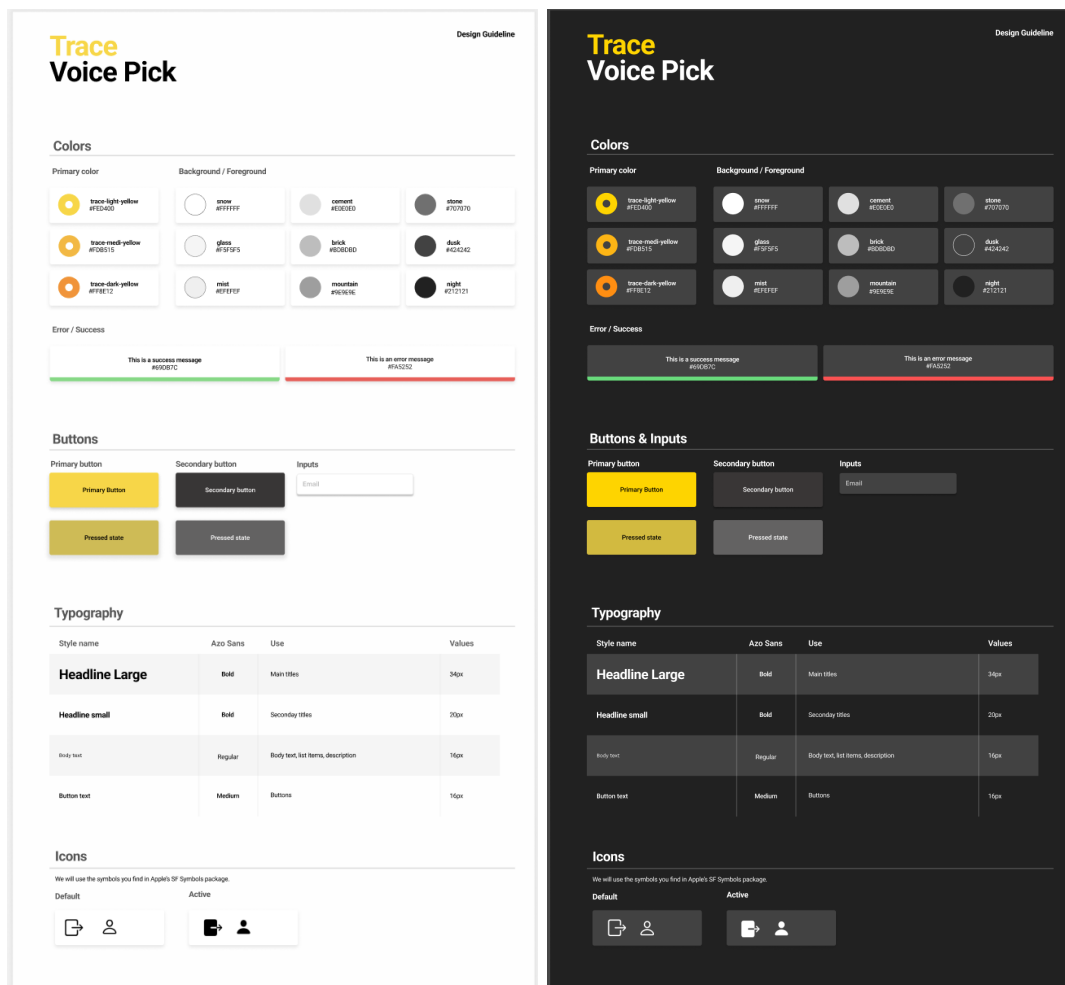


Figure 3.2.1: Design guidelines

3.3 Workflow

3.3.1 Top Down

Our team and advisor decided to use a top down approach for our project, as it would make things simpler and help reduce unnecessary work. By creating the UI of a feature first, we could implement our REST API with the least amount of features for the front-end to work. By not creating the back-end first, we avoid over-engineering our back-end by introducing more logic than necessary.

3.3.2 Git Strategy

Git allows for multiple branching strategies where the two most common are Trunk-Based Development (TBD) and Gitflow [27].

In the Trunk-Based Development approach, developers work on their features in a separate branch and merge their changes back into the master branch, also known as the trunk. This approach best suits smaller teams working on projects with short release cycles.

Gitflow is a more complex branching strategy that involves multiple long-lived branches. It includes a "develop" branch where new features are integrated and a "master" branch that contains only the stable releases. Developers create feature branches off the develop branch to work on specific features, and when complete, they merge these changes back into the develop branch.

Trunk-based git strategy was chosen, as it is more suited for small teams and codebases that do not have closely integrated components, such as our Swift front-end and Spring Boot back-end.

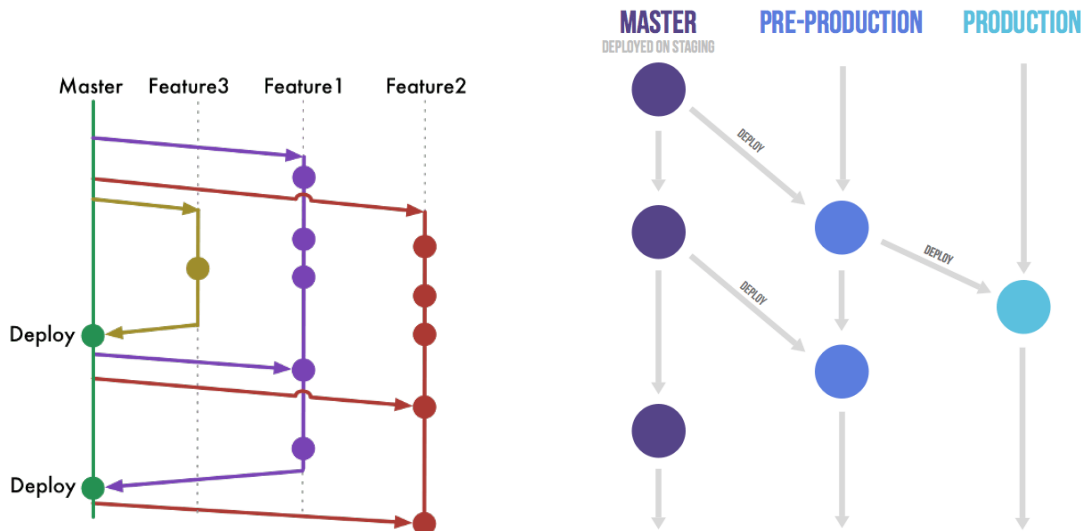


Figure 3.3.1: Git strategies, Trunk (left), Flow (right)

3.3.3 Priority

Jira was used as our main tool for prioritizing tasks. Our backlog was sorted by priority based on feedback from both supervisor and employer as well as each

individual task was labeled with a priority ranging from lowest to highest. This meant that for each sprint, we simply would select as many tasks as we thought we could complete from the backlog. Using this method allowed us to be organized and follow the goals set for each sprint.

3.3.4 Time log

Planning poker has been used during sprint planning to estimate the complexity of issues. To keep track of how accurate the estimations are, we have used Jira's functionality to log hours spent on each individual issue. Throughout the process, we realized that estimation is one of the harder parts of project management, as you never know when you will encounter problems.

Time tracking has also been positive by providing a graphical overview displaying that every team member is contributing to the project. Motivation is kept as you feel like other members are equally active. Figure 3.3.2 shows the statistics of the time spent toward the end of the project.

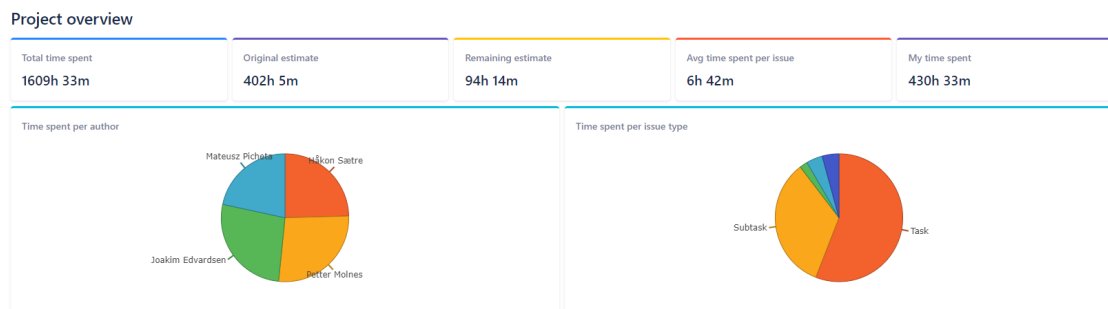


Figure 3.3.2: Overview of time logged

3.3.5 Branching

We have used a trunk-based branching approach in both our front-end and back-end. Together with Jira where we have defined and organized issues, this makes it easy to work only a single issue, one at a time. Jira offers integration with GitLab which we have used to our advantage. To work on a feature, we simply opened up the issue we were to work on in Jira. From the sidebar of the issue, we could create a branch that would automatically be created in the repository. All work related to the issue was developed in the respective branch.

3.3.6 Merge request

When a task was completed, a merge request was opened for review. In the beginning, we simply merged the branch directly into the trunk branch. However, later down the road, the team decided to incorporate merge reviews into our workflow to enforce better code quality.

3.3.6.1 Restriction

To maintain high code quality, we had strict rules for submitting new code. All merge requests had to be approved by another team member, and direct pushing

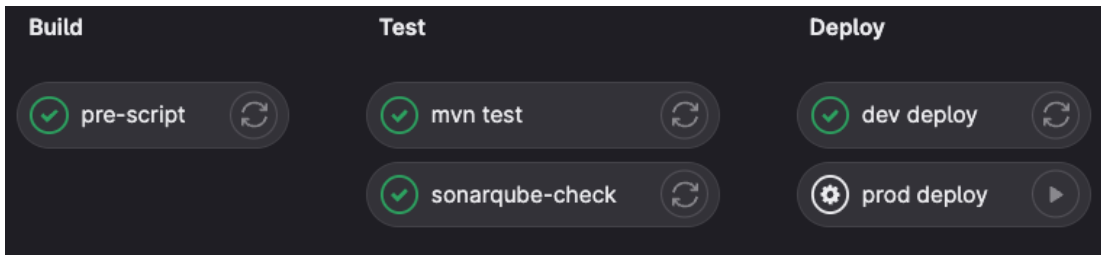


Figure 3.3.3: Pipelines

to the trunk was not allowed. This collaborative approach allows multiple team members to review each other’s code, suggest improvements, and ensure that the code meets the team’s standards. In addition, this ensured that all team members fully understand the application.

3.3.6.2 Commit Naming Conventions

Since Jira was integrated with GitLab, we utilize a standard method of documenting them using Jira issue numbers. This means starting a push or merge request with the relevant Jira issue number, for example, VP-151. This clarified the specific tasks that were implemented within the branch.

3.3.7 Pipelines

Gitlab has a function called pipelines, also called CI/CD. This feature lets us automate actions each time we submit new code to our codebase.

Figure 3.3.3 displays our pipelines that is split into 3 stages; build, test, and deploy. The deployment step deployed two versions of the API, a development deployment, and a production deployment. The development deployment was continuously deployed, which means it redeploys with the newest version each time someone adds new code to the repository. The production deployment was continuously delivered, which lets us deploy it with one click of a button, as this should be a business decision rather than an automated process.

Before deploying any code, it must pass our test stage, which includes running all unit tests and performing a SonarQube analysis. The analysis checks for additional requirements, such as having at least 60% test coverage, passing all tests, and meeting code quality and security standards in all of SonarQube’s categories. Additionally, SonarQube scans the repository, giving feedback on vulnerabilities and code smells. With its dashboard, it gives a quick overview of the state of the application as shown in figure 3.3.4.

3.4 Testing

Testing plays a large role during the development process. As the size of a project scales up, the significance of robust testing increases, as it provides assurance that the code is working correctly.

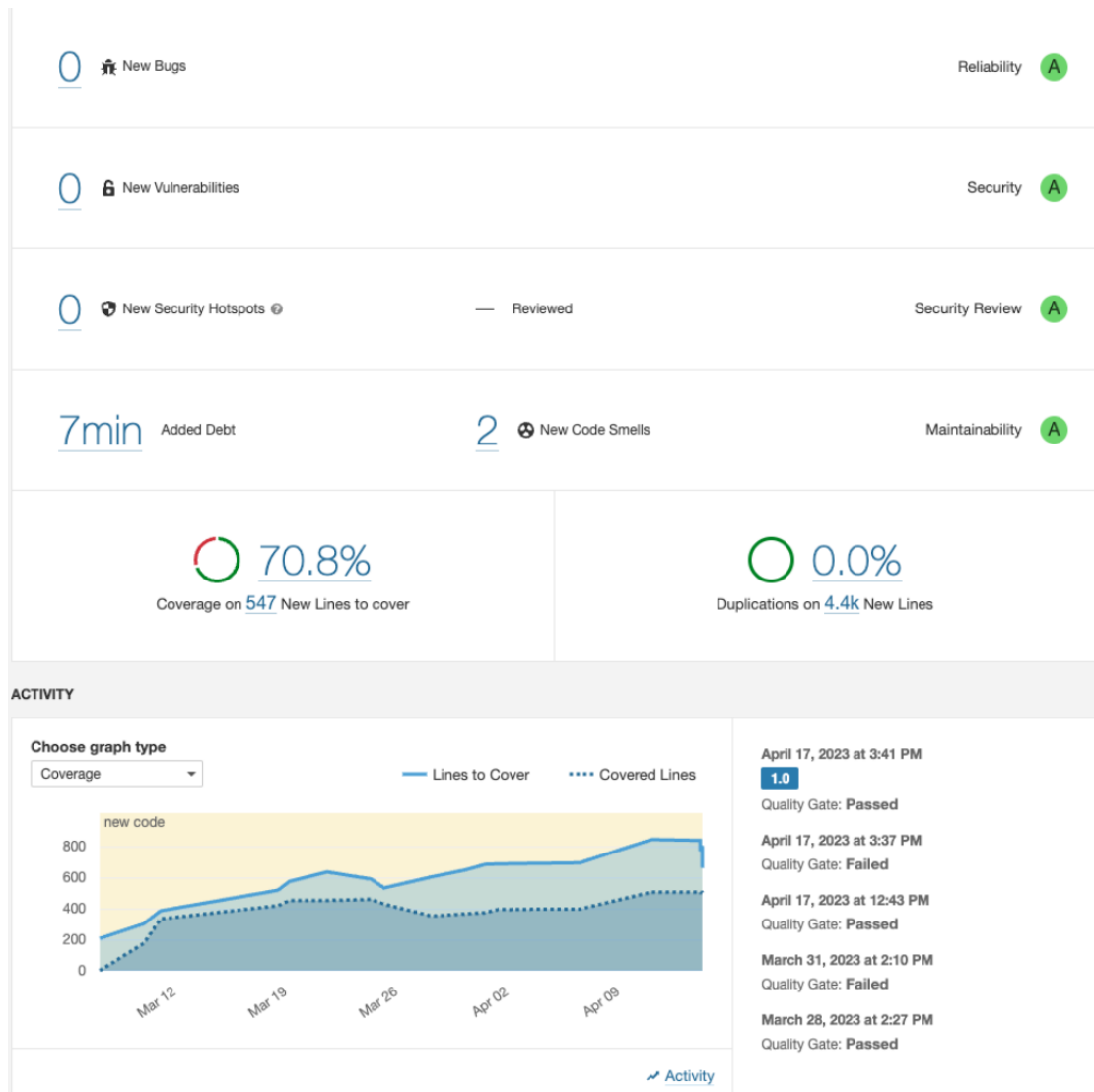


Figure 3.3.4: SonarQube dashboard overview

3.4.1 Test Driven Development

When implementing the API, we strived to follow the test-driven development methodology. This is important because as the application grows a thorough test coverage will help speed up the development process and eliminate future bugs.

3.4.2 Postman

Our initial approach was to utilize Postman. Postman is a tool for testing API and makes it possible to send HTTP requests and validate the response. In Postman you can create teams and invite members so multiple people can cooperate and use the same test suit. However, Postman requires a subscription if your team is larger than three members.

Postman tests can be implemented in a pipeline, although it's simpler to run maven tests defined in the application. As a result of this and the subscription fee, we decided to drop Postman and instead rely on Spring Boot testing. This way our tests are version controlled together with our API all in one place.

3.4.3 Testing Persistence Data

Our API utilizes Spring Boot's in-memory database when executing the testing. This is to make sure the database is in a consistent state before the tests are executed, as a deployed database might change over time. In addition, our tests strictly follow proper stages with setup, execution, validation, and clean-up, further making sure the state of the application is consistent between each test.

3.4.4 User Testing

We've been having a close dialog with both our employer and workers at H.I. Giørtz. They have given us insightful information about our UI/UX as well as the features we've provided in our solution. We tested our final product in Solwr's warehouse since Giørtz was busy at the time. Additionally, we have provided them with a recording of the workflow of the application, making it possible to get some feedback. The link to the video can be found in Appendix C.

