Carsbnb
Software Engineering Project
Uppsala University

Alexander Sellström
Aliyawer Qambari
Avin Kadir
Florin Perpelea
Umer Farooq

May 25, 2023

Contents

1 Introduction

We live in an era where the streets are for cars and not for pedestrians with a few exceptions. The cities of Europe were developed before the car age, and they were built for pedestrians but the cars impact our cities more and more. They bring pollution and eat our living space. The trend that some cities around Europe started is to either ban or implement different strategies to remove the cars from our streets and more people are encouraged to not invest in a car[1].

Without a lot of cars, the overall lifestyle of many will improve but there will arise a problem that the cars solved, namely transportation for various needs. We aim to bring the car for those that need it but cannot afford one or it does not make sense to buy one and use it every other week. Therefore, we hope to achieve our goal of creating a way to make this progress towards a green city where we all can enjoy our streets and not sacrifice that much in the process [1].

To achieve our goal we want to have an application where users can interact and rent cars. With this application, users will be able to use someone else's car and do their day-to-day tasks without the need of owning a car.

2 Problem Formulation

There is a market for this application. Traveling and renting for a few days together form the market of our application. The purpose of the project is to make car-sharing easier for both the client and the owner who wants to share his/her car. It is worthwhile to allow people that do not own a car to have the benefit of owning a car when they need it. Moreover, it helps the users that have a car that they do not use to earn some money from it.

This business idea works because it solves a problem that many people have where they want to find something that they need but do not have. It is demonstrated by other companies that this business works and their approach on the market is described in the coming section. In addition, it helps a city reduce the number of parking lots because there will be less of a need to own a car if there is no absolute need for it.

Our application wants to help users interact and find a car to use. Users should be of two types, owners and clients. The software will offer the owners a way to post cars and make them available on the application to clients. The software will offer the clients an environment to see cars posted by owners. If the clients find a suitable car for them, they should be able to interact and start a chat so the client can rent the owner's car.

3 Similar Software

3.1 Airbnb

Airbnb is a platform where people can either share or rent accommodation. There are two types of users:

- Users who own accommodation
- Users who want to rent accommodation

The first set of users has unused accommodation that they want to rent out. They can either search for possible tenants themselves or use the user-friendly platform to list their accommodation for people to find and book[2].

The second set of users is the clients who want to travel but do not want to stay at a hotel: perhaps because of budget issues. Therefore, the clients can opt to use the platform to search for locally listed Airbnbs and rent something during their travel period[2].

The platform solves a big problem in the market about price range and variety. It also helps greatly with the trust between the users, using a messaging system, ratings, and reviews[3]. The users have the option to leave a review after their stay for other users to know about their experience[4].

Our application differs from Airbnb because it offers a different service. We built an application that has a similar approach to AirBnB but we sell a service that helps clients and owners match and where the owners can rent out their cars.

3.2 GoMore

GoMore is an application in the same category as Airbnb. They offer a platform where users can share or rent a car. There are similar types of users as for Airbnb: users who own or want to rent on a short-term basis. Users have to manually input the desired location of the car as well as select a start and return date. The application will then show if there are any cars available that match the search[5].

The application that we develop aims to have cars being rented for several hours and not put a minimum renting period of one day. This will give more flexibility to the clients to rent it for jobs that take only a few hours.

3.3 Carsbnb

There is an existing project on Github called Carsbnb that serves the same purpose as Airbnb and GoMore.

There is a difference where this Github project does not have a map when you open the application and therefore you just have a list of available cars. There is no specification for chat functionality. Therefore, our application differs from this one even though we have the same name [6].

4 Requirements

4.1 Functional Requirements

- The users should be able to be client and/or owner
- The users should be able to create an account
- The users should be able to restore their password
- The users should be able to add a profile picture
- The clients should see a terms and conditions page before creating an account
- The clients should be able to see available cars on the map
- The owners should be able to add cars to be rented out on the map
- The owners should be able to delete their cars from the application
- The clients should be able to chat with the owners for setting up the contract and agreements

4.2 Non-functional Requirements

- The application should be a cross-platform application usable on Android and iOS devices
- The front-end is developed with Flutter because Flutter offers a good option for cross-platform application
- The back-end is developed in Firebase for ease of use and security

5 Technical Documentation

5.1 User Interface Design

Figma was used to design the user interface. It has tools to create accurate visualizations of user interfaces in collaboration with others[7], free of charge for students[8]. This made it a good fit for this part of the project.

