FineGrind - Developing a mobile app for customers of specialty cafes

Rita Miklán

Haaga-Helia University of Applied Sciences
Business and Information Technology
Bachelor's Thesis
2023

Abstract

Author(s)
Rita Miklán

Degree
Bachelor of Business Administration

Thesis Title
FineGrind – Developing a mobile app for customers of specialty cafes

Number of pages and appendix pages
37 - 6

This thesis follows the workflow of developing a cross-platform mobile app for the customers of specialty cafés.

Specialty coffee is growing in popularity. Small, independent coffee shops are opening to serve customers who want to know more about the coffee they drink and explore new specialty coffee. These places have a disadvantage over big chain cafés with a mobile app for the customers. This app strives to help users to discover more small cafés in their neighborhood. The work describes why such an app is needed and goes through the development steps, from user interviews to deploying the app.

The goal is to build on an already existing MVP version, define possible points of improvement through usability tests, and improve the overall app both from the UX and functionality side.

The theoretical part gives an overview of different approaches to mobile app development. The MVP is presented, and the already existing features are described. The decision to use React Native is also discussed, and the structure of the MVP is explained in detail. The tools and libraries used during the app's development are also introduced.

Data is collected via a user survey and usability testing to define the requirements and create a backlog. This data helps reveal what features and functionalities are lacking in the app.

In the implementation part, the author describes the work done on the code, such as refactoring the code, implementing new features, and improving some already existing features.

Although it is concluded that it will not be possible to finish the complete development of the app during the period of the thesis work, a significant improvement is achieved. For the future, several possibilities are listed, such as creating a café side twin app to work with the end user's mobile app, so the customers can pre-order their drink and also learn about coffee at the same time.

Keywords
Cross-platform mobile app development, React Native, Javascript, Specialty coffee

Table of contents

Glossary

API: Application programming interface

Clean coding practices: Coding practices that enable readable, maintainable, testable, and scalable code.

CLI: Command line interface

DRY coding: "Don't repeat yourself" – coding principle, strives to avoid redundancy in code

JSON: JavaScript Object Notation

MVC: Model – View – Controller architecture pattern

MVP: Minimal viable product

npm: Node package manager

POC: Proof of concept

POI: Point of interest, a specific location marked on maps

Q grader: A professional whose expertise and work is to grade coffee beans based on the standards provided by the SCA.

SCA: Specialty Coffee Association

SDK: Software development kit

Specialty coffee: Coffee that has been graded over 80 points on a scale of 1-100 by a certified grader at several phases of coffee production based on the standards provided by the SCA

Tech stack: A collection of technologies that work together to build an application

TDD: Test-driven development, the practice of writing tests before writing the code

UI: User interface

UX: User experience

3rd wave café: Synonymous with specialty café, a small independent coffee place that sells specialty grade coffee to its customers

1 Introduction

1.1 Background of the project

In this era, smartphones and mobile applications are part of our daily lives, but they have not been around long. Although the concept of a portable device that combined computing and telephone already existed in the 1970s, it took decades before the first prototypes of such a device appeared. Apple launched the first iPhone in 2007, which changed the mobile phone landscape and defined the future of smartphones. (Qualcomm & Want, 2014)

The App Store launched in 2008 to serve as a platform for third-party apps for iPhones since, initially, Apple developed all the apps for iPhones. The app store launched with 500 apps, which could be considered the beginning of the era of mobile apps as we know them today. (Silver, 2018)

Mobile app development is still growing steadily, with the number of downloaded apps increasing yearly. This trend shows that mobile apps are here to stay. This also means that mobile developers are highly sought after in the job market, and it is a good idea to familiarize oneself with at least the basics of it. These are just some reasons why the author became interested in mobile app development in the first place and decided to focus her efforts on deepening her knowledge about this topic.

The choice of app to develop is an app for finding specialty cafes, mainly in Helsinki. Specialty coffee culture is fascinating, and now we live in the era of the third wave of coffee. This means that small café shops open using high-quality beans, and quality rules over quantity. There might be a limited amount of beverages and a personal relationship with suppliers, such as roasteries or even farmers.

These cafes are up against bigger chains, such as Espresso House or Starbucks, where the location, name recognition, and frequency of stores play a significant advantage. They also have their own apps with built-in loyalty programs and offers. These apps serve a significant role in customer retention.

Small cafes do not have these advantages, and sometimes they are also hard to find when visiting a neighborhood we do not know well. The app would help customers to locate these cafes and give sufficient information about them, such as opening hours, addresses, and menu items.

This thesis aims for a working mobile app that helps users explore small, independent cafes. This app is built using React Native, keeping in mind coding best practices and also getting insight from

user interviews and user testing to improve the overall usability and functionality. The first MVP version of the app already exists, and this project will build on that.

1.2 A few words about specialty coffee

Coffee has been a beverage consumed worldwide since the 15th century. Commercial consumption in Europe and the United States began around the 18th century. This was considered the first wave of coffee. Coffee was a commodity that was low in price and consistent in taste, with little regard to tasting notes or quality. Today these coffees live as the mass-produced coffee found in all supermarkets, usually roasted dark, such as Tchibo or Juhla Mokka. These are highly available blends that people can prepare in their homes with a percolator or other simple methods.

The second wave of coffee is attributed to Starbucks, which focuses mainly on the experience and the different, often flavored drinks, such as the pumpkin spice latte during autumn. They gave a space where people could socialize outside of their work or home environment. (Peiper, 2022) They also started the takeaway coffee trend with branded paper cups and had a more extensive drink menu with lattes and cappuccinos. The coffee quality was already higher than the commodity coffee, and they started to write the country of origin on the bags, but it was still not specialty coffee.

The expression "3rd wave of coffee" was coined in the late 90s, but the specialty coffee industry started to evolve already in the early 80s. There the focus moved to the coffee beans. The roasters started experimenting with lighter roast profiles to bring out more flavors from the beans. The beans were graded based on a specific standard the Specialty Coffee Association provided. This standard determines the quality of the coffee, and the grading is done by professionals called Q graders. They score the coffee beans on a scale from 1 to 100 based on size, health, and other sensory characteristics. (Polner, 2021) The SCA was founded in 1982 to help standardize and support the coffee industry. They provide education for graders, roasters, and baristas alike. They value sustainable and ethical coffee operations and an open and supportive coffee community.

Specialty cafes usually work closely with roasters, and roasters usually work closely with the farmers themselves, creating a more transparent and sustainable supply chain from farm to cup. (Roasters, 2019)

1.3 Objectives and deliverables

The project's main objective is to add more functionality to the MVP. These functionalities will be assessed by establishing the target group and conducting surveys and interviews. After

establishing the target group, the author will conduct usability tests to evaluate the current app version. These usability tests will help determine what features and functionalities are missing or faulty in the app. Based on these usability tests, the backlog can be refined.

The planned new features already in the backlog are route planning, editing and adding user information, adding reviews, social sharing, and improving the UI and UX.

The learning objectives of the project are:

- developing general coding skills further
- have a better understanding of software development projects as a whole
- use different testing methods
- use different libraries and frameworks

The project will be complete with proper documentation of the app and its functionalities. It also will follow clean coding practices and strive to create a solid architectural framework.

Clean coding is not defined precisely, but the concept is discussed in-depth by Robert C Martin in his book Clean Code: A handbook of Agile Software Craftsmanship. (Martin, 2009)

In his book, he points out that among his peers, one of the main qualities of clean code is readability. He also mentions clarity, efficiency, simplicity, the importance of tests, and that the code is crafted with care. Some rules for clean code in his book are: giving variables meaningful names, keeping functions small and simple ("Do one thing"), tests should be added before the code (TDD), and tests should also uphold the same principles as the production code.

The more specific deliverables of this project are the results of the user survey and usability testing, defining a backlog that reflects the planned features, refactoring the code where it is needed, and adding the features listed above.

1.4 Apps already on the market

1.4.1 European Coffee Trip's app

When this project started in 2021, the app made by European Coffee Trip was only available for iOS. This was an interesting choice, given the market share for Android phones in the mobile phone market, especially in Europe. (StatCounter, 2023)

Currently, their app is also available for Android phones. The app has a list of cafes, an option to filter and search based on specific terms, shows a map view with the general area of the user's geolocation. It also adds cafes to favorites and shows individual profiles for cafes where they show

the primary information, such as opening hours, menu items, and available services. The app does not have the option to log in and store user data. The navigation to the cafes redirects the user to Google Maps. Also, the cafes do not have an individual "show on map" option.

This app is the closest to what FineGrind wishes to achieve, and the goal is to build an app with better and more functionality than the European Coffee Trip app.

1.4.2 Espresso House app

As mentioned earlier, big chains, such as Espresso House, have the advantage of having multiple stores around town and having the same drinks in every store – thus being reliable in giving customers the same comforting order wherever they might be. The Espresso House app is well-built from the UX point of view. Its features make the customer journey pleasant: showing the nearest coffee shop, showing opening hours, having in-app ordering options, collecting loyalty points, and offering coupons for loyal customers. The app is a great marketing tool for them. It gives customers a better dining experience, for example, skipping the queue by ordering in advance in-app.

1.4.3 Other apps

Other coffee apps currently available on the market focus on brewing methods and beans. Enthusiastic home baristas use these as tools to help them figure out the coffee-to-water ratio, time of brewing for different tools, and other helpful tips to make a good cup of coffee at home.

Some examples are Brew Timer which helps perfect the filter coffee preparation with manual brewing methods, and Coffeely, which enables saving and rating different coffee beans for later reference.

1.5 Out of scope

This thesis will not detail the backend and API development and maintenance. Michel Gillet, a software developer (current position: tech lead at Nokia), developed the backend and API. He used Python and Django to create a JSON format of the data that can be used for the app. He has over 25+ years of experience in developing software and using APIs. Rita Miklán assembled the data for the API. The API endpoint is currently open and can be accessed here: https://api.finegrind.app/cafes/. It provides a JSON of the database, which can be used efficiently throughout the app.

2 Theoretical framework

2.1 Tech stack

The current version of the app was built with JavaScript and React Native. React Native navigation is used for navigating between views. React Native was released by Facebook in 2015, and it is currently maintained by Meta Open Source (formerly known as Facebook). React Native is the mobile version of React, so if someone uses a basic front-end tech stack of HTML, CSS, JavaScript, or Typescript, it is logical to add React to that as a JavaScript framework. (Bahrynovska, 2022)

The users are registered and authenticated with Firebase Authentication, and the user data is stored in the Firebase Realtime database. Context is used for state management, and .env is for storing sensitive information, such as API keys. React Native Maps is currently used to show the cafes on maps based on their coordinates. Expo is used as the dev environment.

Chapter 4, Project tools and libraries, will further detail the tooling, framework, and libraries.

2.2 Comparing tech stacks and methods

The two main operating systems ruling the smartphone market for many years now are Android and iOS. The market shares have been somewhat consistent for the past several years, with Android taking the majority, with a 71.8% market share. (StatCounter, 2023)

With this in mind, several apps still launch for iPhone only, or they launch for Android a lot later. There are many things to consider before deciding whether to do native or cross-platform development, and those are not only based on tech but in the case of more prominent companies. Business- and marketing reasons also play into these choices.

2.2.1 Native development

Native development means implementing code for iOS or Android only. It means using languages and frameworks that work only on one of these operating systems. The main disadvantage here is maintaining two separate codebases. It is more expensive since it needs more developers with different skills, more time for the general project work, and it is more difficult to synchronize the rollout of the new features. However, there are several advantages. For example, a more comprehensive array of phone functionalities can be accessed through native APIs, better overall UX can be achieved, and the code's security is much higher. Since native code works much better

with the environment, the apps created natively can usually be scaled and maintained easier, and they are more intuitive to use. (Jones, 2020)

2.2.2 Cross-platform development

Cross-platform app development means we can use only one codebase and develop for Android and iOS using the same language and framework. This is a convenient choice if the project does not have enough resources or developers to build both native codebases. It has some drawbacks as well, mainly when it comes to UX – even though it is native, but can still be tricky to get certain behaviors right. Not every native device functionality is available, and more external libraries and dependencies are used. This could affect the performance of the code and also cause potential vulnerabilities. Usually, the more dependencies are used, the more difficult it is to handle all the versions of those dependencies, and they can also cause compatibility issues later on. (Kotlin, 2023)

2.2.3 Other solutions

Both native and cross-platform mobile app development has their place in the software world. These are certainly here to stay, but new emerging trends and techniques are also entering the market. We can mention two more types of development: hybrid and no-code/low-code practices. These are both easier than developing a native or cross-platform app. The hybrid solution uses the front-end web development tech stack, meaning HTML, CSS, and JavaScript, and uses specific plugins to envelop the code so it can also run as a mobile app.