3.5 Technology Stack

In this section, we will give you an insight into what technologies our solution uses, and why we chose these technologies.

3.5.1 SwiftUI

SwiftUI was utilized to create our mobile application. Native support was the primary deciding factor, but other pros such as SwiftUI's easy learning curve, reusable components, and dynamic updates, further reinforced the decision. These features allowed us to create efficient UI/UX designs for the complex system of a warehouse. SwiftUI also offers theming, which allowed us to easily create both light and dark modes as requested by the employer.

```
1 usage  ⤴ Joakim Edvardsen
18 List<PluckList> findByUser(User user);
19
1 usage  ⤴ Joakim Edvardsen
20 @Query("SELECT COUNT(PluckList) FROM PluckList p WHERE p.finishedAt IS NOT NULL AND p.user.uuid = :uuid")
21 Integer countCompletedPluckList(@Param("uuid") String uuid);
```

Figure 3.5.1: Example of repository implementation

When deciding on front-end framework, native support was the main factor. However, the developer experience was also taken into consideration, although not as crucial. The XCode editor provides multiple features that enhance the developer experience. Some of them are; canvas, which allows for live preview of UI changes. Documentation of predefined components directly in the editor, and the opportunity to run the application both in emulators and physical devices.

3.5.2 Spring Boot

Spring Boot was our choice of back-end as all team members have previous experience with the framework. Further, it's the primary framework used by Solwr. Our Spring Boot is connected to a PostgreSQL instance. This is also a technology used by Solwr that most of our team members were familiar with.

3.5.2.1 Split Level of Concern

The API uses a standard Spring Boot and REST API structure with controllers, services, repositories, and entities. The entities define the tables in the database, the repository has the responsibility to communicate with the database, the services execute all the business logic, and finally, the controllers handle the requests, map DTOs to the entities, and verify the outcome of the logic executed in the services layer before sending responds respectively.

3.5.2.2 Communicating With Database

Our API communicates with PostgreSQL via the repository components. Each entity has a repository component. These implement the JPA interface which is the ORM used to translate Java to SQL used to query the database. The JPA interface defines predefined queries as well as it allows for customization for advanced queries.

3.5.2.3 Lombok

Utilizing Lombok in our spring boot project, provided us with annotations to reduce repetitive code and simplify the Java development. It automates the generation of boilerplate code, such as getters, setters, constructors, toString, and equals methods. This made it easier for us to focus more on the project logic rather than writing getters and setters. The commonly used annotations in our project are @Data, @Getter, @Setter, @NoArgeConstructor, @AllArgsConstructor, and @RequiredArgsConstructor.

3.5.3 Keycloak

We choose to use Keycloak as our authentication provider for several reasons. Solwr already uses Keycloak for their system, so it was important for us to show that we could take into account their existing choices of technologies. Additionally, it was supposed to be quite straightforward using Keycloak's graphical user interface. It would relieve us of creating a sign-in/sign-up form, as Keycloak handles all that in an in-app browser pop-up. This decision is discussed further in chapter 5.2.

3.5.4 SMTP

SMTP was chosen as a means for sending emails because it's a commonly used protocol for exchanging email messages over the internet. We opted to utilize Apple's SMTP service and connected it with our Spring Boot application to enable this functionality. Our application currently utilizes SMTP to send emails for verification, password resetting, and warehouse invitations. The emails provide templates where randomly generated codes are passed as a means of verification.

3.5.5 Endpoint Documentation

Swagger was used to document every endpoint exposed by the API. It gives an organized overview of all available endpoints with additional information like expected request metadata and response types.

In the following chapter, we will provide insight into the result of our research and development process.

4.1 Authentication Flow

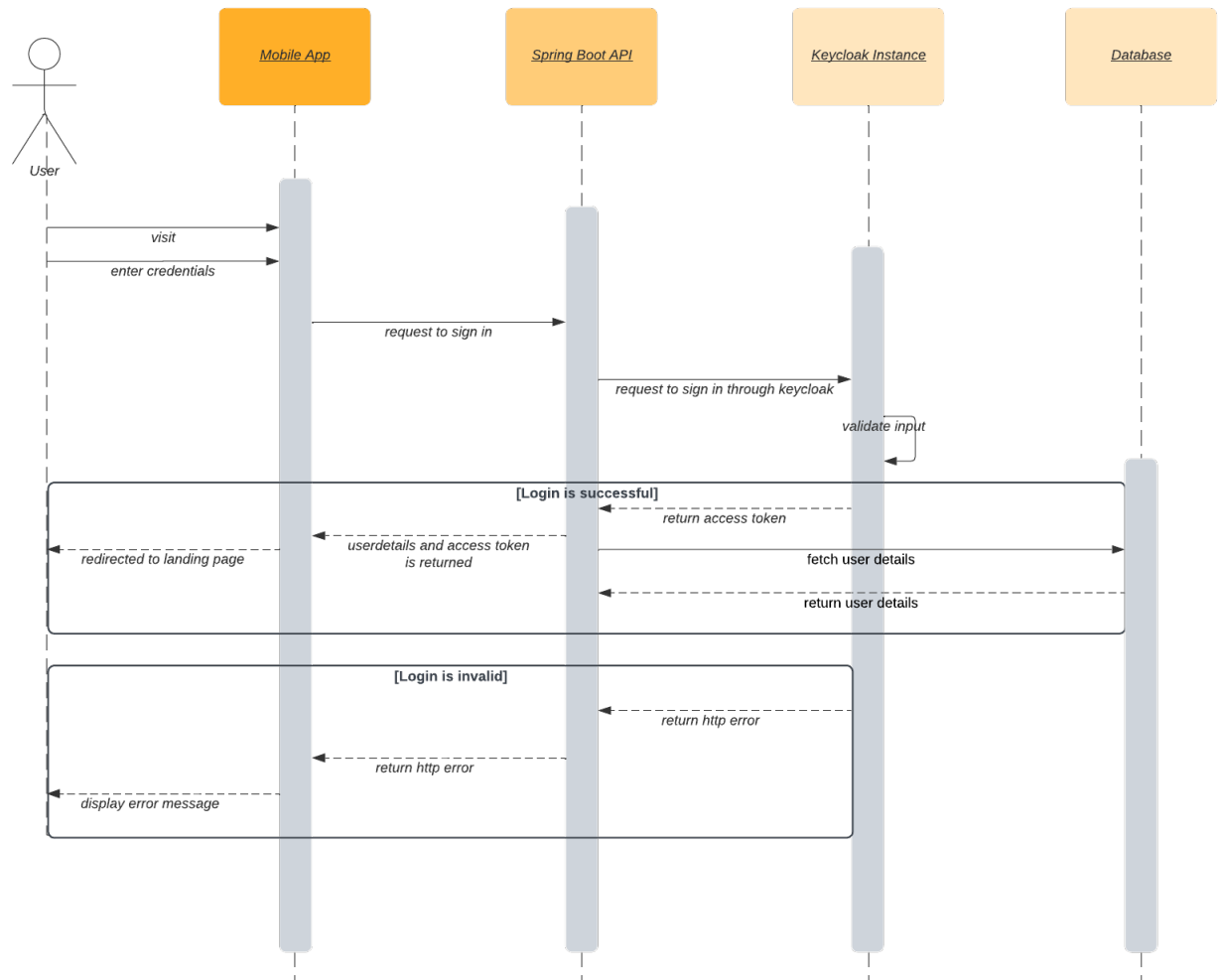
Although authentication was not a primary objective for our project, it was still required, as our application was directed toward a group of users with different roles and authorities. Because of the complexity and similarities of such features, we went with Keycloak as our authentication provider. This lets us use the Keycloak Admin Console to set up roles, access token expiration time, and manage users. Additionally, we use the Keycloak REST API to send requests to our Keycloak instance.

4.1.1 Architecture

In figure 4.1.1 you can see all the actions our system executes when a user logs in through our mobile application. The mobile application will send all its requests through our REST API, and the Spring Boot application will send a request to the Keycloak instance and give a response to the user based on the result from Keycloak. The purpose of this was to maintain the confidentiality and security of the client secret required for communication with our Keycloak instance. By only using the secret in our Spring Boot application, the risk of unauthorized access to our authentication provider was minimized.

4.1.1.1 Keycloak

As mentioned above, Keycloak was used as an authentication provider for our application. The setup of Keycloak realms was fairly straightforward with the Keycloak Admin Console. For the project, we created two realms, one meant for production and one that we as developers could use when testing the application. Keycloak makes it easy to implement role-based authentication by adding custom roles. Keycloak uses JWT where all information about the user and the session is stored.

**Figure 4.1.1:** Login Sequence Diagram

The screenshot shows the Keycloak Admin Console interface. The top navigation bar includes the Keycloak logo and the user 'Bachelor'. The left sidebar contains a navigation menu with sections: Configure (Realm, Settings, Clients, Client Scopes, Roles, Identity, Providers, User, Federation), Authentication, and Manage (Groups, Users, Sessions, Events, Import, Export). The 'Users' section is selected. The main content area is titled 'Users' and features a search bar with a 'View all users' button. Below the search bar is a table of users with the following columns: ID, Username, Email, Last Name, First Name, and Actions. The table contains 15 rows of user data.

ID	Username	Email	Last Name	First Name	Actions
9469a454-61d7-...	haakonfs@hotm...	haakonfs@hotm...	Sætre	Håkon	Edit Impersonate Delete
a407775b-77b5-...	haukaun99@gm...	haukaun99@gm...	SÆTRE	Håkki	Edit Impersonate Delete
ded300bd-cb45-...	invite@me.please	invite@me.please	me	invite	Edit Impersonate Delete
cd136c18-e447-...	inviter@test.com	inviter@test.com	test	inviter	Edit Impersonate Delete
f577ba6-1339-4...	j.oakimedvardse...	j.oakimedvardse...	E	J	Edit Impersonate Delete
5cfa9f1a-01b0-4...	joakim-kul@hot...	joakim-kul@hot...	Edvardsen	Joakim	Edit Impersonate Delete
1908336b-3dff-4...	joakim.edvardse...	joakim.edvardse...	E	J	Edit Impersonate Delete
e9588d89-2c06-...	joakimedvardse...	joakimedvardse...	E	J	Edit Impersonate Delete
79ee8a78-2501-...	joakimedvardse...	joakimedvardse...	Edvardsen	Joakim	Edit Impersonate Delete
748e3db8-4136-...	m@picheta.dev	m@picheta.dev	P	M	Edit Impersonate Delete
5ae1c898-7400-...	pettermolnes@...	pettermolnes@...	Molnes	Petter	Edit Impersonate Delete
6899b1d5-0af6-...	pettermolnes@...	pettermolnes@...	Molnes	Petter	Edit Impersonate Delete
3d98e142-7f22-...	pettermolnes@...	pettermolnes@...	Molnes	Petter	Edit Impersonate Delete
820db795-f142-...	pettermolnes@i...	pettermolnes@i...	me	invite	Edit Impersonate Delete
229a816c-4a39-...	qweqweqwe@te...	qweqweqwe@te...	test	inviter	Edit Impersonate Delete
7e833070-70b9-...	sdfsdf@test.c...	sdfsdf@test.c...	test	inviter	Edit Impersonate Delete
580d5d78-a248-...	user_manager				Edit Impersonate Delete

Figure 4.1.2: Keycloak Admin Console

4.1.1.2 Tokens

The access tokens that the user receives after logging in are in JWT standard. These tokens usually have an expiration date or time. As a result of authentication not being the primary focus of our project, we decided to set the expiration time to an arbitrarily high value. Due to this, we avoid having to implement a refresh-token mechanism in our application. This decision is further discussed in 5.2.2.

4.1.2 Features

With our implementation of authentication, we have support for signing up, signing in, and other features such as resetting your password and email verification. All these features use the architecture mentioned in subsection 4.1.1, where the mobile application communicates with our REST API, which in turn communicates with the Keycloak instance.

4.2 Spring Boot API

Developing an API has not been the primary scope of the project. However, to be able to work more independently from Solwr's already large and complex systems, we were asked to develop our own API that would act as a dummy service for the simulation of a warehouse system. In the beginning, the API provided the front-end with dummy data, however later it was improved and does now function as a standalone WMS where users can create new warehouses and configure them

with locations, products, and members.

4.2.1 Security Configuration

As mentioned in 4.1, our application needed authentication. On the API level, this was done by configuring Keycloak to our Spring Boot application. This way, every request sent to the API goes through a filter chain validating the request. The API can be configured to require certain levels of authorization for different endpoints using roles. Only users with correct authority will be able to pass through the filter chain, evidently getting access to the API.

4.2.2 Entity Relations

We spent a significant amount of time defining the models needed to simulate a warehouse and constructing the relations between the models. Figure 4.2.1 shows a diagram of all the entities with relations stored in the database.

4.2.3 SMTP

SMTP is used to forward emails. The SMTP is configured to be able to send multiple types of email. It is used for password resets, verification, and invite features. In figure 4.2.2 you can see the different emails provided by the back-end. These consist of an invitation email, verification email, and reset password email.

4.2.4 Leaking Internals

In an application, it's common to define entities that relate to entries in the database. It's bad practice to return these entities directly to the client upon request as this can expose confidential data [28]. Instead, our API uses mappers that map these entities into DTOs. The DTOs are specified to only contain data that is required for the request. This results in a safer and cleaner codebase, by having lower coupling and higher cohesion.

4.2.5 Error handling

The Spring Boot API returns different HTTP responses based on the outcome of the request. When an error occurs, our API responds with meaningful HTTP codes and messages. This allows clients to interpret these responses and display correct error messages to the user accordingly.

4.2.6 Documentation

The API is documented using standard Java Docs, where methods are commented about what it does and what type of parameters it expects. In addition, as mentioned in section 3.5.5, all API endpoints are documented using swagger. A complete list of all endpoints can be found at <https://api.bachelor.seq.re/swagger-ui/index.html>. The swagger docs give information about what the endpoint expects and what it returns. Figure 4.2.3 shows a simple endpoint.