Using Figma, an approximate design of the application was created to get an idea of how each page should look like and the overall flow of the application. Moreover, the design was done before entering the development phase.

This process made the work more reliable and easier for the front-end team since it generated some references to look at when implementing the pages. However, further decisions were needed for small details later on that were not discussed during the design phase. It was not considered to be worthwhile to spend a lot of time with Figma to design every functionality of the application in detail.

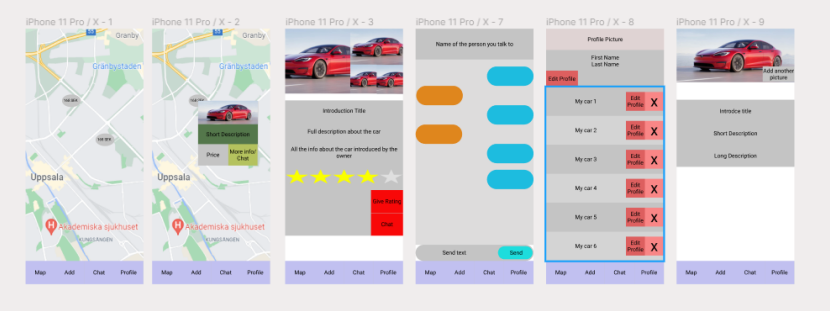


Figure 1: Initial user interface design.

5.2 Front-end

5.2.1 Flutter

Flutter was used to develop the front-end. It is Google's free and open-source UI framework for creating native mobile applications. Released in 2017, Flutter allows developers to build mobile applications for both iOS and Android with a single codebase and programming language. This capability makes building applications for both iOS and Android simpler and faster. Moreover, it uses the emerging language Dart[9].

A Flutter program is built with widgets, which are considered the building block of Flutter applications. Widgets describe what their view should look like given their current

configuration and state. It includes a text widget, row widget, column widget, container widget, and many more[10].

Multiple frameworks can be chosen when developing cross-platform applications. Flutter's approach to developing applications is different from other cross-platform frameworks. React Native, Xamarin, and Titanium try to bring out native (Android/iOS) UI components. React Native uses a JavaScript bridge to convert react components into native views. Flutter renders all of its user interface components by itself. For example, in an application developed in Flutter with any kind of user interface, all the widgets like text, button, and icons are rendered with the help of the Skia engine without using a JavaScript bridge[9].

5.2.2 Dart

The Dart language is crucial for Flutter's success. These are the following benefits of Dart when compared to other languages.

- Dart is ahead-of-time (AOT) compiled into fast native x86 or ARM code for Android and iOS devices.
- Dart can be just-in-time (JIT) compiled.
- Flutter uses Dart 2 – a garbage-collected, object-oriented language with a sound type system and type inferencing.
- Dart has great tooling and IDE support in IntelliJ, Android Studio, and Visual Studio Code.
- Dart has a fast-growing community and an extensive set of libraries and packages that can be used in Flutter apps.
- Dart is very easy to learn for any developer. Some organizations claim that with Flutter it is much easier to hire skilled developers because their background does not matter as much[11].

5.2.3 React Native

React Native was also considered as a front-end solution for the application. It is a popular JavaScript-based mobile app framework that allows for building natively-rendered mobile apps for iOS and Android. The framework lets you create an application for various platforms by using the same code base[12].

Flutter's performance is considered better than React Native's. Unlike React Native, Flutter can manage every pixel of the screen. Moreover, it helps with building responsive

applications for all devices; even older ones. Applications developed on Flutter will work natively on Google's upcoming operating system Fuchsia[13][14].

5.2.4 Google Maps API

Google Maps was used for the application's map view where all the currently available cars are displayed to the user. Google Maps is a web mapping platform and consumer application offered by Google. It offers satellite imagery, aerial photography, street maps, 360° interactive panoramic views of streets (Street View), real-time traffic conditions, and route planning for traveling by foot, car, air (in beta), and public transportation. Google Maps was selected mainly because of its speed and popularity[15][16].