The no-code/low-code solution uses visual app-building tools where users can drag and drop, to build their mobile app with no or just minimal coding required. These solutions might be reasonable for piloting solutions, creating POCs, launching an MVP, and testing how the app performs in certain circumstances. These apps, however, are not as customizable as native or cross-platform solutions, and some features might not be available (such as push notifications, GPS, et cetera), so these are mainly good for lightweight mobile apps. (Holen, 2018)

2.3 Mobile app software architecture

Software architecture is, of course, a much bigger domain in itself to be discussed here in depth. However, before jumping into the code, we must mention some architectural considerations.

Software architecture is not so easy to define, but the Software Engineering Institute of the Carnegie Mellon University provides a concise summary:

> "The software architecture of a system represents the design decisions related to overall system structure and behavior. Architecture helps stakeholders understand and analyze how the system will achieve essential qualities such as modifiability, availability, and security." (Institute, Software Engineering, 2023)

This reflects well that software architecture is more than just defining functionalities. It ensures that all the design and tech decisions contribute towards a maintainable, adaptable, and reliable system.

Generally, a well-rounded system has a separation of concerns, so the business logic, the UI, and the data handled are separated. This can be achieved with an MVC architectural pattern. The Model layer handles the data, the View layer renders the UI, and the Controller layer holds the business logic and interactions. (Cocca, 2022)

We will look at how this applies in FineGrind in chapter 3.7, App structure.

2.4 Unit testing

Software testing is the practice of testing the software or some part of it to expose errors in the code. There are different types of testing, depending on what is needed for the task or project at hand. Some are manual tests, meaning that the tester will manually check that all parts of the system are working as expected. Some tests are automated, meaning they are written with the code and run automatically whenever triggered. These tests ensure that there are no bugs in the code and also that the code is held to a high standard. (Geeks, 2023)

A unit test checks the smallest standalone units of the code, such as a function or a UI element. These unit tests are small and independent, ensuring that all parts of the code work as expected. (Pankaj, 2023)

Even though unit testing seems to be a slow process with writing extra code, ultimately, it will speed up the delivery of the product. It is possible to add new features faster because the tests ensure the code does not break. Moreover, if it does, it is much easier to pinpoint where the error occurred. It is, however, essential to keep the tests up to date as well when adding to the codebase so the code keeps its flexibility.

> "Test code is just as important as production code." (Martin, 2009)

The most optimal workflow is test-driven development. In the process of TDD, programmers are encouraged to write the test before they write the code. This process has multiple advantages, such as ensuring the test coverage is broad and covers the entire codebase.

3 Project background

3.1 What is already there

This project was initially started in 2022, with the initial commit to GitHub on 8th February 2022. It started as the coursework for the Mobile Programming course with Juha Hinkula. The MVP was published on Expo on 17th May 2022.

After this, the codebase of the published app has not changed. The only updates were: adding a dev branch and adding a general description to the README.md file. The navigation refactoring was started but was not merged to `main`.

Features implemented in the MVP version:

- Welcome screen: welcoming the users and offering a button for exploring the café list. Also, offering the options to either login or register
- List cafés: list of specialty cafés in the Helsinki-area
- Individual page for cafés displaying details, such as description, photo, opening hours
- Show café on the map – navigate here from the individual café page
- Registration screen (first-time user)
- Login screen (returning user)
- Save café to the favorites list
- Display the favorites list on the user profile screen

The initial app flow diagram can be viewed in Appendix 1.

3.2 Setting up in a new environment

After not starting this project for six months, several issues were at hand. The laptop on which the work is done was changed between June and December. After setting up all the necessary software on the new laptop, such as installing Visual Studio Code and getting Node.js up and running, the code was pulled from GitHub.

Now there were several problems. One was that none of the processes were documented during the initial implementation of the MVP. Any record of which installed libraries were actually used was missing. There was no documentation on how to add the libraries needed. Also, there was no record of how to do possible setups and configurations.

The other major issue was that most libraries were outdated, and some were deprecated. The dependencies were severely broken.

First, Expo was added globally with the `npm install --global expo-cli` command. As a second step, the `npm install` was run in the local repository. This was when the dependency issues started. All the packages were several releases behind. At first, the author attempted to upgrade the Expo SDK, but there were several backward compatibility issues, and the upgrades failed.

After trying several different approaches, the decision was that the best would be to entirely delete the following files from the local repo: package.json, package-lock.json, babel.config.js, the .env, and the .expo-shared folder. After this, the author initiated the project again with the `npx create-expo-app` command, which recreated all these folders (except the expo-shared folder and the .env) and reinstalled all the basic dependencies with the correct versions.

After this, the author attempted to start the project with the `npx expo start` command. It still did not work since many other libraries were still missing, but now the log gave the information on what other libraries are needed for the project to run. Following the error messages in the log, figuring out what was still needed was easier.

Firebase, dotenv, react-navigation, and react-native-maps were added with the npm package manager. Chapter 4.2 Libraries details the exact commands, versions, and configurations.

Since the published version was also running on the deprecated Expo SDK and the expo-shared folder was deleted, the MVP must be republished at some point.

This process also gave an excellent example of why detailed documentation is essential for every software project.

3.3 Planned features

Of course, the list of possible features to implement can be endless. However, since this project has a relatively short period and limited resources (just one developer), the focus will be on features that make the app more usable and more helpful to the audience. These features are as follows:

- Usability features, such as alerts and button feedback. The user must always be aware of what is happening in the app. When a button is pressed, it should have visual clues. Also, the text fields should have some error avoidance/error handling mechanism.
- Plan the route to the café from the user's location. The user's location should be visible in a map view, and the user should be able to find a route to the nearest (or any) café.
- Search and filtering of cafés based on location, opening hours, menu items, et cetera.

The precise list of features will be defined after user surveys and usability tests have been conducted, and the author gets more profound insight into what the users would benefit from the most.

3.4 Project management methods and tools

The project follows a Kanban-style process. A Trello board is used to create a backlog of tasks and to choose what task will be worked on next. Since this is a one-person project, keeping it as simple as possible and having only one simple Trello board for following the development seemed fitting.

Also in use is Toggl Track, which is a time-tracking web app. This enables the user to track time spent on different tasks and projects.

Calendar blocking is also used to make sure enough time is reserved throughout the weeks for completing this thesis.

All the files are held in a Microsoft OneDrive folder to ensure everything is backed up properly and that every document is saved automatically.

3.5 Software development methodology

For a small, one-person project like this, thinking about software development methodology might be odd or redundant when there is no team or other stakeholders. However, it is still essential to think about some basics, such as the way of working or the definition of done. This helps to structure the work and makes the code changes followable since it is not just randomly decided what to do next. Although with only one person, sprints make no sense, but certain elements, such as backlog grooming, planning, and executing changes in smaller increments, are good to keep. The definition of done is also not as strongly defined as it would be in a team, but the importance of working code and clean coding practices, such as DRY coding, is upheld.

3.6 FineGrind tech stack decision

The decision to use React Native for this project was made based on several factors. One significant decisive factor was that the author had already used JavaSctipt, React, and React Native in previous front-end projects, so she was familiar with React and JavaScript. Stepping toward React Native was a natural and logical step in the process.

Another consideration was that this app could exist on Android and iOS devices without developing different codebases in different languages. However, the app might be migrated to Typescript instead of JavaScript for more consistent and concise coding.

Also, the Expo SDK API allows access to a wide array of device features and fully supports React Native. This brings back the advantage of native development to this cross-platform solution. Overall, maintaining one codebase and using tools and languages already familiar was the most reasonable decision for this thesis work.

3.7 App structure

Regarding the app's folder structure, the idea was to have a clear overview of what is happening inside the app. The data is stored and handled within the app with Context, so once there is any action from the user's side (login, save to favs), that data is stored in one place in the app and is provided to the rest from there. This makes sure that there are no unnecessary calls to the Firebase database. Since this is a simple mobile app, there is not much need for an elaborate architectural pattern. However, we can see a certain separation of concerns happening.

The user name, uid, and other relevant information regarding the logged-in user are all saved in Context, which provides them to the other app components.

The call to the coffee API is also extracted to a separate hook. In that sense, the UI is separated from the business logic handling the data.

The Firebase helper functions are also removed from the UI, so all API calls happen removed from the view. The user interacts with the UI via the form fields and buttons, controlling what data is sent to the database and what is retrieved.

Even though this application is relatively simple and does not need to implement an architectural pattern, strictly speaking, it is good to have a higher-level overview of how the data is handled throughout the app and how the user interacts with it. This gives the code a solid foundation that can be replicated later on in other projects if needed. Other benefits are that the code is more scalable, and it is easier to implement changes if needed. Good architecture also has other benefits for larger-scale projects, such as cost savings and reduced development time. (Novoseltseva, 2021)

The navigation has a separate folder as well. The App.js is the entry point that calls the Context and the navigation, and the navigation accesses all the Screens. Some hooks and components have been extracted to make the code more modular and reusable. The screens can render components as needed, and the same code does not need to be rewritten many times. The styles come from a global stylesheet to avoid inline and internal styling. These styles are also reusable as much as possible. If any changes are needed, the change has to be done in only one place. The app follows a logical flow, so the user knows how to interact with the app intuitively. The file structure is demonstrated in Figure 1.

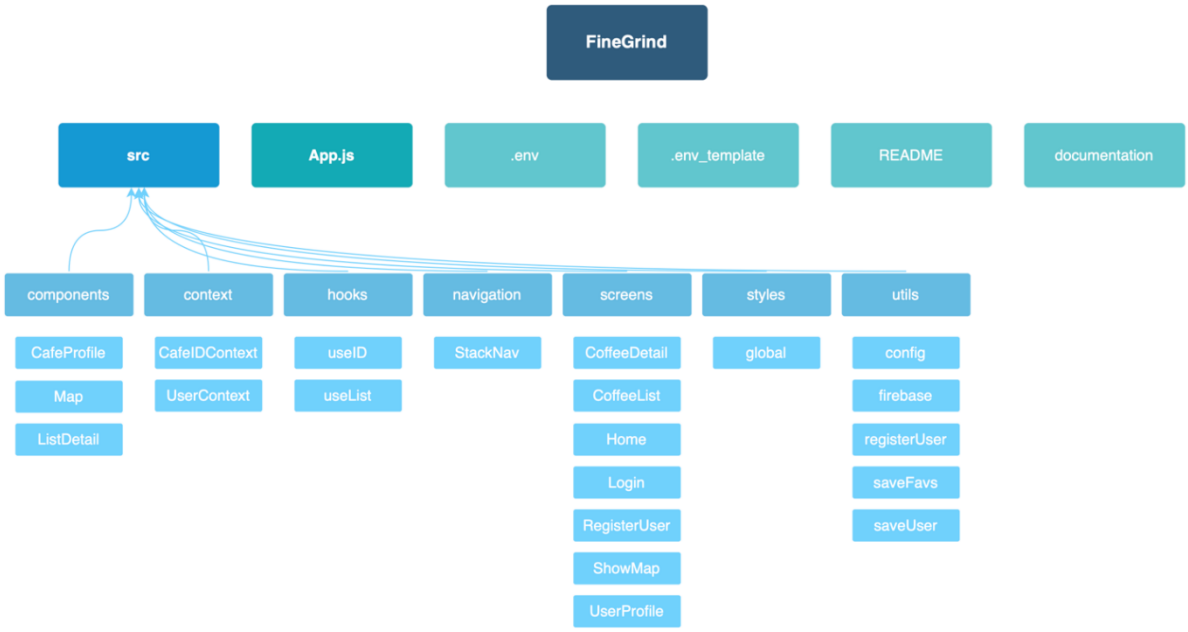


Figure 1. FineGrind file structure (created by the author)

3.8 Implementation plan

For the first phase of the work, the following tasks are planned:

- Set up the environment on the current MacBook: in-between now and when this project was initially started, the laptop in use has changed, so the repository will have to be cloned and set up on the new laptop
- Start a good and thorough documentation: documentation is an integral part of every project. This ensures that future development will be easier since all the knowledge regarding this project will be in the same place. If any future developer wants to continue

the work, they will be able to see the tools and resources and the project's current state, thanks to the documentation.