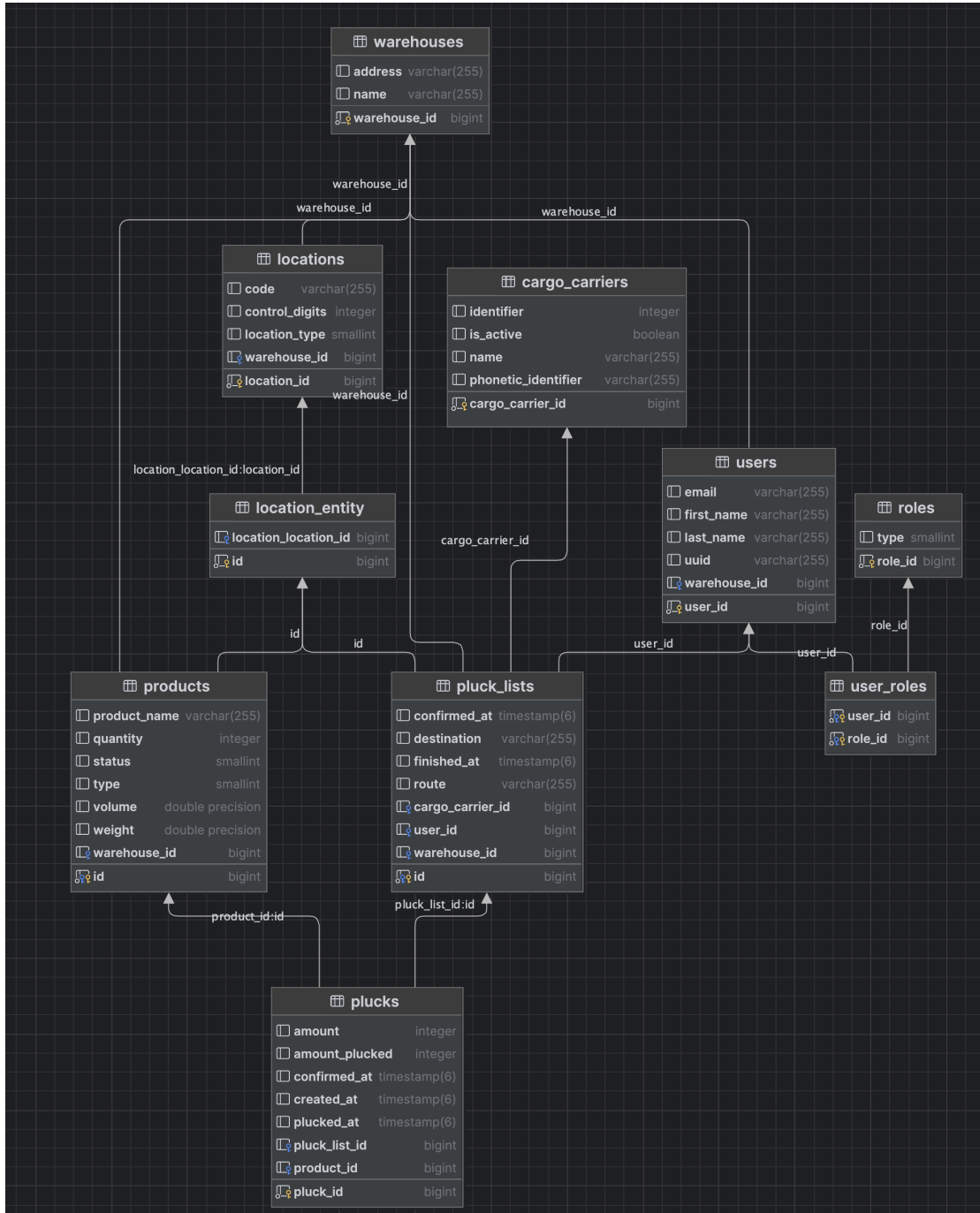


Figure 4.2.1: Database Diagram

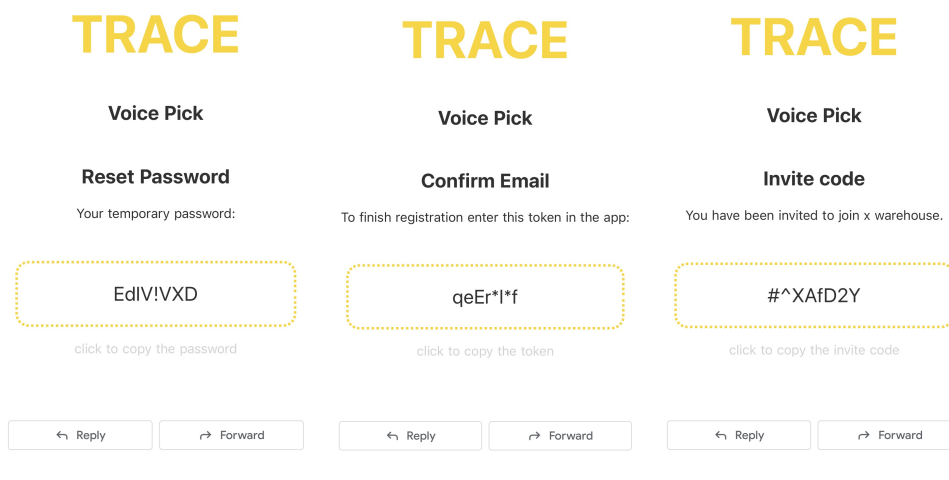


Figure 4.2.2: Emails sent with SMTP

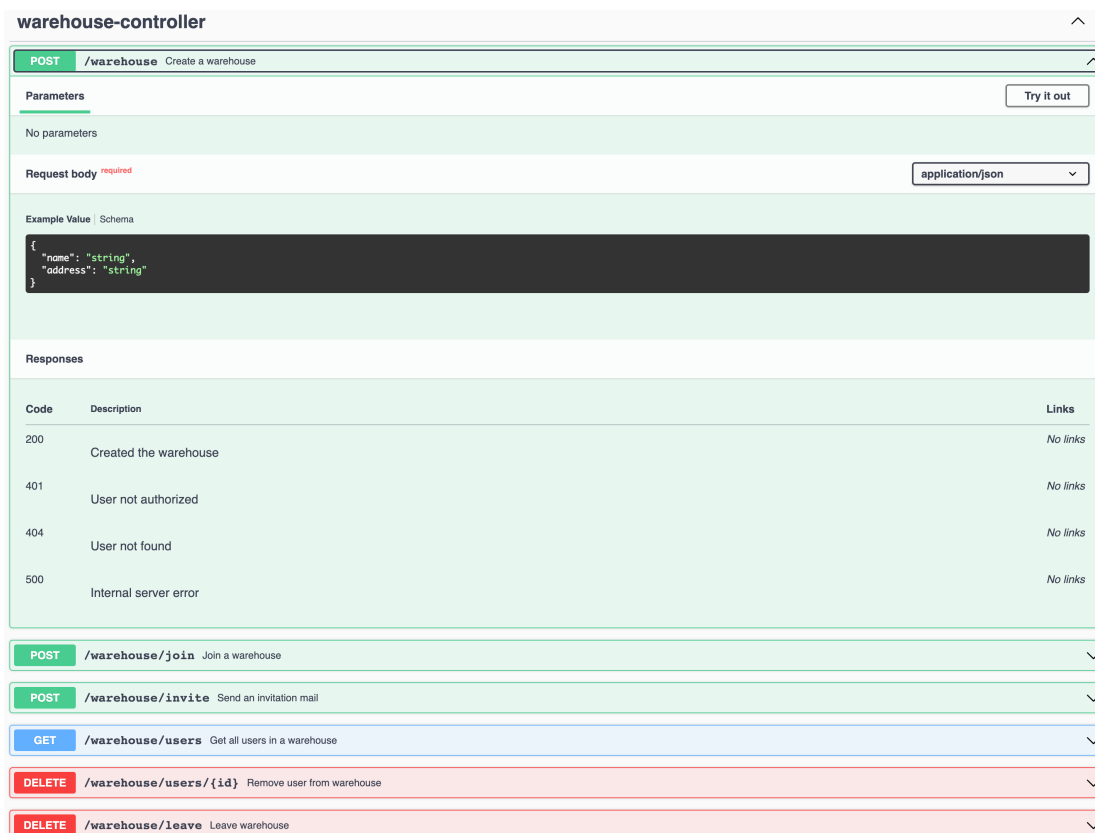


Figure 4.2.3: Swagger-UI

4.3 Deployment

Our backend is deployed within a Docker container, communicating with our Keycloak container and database.

We've used Terraform to create and configure our Azure resources, including our VLAN, firewall, and virtual machine. Thanks to our containerization of the application, integration with Solwr's existing Kubernetes clusters is straightforward. And with our use of Keycloak, integrating with their preexisting deployment becomes effortless.

We've also implemented a continuous deployment process by using GitLab CI/CD pipelines. They run tests on our code and provide us with the option to deploy the code manually. This makes sure that our code is tested before any deployment, allowing us to maintain a high level of quality and reliability in our software releases.

4.3.1 Diagram

In figure 4.3.1 you can see our system diagram. The keylocks represent HTTPS connections our Nginx proxy enables. You can also see that GitLab is connected with our GitLab runner, which continuously deploys pushed code to our development API, and tests it in our SonarQube instance. In addition, most of our services are containerized as shown by the dotted lines.

4.4 SwiftUI Application

As a result of our research and development, we have created a complete iOS application, ready to be released on App Store. The application serves pages that let any user download the app and set up a warehouse from scratch, or join an existing one, and use our voice-plucking system to prepare pallets for delivery. In this section, we provide a thorough description of our application.

4.4.1 Application layout

We have designed an application that consists of several pages as seen in figure 4.4.1. These pages include an authentication page, an email verification page, and a page for joining or creating a warehouse, figure 4.4.2. Furthermore, the application includes "pluck" pages for the pluck flow, and a warehouse configuration page for managing locations, users, and products. Additionally, we added a "log page" that displays messages from the recognized words by the user, and the uttered words by the application. Finally, there is an account page where users can adjust their settings and manage their accounts.

By implementing this diverse set of pages, we provide an application that can be downloaded and used by anyone in need of a WMS. The combined features of our front-end and back-end result in an efficient and user-friendly application that can be used to prepare orders, and manage your warehouse.

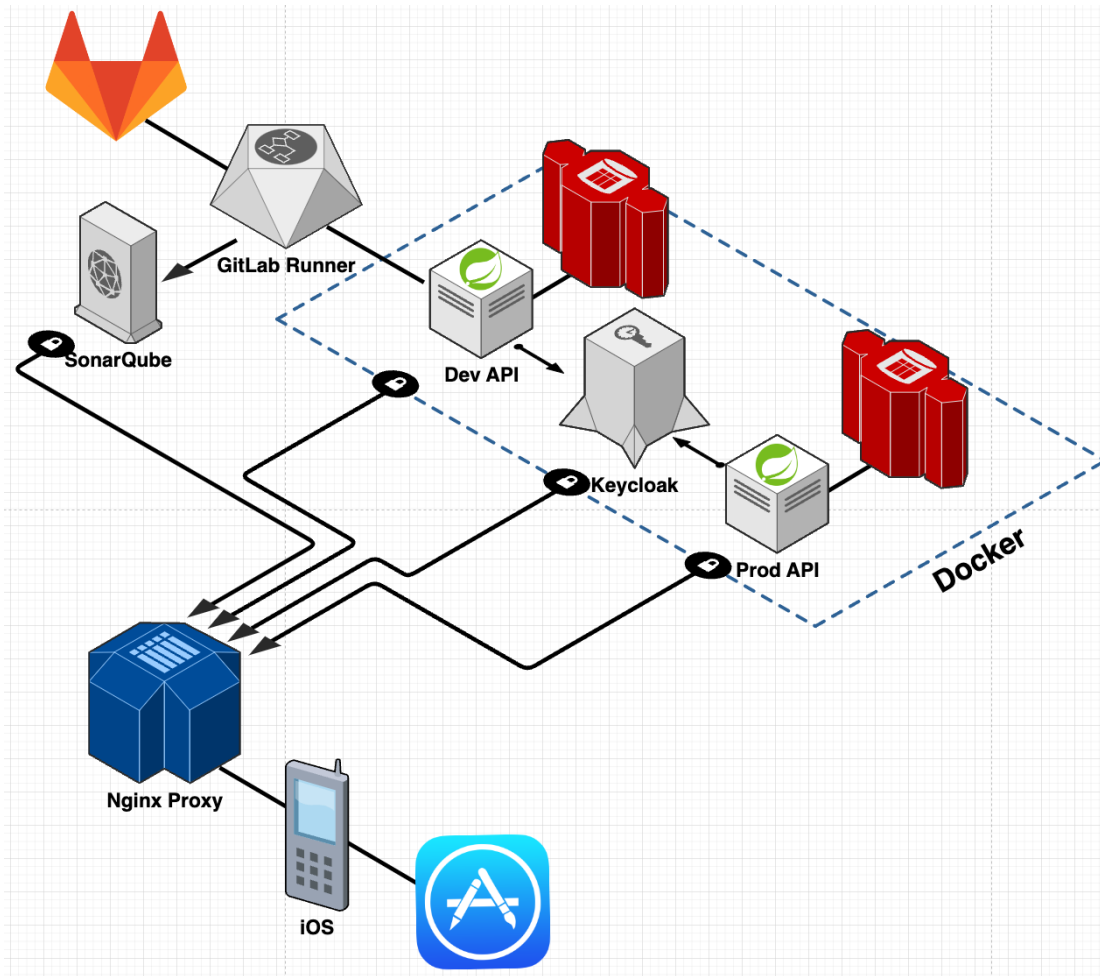


Figure 4.3.1: System Diagram

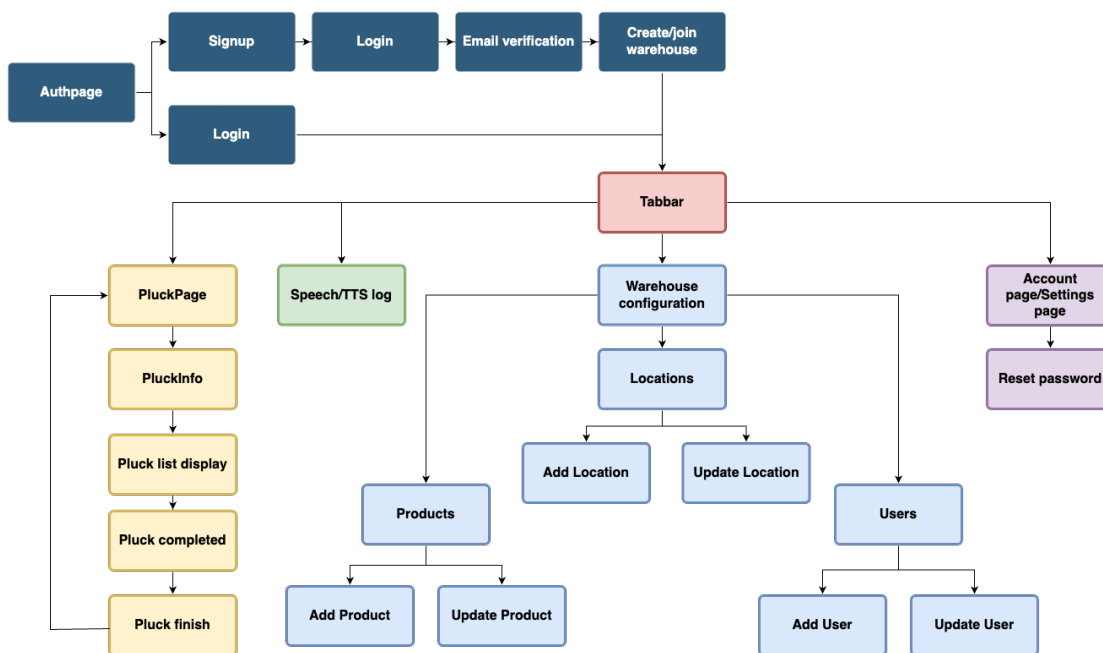


Figure 4.4.1: Swift app flow diagram

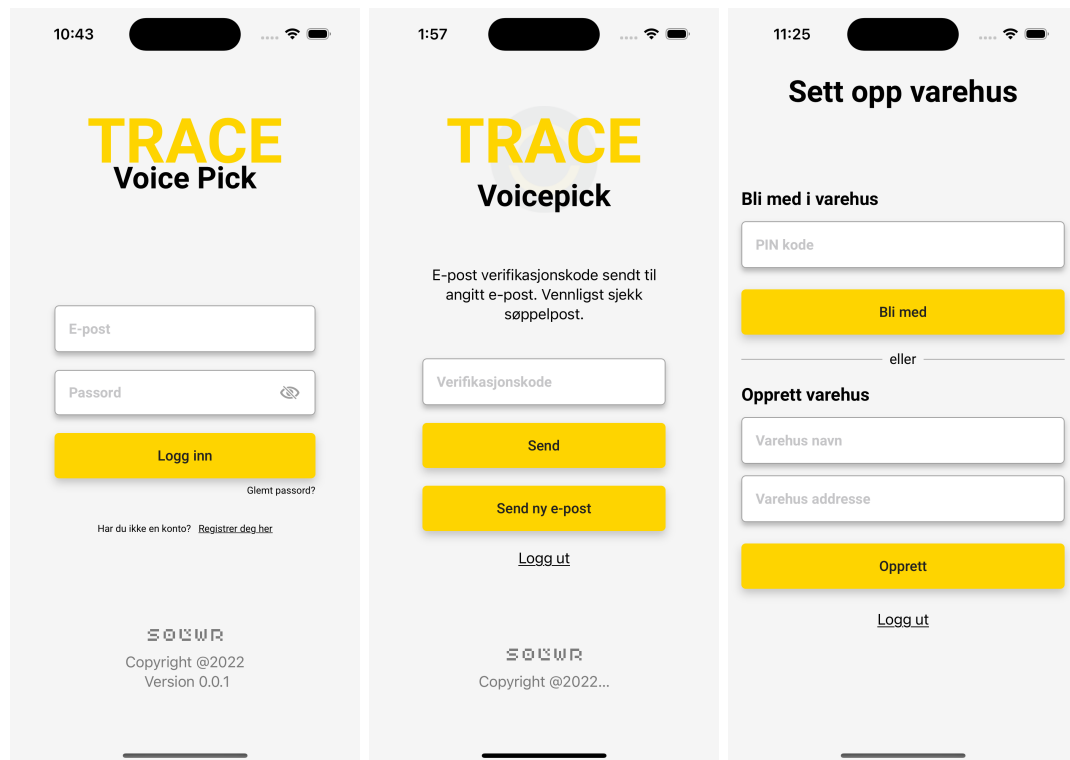


Figure 4.4.2: Authentication flow

4.4.2 Speech Recognition

Since this project takes a considerable part in developing a voice interaction app we need to talk about how we made it possible.

The Speech framework provides several different features for speech recognition, including speech-to-text conversion, language detection, and real-time transcription. These features made it possible for us to create a powerful application that allows users to interact with their devices using natural language commands.

4.4.2.1 Implementation

Implementing a speech recognition system involves several steps; requesting user authorization, checking for an authorized microphone, managing audio input recordings, handling speech recognition, verifying and enabling on-device recognition, and adding a custom vocabulary used for keyword detection. This is done through an API provided by Swift called `SFSpeechRecognizer` [29].

When a user signs in for the first time, the application prompts the user to allow the application to use the device's built-in microphone, as seen in figure 4.4.3. This is a privacy precaution, as iOS applications are not allowed to use microphones unless given access by the user. The application can be used without access to speech recognition, however, the user will have to use the touch interface.