5.3 Back-end

5.3.1 Firebase

Firebase was used for the application's back-end. It is a back-end-as-a-Service that provides developers with a variety of tools and services to help them develop quality apps, grow their user base, and earn profit. Firebase is built on Google's infrastructure. Moreover, it is categorized as a NoSQL database program, which stores data in JSON-like documents. There are also Firebase libraries for Flutter that make communication with the back-end easier. Reading data from the database is made easy and there is no need to specify HTTP methods or dealing with websockets[17][18][19].

Firebase consists of multiple different services, of which Cloud Firestore, Firebase Authentication, Cloud Storage, and Cloud functions have been selected for this application. Cloud Firestore is a serverless NoSQL database. Firebase Authentication provides an easy way to build secure authentication systems and supports email and password accounts as well as federated authentication et cetera. Cloud Storage stores and serves user-generated content. Cloud Functions let developers write and run back-end code server-side without the need of setting up a server. These services are easy to integrate with each other to create a full back-end solution for the application[20].

Besides Firebase, there were other back-end technologies considered for this project, namely Django and MongoDB. Django is a high-level Python web framework that allows for rapid development and results in an API with endpoints that are integrated with the front-end. Django's database API functionality requires a running database server. The framework supports many different databases, such as PostgreSQL, MySQL, and SQLite. The framework's main advantages are that it is secure, scalable, and makes development quicker and easier. MongoDB, on the other hand, is a NoSQL database that can be operated on-premise or on the cloud. Both MongoDB and Firebase are post-relational databases

with similar JSON-like document data models and schemas, and both are built to make application development easier and for horizontal scalability. MongoDB is, however, said to have better performance than Firebase[21][22][23].

In the end, Firebase was selected for the back-end. The main reason for this was that it was the only solution that offered a full back-end solution with all the services mentioned earlier, while also being serverless. The solution being serverless means that no time needs to be spent on finding a server and deployment. It also has the advantage of the Firebase libraries for Flutter to make the front-end integration easier.

5.3.2 Authentication

Firebase Authentication is a back-end service that handles registration, login, and authentication of users[24]. Logged-in users receive a digitally signed *JSON Web Token* (JWT)[25] which can be used to securely authenticate them. All requests sent from the app to any of the back-end services provided by Firebase include this JWT, allowing rules to be put in place to prevent unauthorized access.

Firebase Authentication provides registration with OAuth[26] through many federated identity providers such as Facebook, Google, or Apple. In this project, Google was chosen as a login and registration method in addition to email and password. Firebase Authentication also handles sending out emails to new users or users who wish to reset their password.

Firebase Authentication uses its own database for storing users which is inaccessible to developers. When a new user is created through Firebase Authentication an event is fired, allowing event-based Firebase Functions to create the corresponding user's document within the Firestore database.

5.3.3 Cloud Firestore

Cloud Firestore is a flexible and scalable NoSQL cloud-hosted database to store and sync data for client-side and server-side development. It keeps the data in sync across client apps with the help of real-time listeners. In addition, it offers offline support for mobile and web so that applications can stay responsive regardless of network latency or internet connectivity. Cloud Firestore also allows for an effortless integration with Firebase Cloud Functions, which will be described in more detail in an upcoming section.

The database's NoSQL data model works by storing data in documents containing fields mapping to values. These documents are stored in collections, which are used to organize the data and query it. Moreover, subcollections within documents can also be created. The documents support many different data types, such as strings, numbers as well as complex, nested objects.

The data in the Cloud Firestore database can be secured by using Firebase Authentication and Cloud Firestore security rules, which are both used in this project. Another important feature is that the database is designed to scale automatically. This allows for a tough database workload and large database, without having to worry about the application breaking in case of increased popularity.

We refer to [27] for details of Cloud Firestore.