- Define a target user group and how to reach them: a target group is essential for surveys and user testing. Also, we need to know our target group to streamline the development to serve them better.
- Survey members of the user group about what features they would find helpful: a survey will help understand the group's habits when it comes to smartphone use and coffee consumption
- Plan the overall architecture of the app: this will help design a maintainable and concise system.
- Plan the tasks for the backlog: refining the tasks will help to build the workflow.

Second phase:

- Start user interviews and user testing of the MVP: usability tests will help see the app's shortcomings and refine the backlog.
- Refactor existing code: see if there are any places where improvements can be made.
- Migrate to Typescript – this might not be a feasible task to carry out in this timeframe
- Start implementing the planned features
- Implement testing
- User validation upon newer releases
- Update documentation

Third phase:

- Write about the project work and learnings
- Possibly release the app to the public

4 Project tools and libraries

4.1 Tools

This project was initially started on a Windows environment, but now further development happens on a MacOS. Since all the tools used are available for both Windows and MacOS, the development can be done either on Windows or MacOS. The project uses Expo, so that means we do not need to have any emulators installed on the laptop itself.

4.1.1 Visual Studio Code

This project uses the latest version of VS Code, the 1.773 Universal version. This is a simple and free IDE that fits the purpose well. For formatting the code and keeping it consistent, the Prettier Code Formatter extension is recommended. Prettier formats the code on save and makes sure the code looks consistent. This is also important for the readability of the code.

4.1.2 Terminal

A terminal is a fundamental tool for this (or any) project to run the package manager, add libraries, and manage the git repository. The terminal is included in all operating systems by default. However, on the MacOS environment, the author uses iTerm2 with zsh and Oh My Zsh packages for additional features, such as multiple tabs in the terminal window.

4.1.3 npm

The project uses the npm package manager. The current version used is 9.5.0. The npm is part of the node.js package, so even though this project does not use Node.js code, we still have to install it to access npm. Another alternative would be to use yarn for package management. It is important not to mix these two in the same project because it can lead to errors or warnings because these handle the lock files and dependencies differently.

4.1.4 GitHub

GitHub is used for version control. The two branches are `main` and `dev`. The `main` branch is protected, and the code can only be merged with a pull request to avoid accidentally pushing code to the `main` branch. The `dev` branch serves the development environment. The git workflow is as follows:

- All new edits and changes are done on the `dev` branch. Since only one developer is working on the project, there is no need to check out feature branches from the dev branch. However, the author decided to use feature branches, making the changes easy to track. This also allows following the backlog more clearly, since feature branches give a frame for not jumping from one task to the next one randomly.

- When the code reaches a stable state or when a new feature is added, it can be merged into `main`

- After merging to `main`, the code must be pulled to the local environment, and the `dev` branch is rebased so the `dev` and `main` point to the same commit.

Another branch in existence is the `startingpoint`, a read-only branch that keeps the state of the code at the start of the thesis project. The point of this is to have a reference point for how the code has developed during the project. Snapshots of the GitHub workflow examples can be found in Appendix 2.

4.1.5 Expo

This app is created and bundled with Expo. It uses the current latest version of Expo SDK 48.

Downloading also the Expo Go mobile app is recommended for live testing the project on an actual device, without the need for downloading or installing anything else.

Expo also has packages to allow access to the device's functionalities.

Quick start with Expo:

```
npm install --global expo-cli
npx create-expo-app my-app
```

4.2 Libraries

4.2.1 React and React Native

The Expo command will set up the project already installing these two libraries with the latest stable version, which is compatible with the Expo SDK. Currently, the Expo SDK 48 supports React 18.2.0 and React Native 0.71.6

These libraries are essential for building anything and give the basics of the whole app.

4.2.2 Firebase

The project uses Firebase for authenticating and storing user data. Firebase is a platform provided by Google, which allows app developers, among others, to add an easy-to-setup, cloud-based backend to their app. It has different services, such as authentication and a document-based database. To add it to the project, run these commands:

```
npm install Firebase
npm install --save @react-native-firebase/
```

This project uses the Firebase Realtime Database, which might not be the best fit as the project evolves. Cloud Firestore is a newer version of this database with better functionalities and documentation. However, for now, the Realtime Database is enough for the current needs.

4.2.3 dotenv

The dotenv library loads environment variables from a .env file from the root folder. It is a safe way to store secrets, such as API keys or links, that end users or others should not access.

To set it up, run this command in a terminal.

```
npm install -D react-native-dotenv
```

After adding the dotenv library, we need to set it up like so:

Add this snippet in the babel.config.js file under the presets line:

```
plugins: [
["module:react-native-dotenv", {
"moduleName": "@env",
"path": ".env"
}],
]
```

Add a .env file in the root directory, where all the environment variables are added, for example:

```
API_URL = http://apiurl.com
```

Also, add .env to the .gitignore file so it does not get pushed into the GitHub repository.

Create a .env_template file as well, where all the env variables are added (without the actual keys or links) as a future reference if the .env file needs to be recreated.

Then create a Config.js file in the utils folder for creating an import and export

Config.js:

```
import {API_URL} from "@env";
export default {API_URL};
```

Now the API_URL variable can be used in the codebase everywhere where the API URL needs to be accessed. The only import needed in the file where it is used is:

```
import config from "./config";
```

The Firebase credentials are also added to the .env file.

4.2.4 React Native Navigation

The app uses React native navigation, more precisely, both stack and tab navigation. For running the navigation as it is described in Chapter 6.2.1. Navigation, these three libraries are needed:

- react-navigation/bottom-tabs
- react-navigation/native
- react-navigation/native-stack

For this, run these commands in the terminal:

```
npm install @react-navigation/native-stack
npm install @react-navigation/native
npm install @react-navigation/bottom-tabs
```

4.2.5 React Native Maps

This is the MapView component for iOS and Android on React Native. This library allows us to show locations on Google Maps, mark a location with a pin, scroll and zoom the map, and track location. To install this library, run the following command:

```
npm install react-native-maps
```

4.2.6 Expo location

The Expo location library is used to access the device's location. When launching the app, the user is prompted to permit the app to see the device's location. When the user agrees, a map will render to show the user's location on the map. We install this library using the following command:

```
npx expo install expo-location
```

5 Requirements

5.1 Defining target user group

Almost every adult today consumes coffee in one way or another, and Finland ranks as the top coffee-consuming country in the world per capita. (MacDonnell, 2023) Coffee, as well as any other consumer good, has different categories and varying availability. Specialty coffee is at the high end of this scale. This app is for coffee lovers, who are open to trying new places, want to explore more about coffee, and tech-savvy enough to use a mobile app. Therefore, the target group is people who love coffee, have smartphones and are open to exploring new cafés and learning about specialty coffee. After the initial survey, some users will be selected based on their answers to some questions for further user testing.

5.1.1 Online survey

The online survey is the first step for gathering some basic information about the general habits of visiting coffee shops. The complete set of questions can be found in Appendix 3.

The survey was sent out to a general audience on 31st December 2022 via an Instagram story. On 2nd January, it was also shared on LinkedIn, the author's personal Facebook profile, and one of the general Slack channels at the author's workplace.

The survey was closed on 6th January 2023. It accumulated 84 answers. This already shows that there is a general interest in the topic.

Of the 84 replies, 67 came from Finland, six from Hungary, five from Germany, three from Sweden, and one from France, Poland, and the United States, respectively. Of the answers from Finland, 54 came from the Capital Region (Helsinki, Espoo, Vantaa, Kauniainen).

Most of the answers came from people between 25 – 45 years old, giving 80.9% of the replies.

86.9 % of responders indicated drinking coffee daily, 46.4 % said they visit coffee shops a few times per month, and 8.3% a few times per week. Based on this, about 50% are regulars when visiting coffee shops. Figure 2. represents this as a pie chart.

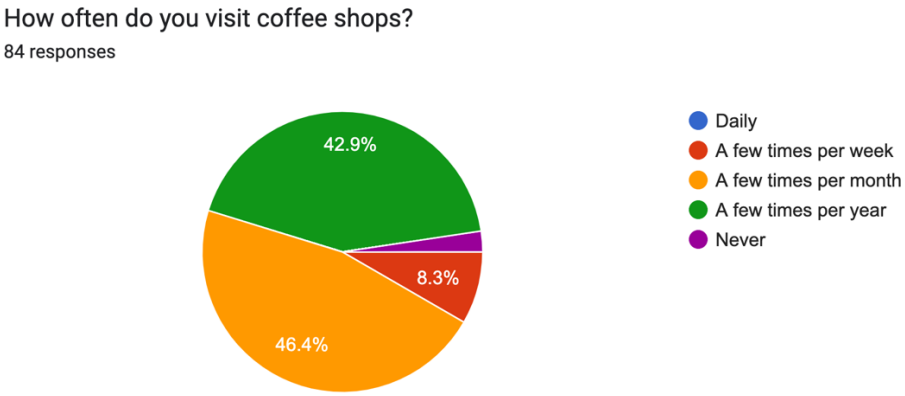


Figure 2. How often do you visit coffee shops – graph (from user survey results)

98.8 % own a smartphone (only one responder said no), and the majority, 79.5%, said they are at an advanced user level.

The next question was, "How familiar are you with specialty coffee?" and it was interesting to see how people responded.

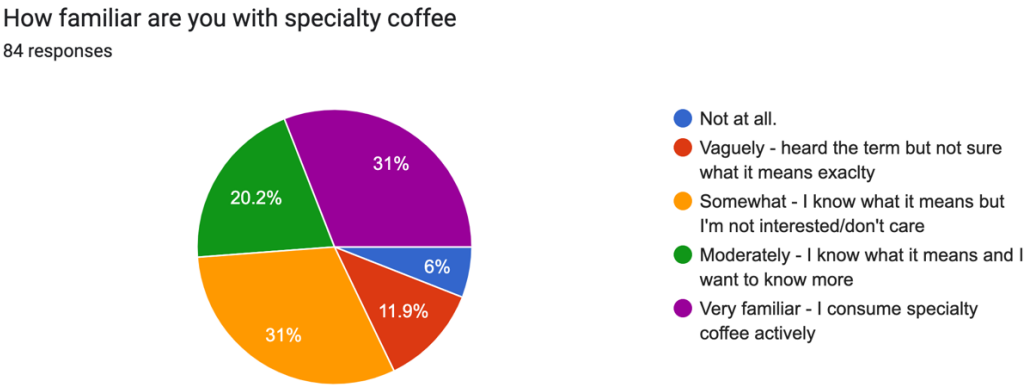


Figure 3. How familiar are you with specialty coffee - graph (from user survey results)

Only 17.9 % of people replied that they were unfamiliar with or not interested. However, on the pie chart in Figure 3., we can observe that 82% of the responders were at least somewhat familiar with and definitely interested. This shows that even though not everyone knows what specialty coffee is, there is a definite interest and curiosity in discovering more about it. This shows that an app like this certainly has a right to exist.

Even though specialty coffee has existed since the 80s', there is still much newness about it. If we look at the wine industry, for example, we see that there are different wineries, years, flavors, grape varieties, et cetera. The same exists in the coffee industry, but many people still have not discovered it in depth. So, if we allow people to find out more with a simple, easy-to-use app, they may explore more. Also, this would help the specialty cafes to bring in new customers.

58.3% of responders indicated that they use some sort of app to find out information about places, and the most popular apps named were Google and Google Maps. Some other apps mentioned were Edenred (used by companies to handle the lunch benefit) and some other reward apps. Instagram and Facebook are also used to find information about places, such as location, opening hours, and menu items.

At least one responder wrote that they used the Espresso House app, and the loyalty program offered through the app was an essential factor in their decision to visit Espresso House.

Only one responder wrote that they use the European Coffee Trip app.

From this simple survey, it is already clear that there is curiosity towards exploring coffee more and finding new places. In Figure 4, we can see that most responders would like to explore more.