Employees at a warehouse should only be able to interact with the application using a set of keywords. Our application implements this by defining a vocabulary used by the `SFSpeechRecognizer`. When the recognizer picks up a recording, it compiles a transcription. This transcription is then filtered to only look for the keywords defined in the vocabulary. Examples of the keywords are; "help",

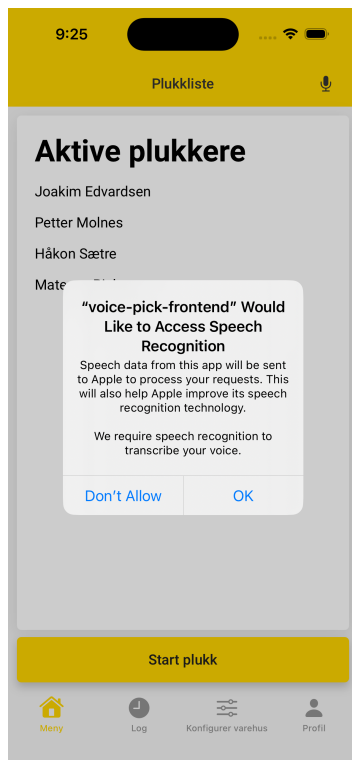


Figure 4.4.3: Prompt when logging into the application

"repeat", and "next". While certain keywords remain applicable throughout the entire plucking process, others are limited to specific stages and are only available when the worker is working on those particular stages.

In addition to keywords, the speech recognizer also picks up any number. This is used for the user to both confirm the correct control digits of a location as well as for the user to select the amount they pluck. When prompting with a number between zero and nine, the recognizer transcribes this as the string value. For example, when saying 1, it's recognized as *one*. However, when a number larger than nine is recognized, it transcribes it as the actual number. E.g. saying 202, results as 202. In our case, it was required to always transcribe the number value. To overcome this, the application checks whether the transcript is a word between zero and nine, if so, it converts the word to the number value.

At first, when saying a number, the recognizer started transcribing the result before the user was finished saying the complete number. For example, when a user said 245, the transcript would compile 2, 24, and 245. To make sure the recognizer would wait for the user to finish we used a debounce technique, where the recognizer only transcribes the result 500 milliseconds after the recording stops. This implementation can be seen in figure 4.4.4. Using the debouncing technique also helped solve the accuracy problem mentioned in 3.1.2.1.

4.4.2.2 Voice Requirements

As mentioned in chapter 3.1.2.3, a user should be able to use different inputs. Our application allows this by checking for different inputs when the application is launched, 4.4.5. For example, if AirPods are connected, the default microphone will be set to the AirPods. If no other external microphone is detected, the


```

159  /**
160   * Processes the result gotten from the recognizer
161   *
162   * - Parameters:
163   * - result: a string to be processed
164   */
165  func processRecognitionResult(_ result: String) {
166      recognitionTimer?.invalidate()
167      // debounces recognition X seconds
168      recognitionTimer = Timer.scheduledTimer(withTimeInterval: RECOGNITION_DEBOUNCE_TIMER, repeats: false) { [weak self] _ in
169          guard let self = self else { return }
170          // Refresh the value stored in the published value
171          let result = self.filterKeywordsAndNumbers(from: result)
172          self.onRecognizedTextChange?(result)
173          // Reset the recognition task so it will start from empty state
174          self.stopRecording()
175          self.startRecording()
176      }
177  }

```

Figure 4.4.4: Speech filtering implementation

```

61  /**
62   * Checks if any bluetooth devices are connected
63   */
64  private func isBluetoothConnected() -> Bool {
65      let routes = audioSession.currentRoute
66      for output in routes.outputs {
67          if output.portType == .bluetoothA2DP || output.portType == .bluetoothLE || output.portType == .bluetoothHFP {
68              return true
69          }
70      }
71      return false
72  }

```

Figure 4.4.5: Code checking for connected devices

application uses the default microphone of the device.

4.4.2.3 On-device Recognition

In our project, we implemented on-device recognition, 2.7.3.1. As a result of this, our application demonstrated improved performance and efficiency, as it no longer relied on internet connectivity or remote server processing for speech recognition. This was a sought-after feature from our employer, due to poor connection in the warehouse. While it is not as accurate as server-based recognition, the precision is good enough for our purposes and supports a wide variety of languages.

Additionally, the server-based recognition had some restrictions related to a daily request limit and a maximum listening duration of 30 seconds. On-device recognition solved these issues.

4.4.2.4 Toggling Speech Recognition

As workers at a warehouse sometimes need to communicate with one another, the ability to toggle speech recognition was highly requested. As seen in figure 4.4.6, there is a microphone icon in the header that toggles speech recognition. Furthermore, a user can say "mute" to mute the speech recognition. While in "mute" mode, the recognition only responds to one keyword, "listen", which then unmutes the microphone.

4.4.2.5 Languages

Our application uses English for speech recognition. `SFSpeechRecognizer` lets you choose other languages too, however, English is slightly more accurate than the others. With the `SFSpeechRecognizer` it is possible to change the language

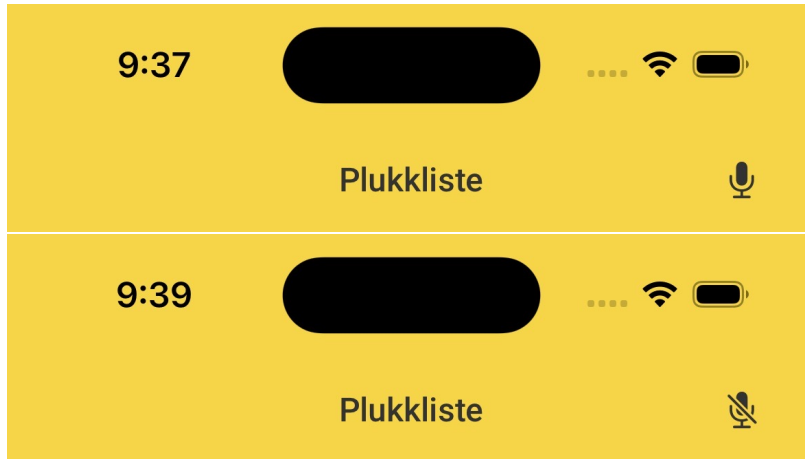


Figure 4.4.6: Mute options in the application header

```
SFSpeechRecognizer(locale: Locale(identifier: "en-US"))
```

Figure 4.4.7: Setting the Locale of the SpeechRecognizer

detection in runtime. A list of available languages can be exposed using the `supportedLocals()` method. Together with a variable storing the currently selected language, the `SFSpeechRecognizer` can be configured to use this variable instead of a constant string as seen in figure 4.4.7.

4.4.3 Text to Speech

Apple's TTS solution has enabled the conversion of hard-coded text into natural-sounding spoken words in our project, thereby improving the user experience during the plucking process.

Leveraging advanced algorithms and neural networks, Apple's TTS technology analyzes input text and generates speech that closely mimics human voices. In our project, we have utilized this technology to better the user experience, providing real-time updates and guidance through a natural-sounding voice [30].

4.4.3.1 Implementation

The TTS is configured the same way as speech recognition. It is initialized by using the `AVAudioSession` API provided by Swift [31]. On application launch, the application checks for any connected Bluetooth speakers. If there are no available connected speakers, it falls back to the default built-in speaker in the device.

By implementing a `speak()` function that takes in a string. The string is then played through the speakers, 4.4.8. Any subsequent call to the `speak()` function, before the first utterance is finished, will be placed in a queue and played when the `AVAudioSession` is available.

In addition to the `speak()` function, we looked into the possibility to interrupt the utterance. This feature was requested as warehouse workers do not always want to listen to the whole utterance. The TTS API exposes a `stopSpeaking()`

```

32  /**
33   Plays a string to the device audio output
34
35   - Parameters:
36   - utterance: the string to be played to the speaker
37   - fromVoice: a boolean describing if commands were given via voice or touch.
38   If they were given through voice, the utterance is played on the speakers.
39   If the commands are given through touch, the utterance is not played on the speakers
40   */
41  func speak(_ utterance: String, _ fromVoice: Bool) {
42      if (fromVoice) {
43          voiceService.muted = true
44          let speechUtterance = AVSpeechUtterance(string: utterance)
45          speechUtterance.rate = self.rate
46
47
48          if (selectedVoice == nil) {
49              speechUtterance.voice = AVSpeechSynthesisVoice(language: "en-US")
50          } else {
51              speechUtterance.voice = AVSpeechSynthesisVoice(identifier: selectedVoice!.rawValue)
52          }
53
54          speechUtterance.volume = self.volume
55          speechUtterance.pitchMultiplier = 1.0
56
57          DispatchQueue.main.asyncAfter(deadline: .now() + 0.5) {
58              self.voiceLog.addMessage(LogMessage(message: utterance, type: LogMessageType.OUTPUT))
59          }
60          synthesizer.speak(speechUtterance)
61      }
62  }

```

Figure 4.4.8: Speak function in TTSService

method which can be called to stop the utterance immediately. This also removes any utterances stored in the queue.

4.4.3.2 Text to Speech Settings

With the implementation of TTS, a user has the option to change the utterance volume and speed, making a more personalized experience. For workers at a warehouse, this means they can configure the text-to-speech to fit their needs.

During the development phase, it was discovered that the application’s voice output had a robotic quality which could have potentially been problematic and irritating for employees working extended shifts. We identified methods to localize the voices installed on the device and modify the voices accordingly. On the application’s account page, there is a section where you can change the voice to either Nathan or Evan, 4.4.13. If the voice select is not installed on the device, the user is prompted with a message explaining how to install the voice on the device. When the voice is installed, the application can make use of the voice for utterance.

4.4.3.3 Voice looping

When played on a speaker, the speech recognition would sometimes recognize the utterance played by the device, causing a feedback loop. The problem was not too concerning as the employers would mainly use the application with a headset where the microphone would not pick up sound from the headset speakers. However, it was not desired as it removed the possibility of using the application in speaker mode. At first, the group thought of disabling speech recognition as the device

```
14 enum Steps {
15     case START
16     case SELECT_CARGO
17     case SELECT_CONTROL_DIGITS
18     case CONFIRM_PLUCK
19     case DELIVERY
20 }

79 switch currentStep {
80 case .START:
81     handleStartActions(keyword, fromVoice)
82     break
83 case .SELECT_CARGO:
84     handleCargoAction(keyword, fromVoice)
85     break
86 case .SELECT_CONTROL_DIGITS:
87     handleControlDigits(keyword, fromVoice)
88     break
89 case .CONFIRM_PLUCK:
90     handleConfirmPluck(keyword, fromVoice)
91     break
92 case .DELIVERY:
93     handleDeliveryAction(keyword, fromVoice)
94     break
95 }
```

Figure 4.4.9: Pluck steps - Code snippet

was playing a sound. However, this would remove the feature of canceling the utter, which is a very important feature for employers. Eventually, this was solved by switching out the vocabulary of the speech recognizer as the device was playing sounds. When playing sound, it only listens for the "cancel" command. When the device is no longer playing a sound, the vocabulary is changed back making it listen to all commands.

4.4.4 Touch and Voice Concurrency

As mentioned in 1.2.3, concurrency between voice and touch when performing a pluck round was one of the most important aspects of our project. Our application solves this using a `PluckService` class that keeps track of the current step of the pluck round by using an enum as seen in figure 4.4.9. The service exposes a method `doAction(keyword: String)` which is used whenever a keyword from the voice recognizer is detected, or when the user performs an action through the touch interface. The method will switch over the possible steps, and perform the correct method based on the current step, as shown in figure 4.4.9. The respective method will then switch over the possible keywords available for the current step and perform the correct action based on the user input.

This solution lets us call the same method when the user does an action, regardless of the user interacting with the voice interface or touch interface. An alternative approach will be discussed in section 5.5.

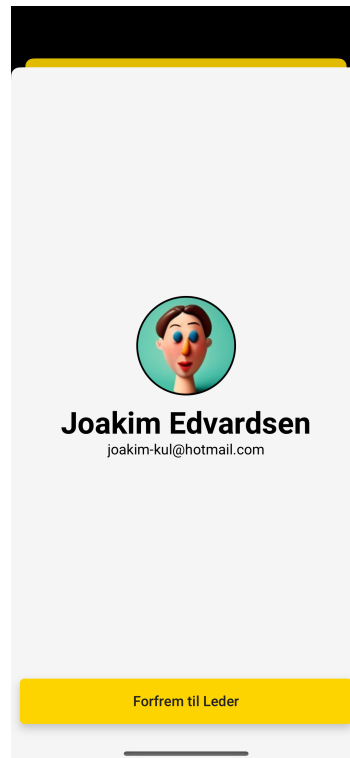


Figure 4.4.10: Detailed employee view. A leader can change employee role

4.4.5 Warehouse Configuration

To increase the application's functionality and utility, we have implemented warehouse configuration pages, allowing users to set up their own warehouse, 4.4.11. As a result, the app can cater to multiple warehouses, rather than being limited to a single warehouse, broadening its scope and adaptability. With this, the application is available for any business.

When registering a new account, the user has the option to either join or create a warehouse. To join a warehouse, a leader must enter the email of the user to invite. The user receives an email and enters the code into the application and gains access to the warehouse.

For the configuration part, when you create a new warehouse, you are automatically assigned a leadership role in that warehouse. Leaders of a warehouse have the functionality to add, remove, and edit the locations, products, and users in their warehouse. As a leader, you can also change the roles of other members in your warehouse, 4.4.10.

The user, location, and product pages have the same styling and the same CRUD operations. Entering the pages, the user sees a list of the locations, products, or users in the warehouse. When updating, adding, or removing entities, there are loading indicators and banners showing success or error status. By pressing on a list item, a detailed view is opened of the specific entity, where the user can edit its properties. We have implemented great error handling, responses and confirm dialogs. This is especially important for actions such as removing a member from your warehouse, to prevent accidental removals.

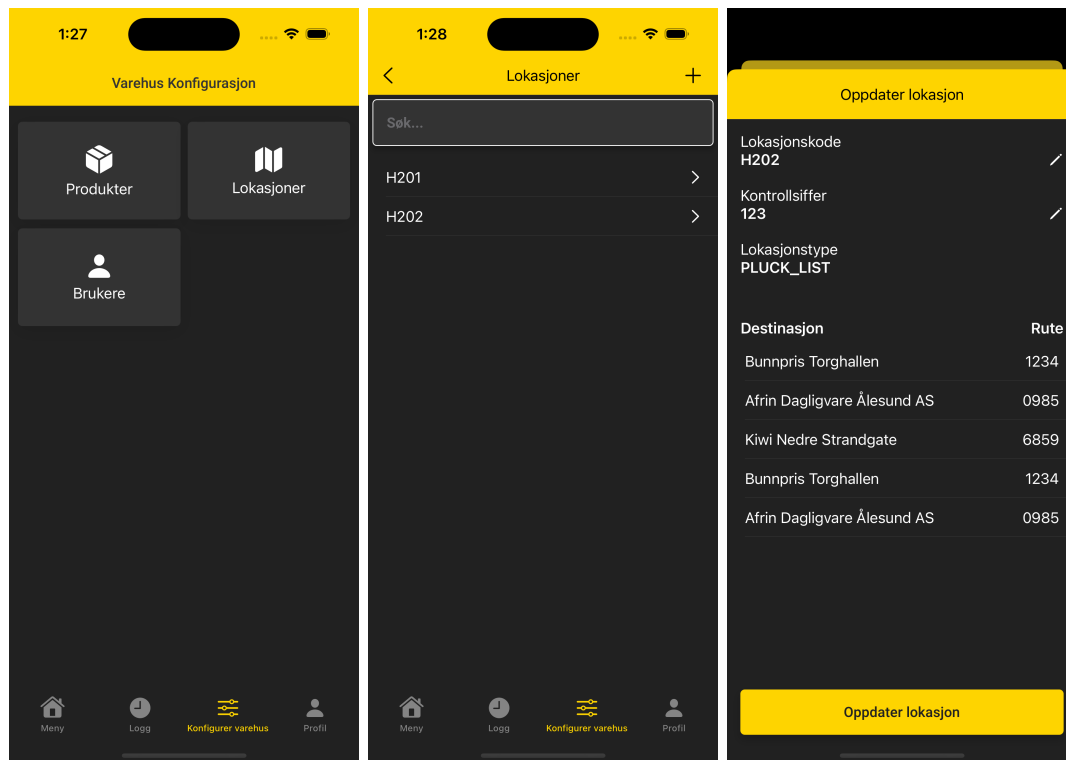


Figure 4.4.11: Warehouse config - Location Page - Update Location /w product on location

4.4.5.1 iOS Scanner

One of the discussions with our employer was to implement a barcode scanner that could scan the bar codes of products to automatically add or register products. Figure 4.4.12 shows the implemented barcode scanner using a scanner package available for iOS devices running iOS 13.0 or later [32]. The scanner was implemented with the thought of replacing the current scanner which is a separate system. Our implementation is not necessarily more efficient, however, it helps keep all tools needed for the employees at a warehouse in one place, further enhancing the application experience.