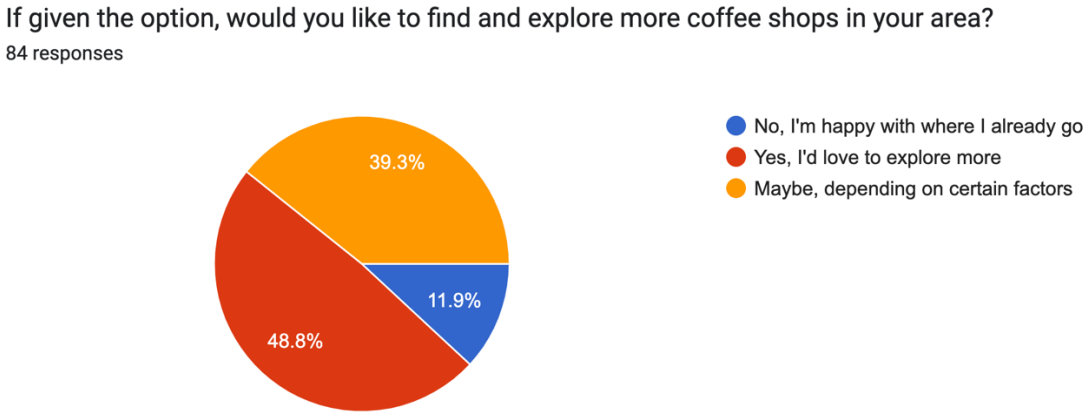


Figure 4. Willingness to explore more coffee shops - graph (from user survey results)

Another interesting finding is that several responders wrote that their priorities differ during average days and vacations. During regular days they might go for something fast, takeaway, which is on their way, but when they travel or are on vacation, they prefer to sit and dine in and chat with friends. This is also an important point to keep in mind when thinking about the user experience.

5.2 Usability testing of the MVP

While a survey is an excellent tool to test the idea and check if the product is feasible with the target user group, this is just the first point of contact with the users. Usability is critical, especially in 2023, where many apps with similar functionalities compete for the same users' attention. For successful software development, we must always consider the users at every step. While the wireframe was not tested with users in this project (typically, that would already be tested), usability tests were conducted on the MVP to gather more information.

These tests were done face-to-face, so the interviewer could observe how the users executed the tests given to them. Afterward, the interviewer asked a few general questions to gather more insights. These tests revealed already a few issues that need to be improved.

5.2.1 Test cases

1. Find We Got This café in the café list
    - Click café list
    - Scroll down until We Got This in the list
2. View Café on the map
    - Click café list
    - scroll down
    - tap on café to open the café profile
    - click view on the map
3. Return to the home screen
4. Register user profile
5. Add café to favorites
6. Check user profile
7. Log out / Log in again

5.2.2 Usability test results

The MVP was tested with five users. The number of users to test with was decided based on the recommendation of Nielsen, who pointed out in his article that five users are giving sufficient data and will most likely uncover all the shortcomings of the UX. (Nielsen, 2000)

We can see the results of these tests at a glance in Table 1. Usability test results.

Table 1. Usability test results

	Find We got this in the café list	**View café on the map**	**Return to the home screen**	**Register user profile**	**Add café to favs**	**Check user profile**	**Log out / Log in again**
USER 1	PASS	PASS	PASS	PASS WITH MINOR ISSUES	PASS	PASS	PASS
USER 2	PASS	PASS	PASS	PASS	PASS	PASS	PASS WITH MINOR ISSUES
USER 3	PASS	PASS	PASS	PASS	PASS WITH MINOR ISSUES	PASS	PASS
USER 4	PASS	PASS	PASS	PASS	PASS	PASS	PASS
USER 5	PASS	PASS	PASS	PASS WITH MINOR ISSUES	PASS WITH MINOR ISSUES	PASS	PASS

Most test cases passed, and there were only a few test cases where minor issues were observed. However, these issues gave insight into defining what features could be improved and implemented.

Some of these issues are tied to accessibility and design, and some to the app flow and user behavior. Based on these findings, the author refined the backlog and better understood how the app should flow for an optimal user experience.

Usability improvements are needed in the following areas, based on the 10 Usability Heuristics for User Interface Design (Nielsen, 1994)

- Visibility of system status
    - no feedback on the button press
- User control and freedom
    - return to home, remove café from favorites
- Error prevention
    - password feedback
- Efficiency of use
    - Favorite list icon in a visible place, map on the front page

6 Implementation

6.1 Planning features and defining backlog

Based on the feedback from the usability tests and also from the survey, the following items are high-priority in the backlog:

- The map should already be visible on the main page of the app
- Feedback on button pushes, and other user actions are needed
- Home button to return to the main page quickly with one click
- Favorites should be accessed from a bottom tab
- Filtering and search options on the café list
- Log out button should be on the user profile and not on the main page (the logout is not that important function when it is a single-user device, such as a mobile phone)
- Café profiles should have menu items added
- The app should ask for the user's location and show cafes nearby

A snapshot image of the project Trello board with the refined backlog can be found in Appendix 4.

Based on these, we can observe that some areas need a major overhaul in the app. One of these is the navigation, the other is the home screen itself, and the favorites are also brought up during the usability test interviews. We must also take care of the heuristic aspects and provide user feedback on button presses and error messages upon incorrect inputs.

These are all related to each other, and it is tempting to tackle them head-on. However, the way of working will be kept to follow the basic best practices outlined in Chapter 3.5, Software development methodology. The changes are followable, and the development happens in small increments rather than big random chunks.

The incremental approach also helps to safeguard the work that has already been done. If some changes need to be reversed or entirely break the code, returning and finding the last stable state is much easier. Also, working with feature branches allows us to try out different things without risking an irreversible mistake.

Another advantage is that we can develop a feature bit by bit. It is less daunting to implement a feature if it is broken down into smaller tasks and done step-by-step. This also gives a chance to test how the new feature behaves and in what way it should be developed further. Figure 5. shows us how the app looks before any refactoring and new features.

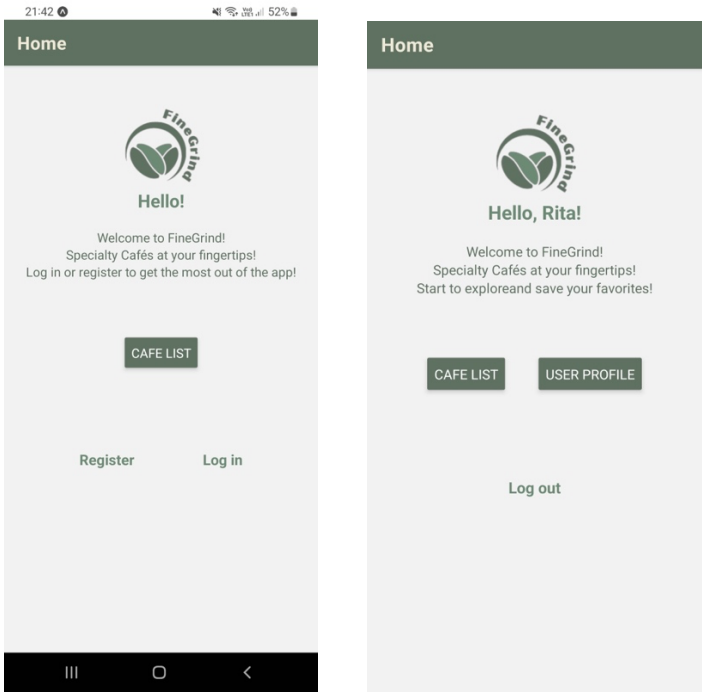


Figure 5. Starting point before refactoring

6.2 Refactoring and adding new features

6.2.1 Navigation

Redesigning the navigation was the first significant step in the workflow. The MVP was initially designed with stack navigation, so the screens could have buttons to navigate to other screens. Since usability testing results showed a need for a Home button and a Favorites button, it was logical to add a bottom tab navigation containing these. These would allow the user to return to Home with one click instead of navigating back through multiple screens. It would also be logical to place the Profile and Login buttons there. These could be rendered conditionally based on the signed-in state of the user.

Although this is an excellent solution from the UX point of view, the author still wanted to keep the stack navigation as well, so buttons could also be implemented on the screens as needed.

This needed a rather creative solution of nesting the stack navigation inside the tab navigation. For accessing the state of whether the user is logged in, Context is used here as well. If the user is not logged in, then the tab shows Home and Log in. The tab shows Home, Favorites, and Profile when the user is logged in.

The Logout button was moved to the profile page after considering the usability testing findings.

For each bottom tab nav icon, there is a corresponding navigation stack. These include all the screens that can be reached, starting from the screen that is rendered by the bottom tab nav icon.

```
function HomeStack() {
  return (
    <Stack.Navigator
      screenOptions={{
        headerStyle: {
          backgroundColor: color.darkGreen,
        },
        headerTintColor: color.light,
        headerTitleStyle: {
          fontWeight: "bold",
        },
      }}
    >
      <Stack.Screen name="Home" component={Home} />
      <Stack.Screen name="CoffeeListScreen" component={CoffeeListScreen} />
      <Stack.Screen name="ShowMap" component={ShowMap} />
      <Stack.Screen name="CoffeeDetailScreen" component={CoffeeDetailScreen} />
      <Stack.Screen name="RegisterUserScreen" component={RegisterUserScreen} />
      <Stack.Screen name="LoginScreen" component={LoginScreen} />
      <Stack.Screen name="UserProfile" component={UserProfile} />
    </Stack.Navigator>
  );
}

function RegisterStackNavigator() {
  return (
    <Stack.Navigator
      screenOptions={{
        headerStyle: {
          backgroundColor: color.darkGreen,
        },
        headerTintColor: color.light,
        headerTitleStyle: {
          fontWeight: "bold",
        },
      }}
    >
      <Stack.Screen name="LoginScreen" component={LoginScreen} />
      <Stack.Screen name="RegisterUserScreen" component={RegisterUserScreen} />
      <Stack.Screen name="UserProfile" component={UserProfile} />
    </Stack.Navigator>
  );
}
```

Then these stacks are exported and linked to the bottom tab nav icons instead of individual screens.

```
<Tab.Navigator
   initialRouteName="Home"
   screenOptions={({ route }) => ({
     tabBarIcon: ({ focused, color, size }) => {
       let iconName;
       let rn = route.name;
       if (rn === "Home") {
         iconName = focused ? "home": "home-outline";
       } else if (rn === "Log In") {
         iconName = focused ? "login": "log-in-outline";
       }
       return <Ionicons name={iconName} color={color} size={size} />;
     },
     tabBarActiveTintColor: color.darkGreen,
     tabBarInactiveTintColor: color.lightGreen,
     tabBarShowLabel: false,
     headerShown: false,
   })}
 >
   <Tab.Screen name="Home" component={HomeStack} />
   <Tab.Screen name="Log In" component={RegisterStackNavigator} />
 </Tab.Navigator>
```

The App.js accesses the Tab navigation, and the Tab nav accesses the Stack navigation.

```
export default function App() {
  return (
    <UserProvider>
      <SafeAreaView style={{ flex: 1, backgroundColor: color.white }}>
        <NavigationContainer>
          <TabNavigation />
        </NavigationContainer>
      </SafeAreaView>
    </UserProvider>
  );
}
```

The app now threw a warning on starting up, stating that there are multiple nested nav elements called "Home," which might result in some unexpected behavior, so the Home view was renamed to HomeScreen, to make the distinction better.

Within the stack navigation, the stack screens were also updated by a title option, so the titles are not the default screen names (such as CoffeeListScreen) but more natural titles, such as *List of Cafes*.

6.2.2 Map on the Home Screen

One of the main points of the feedback was that the Home screen is almost pointless because nothing is happening there. The map should have a central role in the app to give the users maximal usefulness and a good user experience. Although a map component is already used to render the cafes' locations on their individual map pages, a separate map component is now rendered on the Home screen.

For this, getting the user's permission to use the device's location was imperative. The expo-location Expo SDK is used to obtain the user's permission and get the coordinates.

Triggering the function when the screen is loading, it will first prompt the user to give permission. Once permission is granted, the location will be provided in a JSON format. This can be saved in a variable and used later to display the user's location on the map. The `useEffect` hook is used, so the function only triggers once when the screen is rendered for the first time.

The map is rendered by the react-native-map library, which is a community-run project. The React Native Community consists of volunteers and companies who depend on these libraries, so these are usually well-maintained and safe to implement in projects. These libraries can be found in the npm registry. (React Native, 2023). Among the React Native Maps contributors, we can find, for example, AirBnb. It is, of course, always important to ensure that external libraries come from reliable sources and have long-term support so the project does not become vulnerable or break entirely when the library stops working with other dependencies.

The `MapView` component renders the map, and it has a `provider` prop that defines if the map will be rendered as a Google Maps or Apple map. If we choose Google Maps, the map will be provided by Google Maps.

The React Native Map library provides many different component APIs to implement the map with different options. One of them is the `Marker` component, which can be used to pinpoint a location on the map, for example, the user's location, based on the coordinates supplied by the device.

However, after this, the map still only shows the user's location, which is not very useful on its own. Most likely, the user will be aware of their location and wants to see where the nearest café is. For this, we need to add more markers with the coordinates of the cafés. For this, the `useList` hook is used. The `useList` hook is a custom hook that is extracted to fetch and list all the cafés from the FineGrind API. Extracting hook logic allows us to have a reusable function. Since we need to have the coffee list in multiple places of the codebase, extracting the function that fetches the data from

the API makes sense. Now we do not repeat the code (DRY coding principle); we removed the business logic from the UI part and only have one API call.

This hook also returns a JSON with all the data for the cafés. To extract the coordinates and render the markers, we use the `map()` method. It is imperative also to pass a `key` prop to the `Marker`, as well as the coordinates. Without the `key` prop, multiple markers will not render in the `MapView` component.

Now we have all the cafés on the map marked by the red pin icon. Also, the user location is marked with the same red pin. We also have all the other labels, such as street names, area names, businesses, landmarks, and other POIs. This makes the map look rather cluttered.

As a first step, the author changed the red pins for the cafés. This can be done with the `image` prop of the `Marker` component. We need to add the image we want to the assets folder in the project's root for using it in the Marker. Here a png of a coffee cup was used as the icon.

The next step was to style the map itself. This was done by using an online styling wizard, where the sizes of labels, frequency of landmarks, and some other options can be selected. It generates a JSON object that can be passed in the `customMapStyle` prop of the `MapView` component. Since this file was over 40 lines long, a separate file was created for the custom map styles within the styles folder. This also makes it reusable in other parts of the code.

Another component of the React Native Maps library is the `Callout`. This adds a tooltip to the `Marker`. This also can be fully customized to fit the app's design and needs. A `Callout` was added to all the pins marking cafés in the app. They show the name of the café when the user taps on them. On the Home screen `MapView`, the `Callout` tooltips link to the individual café pages, and navigate there on press.

6.2.3 Button feedback and delete favorites

One major criticism the MVP got during the user tests was that the "Add to favorites" button did not have any immediate feedback to the user. Also, there was no way to remove a café from the favorites.

This is a major flaw in the UX and also a core functionality to have. To follow which café was added to the favs by the user, the id of the café is also saved in Context. If the café's id can be found in the Context fav list, then the button will be a "remove" button. Otherwise, it will be an "add" button. This is done by conditional rendering of the button.

Now the button immediately changes to "remove from favorites" when the button is pressed, also changing to a different color and giving visual feedback to the user, besides giving the option also to remove it and revert the change.

In the business logic, a helper function was added to remove the favs from the database to the `src/utils/firebaseFavs.js` file. Initially, this file was called `firebaseSaveFavs`, but it was logical to move the `removeFavs` function here as well and rename the file. This way, both helper functions that deal with the favorites are in the same file.

The Firebase Realtime Database has a `remove()` method, and the author initially used that to implement the functionality. However, that method did not work as intended and as it was demonstrated in the documentation. Instead of removing just one element from the user's fav list, it removed the entire `users` document.

Reading the documentation on the Firebase website, the suggested workaround was using the `update()` function instead and updating the value in the database to NULL. This solution was implemented and it was done by initializing a JavaScript dictionary as a new variable. As the key, a template literal is used, which points to the id of the café in the favorite list of the user:

```
`users/${uid}/favs/${favID}`
```

As the value, `null` is given. Then in the `update()` function, this variable is the passed argument.

As mentioned earlier, Firebase Realtime Database might not be the optimal solution, and later switching to Firebase Cloud Firestore might be considered.

6.2.4 Favorites list

Originally the favorites were listed on the profile page, which was available from the main page through stack navigation once the user logged in. A new menu icon for favorites was added when redesigning the navigation and adding the bottom tab. A new screen for favorites was created in the `src/screens` folder.

Since most of the code already existed in the `UserProfileScreen.js`, it was easy to move that to the `FavoriteScreen.js`. The code first creates a snapshot from the database and checks if there are favorites. Then it loops through the café list, and if the café id matches the ids from the snapshot data, it pushes the café to a new array. Then the new array is rendered by a `FlatList` to display the list of cafes.

Now the favorite list is available from anywhere in the app with one button in the bottom tab that is represented by a heart icon. When a café is in the favorite list, the button displays the "Remove from favorites" option.

6.2.5 Styling

The original app folder structure (Figure 1.) shows that the styles were already moved to their separate file. This decision was made to avoid inline and internal styling and repeating the same styles throughout the app. It also helps to keep the different parts of the stylesheet organized; for example, the styles for containers, texts, images, and so on all have their section within the file. All the different styles have a descriptive name, for example, headerText, mainText, plainText, and errorText. This helps differentiate immediately within the different styles and helps to find a specific one when an update is necessary.

The colors were also added as variables in a separate color.js file, so they can be easily updated whenever needed by changing the hex codes only in one place. The color has already been updated from a mostly green-dominated color palette to a mostly blue-dominated one. The color palette was chosen from the ColorHunt website and can be accessed here: https://colorhunt.co/palette/f5efe6e8dfcaaebdca7895b2

The icons for the tab navigation are from IonIcons. Currently, no external UI libraries are used in the project because that would require more extensive research and possibly collaboration with a UI/UX designer.

Some updates in the UI were implemented during the development phase, although these reflect an overall UX and accessibility point-of-view rather than strictly a better design. Some text sizes were increased, and all the buttons were changed to a darker shade of blue so they stand out more. Some texts were given more padding and margin to have better readability and make the screen appear less cluttered.

6.2.6 Form error handling and validation

Although there are some external libraries for form validation and error handling, in this project now, the built-in Firebase features are used. The Firebase Authentication is an out-of-the-box solution for registering and logging in and out users with their email addresses and password. This has been implemented in the app already at the MVP stage.

The Firebase Authentication already checks that the email has the correct format and that the password is at least six characters long. When the user types in their credentials and presses the

register/login button, it triggers a function with a `.try() .catch()` syntax. If there is an error in the user input, Firebase will return an error message that we can catch and either save to a `useState` or log in to the console.

To keep this simple, first, a state was created for the error message set to null. Then a simple `if` check was added to the code. If the error state is not null, a message will be displayed in the UI stating that some input was incorrect.

A simple message is displayed for the password field until the password reaches six characters in length. This way, the user can fill all the input fields correctly, or they can be aware if a mistake was made. Figure 6. shows some screengrabs of the updated look and feel of the app with some of the new features implemented, and Appendix 5. gives an overall look at the app's UI as it is after all the new features discussed here are added.

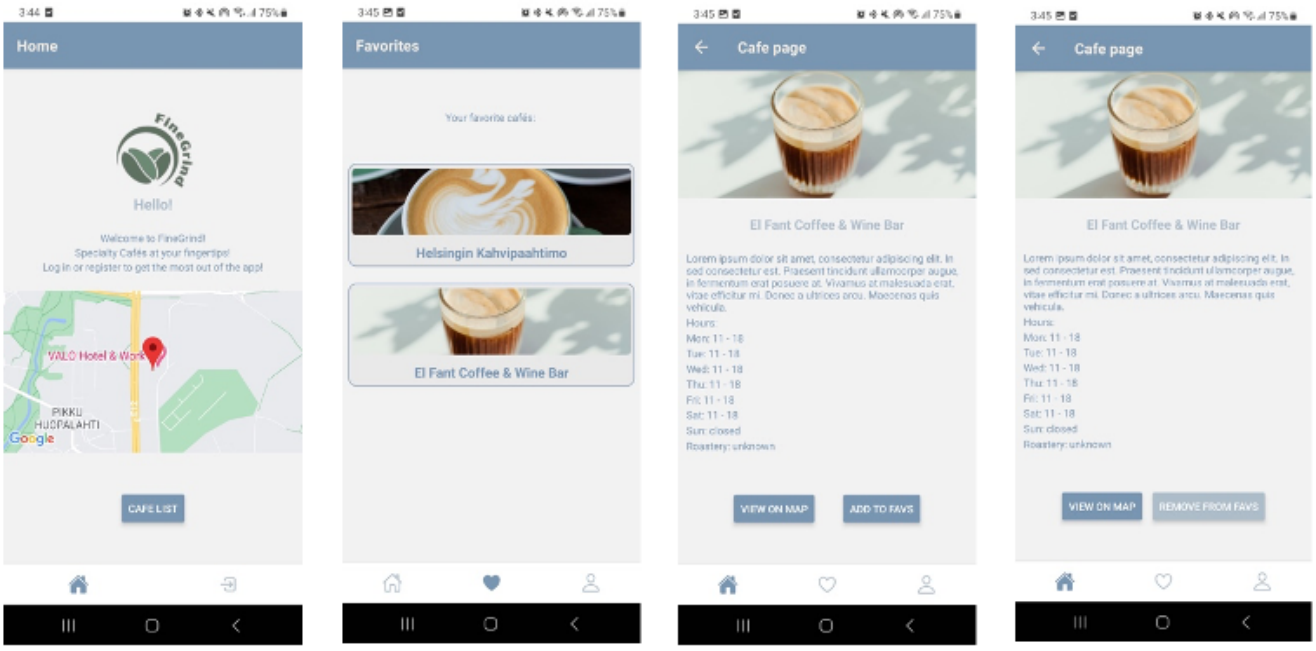


Figure 6. Redesigned app - work in progress (screenshots)

7 Discussion

7.1 Results achieved

While developing this app, it became clear that one developer is insufficient as the sole resource for this type of work. While discussing this app with peers, one developer colleague pointed out that they work on a similar application – except they have been working on it for over a decade with a team including several people. Compared to that, having only one developer who implements an application within a few months sounds impossible.

During the project, it became apparent that this app might never be fully ready. This is one of the beauties of the software development world: there is always something to improve, and there is always something to learn. Continuous improvement is one of the requirements of a software developer.

Even though the app was not finished during the timeline of the thesis work, the project resulted in significant improvement of the app. This was possible by conducting the user survey and the usability tests. Proper planning and analyzing the problem we strive to solve is essential to the systems development life cycle. We cannot develop efficient, functional, usable, and maintainable software without giving it the proper foundation. During this project, the surveys and the usability tests gave the needed insight to know where to take the development.

This way, it was possible to focus on the features that the users found most valuable and what needed the most improvements from the usability perspective. One of those was the navigation, which is now updated to fit the user's needs better. Now, with the Stack navigation integrated into the Tab navigation, the user gets the best of both worlds, and it is easy and intuitive to get around in the application.

The app's usability also improved from other angles, such as button press feedback to the user, feedback about form input errors, better use of text sizes and colors, and the ability to reverse actions (add and remove, log in and log out). These prevent the users from making mistakes, and even if they make mistakes, there is guidance on how to correct or reverse them quickly.

Of course, this does not mean that there are no more improvements possible on the app. The backlog was quite extensive, and it was impossible to tackle every single item during the allocated timeframe.

Some other issues could not be resolved within the scope of this project. For example, the warning that comes up when starting the project says that the "Async Storage has been extracted from

react-native core." After looking at some forums, such as Stack Overflow, and reading some articles about it, it became apparent that this is part of the Firebase library that still uses the deprecated react-native package. This is one of the challenges of cross-platform app development. Most of the time, when one package gets an update, there might be other packages that depend on that, which are not yet compatible with the updated package. This compatibility issue can represent warnings at first, and the project might still run, but it is advised to avoid these dependency issues. They can quickly get out of hand and cascade into more significant failure. When this happens in a production environment, the app stops working.

This can be avoided with thorough documentation, several different ways of testing, and of course, active maintenance of the software. This does not only mean writing good code but also removing unused code and ensuring that all the dependencies are up-to-date and compatible. When updating a package to the latest version, it is always a good idea to make sure that it supports the core dependencies of the app.