4.4.6 Additional pages

4.4.6.1 Log page

As the user communicates with the application through voice, all commands, both spoken to and uttered by the application are saved and can be viewed in a log. The log is displayed like a chat, with the spoken commands on one side and the uttered commands on the other, figure 4.4.14. This makes it clear to the user what has transpired.

4.4.6.2 Account page

As mentioned earlier, the user can change the settings on the account page, 4.4.13. Additionally, there is the option for deleting the user, leaving the warehouse, resetting the password, and logging out. The account page also provides the



Figure 4.4.12: Barcode scanner in the application

option for selecting profile pictures for the user account where you can select from a list of predefined pictures. These pictures were generated using MidJourney. We have also implemented a fun ranking system for the workers where they get ranked based on how many pluck lists they have completed. For example, if you complete 10 000 pluck lists you achieve the title "Ekspert plukker".

4.4.7 Keychain

Keychain is a tool available in Swift and allows storing data securely in the application [33]. This is incorporated into the application to securely store sensitive user data like access tokens. Due to this, data is encrypted and persisted when the application is closed.

4.4.8 Communicating with API

The SwiftUI application directly communicates with the API using the HTTPS protocol. In the application, a `RequestService()` can be initialized. This service exposes `GET`, `POST`, `PATCH`, and `DELETE` methods, 4.4.16. In addition, the service exposes a published field, `isLoading`, for checking if the request has been processed or not. This field is then used to display loading indicators in the application, giving the user feedback that something is being processed. Upon an error from the API, the application checks the error status and prompts the user with error messages accordingly. Figure 4.4.17 shows how the exposed methods can be used.

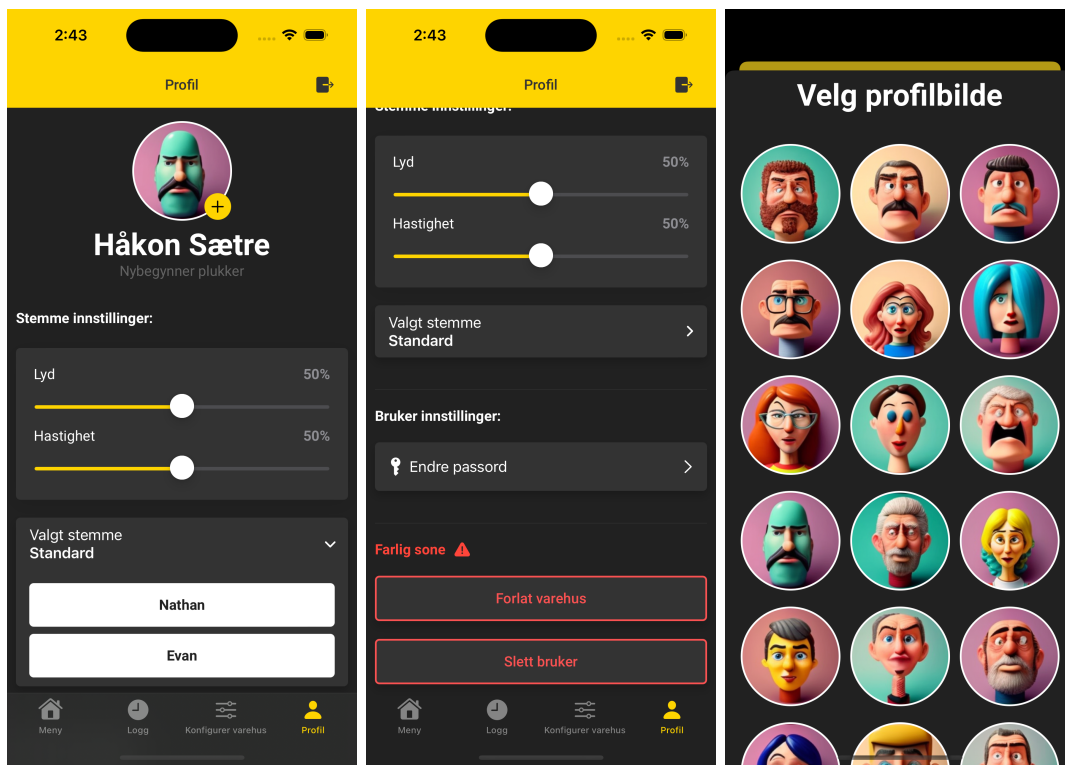


Figure 4.4.13: Accountpage

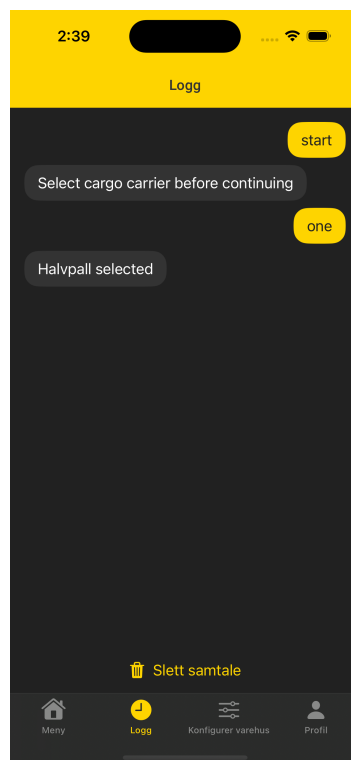


Figure 4.4.14: Log page

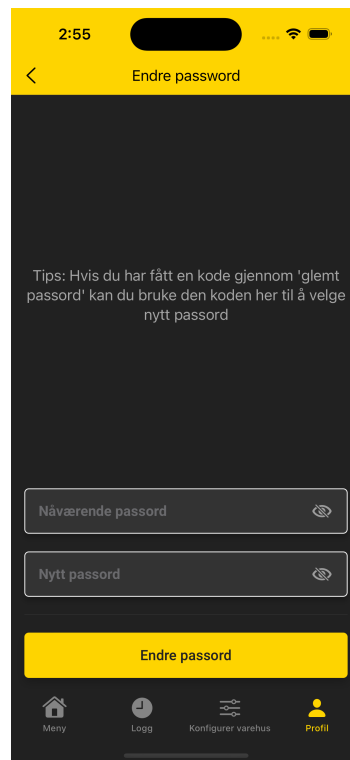


Figure 4.4.15: Forgot password page

```

153  /**
154   Returns an object of type U. The object has to conform to `Codable` so it can be mapped to and from JSON
155  */
156  - Parameters:
157  - path: The relative path to the endpoint
158  - token: A jwt token that should be added to the headers
159  - body: The additional information as type T to attach the requests body
160  - responseType: The type of the response you expect to get
161  - completion: A function that is called whenever the request is completed. Should handle both success and error conditions
162  */
163  func post<T: Codable, U: Codable>(path: String, token: String? = nil, body: T, responseType: U.Type, completion: @escaping (Result<U, Error>) -> Void) {
164      request(
165          "POST",
166          path,
167          token,
168          body,
169          responseType,
170          completion
171      )
172  }

```

Figure 4.4.16: Post method exposed by the RequestService

```

71  /**
72   Fetch all employees in the warehouse
73  */
74  func getEmployees() {
75      requestService.get(path: "/warehouse/users", token: authenticationService.accessToken, responseType: [User].self, completion: { result in
76          switch result {
77              case .success(let employees):
78                  self.employees = employees
79              case .failure(let error as RequestError):
80                  handleError(error.errorCode)
81              default:
82                  showBanner = true
83          }
84      })
85  }

```

Figure 4.4.17: Example usage of RequestService

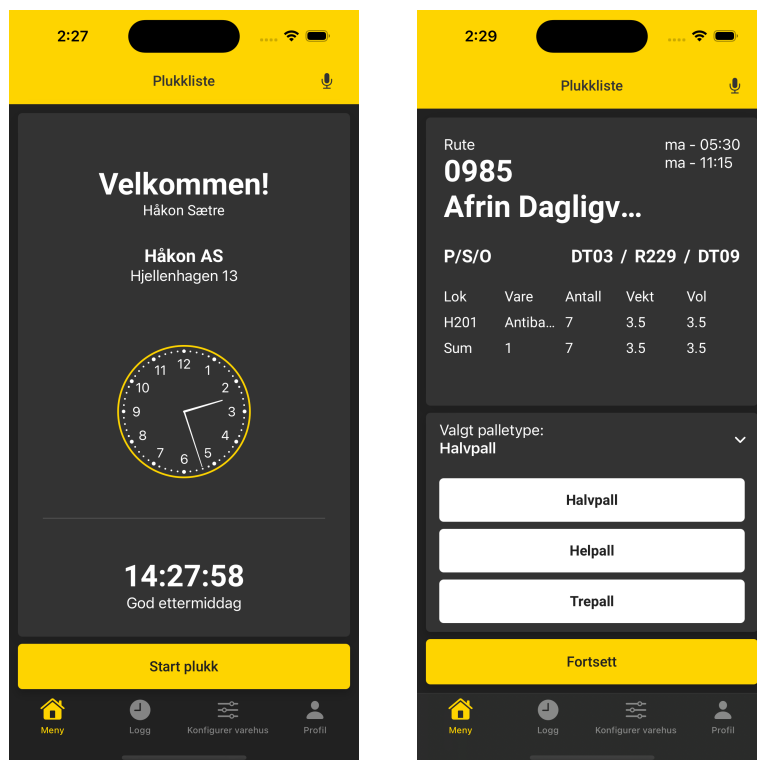


Figure 4.5.1: Pluckflow Info/lobby

4.5 Real Life Environment

4.5.1 Pluck-flow Example

Upon logging into the app, users are directed to a page where they can initiate the plucking workflow. This workflow can be executed using voice, touch, or a combination of both. By either pressing the "Start" button or using voice commands to say "Start," a pluck list is retrieved from the database, as seen in figure 4.5.1.

After starting a pluck, you are directed to a page displaying information about the pluck. Here, you must choose which cargo type to use, either using your voice or the drop-down menu, 4.5.1. Once you choose a cargo type, press or say "next" to move on to the stage where the plucking begins. This is when you enter control digits and the number of items you need to pluck, 4.5.2. When all of the plucks are done, you have to drop off the cargo at the correct location 4.5.3. All instructions are uttered by the speaker, which lets the user complete the pluck round without having to interact with the touch interface.

At the beginning of each pluck round, a user may choose to rearrange the plucks in a better-suited order. Users can manually rearrange the products in their desired plucking order by utilizing the implemented drag-and-drop feature. A user can complete all other actions in a pluck round using just their voice.

We recommend watching the YouTube video linked in Appendix C, where a user completes a pluck round using our application. It depicts our application used in a real-life environment and can help better understand the use case of our solution.

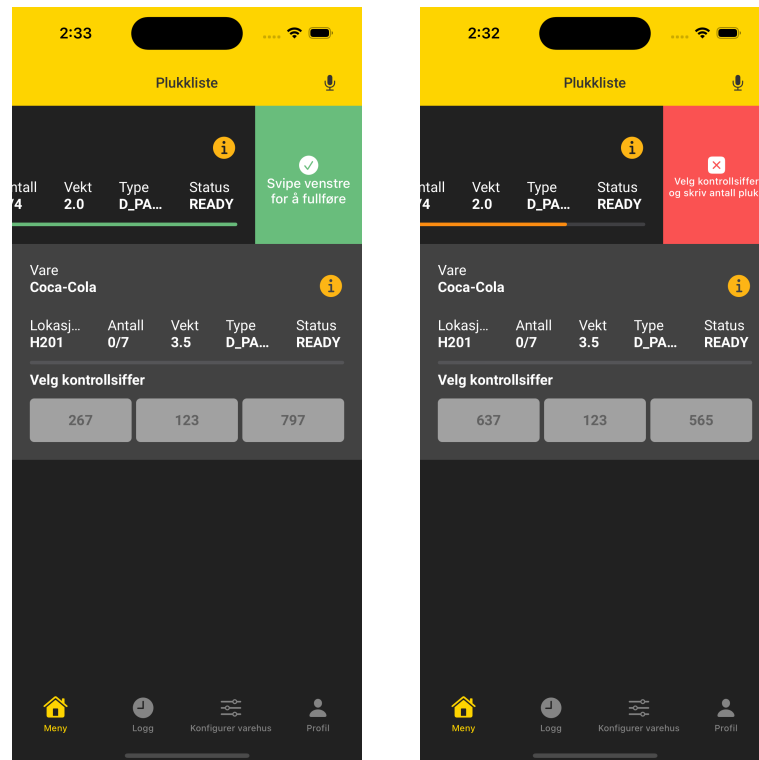


Figure 4.5.2: Plukkflow product list success/error

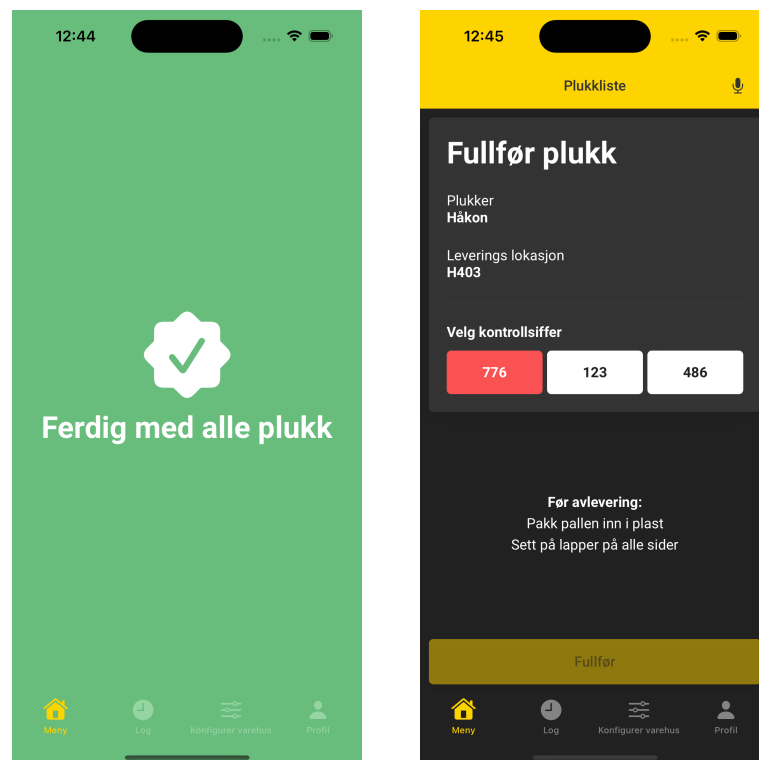


Figure 4.5.3: Plukkflow pluck finish/finish page

4.5.2 Efficiency

The fact that users can choose between voice and touch as desired helps increase efficiency. Normally, workers prefer to use voice as this makes both hands available for driving the truck as well as plucking products from shelves. However, some actions are more complex and the addition of a user interface that can be interacted with touch will help streamline these actions.

4.5.3 Removed Redundancy

Simplifying the app and streamlining the required data were among our top priorities during development. We aimed to create an application that is simple and efficient for users. While the app may appear straightforward on the surface, a significant amount of time and effort was invested in achieving this level of simplicity. A storage facility contains a lot of data, including various locations, products, and users, which the application efficiently handles while maintaining its user-friendly interface.

4.5.4 Bluetooth devices

We have thought closely about how the application would work in a facility with lots of noise and other workers. Bluetooth devices have therefore been tested to see if noise and other voices can interrupt another user's system. As a result of this, we figured out that an option for this is the use of Bluetooth devices or bidirectional microphones.

4.6 Android Implementation

As a result of our research and dialogue with Solwr mentioned in subsection 5.3.1, we used Swift and SwiftUI, thus creating an iOS only application. However, Solwr is also interested in knowing if the application can be created for Android applications as well. In this section, we will walk you through the result of our research on implementing an Android solution.

4.6.1 Implementing Voice-Pluck

Our implementation of voice and touch concurrency mentioned in 4.4.4 is implemented in a way that different choices of voice and/or utter libraries should not affect the concurrency. Therefore, we have mostly conducted research on the best libraries to use for speech recognition and text-to-speech in Android applications.

4.6.1.1 Speech Recognition

As mentioned earlier, we used `SFSpeechRecognizer` for iOS [29]. For Android devices, we found a class for speech recognition similar to `SFSpeechRecognizer`, [34]. It contains a lot of the same features that `SFSpeechRecognizer` does, such as on-device recognition.

Additionally, there are obvious choices such as Google’s Speech to Text service. The aforementioned option is undeniably better than other open-source alternatives. However, Solwr wants to take a step away from paid services, so we have looked at other alternatives such as PocketSphinx [35] and Kaldi [36].