Before conducting any user research, these were the planned updates in the app:

- Usability features such as button feedback and error handling
- Plan route to cafes
- The user's location is visible on the map
- Search and filter cafes based on location, opening hours, menu items, et cetera.

This was the starting point, and this list changed somewhat after the user interviews and during the development phase. This is typical during iterative development. There is always an opportunity to revise, rethink, and change direction if needed. In the end, these were the items completed during the implementation phase:

- Redesign navigation
- Button click feedback
- "remove from favs" button
- Log out button moved to the user profile screen
- Favs moved to favorite screen
- Input field error handling
- Styling updates
- Home screen map view listing all the cafes
- Dependency updates

Out of the original plans, the in-app route planning and the filtering of the café list were not implemented. These are both more significant tasks and were pushed down on the priority list. Instead of route planning, HomeScreen got a map that displays all the cafés and the user's location.

For the filtering options to be helpful, the café database should be updated first with valid data. Currently, except for the café's name and address, all the other data is dummy data, so there is not much to filter or search there. Once the database is updated with valid and searchable data, this could be a significant functionality in the app.

Another part that was planned but not implemented at the end is unit testing. The unit testing needs external libraries, such as Jest and possibly a React testing library. The problem here as well was that the dependencies and these libraries did not work with each other. The setup of the testing libraries is poorly documented, and all the answers to the different error messages online are several years old. Even though the author spent a substantial amount of time and reached out to her co-workers as well for help, in the end, unit testing was not added to the project at this stage.

Although not all the planned features were implemented at this stage, the app development progressed steadily. The latest features are released in the Expo environment, and the app can be tried with the Expo Go app.

8 Conclusion

This project was a great learning experience. Besides a full-time job as a software developer, it was challenging to do this, but with good time management and adequate planning, it was possible to allocate enough time for the work.

Working as a software developer in the meantime also helped to gain perspective on an actual client project and see why software development methodologies are essential. The project will most likely fail without a consensus about the ways of working, priorities, and proper planning. In the center of all these, the most important is clear communication between the members of the development team and other stakeholders.

In a one-person project, the most important is planning and collecting enough information to implement a successful application.

8.1 Learning points

As Mark Watney says in the movie The Martian, at some point, everything will go wrong, and then we get to decide if we accept it or get to work. Of course, software development is not nearly as dramatic as space travel, and lives are not at stake most of the time, but one important lesson is how we react to errors in the code.

While working on this app and her full-time software developer job, the author experienced several setbacks, errors, and bugs. When the build fails, the terminal has hundreds of lines of log, and the screen is red from the error messages, it is easy to feel overwhelmed and ready to give up. However, one crucial skill of a good software developer is a certain stubbornness. When the failure happens, it is time to do the investigative work and find out why and how it happened and how it can be resolved.

Sometimes coding is reading forums, and articles, watching tutorials, and asking others. Also, coding is definitely a "team sport." It is not accidental that Hirotaka Takeuchi and Ikujiro Nonaka chose the term "scrum" in their paper *New Product Development Game* published in 1986. (Hirotaka Takeuchi, 1986).

Software development is much more efficient and enjoyable when surrounded by a team. Moreover, the resulting application is more multi-faceted, the more multidisciplinary and diverse the team is. So, while developing an app alone is not impossible, it is much more challenging without reliable and skilled peers. The value everyone brings to a project is more than the sum of their skills, and

looking at a problem from different perspectives will provide a broader scope of proposed solutions.

As for the specific goals of the thesis, these were the learning outcome goals stated at the beginning of the thesis:

- developing general coding skills further
- have a better understanding of software development projects as a whole
- use different testing methods
- use different libraries and frameworks

Out of these objectives, as pointed out already in the Discussion chapter, the testing fell short, so unit tests were not added to the project. Other than that, all points were touched on during the thesis work.

The author definitely improved her coding skills and now feels more comfortable coding React Native. One specific example is the MapView on the HomeScreen. An error message came up in the dev environment, and the page did not render, even though the code seemed correct. After some debugging, the author realized that during the first time use of the app, the page renders before the user grants location permission. The MapView cannot render without the user's location data, and since it has a null value before the user grants permission, the MapView throws an error and breaks the page. This was avoided with a simple if check: if the location state has a value, the MapView will render, and even if the value is null at first, the page will not break.

These solutions and checks come with a problem-solving mindset, but also getting to know how the code behaves in some instances, so when an error occurs, it is easier to debug and find where the error is in the code.

As for using external libraries, the most important part is to learn how to find and read the documentation. This is also important when someone joins a project with an already existing codebase and has to familiarize themselves with how the code works.

This thesis work also helped to give insight into why planning, user testing, backlog grooming, and other non-coding activities are also essential for a software project, even though this work was not done in a team. Nevertheless, when the unit test bugs came up, the author had debugging and sparring sessions with her peers that showed her different methods and point of view on solving the same problem. This was also a great learning point.

8.2 Future possibilities

The possibilities for this app can lead to many different paths. One way it could be developed further is by collaborating with other developers with similar interests. Several people make these small apps just as hobby projects, and those could be united and merged into one solid application.

From a technical standpoint, the code can still be significantly improved. One task that was not done within the scope of the thesis was to migrate the project to TypeScript instead of JavaScript. Type-checking helps avoid minor bugs that go unnoticed in JavaScript but could cause bigger issues later in development.

Another improvement mentioned earlier would be to swap the database from Firebase Realtime Database to Cloud Firestore, which is more flexible and scalable than the Realtime database.

Once the project is migrated to TypeScript, the unit testing setup will hopefully become more straightforward. Unit testing would help ensure that all the different components work as expected and that nothing breaks when the code is changed. Alternatively, if something breaks, it is easier to find out what caused an issue once we look at the test log of the failing test. With unit tests, we could test the rendering of UI elements, for example, buttons, headings, lists, and so on. As the unit tests would cover most of the codebase, it would also be easier to maintain the code and ensure that future production releases improve the overall quality, but also, the code stays flexible.

Another important topic that needs to be addressed is accessibility. About 16% of the world's population is estimated to be differently-abled. That translates into 1 in 6 people. (WHO, 2023). This means that nowadays, accessibility must be at the front and center of every project that develops any service, app, website, or product. Today it is not enough to provide a good user experience. It is essential to provide a good user experience for everyone, including differently-abled users.

As for the list of possible features, this app has many possibilities. Besides helping users find and explore new cafes, it could also serve as a marketing tool. With a twin app deployed on the café side, the cafes could offer users a loyalty program, special offers, or other options. These days apps for fulfilling basic human needs, such as hunger, are becoming the new norm. An app that could be used to order a cup of coffee in the user's favorite place and be picked up on the way to work, and provide interesting information about the coffee beans, could fill the gap in the app market that currently exists.

Bibliography

Bahrynovska, T., 2022. *What Is a Tech Stack: A Package of Means for Achieving Your Goals.* [Online]
Available at: https://forbytes.com/blog/what-is-a-tech-stack/
[Accessed February 2023].

Cocca, G., 2022. *The Software Architecture Handbook.* [Online]
Available at: https://www.freecodecamp.org/news/an-introduction-to-software-architecture-patterns/#mvc-folder-structure
[Accessed February 2023].

Geeks, G. f., 2023. *Geeks for Geeks.* [Online]
Available at: https://www.geeksforgeeks.org/types-software-testing/
[Accessed April 2023].

Hirotaka Takeuchi, I. N., 1986. New New Product Development Game. *Harvard Business Review,* January.

Holen, H., 2018. *Medium.* [Online]
Available at: https://medium.com/crowdbotics/the-best-low-and-no-code-mobile-app-development-platforms-53bc0b4f3558
[Accessed April 2023].

Institute, Software Engineering, 2023. *Software Architecture.* [Online]
Available at: https://www.sei.cmu.edu/our-work/software-architecture/index.cfm
[Accessed February 2023].

Jones, M., 2020. *Medium.* [Online]
Available at: https://martha-7987.medium.com/top-advantages-of-native-mobile-app-development-for-businesses-ab5c016b2e94
[Accessed April 2023].

Kotlin, 2023. *Kotlin.* [Online]
Available at: https://kotlinlang.org/docs/cross-platform-mobile-development.html
[Accessed April 2023].

MacDonnell, K., 2023. *Coffee Consumption by Country in 2023: Top 10 Countries.* [Online]
Available at: https://coffeeaffection.com/coffee-consumption-by-country/
[Accessed March 2023].

Martin, R. C., 2009. *Clean code: a handbook of agile software craftsmanship.* s.l.:Upper Saddle River.

Nielsen, J., 1994. *10 Usability Heuristics for User Interface Design.* [Online]
Available at: https://www.nngroup.com/articles/ten-usability-heuristics/
[Accessed February 2023].

Nielsen, J., 2000. *Why You Only Need to Test with 5 Users.* [Online]
Available at: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/
[Accessed February 2023].

Novoseltseva, E., 2021. *15 Benefits Of Software Architecture.* [Online]
Available at: https://apiumhub.com/tech-blog-barcelona/benefits-of-software-architecture/
[Accessed March 2023].

Pankaj, P., 2023. *Geeks for Geeks.* [Online]
Available at: https://www.geeksforgeeks.org/unit-testing-software-testing/
[Accessed April 2023].

Peiper, H., 2022. *Reimagining the Third Place: How Starbucks is evolving its store experience.* [Online]
Available at: https://stories.starbucks.com/stories/2022/reimagining-the-third-place-how-starbucks-is-evolving-its-store-experience/
[Accessed March 2023].

Polner, E., 2021. *What is a Q Grader and How Do You Become One?.* [Online]
Available at: https://beannbeancoffee.com/blogs/beansider/professional-coffee-tasting-with-q-graders
[Accessed April 2023].

Qualcomm, N. I. & Want, R., 2014. Smartphones: Past, Present, and Future. *IEEE Pervasive Computing,* pp. 89 - 92.

React Native, 2023. *React Native.* [Online]
Available at: https://reactnative.dev/docs/libraries
[Accessed April 2023].

Reenskaug, T., n.d. *MVC XEROX PARC 1978-79.* [Online]
Available at: https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html
[Accessed March 2023].

Roasters, D. C., 2019. *The Differences Between 1st, 2nd, and 3rd Wave Coffee.* [Online]
Available at: https://www.drivencoffee.com/blog/coffee-waves-explained/
[Accessed February 2023].

Silver, S., 2018. *Apple details history of App Store on its 10th anniversary.* [Online]
Available at: https://appleinsider.com/articles/18/07/05/apple-details-history-of-app-store-on-its-10th-anniversary
[Accessed February 2023].

StatCounter, 2023. *Mobile Operating System Market Share Worldwide.* [Online]
Available at: https://gs.statcounter.com/os-market-share/mobile/worldwide
[Accessed February 2023].

WHO, 2023. *Disability.* [Online]
Available at: https://www.who.int/news-room/fact-sheets/detail/disability-and-health
[Accessed March 2023].