Unlike Android class [34] which is maintained and created by Android, these alternatives are maintained by independent institutions and authors. As a consequence, you cannot be sure whether these libraries will keep being maintained. Therefore, we would highly recommend using Android’s built-in classes, as they provide a reliable set of features.

4.6.1.2 Text To Speech

Concerning text-to-speech, we found another class developed and maintained by Android [37]. The class can be utilized the same way we utilized Apple’s Speech Synthesis class. It is possible to find third-party options as well. However, we still recommend choosing built-in classes as they are more reliable and likely to be maintained.

4.7 Smart Watch Companion App

A part of our project was to research the possibilities of creating an app for different devices, such as smartwatches. Early in the process, we created sketches for smartwatches, seen in figure 4.7.1. The sketches depict how you can show a round of plucks, with the functionality to complete it through a touch interface.

Later in the process, we concluded that a potential smartwatch application should be a companion app. Having it as a standalone application would propose several issues related to performance and support.

4.7.1 Apple Smart Watch App

We used Apple’s tutorial [38] to get a quick introduction to Apple Smart Watch applications.

Apple’s `SFSpeechRecognizer` class that we use for recognizing words is only supported on iOS. Therefore, we concluded that with our approach, it is not possible to create a standalone smartwatch application. Despite this, it is still feasible to create a companion WatchOS app that increases efficiency. The application does not need to support speech recognition and utterance, but could rather serve as an accessible touch interface while the phone is mounted on the forklift. Implementing two-way communication between the iOS app and its paired watchOS app is easily done through Apple’s Watch Connectivity framework [39].

4.7.2 Android Smart Watch App

Our group focused on iOS development. However, Solwr develops for Android as well, so we have researched the possibility to create an Android Smart Watch application.

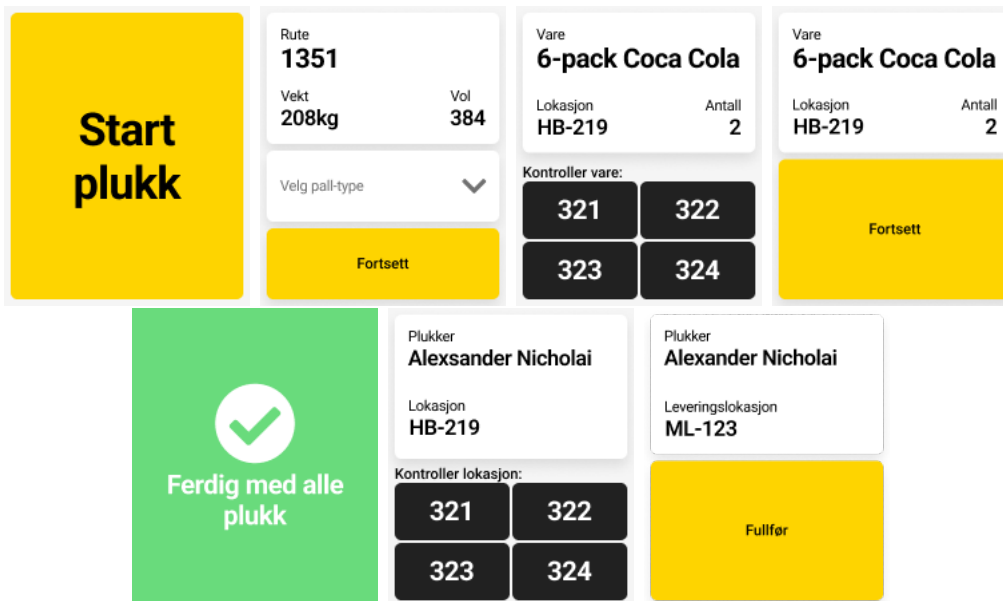


Figure 4.7.1: Smart Watch Sketches

Unlike Apple’s speech recognizer class, Android’s speech recognizer class [34] works with smartwatches as well. Naturally, this is contingent on the smartwatch having the proper hardware, so the performance can vary from model to model.

We concluded that it is possible to create a standalone smartwatch application for Android watches. However, several problems related to performance could surface. Various models will not have sufficient hardware to run the application efficiently. Therefore, we recommend the approach in the aforementioned subsection, 4.7.1.

DISCUSSION

In this chapter, we will discuss the choices in technology and the troubles we overcame.

5.1 Project Structure And Architecture

While developing, we ran into some difficulties related to project structure and architecture.

5.1.1 Project Plan and Roadmap

As mentioned in chapter 3 we started with an extensive planning phase, where we created a roadmap for our project. This roadmap can be seen in Appendix A. After several sprint meetings, we were advised to focus on creating a finished mobile application, instead of creating the smart watch application as well. Because of this, the roadmap has changed a bit over the course of the project. This was expected and with the workflow discussed in section 2.2, it was easy to adapt to other priorities.

5.1.2 Reflection on Project Structure Choice

Our choice of project structure is well-suited for large projects. However, some team members were not familiar with the structure and found it more confusing rather than organized. On the other hand, the application might be developed further, justifying the choice and helping feature extraction as the application grows.

5.1.3 Matching DTOs Between Applications

As mentioned in subsection 4.2.4, we used DTOs to transfer data between our applications. However, combining this with our top down approach caused a lot of time-consuming debugging. Whenever there was a small difference between a DTO in the back-end and the same DTO in the front-end, the front-end would fail to decode the response from the API to the proper object. This could, for example,

be a field called 'firstName' in our back-end and 'firstname' in our front-end, where the only difference was capitalization.

Had we utilized a bottom-up approach, developing our entire API from the beginning, we would not encounter this problem as often. This could potentially have saved us some time that could have been better spent.

5.1.4 Shared Services

In our front-end application, we created several services, such as an authentication service. The service was responsible for storing access tokens and user information. Initially, we instantiated this service several places in our application. Obviously, this caused issues as our application would contain multiple objects of the same service, with different values.

To circumvent this issue, we made our service conform to `ObservableObject`, which were instantiated once at the top level they were required and passed around as `EnvironmentalObjects`. `ObservableObject` works by "emitting the changed value before any of its `@Published` properties change" [40]. This was a better solution than creating them as singleton classes, as the `@Published` fields in the `ObservableObject` would automatically update the views listening to changes.

5.2 Authentication

As mentioned in section 4.1 authentication was not a primary focus of our project, but it was still required. Due to it not being a primary focus, we spent time researching the easiest and best way to implement it.

5.2.0.1 Keycloak Problems

As it turned out, it was not straightforward to use Keycloak's graphical user interface. To use it with iOS you have to use an adapter, such as `AppAuth` [41]. The existing adapters were initially built to support the old standard for creating UIs in Swift; `UIKit` [42]. However, we are using `SwiftUI` to build UIs. Finding an adapter that was easily integrated with `SwiftUI` was difficult.

The Keycloak Admin API dependency [43], exposes methods that simplify actions like adding roles to users, etc. Unfortunately, this dependency did not yet support Spring Boot 3. This resulted in us having to send HTTPS requests to the Keycloak API from our Spring Boot instance.

Due to the reasons mentioned above, we had to use the Keycloak REST API, thus having to do a lot of manual development for the authentication features. As mentioned earlier, this was undesirable as authentication was not the primary focus of our project. It took away valuable time we could have spent perfecting our application's voice-plucking part.

Additionally, deploying Keycloak was not as straightforward as we hoped. There was little information on which docker containers were official and maintained. Additionally, all of them had their own unique way of setting up HTTPS which made it difficult to work with.

5.2.1 Firebase

Keycloak turned out to be more work than we anticipated. Therefore, we decided to test FirebaseAuth since we had already used it in the course, IDATA2503 [1]. FirebaseAuth has an Admin SDK dependency that exposes methods to easily interact with your database through privileged environments [44].

Unfortunately, we stumbled into problems where Spring Boot wouldn't decode JWT tokens returned from Firebase properly. It was unclear if this was because of our existing code. As a consequence, we continued with Keycloak as our authentication provider.

5.2.2 Tokens

To authenticate users, we attach access tokens in the JWT standard for every request after signing in. Normally, these tokens are short-lived, and the user has a refresh token to refresh the access token.

We had already spent a lot of time implementing authentication features. Therefore, we decided to put the token expiration time to an arbitrarily high value. This lets us avoid having to implement a refresh mechanism for our tokens.

This is not the most secure way to use tokens. In a production environment with real customer data, it would be more secure to have a short-lived access token and use refresh tokens to refresh it continually. However, we are not using any real data. Additionally, if Solwr decides to scale this project, they will use their own API and Keycloak instances. This benefited us, as we could shift our focus to the voice-recognition, and it eased the developer experiences for us, by not having to think about expiration time on tokens as much.

5.3 Choice of Front-end Framework

Each framework considered was thoroughly researched with article readings as well as tested by creating some simple applications. A detailed description of the research can be found in the appendix D.

Before development, we considered if we could create the application as a responsive web app. However, we came to the conclusion that the native support for voice recognition and utterance in mobile applications was too important. In addition, we considered creating an Admin-Panel web application, but due to time restrictions, we focused on the primary objective; voice-plucking with the mobile application.

5.3.1 Decision

The application to be developed heavily relies on native support from the devices. Because of this, both Flutter and React were omitted by the group despite, three out of the four group members already having experience with Flutter.

Solwr already has applications in production using the old Android standard. This is a bit outdated and we instead looked into JetPack Compose, the new way of writing Android applications. JetPack Compose was only in beta at the beginning of the project. Because of this, we were afraid that large updates could

cause implications later down the road. After close communication with Solwr, our team decided to go with SwiftUI as the front-end technology, despite the group wanting to use Solwr's original technology stack

5.3.2 Reflecting on iOS Development

During the development process, we experienced several pros and cons of iOS development. The clean syntax and easy learning curve of Swift and SwiftUI allowed us to quickly build visually appealing views.

The most significant challenge we faced with iOS development was the project file. When new files were added to the project, the project file would update accordingly. Whenever two team members did this simultaneously, it would result in a merge conflict that could not be easily solved. We spent a lot of time resolving conflicts because of this file. After doing research, it seemed like there were no convenient ways of solving these conflicts.

Additionally, SwiftUI is relatively new and is therefore only supported on devices running iOS 13 or newer. This could exclude potential users. However, our applications require up-to-date hardware, making older devices inapt.

Related to SwiftUI being relatively new, it was quite hard to find third-party resources and articles. This resulted in us having to turn to official documentation a lot. However, it would be nice to have had more examples from articles and other sites.

Another possible disadvantage of developing in Xcode is its performance. Xcode's indexing and building can be slow, and its performance is not entirely flawless.

Developing iOS applications using Swift, SwiftUI, and Xcode offered an easy learning curve and clean syntax. Despite the limitations mentioned in this subsection, the benefits outweighed the drawbacks we experienced underways.

5.4 Text To Speech Alternatives

The application utilizes TTS voices available on iOS devices. This works well as Apple offers high-quality voice models that sound like real-life humans. In addition to the voices available by Apple, the team discussed other alternatives with the employers. These alternatives were; pre-recorded audio files or using AI models due to the rapid advancement of quality. However, the high-quality voices Apple offers are good enough for our purpose. Our implementation can also be extended to support a broader set of these voices, providing more customizability for the end user.

5.4.1 Phonetic Alphabet

Letters can sometimes be difficult to distinguish from each other. Especially letters like M and N. It is not currently implemented in the application, however, the team thought of using the phonetic alphabet as a solution. Ideally, this should be a user-based setting allowing the workers to choose their preferences. It would further improve the quality of our voice interface, but it was not a priority as our solution was adequate.

5.5 Voice and Touch Concurrency Challenges

As we mentioned earlier in 1.2.3, voice and touch concurrency was one of the most important features of our application. Our solution works incredibly well. However, since it was such a big part of our project, we have thought about other approaches to solve it.

5.5.1 Alternative Approach

Instead of our current solution explained in 4.4.4, we thought about a solution using a doubly linked list 5.5.1. A DLL is a data structure where each node in the list points to the next and previous node, the last node only points to the previous node, and the first node only points to the next node. A Task (step in the pluck flow), contains a map with keywords as keys, and a function as the value. This implementation would circumvent one of the current switch cases, as we could look up the correct function based on the entered keyword, instead of switching over possible keywords in each function.

The biggest advantage of the DLL approach is that it would allow us to add keyword-function pairs at runtime without having to modify the logic of our code. This flexibility would allow us to implement a feature where the user could decide their own keywords, further increasing customizability. However, the DLL approach would also increase the complexity of our code.

Even though the DLL solution would provide our application with a broader set of features, user-selected keywords were not a part of our scope. In addition, we were already far along with our first solution when we thought of the DLL approach.

5.6 Back-end

5.6.1 REST API and WebSockets API

While recognizing the benefits of using Websockets for event-driven push notifications, we still decided to use REST architecture for our API. REST has proven to be effective for a wide range of use and offers a simpler approach to our current application requirements.

It is worth noting that the addition of WebSockets could provide advantages in terms of bidirectional communication between the client and server. This would give a more responsive user experience, but also reduce the server load.

5.6.2 Reflection on Test Driven Development

Since test-driven development has been a focus throughout the project, the team has gotten some key insight into the workflow. In the beginning, it can be a bit frustrating and tedious as the outcome of testing is not as transparent. Nevertheless, as the project expands, the significance of a comprehensive test suite becomes increasingly valuable.

One of the key problems was that the team started writing tests without following the correct stages; arrange, act, assert, and tear-down. The team had to

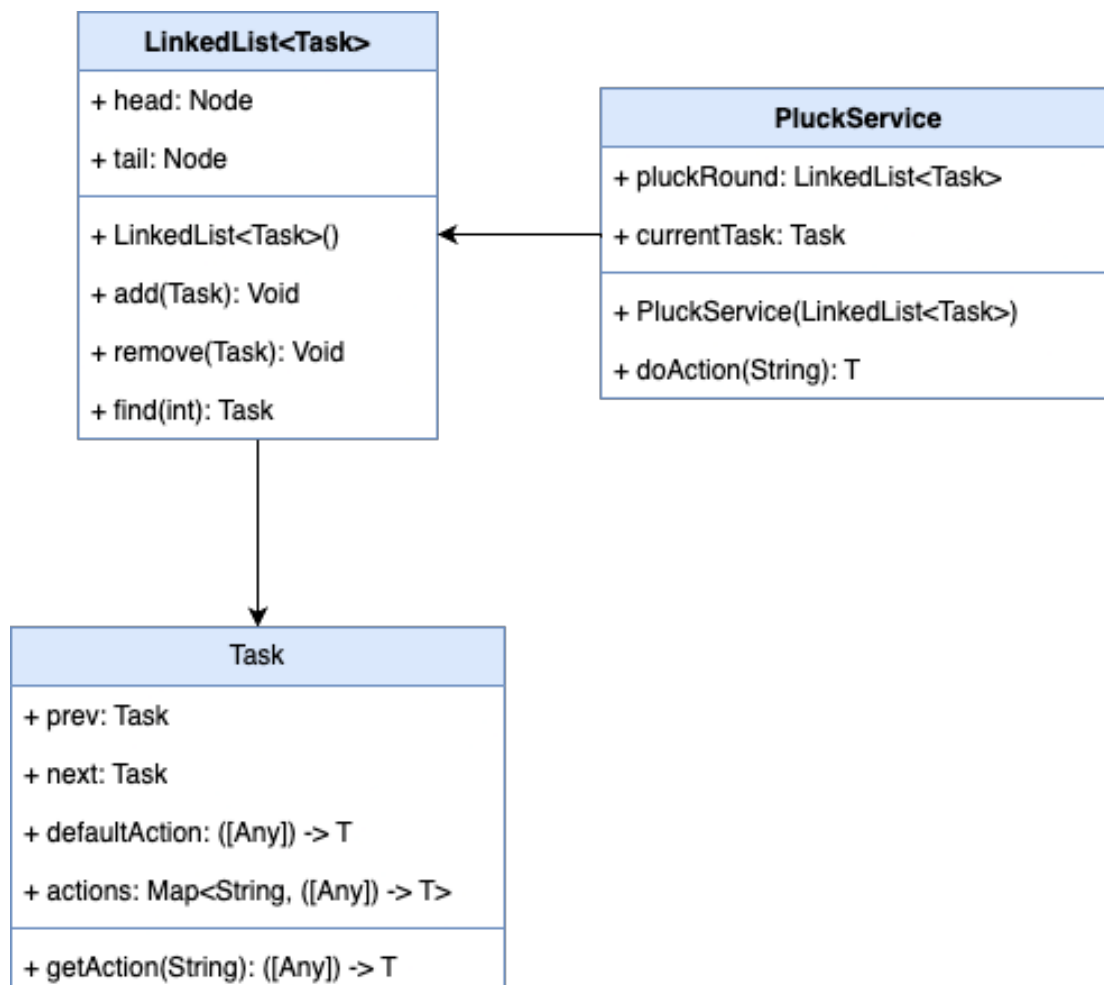


Figure 5.5.1: Linked List Solution

examine each test and use a lot of time rewriting them to follow the proper stages. After the rewriting, the tests gave a much higher value to the project.

The top-down approach followed throughout the project has also caused some issues with the testing. Mainly when features would be extended, the test would also have to be rewritten to take into account the addition. There might be solutions to overcome this challenge, however, the team's limited experience with this approach resulted in a significant amount of time being spent on rewriting the tests.

On the contrary, when adding new features, the testing has helped ensure that the previously added features still work as intended. This gave us confidence that new features did not brake the old ones.

TDD is an effective and efficient workflow if implemented correctly. However, if it's poorly executed it can inflict more harm than it provides assistance. Therefore, it's worth investing time in learning about TDD, ensuring that you do not fall into the same pitfalls as we have experienced.

5.7 Database Challenges

5.7.1 Testing Persistence Data

Testing persistence data was a challenge in the early stage of the project as most of the tests did not follow a proper structure. Spring Boot allows for ordering tests, making sure they are executed in a specific order every time. This was utilized to fix the original problem. However, we quickly discovered that in our use case, it was an antipattern. Each test should be independent and therefore not need to be executed in a specific order. Instead, refactoring every test to follow a proper structure with arrange, act, assert, and tear-down was a more optimal way of solving the problem.

5.7.2 Migration

As mentioned in chapter 3.3.1, we wanted to develop the project using a top-down approach. When adding or updating features, we often had to update the schema of the database. This has caused some issues along the way as nobody on the team knew exactly how to tackle this problem. After we realized this was a recurring problem, we started looking into database migration as this seemed to be the solution to our problem.

Spring Boot hibernate has an option called `spring.jpa.hibernate.ddl-auto` that can be set to `update`. This will enable hibernate to update the schema of the database based on the entities defined in the application. Our original approach was to use this feature, however, this is considered bad practice as you do not have full control over how hibernate will migrate the database. In production, this can cause data inconsistency and loss of data. This option should be set to `none` and updating the database schema should be done via migration scripts.

It's common to define the schema of the database in SQL scripts. As the database change, the migration scripts is the single source of truth. In migration scripts, it can be defined exactly how the database should be migrated resulting in the developers having full control of the database schema at all times.

5.7.3 Connection Pool

During the development, we had some challenges regarding overstepping the default connection pool size. We hosted our own database with two schemas. One for production and one for development. Because we all connected to the dev schema when developing, this caused a lot of connections as a default Spring Boot application normally creates ten connections to its data source. To address this issue, we changed the default lifespan of a connection, evidently fixing the problem.

CONCLUSIONS

6.1 Conclusion

The main objective of our assignment was to remove the need for a third-party paid service for voice recognition and utterance. Additionally, it was important to solve the problems with the current system, such as rearranging the pluck order.

We have developed a fully functional production-ready voice and touch-controlled system to assist warehouse workers with plucking. Additionally, the back-end WMS was built from scratch using the same technologies as Solwr, to assist with future integration and merging. With this, our API supports features for role-based authentication, user management, email transfer, and general CRUD operations for warehouse entities, making it a complete WMS.

The mobile application utilizes our WMS. Our application supports a complete set of WMS features. These features include joining/creating a warehouse, and adding, removing, and editing products, locations, and members of a warehouse. Above all, our application supports plucking orders using both voice and touch concurrently, without the use of a paid library, solving the main objective of our assignment. The addition of the touch interface solves problems such as rearranging the pluck order and managing your warehouse. In addition, this solution is designed to be reusable and easily configurable, allowing any company to adapt it to its specific warehouse layout. By prioritizing user-friendliness and intuitiveness, we have created an application that is accessible to a wide range of users, regardless of their technical expertise.

The group has reflected on alternative solutions to our problem and given extensive arguments for both sides. Furthermore, we have conducted valuable research on possible implementation for an Android mobile application, a watchOS application, and an Android Smart Watch application.

We have received positive feedback from our employer during our development process. All desired features are implemented and other approaches, along with future work have been discussed.

6.2 Future Work

The application has been designed with the potential for further additions in mind. The following section will outline some ideas that can be expanded upon to enhance its functionality.

6.2.1 Apple Watch

Currently our application is designed and tested specifically for iOS. The code can be easily reused to make a companion Apple Watch application as discussed in 4.7.1

6.2.2 Truck Inspection

The warehouse workers perform a mandatory daily truck inspection in order to check that the trucks have enough battery, and battery acid and are overall safe to operate. They then have to fill out a paper sheet documenting this inspection. This documentation process could be implemented in our app.

6.2.3 Feature Toggling

As more and more features are implemented, the application and warehouse businesses might benefit from feature toggling, like disabling truck inspection or other features. Further improving the tailored experience, by allowing them to only select the features necessary, thus reducing any overhead that might cause confusion.

6.2.4 Landscape Mode

As some of the warehouse workers might want to use the application strapped onto their wrists, they might benefit from a layout that is designed specifically for landscape. Currently, nothing stops users from using the application in landscape mode, however, the currently implemented layout is not designed for this use case.

6.2.5 Motion Detection

Since the warehouse workers are prohibited from using their mobile phones while driving the forklifts. The application would benefit from using motion detection to decide when the user can use the touch interface. This would increase the safety and enforce the company's health, environment, and safety rules on the workers.

6.2.6 Linked List Approach

The application could benefit from the approach discussed in 5.5.1. It would enhance the user experience by introducing even more customizability.

SOCIAL IMPACT

Ethical aspects

Accessibility

The introduction of a touch interface makes the application more accessible. In addition, the application introduces personal options for voice such as speed and volume further enhancing the accessibility aspect.

Privacy

Our application asks for permission to use the device's speaker and microphone. The user can decide if they want to use these features, as the application is still usable without these permissions, but at the cost of the features.

Additionally, sending voice recordings across servers to be processed raises concerns about privacy. In our application we have utilized on-device recognition, removing the need for sending the recordings between servers. This ensures that the user does not have to worry about privacy, as the voice recordings never leave the device.

Security

Our system's communication is encrypted, all APIs and sites deployed use HTTPS. Endpoints are secured and require authentication so that no unauthorized user can access them without the required privileges. The system is deployed on its own personal VLAN and does not expose any unnecessary ports to further increase security. In addition, personal devices store sensitive data in a keychain where everything is encrypted. Finally, user passwords are encrypted and stored in Keycloak, ensuring that passwords are securely stored, minimizing the risk of attacks.

Economic effects

Productivity and Efficiency

Combining touch and voice will help streamline the process of preparing orders from grocery stores. This will help increase the productivity and efficiency within the warehouse. With the combination, the workers can work in a way that suits them best, helping speed up the process for each individual pluck order. Warehouses can expect to use less time to prepare each order, therefore having the capacity to prepare more orders.

Cost Savings

If Solwr replaced the current system with our system, they would save a significant amount of money that they use for the VoiceConnect voice recognition service. End users, such as H.I. Giørtz would also save costs because our application increases productivity and efficiency at the warehouse.

Quality Control

The application gives an overview of entities in a warehouse. This overview can help reduce errors in warehouse inventory management. With fewer errors, there will be spent less time to fix the errors, enabling workers to focus on preparing orders.

The Application's Contribution to the UN's Sustainable Development Goals

9. Build Resilient Infrastructure, Promote Inclusive and Sustainable Industrialization and Foster Innovation

In goal 9, the UN explains how the world needs to develop its infrastructure to support substantial economic development and quality of life [45]. The logistics behind trade of goods have a significant impact on the world. Our application helps streamline a small part of the overall logistics process. However, small improvements can lead to significant impact as actions are repeated multiple times, and the cumulative effect of streamlining can be substantial. Furthermore, the extension of a touch interface makes our application more inclusive, enabling more people the opportunity to work in a warehouse.

12. Ensure Sustainable Consumption and Production Patterns

The UN explains their 12. goal by stating that the world needs to achieve more with fewer resources [46]. A grocery warehouse is an example of an area where our application can be used. Since food items have a relatively short expiration time, a comprehensive warehouse management system that provides a good overview can

reduce waste. By effectively managing inventory and ensuring rotation of goods, our application can contribute to minimizing waste in the food industry.

REFERENCES

- [1] NTNU. *Emne - Mobile applikasjoner - IDATA2503 - NTNU*. URL: <https://www.ntnu.no/studier/emner/IDATA2503#tab=omEmnet> (visited on 04/27/2023).
- [2] Claire Drumond. *Scrum*. 2023. URL: <https://www.atlassian.com/agile/scrum> (visited on 03/01/2023).
- [3] *What is a Sprint in Scrum?* URL: <https://www.scrum.org/resources/what-is-a-sprint-in-scrum> (visited on 03/01/2023).
- [4] Dave West. *Agile scrum roles and responsibilities*. Atlassian. URL: <https://www.atlassian.com/agile/scrum/roles> (visited on 04/02/2023).
- [5] Deepti Sinha. *What is an Epic in Agile? examples and Differences*. knowledgehut. URL: <https://www.knowledgehut.com/blog/agile/what-is-an-epic-agile> (visited on 04/02/2023).
- [6] *Task*. Agile Academy. URL: <https://www.agile-academy.com/en/agile-dictionary/task/> (visited on 04/02/2023).
- [7] *What is version control?* Atlassian. URL: <https://www.atlassian.com/git/tutorials/what-is-version-control> (visited on 05/01/2023).
- [8] *What is a REST API?* Red Hat. May 2020. URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api> (visited on 04/24/2023).
- [9] *The WebSocket API (WebSockets)*. mdn web docs. Mar. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (visited on 04/23/2023).
- [10] Auth0. *Role-Based Access Control*. 2022. URL: <https://auth0.com/docs/manage-users/access-control/rbac> (visited on 04/25/2023).
- [11] *JWT.IO - JSON Web Tokens Introduction*. URL: <https://jwt.io/introduction> (visited on 04/25/2023).
- [12] *Spring Boot - Introduction*. Tutorialspoint.com. 2019. URL: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm (visited on 04/24/2023).
- [13] *Core Technologies*. Spring. URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-introduction> (visited on 04/24/2023).
- [14] *Swagger tutorial*. Java T Point. URL: <https://www.javatpoint.com/swagger> (visited on 04/24/2023).

- [15] Fatima Junaid. *What is SwiftUI?* educative. URL: <https://www.educative.io/answers/what-is-swiftui> (visited on 04/24/2023).
- [16] Jamis Charles. *What is Debouncing?* Medium. URL: <https://medium.com/@jamischarles/what-is-debouncing-2505c0648ff1> (visited on 05/15/2023).
- [17] Alexander S. Gillis. *Voice Recognition (Speaker Recognition)*. URL: <https://www.techtarget.com/searchcustomerexperience/definition/voice-recognition-speaker-recognition> (visited on 04/28/2023).
- [18] *What is text to speech?* URL: <https://www.naturalreaders.com/online/> (visited on 04/28/2023).
- [19] Ian Buchannan. *Feature flags*. Atlassian. URL: <https://www.atlassian.com/continuous-delivery/principles/feature-flags> (visited on 04/28/2023).
- [20] *What is a relational database?* Google. URL: <https://cloud.google.com/learn/what-is-a-relational-database> (visited on 04/11/2023).
- [21] *Database migration: Concepts and principles (Part 1)*. Google. Oct. 2022. URL: <https://cloud.google.com/architecture/database-migration-concepts-principles-part-1> (visited on 04/11/2023).
- [22] *Flyway*. Flyway. URL: <https://flywaydb.org/> (visited on 04/11/2023).
- [23] *Welcome to the Liquibase Community*. Liquibase. URL: <https://www.liquibase.org/> (visited on 04/11/2023).
- [24] Thomas Hamilton. *What is Test Drive Development?* 2023. URL: <https://www.guru99.com/test-driven-development.html> (visited on 03/17/2023).
- [25] Martin Sandin. *Four Strategies for Organizing Code*. Feb. 2016. URL: <https://medium.com/@msandin/strategies-for-organizing-code-2c9d690b6f33> (visited on 04/01/2023).
- [26] Web Dev Simplified. *Junior vs Senior React Folder Structure - How To Organize React Projects*. Youtube. July 2022. URL: <https://www.youtube.com/watch?v=UUga4-z7b6s&t=581s> (visited on 04/01/2023).
- [27] Rowan Haddad. *Git branching strategies*. Flagship. Mar. 2022. URL: <https://www.flagship.io/git-branching-strategies/> (visited on 04/24/2023).
- [28] Amigoscode. *10 Spring and Spring Boot Common Mistakes You Need To STOP*. Apr. 2023. URL: <https://youtu.be/CT8dbbe783s?t=22> (visited on 04/24/2023).
- [29] *Apple Developer Documentation*. URL: <https://developer.apple.com/documentation/speech/sfspeechrecognizer> (visited on 04/28/2023).
- [30] Apples Machine learning research. *On-device Neural Speech Synthesis*. URL: <https://machinelearning.apple.com/research/on-device-neural-speech> (visited on 05/12/2023).
- [31] *AVAudioSession*. URL: <https://developer.apple.com/documentation/avfaudio/avaudiosession> (visited on 05/16/2023).
- [32] Paul Hudson. *Scanner*. URL: <https://github.com/twostraws/CodeScanner> (visited on 04/27/2023).

- [33] Bill Morefield. *How To Secure iOS User Data: Keychain Services and Biometrics with SwiftUI*. Aug. 2020. URL: <https://www.kodeco.com/11496196-how-to-secure-ios-user-data-keychain-services-and-biometrics-with-swiftui> (visited on 05/01/2023).
- [34] *SpeechRecognizer | Android Developers*. 2019. URL: <https://developer.android.com/reference/android/speech/SpeechRecognizer> (visited on 04/28/2023).
- [35] *cmusphinx/pocketsphinx*. URL: <https://github.com/cmusphinx/pocketsphinx> (visited on 04/28/2023).
- [36] *Kaldi Speech Recognition Toolkit*. Feb. 2023. URL: <https://github.com/kaldi-asr/kaldi> (visited on 04/28/2023).
- [37] *TextToSpeech | Android Developers*. 2019. URL: <https://developer.android.com/reference/android/speech/tts/TextToSpeech> (visited on 04/28/2023).
- [38] *Apple Developer Documentation*. URL: <https://developer.apple.com/tutorials/swiftui/creating-a-watchos-app> (visited on 04/30/2023).
- [39] *Watch Connectivity*. URL: <https://developer.apple.com/documentation/watchconnectivity> (visited on 04/30/2023).
- [40] *ObservableObject*. URL: <https://developer.apple.com/documentation/combine/observableobject> (visited on 05/01/2023).
- [41] OpenID. *openid/AppAuth-iOS*. 2023. URL: <https://github.com/openid/AppAuth-iOS> (visited on 04/26/2023).
- [42] Apple. *UIKit | Apple Developer Documentation*. 2020. URL: <https://developer.apple.com/documentation/uikit> (visited on 04/26/2023).
- [43] *Set up Keycloak in Spring Boot using the Keycloak Admin API*. URL: <https://gauthier-cassany.com/posts/spring-boot-keycloak-admin-api> (visited on 04/27/2023).
- [44] Firebase. *Add the Firebase Admin SDK to your server*. URL: <https://firebase.google.com/docs/admin/setup> (visited on 04/27/2023).
- [45] *9 innovasjon og infrastruktur*. UN. URL: <https://unric.org/no/mal-9/> (visited on 05/20/2023).
- [46] *12 ansvarlig forbruk og produksjon*. UN. URL: <https://www.fn.no/om-fn/fns-baerekraftsmaal/ansvarlig-forbruk-og-produksjon> (visited on 05/20/2023).
- [47] *React Native*. 2023. URL: <https://reactnative.dev/> (visited on 03/29/2023).
- [48] Maksym Churylov. *Pros and cons of React Native for cross-platofmr app development*. URL: <https://www.mindk.com/blog/react-native-pros-and-cons/> (visited on 03/28/2023).
- [49] *Make any app. Run it everywhere*. URL: <https://expo.dev/> (visited on 03/28/2023).
- [50] *What are the different kinds of cases?* Jan. 2022. URL: <https://stackoverflow.com/questions/17326185/what-are-the-different-kinds-of-cases> (visited on 03/28/2023).

- [51] *Rapidly build modern websites without ever leaving your HTML*. URL: <https://tailwindcss.com/> (visited on 03/28/2023).
- [52] Amazon. *What is flutter?* URL: <https://aws.amazon.com/what-is/flutter/> (visited on 04/25/2023).
- [53] *Null Safety Support for Flutter and Dart*. Section. Oct. 2021. URL: <https://www.section.io/engineering-education/null-safety-support-for-flutter-and-dart/> (visited on 04/27/2023).
- [54] Kotlin. *Multiplatform*. URL: <https://kotlinlang.org/lp/mobile/> (visited on 04/25/2023).
- [55] Android. *Jetpack Compose*. URL: <https://developer.android.com/jetpack/compose> (visited on 04/25/2023).