Appendices

Appendix 1.: MVP app flow diagram

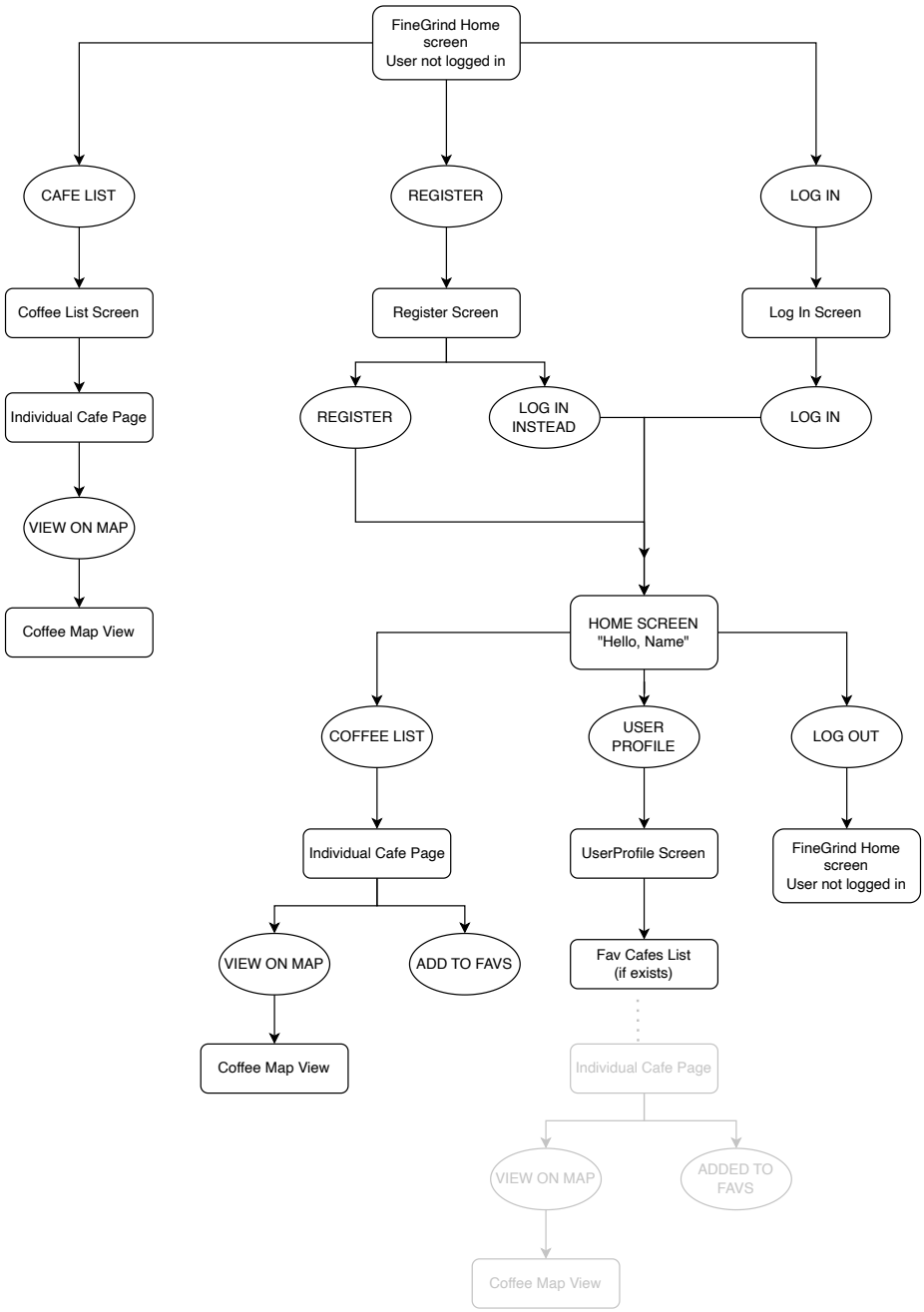


Figure 7. MVP app flow diagram (created by the author)

Appendix 2.: GitHub workflow

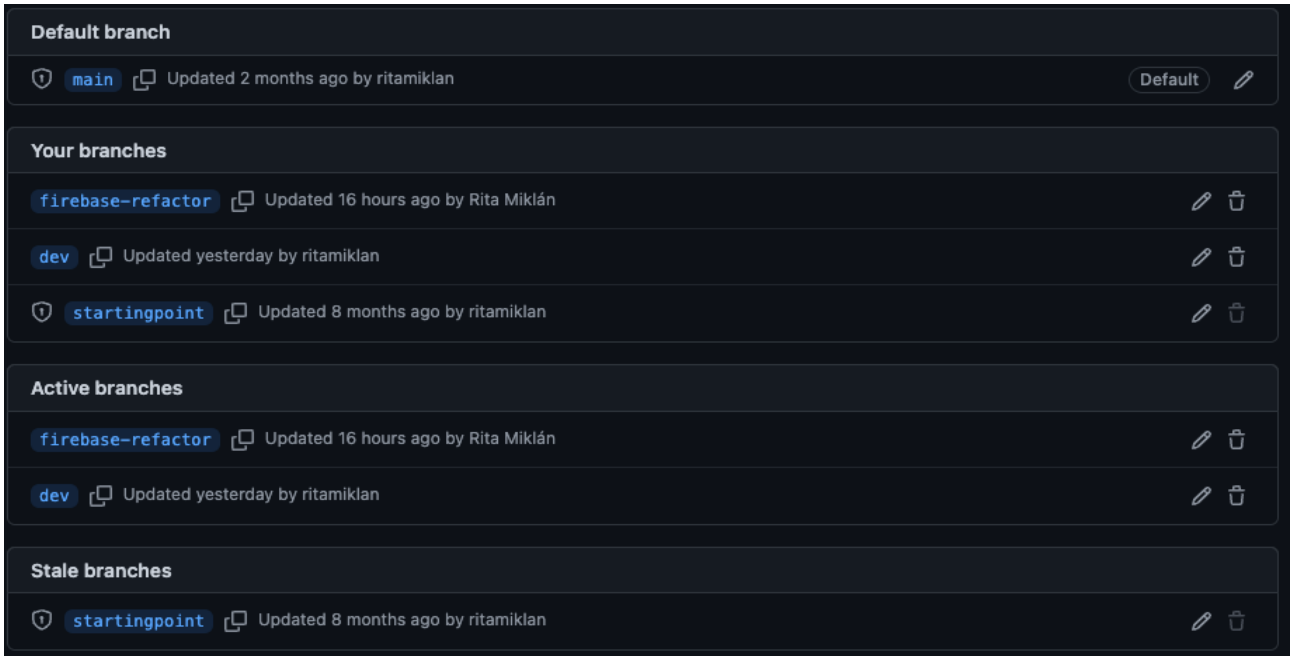


Figure 8. GitHub branches (screenshot)

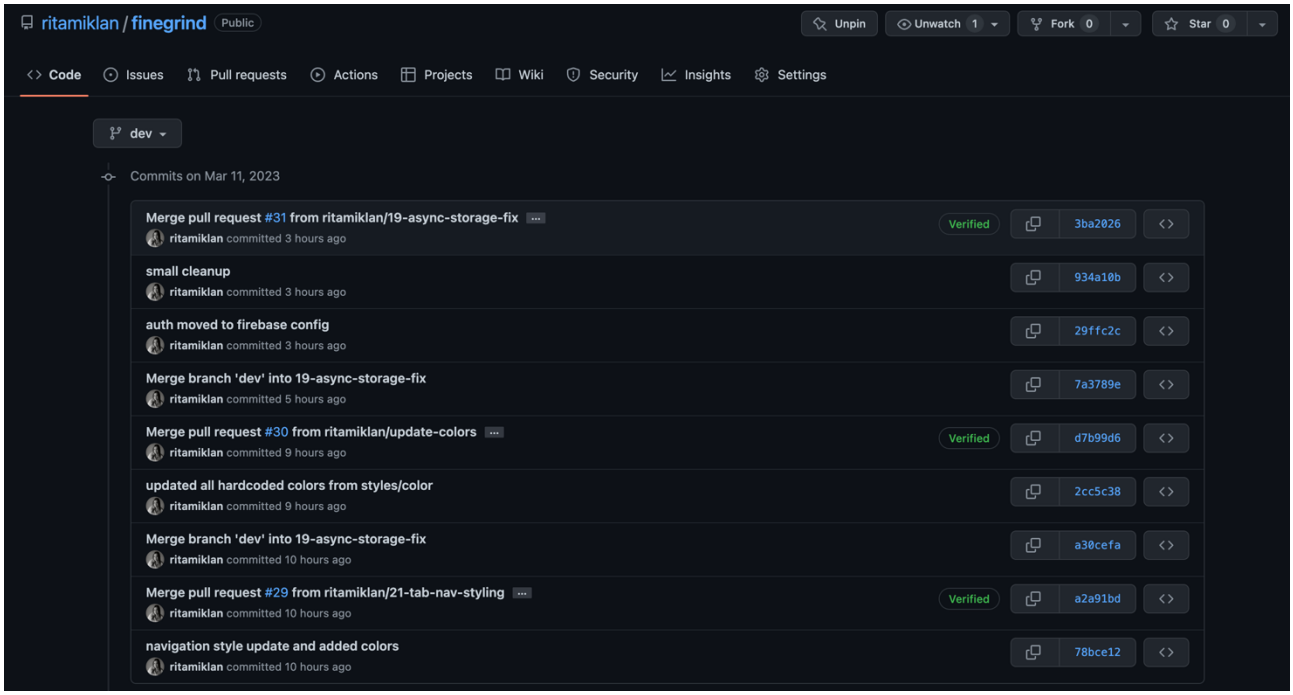


Figure 9. GitHub workflow example (screenshot)

Appendix 3.: Survey questions

1. What is your location (city)?
2. What is your age group?
    - Under 18
    - 18 – 25
    - 26 –35
    - 36-45
    - 46-55
    - Over 56
    - I'd rather not tell
3. How often do you drink coffee?
    - Daily
    - A few times per week
    - Occasionally
4. How often do you visit coffee shops?
    - Daily
    - A few times per week
    - A few times per month
    - A few times per year
    - Never
5. Do you own a smartphone?
    - Yes
    - No
6. What is your user level with smartphones?
    - Basic /calls, texts, taking photos
    - Medium/email, calendar, social media
    - Advanced / use a variety of apps to manage different aspects of life
7. How familiar are you with specialty coffee?
    - Not at all
    - Vaguely – heard the term, but not sure what it means exactly
    - Somewhat – I know what it means, but I'm not interested/don't care
    - Moderately – I know what it means, and I want to know more
    - Very familiar – I consume specialty coffee actively
8. When choosing coffee, is quality important for you?
    - No, I don't know the difference
    - Maybe, if I have some help choosing
    - Yes, I prefer high quality
9. How do you choose where to get coffee?
    - Whatever is closest
    - Big chains, like Espresso House, Starbucks
    - R Kioski, K market, or other supermarket coffee
    - I prefer small local coffee shops (La Torrefazione, Johan och Nystrom)
    - I like to explore small, relatively unknown coffee shops
    - Other:
10. What is also an important deciding factor in where to go? (Multiple choice)
    - Child-friendly
    - Can access it with a pram / Mobility aid
    - Has vegan/gluten-free options
    - Has seasonal drinks (such as pumpkin spice latte)
    - Opening hours for my schedule
    - Laptop-friendly
    - Dog friendly

- Plays good music
- Good public transportation options to get there
- Baristas are friendly, good service
- Other:

11. Do you use apps to find out information about places?
    - Yes
    - No
12. If yes, which apps do you use?
13. Would you use an app to find out more about coffee shops?
    - Yes
    - No
    - Maybe, it depends
14. If given the option, would you like to find and explore more coffee shops in your area?
    - No, I'm happy with where I already go
    - Yes, I'd love to explore more
    - Maybe, depending on certain factors
15. What is the most important for you when visiting a coffee shop?