APPENDICES

APPENDIX

A

PRELIMINARY PROJECT PLAN

Preliminary Project Plan

eDate	Version	Descriptions	Author
11.01	0.1	Template version	@ Joakim Edvardsen
18.01	0.2	Goals & Boundaries	@ Håkon Sætre
19.01	1.0	Report first version	@ Petter Molnes @ Håkon Sætre @ Joakim Edvardsen
20.01	1.1	Cleanup, remove duplicates	@ Mateusz Picheta @ Petter Molnes

Goals & Boundaries

1.1 Overview

We knew that Solwr would supply NTNU with a couple of Bachelor Assignments since three of the members in this group work there. We expressed our wish in getting one of their assignments. There were several reasons for this: 3 of our group members had the course Mobile-Development during our 5th semester, and all of us found the course and the work intriguing. Additionally, all of us got excellent grades and feedback. Furthermore, the three of us who work at Solwr have a great relationship and experience working for them. We all wanted to learn more about Mobile-Applications and work more with Solwr, so, therefore, we applied for this assignment as our first choice.

1.2 Problem statement / Project description and results goals

Help Solwr develop a mobile application improving the process of picking products in large warehouses. Currently, Solwr has a voice pick system controlled by employees using their voice together with a headset. They want to develop a mobile application, keeping the benefits of their current implementation with voice pick and the flexibility of a touch screen on a touch device.

We will use this as an opportunity to explore and develop a solution with technologies that Solwr currently does not have the resources to look into. For example, integration with smart watches, scanners and/or sensors, or even smart glasses. Ultimately helping Solwr explore technologies that can help improve the process of picking products in large warehouses.

The result goal we strive for is to create an accessible application that fits in well with Solwr's ecosystem of software and applications. It should not stand out from their other software and work on a wide range of devices. Aswell as do its main task, voice picking effectively.

1.3 Effect goals

Our group's goal with this project is to enhance our knowledge of Mobile-Applications and develop in a team with SCRUM methodology. Also, the project gives us the opportunity to explore technologies that we haven't used before, such as Swift, React Native, Kotlin, voice-recognition, Smart Watches, etc.

Our prototype will be used by Solwr to look at the possibility to implement this on a wider scale as a marketable product. By utilizing the result of our project, Solwr can increase its productivity if they decide to further develop this idea. They can look at what we did well, and what could have been better so that they don't make the same mistakes that we might do. Currently, the user of the existing pick system can only use their voice to interact with the system they are using now. If Solwr decided to further develop this prototype, their customers will have a more effective and flexible workflow. Ideally, the customer would be able to use their voice through their iPhone or Apple Watch to interact with the application, additionally, they will have the possibility to use the on-screen interface to adjust picking-order, and mark items as complete. Furthermore, the employer at warehouses would not have to buy as many devices, as the employees would be able to use their own devices.

In conclusion, this will be a research-driven project that Solwr can decide to further develop to make a marketable product that increases the productivity and effectivity at warehouses, and decreases the upfront cost for physical devices.

1.4 Boundaries

For this project, we don't have much need for money or any equipment. What we need is a cloud server to run our back-end. The cloud server will be provided by either Solwr or NTNU. If publishing the app to App Store is a priority, Solwr will provide Apple Developer Accounts for us so that we can go through that process.

Our employer at Solwr will meet with us every two weeks at the end of our sprints, to have a review with us. Mainly we will be seated at NTNU when we work on our project, but if we want or need to sit at Solwr's office, they have the possibility to assign us some desks.

Additionally, we have requested a tour of the warehouse where the application would be used. This is so that we as a group can have a better understanding of the workflow and take that into account when we develop the application. Gjørtz agreed that we can come over and get an introduction to how they work and try it ourselves.

2. Organizing

Project manager:

Our coordinator and project manager are Mikael Tollefsen from NTNU, which is going to ensure that the bachelor structure and content are on track.

Development team:

Our bachelor group consists of Joakim Edvardsen, Petter Molnes, Mateusz Picheta, and Håkon Sætre. We are the ones that are responsible for designing, building, and testing the application.

End users:

Our target is companies that need an efficient way to have control over their products. The target for our product is Gjørtz, which is the leading logistics firm in Ålesund.

Quality assurance team:

Gjørtz, we want to have good communication with our customers and make sure they test our product for further improvements. Testing will also be continually done by our group and Solwr. Mikael Tollefsen as our NTNU supervisor will also supervise our work-process and report-writing.

3. Implementation

3.1 Main activities

Listing of main activities.

Our project will be separated into several phases of activities.

- Preparation - we will spend our first week having a kick-off meeting with Solwr to get an understanding of our working relationship during this project. As well as signing off on contracts between our group, and also with Solwr. Furthermore we will spend time setting up Jira and proper documentation for our project. This is done so that we have a streamlined process where every group member knows where to find information.
- Planning - afterwards, we will analyze the problem domain and create epics, user-stories and tasks based on what is written in the project proposal. Have a meeting with Solwr where we get feedback on this and what should be prioritised.

Plan our sprints so that we have a clear view of when we need to start and finish our main activities.

The group members will plan the architecture and infrastructure of our system and document this properly. Furthermore we need to design how the app will look and feel for our end-users, with input from Solwr.

By doing this, we hope to achieve a more effective and easier process when we develop our application.

- Research - before beginning development, the group members need to look into what tools we need to use based on the feedback from Solwr. We need to look into the different technologies that we consider using and what suits our needs the best. This is done so that all our group members will be familiar with the different choices we have, and become comfortable with the technologies that we end up using.
- Development - after having researched, and decided which route we want to go, we will go into the longest phase of this project, development. In this phase, every group member will be responsible to make sure we can deliver a product on time, which will satisfy the needs of our employer.
- Report writing - this will also be done concurrently with all the other phases, since it is the most important aspect of our Bachelor. However, we will use the last phase to finalize and fine tune our report. We will get some feedback from our supervisor at NTNU before we hand it in.

Every part of the process should be documented on our Confluence and Jira page, so that we have all the necessary information we need, when we write our project report.

3.2 Milestones

- Preliminary project plan - 28. January
- Poster - 28. mars
- MVP in middle of April.
- Oral presentation (English) - week 13 (*date TBD*)
- Documentation of work process (Status reports, meeting notices, meeting notes, log) - 20. May
- Oral presentation of final product - 27. May

4. Follow-up and quality assurance

4.1 Quality assurance

To ensure good quality of our project we will have good sprint reports to make sure that everything is done and everyone is working on the task they have been assigned. We as a group want to set clear guidelines for the project and will ensure that everyone in the group, our employer, and our supervisor understands what is expected of them and us to provide good quality.

Having good communication is crucial for ensuring quality. To regularly have good communication with our customers and get feedback on what has been done and what needs to be done. By continuous evaluation and improvement of the process, we will make sure that the project is moving in the right direction.

Documentation of the whole process will make sure that every party is aware of the guidelines and will make it easy to identify errors in the process.

4.2 Reporting

We will work in two-week sprints. Each sprint consists of some internal meetings with only team members, some meetings with the supervisor, and some meetings with the employer.

In internal meetings, we will discuss daily stand-up, sprint planning, and spring retrospective. During daily stand-ups, we will report if there is a problem with any issues or if some of the estimations have to be adjusted. Sprint planning will happen once every spring when the sprint starts. During the sprint planning, we will plan the next sprint, what the goal is and what tasks we should work on during the spring. Lastly, at the end of each sprint, we will have a sprint retrospective meeting to discuss what went well, and what can be done better.

At the end of each sprint, we will also have a sprint review with the employer where we discuss what we have done during the sprint. Also, get feedback from the employer and input on what should be priorities. Sprint review will be combined with status rapport since the supervisor will always be present. A meeting notice should be sent to both supervisor and employer to notify specific time and date, with the relevant agenda.

5. Risk assessment

Risk analysis that assesses vulnerabilities in the project (event, probability, consequence, and measures)

In this project, we have some points that could potentially affect the project negatively. If one of our members for some reason can't attend or is sick, it might impact the project by not having all of the necessary parts correct or on time. The probability of this happening is low, cause most of our tasks can be done remotely except for spring and group meetings. If this happens and someone in our group is very sick, we would need good communication to probably split up the task for others to do.

When it comes to risk assessing the application. We need to make sure the application is secure, that all our APIs are secured and function as expected. Make sure the application is compatible with various devices as well as making sure the application is performant.

Another risk is the time we have, we need to plan out our schedule to make sure we achieve all our goals before the deadline. Lastly, we have to make sure not to overreach with the scope of the project. We do not want to end up with a too wide scope where we haven't actually solved anything.








6. Attachments

The following documents are delivered as separate files when submitted in Blackboard in January (mandatory work requirement), but not in the final delivery of the main report on 20 May!

6.1 Schedule

Here is the first iteration of our roadmap. This could of course change over the course of the project.

	JAN	FEB	MAR	APR	MAY
Sprints	VP Sprint 1	VP Sprint 2			
VP-117 Write project report	[Task bar spanning from start of JAN to end of MAY]				
VP-9 Forprosjektplan	[Task bar in JAN]				

> VP-13 Setup tools we are using for the project					
> VP-19 Project Planning					
> VP-24 Research of Technologies					
> VP-32 Create Figma design sketches & design gui...					
> VP-35 Authentication					
> VP-52 Managing Warehouse					
> VP-82 Picking through voice					
> VP-98 Picking with touch interface					
VP-118 Create application for smart-watches					
> VP-107 Deploy application					
> VP-88 SCRUM					

6.2 Agreement documents

6.3 Work contract for bachelor group

This is a link to our group contract in Confluence, if you don't have access, look for pdf attached to the hand-in.

[Group contract signed](#)

3-party agreement

This is a link to our Standard Agreement in Confluence, if you don't have access, look for pdf attached to the hand-in.

[Standard Agreement](#)

GITLAB REPOSITORY

All code developed during the project are included in the GitLab repositories linked below. Further explanations are given in the readme-file.

B.0.1 SwiftUI GitLab Repository Link

- <https://gitlab.com/IDATA-2900-Group-1/voice-pick-frontend>

B.0.2 Spring Boot GitLab Repository Link

- <https://gitlab.com/IDATA-2900-Group-1/voice-pick-backend>

B.0.3 Iac GitLab Repository Link

- <https://gitlab.com/IDATA-2900-Group-1/voice-pick-iac>

PLUCK FLOW EXAMPLE

In this appendix, you will find a youtube link with a video of one of our developers testing out our application for completing a pluck round.

- <https://www.youtube.com/watch?v=ldt2QxZ4jbU>

RESEARCH

D.1 React Native

React Native is a cross-platform framework developed by Meta that allows for the creation of mobile applications. The framework utilizes a write-once, run-anywhere strategy, enabling developers to create applications that can be deployed on multiple mobile platforms, such as iOS and Android [47].

The framework is written in JavaScript, which allows for similarities with other frameworks commonly used for web development [47]. The React Native compiler converts the JavaScript code written by the developers into native code, thus enabling the use of the same native platform APIs as other applications.

D.1.1 Native Support

React Native, while offering cross-platform compatibility, still experiences limited compatibility with native APIs. Due to the framework's JavaScript-based architecture, some native features may require the use of third-party libraries or custom modules. This can result in longer development time as well as increased maintenance costs. Additionally, compatibility with newer versions of native platforms may also be limited and require additional updates to the framework or its associated libraries [48]. These limitations may hinder the ability of developers to utilize certain native functionalities, potentially impacting the overall user experience of the application.

D.1.2 TypeScript Support

In addition to its core functionality, React Native also offers support for TypeScript, a typed superset of JavaScript that provides additional features such as type-checking and auto-completion. This feature furthers the development experience, allowing for more efficient application creation.

D.1.3 Expo

Expo Go is a framework that facilitates the development of React Native applications by providing a set of tools and services for the developer to utilize [49].

The framework offers an alternative approach to the traditional React Native CLI, providing a more user-friendly and streamlined development experience.

After creating an application with Expo Go, developers can run the application and receive a QR code in the terminal. This code can be scanned using the Expo Go app, available on both the App Store and Google Play, to open the application on any mobile device. The application will automatically hot reload and reflect any changes made to the code, providing a smooth developer experience. Additionally, developers can share the QR code with others to receive feedback on the application.

```
Starting Metro Bundler



> Metro waiting on exp://10.22.184.86:19000
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

> Press a | open Android
> Press i | open iOS simulator
> Press w | open web

> Press j | open debugger
> Press r | reload app
> Press m | toggle menu

> Press ? | show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
iOS Bundling complete 17969ms
```

Figure D.1.1: Expo Go output in the terminal when running the application

D.1.4 Styling

As mentioned above, React Native is a JavaScript framework, meaning it's quite similar to other front-end frameworks used to develop websites. This compatibility can benefit teams who have expertise in these JavaScript frameworks. Another familiarity React Native shares with web development is how the structure and layout of the application are designed as well as how each individual component is structured. By default, React Native offers a language almost the same as CSS used in web development. The main difference is CSS uses a kebab case scheme for its naming conventions, whereas the CSS offered by React Native changes this to a camel case scheme [50].

Furthermore, Tailwind can easily be integrated into a React Native app, just like any other web development framework. Tailwind is a vast library that comprises utility classes, which can simplify the application development process [51]. Since only the utilized classes are included in the final application during compilation, the application size is kept to a minimum, eliminating any concerns of overload.

D.2 Flutter

Flutter is a framework developed by Google, that is used for creating modern frontend and full-stack application user interfaces. Being cross-platform, Flutter can compile to multiple different devices using a single codebase. Flutter uses the programming language Dart, which Google also developed. With the use of dart and flutter's widgets, you can create visually appealing user interfaces. In Flutter, widgets are the main components for building UI layouts, meaning that everything visible in the app is made out of widgets. [52]

D.2.1 Null Safety

One of the best additions to Flutter is null safety. With null safety, the compiler forces the developers to avoid null values in variables [53]. This feature is supported in multiple languages and the implementation in Flutter enhances the development experience.

D.2.2 Previous Experience

Flutter is a framework some of us have used before. It's great for making cross-platform apps and has a nice UI with many customizable widgets and a modern design. While Flutter performs quickly, it does have limited library choices and lacks native support.

For our project, we will not be using Flutter. The cons of Flutter outweigh the pros. We need good native support for our voice commands, and performance is really important for our customers since they want to be able to do their job as effectively as possible. Also, the focus of our project isn't really to have an application for both iOS and Android but streamline the workflow for our users with both a touch interface and a voice interface.

D.3 Android

D.3.1 Kotlin

Kotlin is a framework developed by Android. It is designed for creating cross-platform applications. Kotlin is trusted by many of the world's leading companies and used by over 60% professional Android developers. It reduces the time spent writing and maintaining the same code for different platforms while retaining flexible native programming [54].

D.3.2 Jetpack Compose

Jetpack Compose, is a modern toolkit for building native UI. Jetpack Compose makes it possible to make appealing applications with less code and is therefore simple and easy to maintain. The code is written only in Kotlin, rather than having it split between Kotlin and XML, offering a more unified development experience. [55]

D.4 SwiftUI

SwiftUI is a modern framework developed by Apple for building user interfaces across its operating system ecosystem, including iOS, iPadOS, macOS, watchOS, and tvOS. The framework provides a simple and intuitive syntax for constructing user interfaces, making it easier for developers to create visually appealing and functional apps. It uses the powerful features of the Swift programming language to offer automatic support for critical elements like accessibility, localization, and dynamic type. SwiftUI is designed to work seamlessly with other Apple technologies, including Core Data, Combine, and Xcode, providing a comprehensive solution for app development. With its unified tools and APIs, SwiftUI allows developers to create apps for all Apple platforms using a single codebase, streamlining the development process. [15]

D.4.1 Styling

SwiftUI has a powerful styling system that lets developers make their user interfaces look good. The system uses ViewModifiers, which are small pieces that can be put together to create different UI elements. SwiftUI has many built-in ViewModifiers for buttons, text, and images. Developers can also make their own custom ViewModifiers.

Consistency with styles is easy to maintain with SwiftUI. Like changing style to text, font, color, and other features. Swift supports themes, which are collections of styling that can be applied throughout the whole app.

Automatic support for dark mode, so the app can switch to a dark look when the user sets their device too dark mode. With all these tools, SwiftUI helps developers make their apps look great and work well. Dark mode was highly requested by our employer as well.

D.4.2 Icons

The framework provides a wide range of built-in icons, including system icons and customizable icons, that can be used to improve visual interests.

Developers can also create custom icons, either by creating them from scratch or by using third-party libraries. SwiftUI makes it easy to use custom icons like built-in icons, making it simple to add a unique touch to an application's user interface.

D.4.3 Speech to Text

With SwiftUI applications can make use of APIs and libraries to add functionality with SST. The speech framework provides access to the device's STT engine, allowing developers to convert spoken words into text.