Appendix 4. Project board on Trello

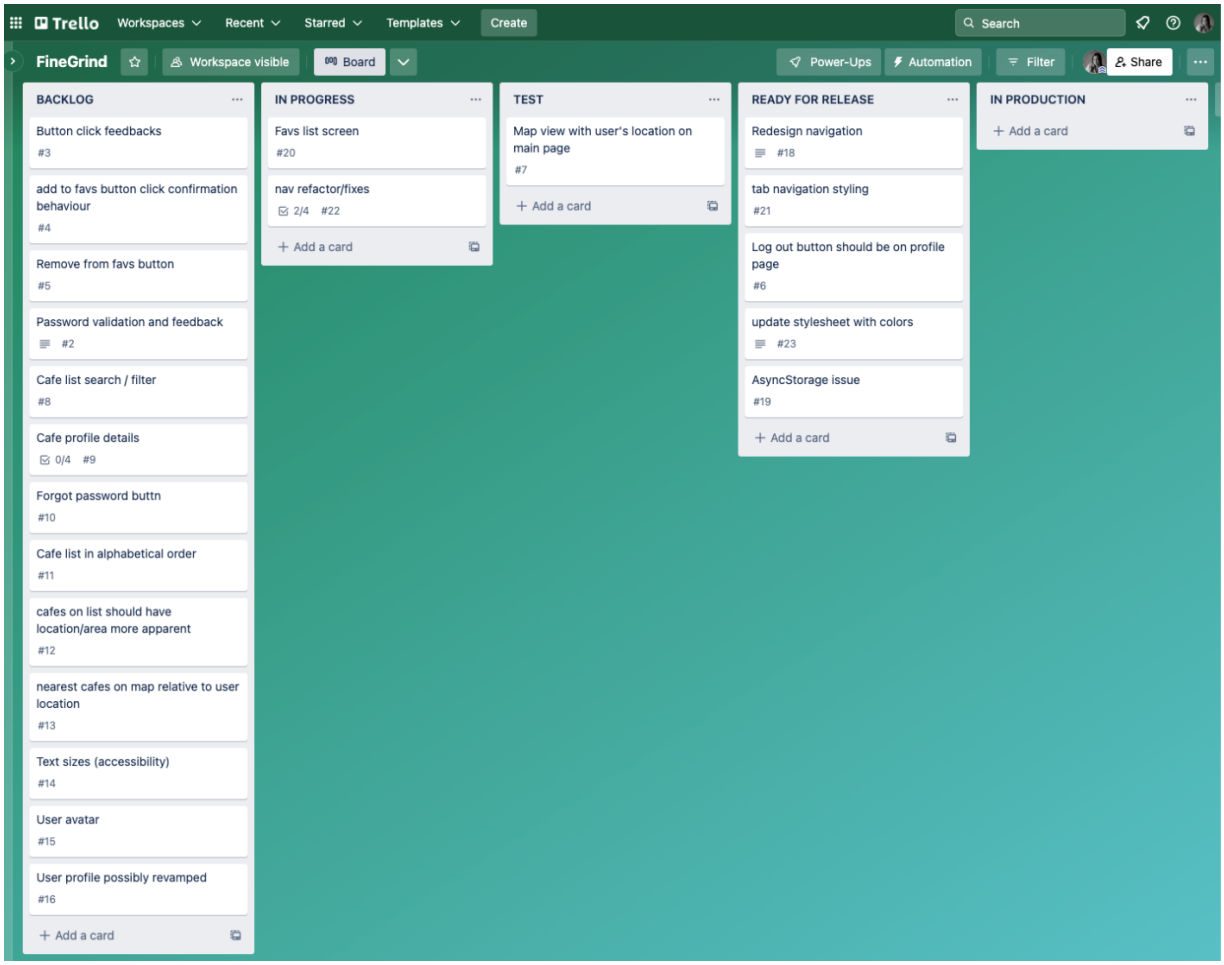
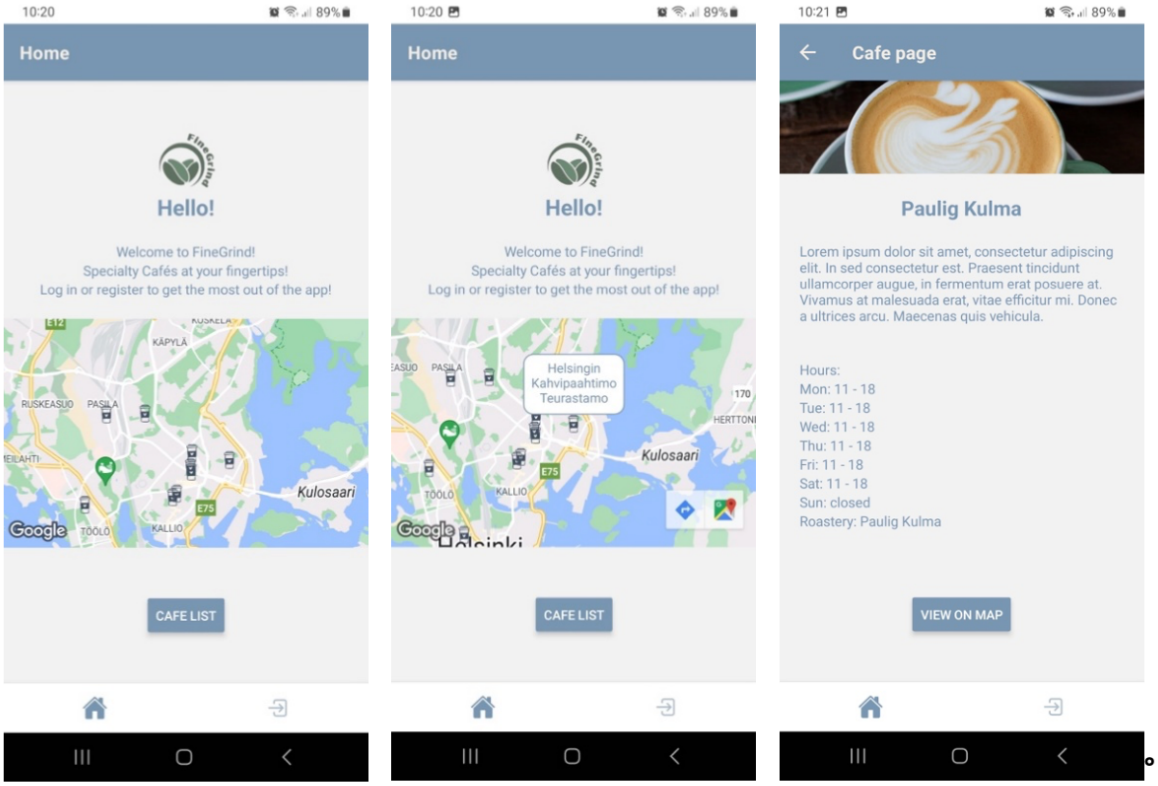
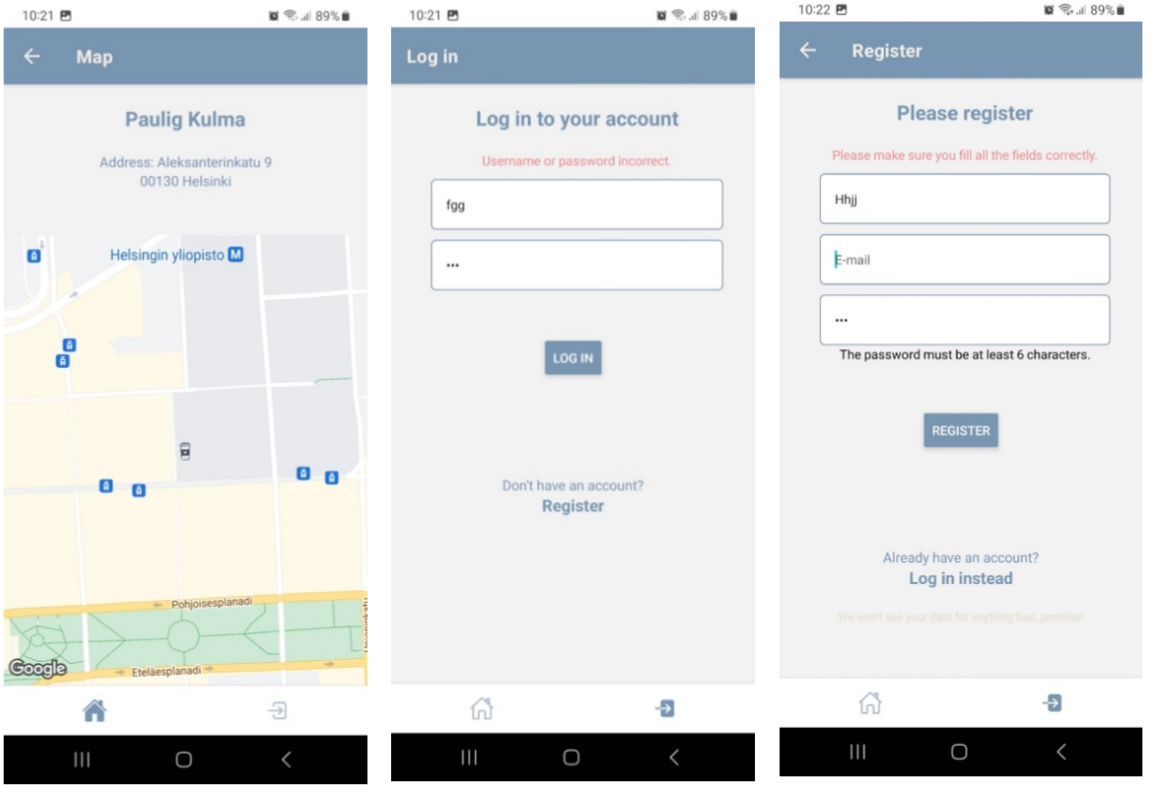
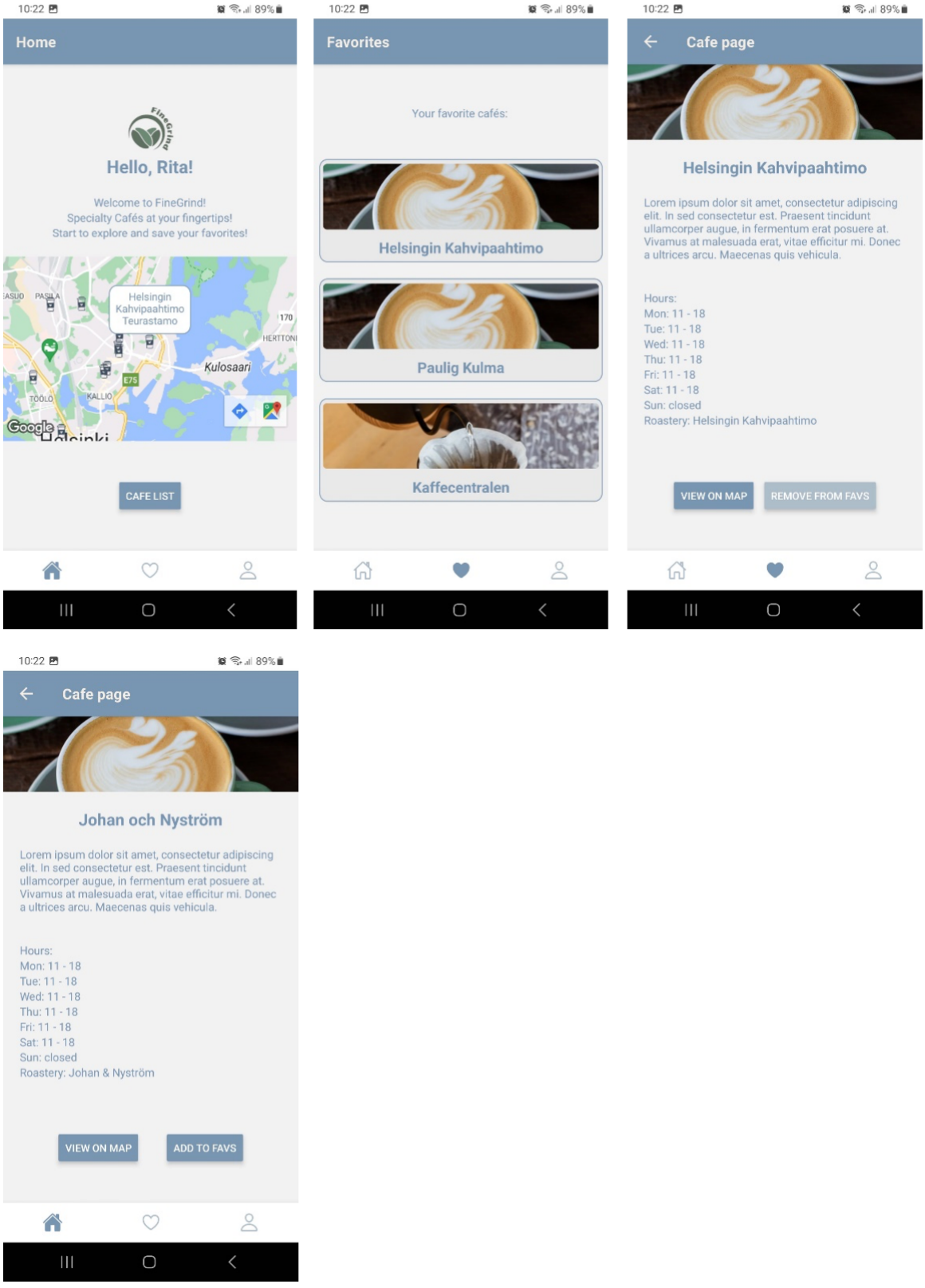


Figure 10. Project Trello board (screenshot)

Appendix 5. Final UI with the new features implemented







Figures 11-20. App UI with the newly implemented features (screenshots)