5.3.4 Database Structure

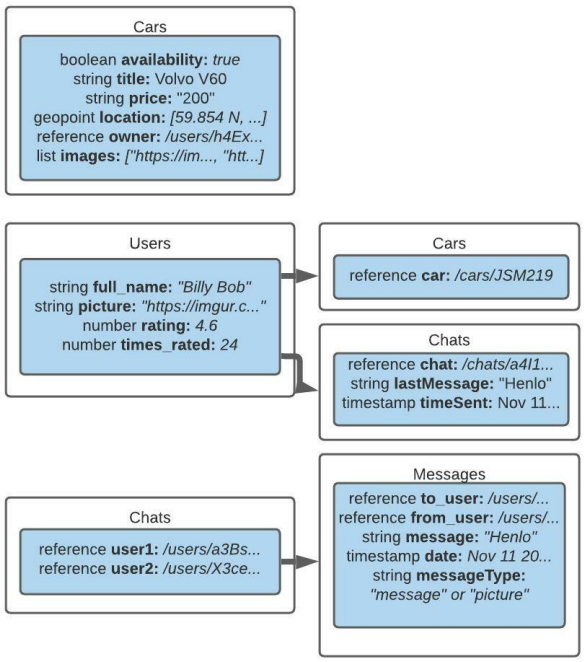


Figure 2: Database structure.

The database structure is illustrated in Figure 2. The white boxes are meant to represent the collections while the blue boxes represent the documents within the collections. Moreover, the arrows symbolize that the pointee is a subcollection within the pointing collection

There are three main collections in the database: **Users**, **Cars**, and **Chats**. The documents in each of these collections all have auto-generated names.

Users stores documents for every user in the system, regardless of whether they used email and password registration or federated login. The documents contain fields that store the user's full name, profile picture with the URL to the image as well as the user's rating and

the number of times it has been rated. The collection also consists of two subcollections, Chats and Cars. Chats stores documents for each chat that the user is participating in. The fields contain a reference to the chat document in the main **Chats** collection, the last message, and the time the last message was sent in that specific chat. Cars stores documents containing references to the car documents that are owned by the user, to access the correct user cars in the front-end. Another approach that was considered for this problem was to simply store the user ID as a field in the car documents and query all the car documents containing the authenticated user's ID in the front-end. The reason why this approach was not selected was the fear that reading would take a long time in case a significant number of cars are stored in the database.

Cars, on the other hand, contains documents for each added car in the application. The documents consist of fields that store the current availability of the car as a boolean, the title and price, a reference to the owner's document as well as a list of URLs to the user's uploaded images of the car.

Chats stores documents for each chat between two users that exist in the system. There are two fields in each document that contain references to both of the partaking user's documents in the **Users** collection. In addition, the documents consist of a subcollection named Messages, which stores each message in the chat as a document. The fields in a Messages document contain information about who was the sender and receiver, the time it was sent, the message itself, and the type of message it was. If the field *message Type* stores the string "message", it means that the message was an ordinary text message and the field *message* will then contain the text as shown in Figure 2. On the other hand, if *message Type* happens to be "picture", it means that the message sent was an image. The field *message* then stores a URL to the image that was sent by the sender. *message Type* exists to differentiate the type of message in a simple way for the front-end integration.

5.3.5 Cloud Storage

Firebase Cloud Storage is a service for storing files such as images that are too large to be stored in Firestore[28]. In this project, Cloud Storage is used to store the profile pictures of users and pictures of cars. Cloud Storage generates URLs to the images which are stored in the Cloud Firestore database.

5.3.6 Cloud functions

Cloud Functions are used for all create, update, and delete operations in Firestore and Cloud Storage. The function for registering new users is an event-based function that gets triggered when a new user is created by Firebase Authentication. All other cloud functions are invoked by the application. Our Cloud Functions only return 1 or 0 to indicate if they

completed successfully: There is no meaningful data sent back to the mobile application.

5.4 Integration

Integration of the back-end with the front-end was mainly done in two ways, namely reading data from the database and calling cloud functions to make database operations.

Reading data was achieved with the help of StreamBuilders, which is a widget that makes it possible to use data from real-time snapshots taken from the database[29]. Figure 3 shows an example of how the data about the user's cars is read from the database in the front-end. The StreamBuilder's *builder* property uses the snapshots taken from the *stream* property, which in this case streams the collection storing documents about users' cars. The data from the documents in this collection is then retrieved in the *builder* by using the data attribute followed by the docs attribute.

```
body: StreamBuilder<QuerySnapshot>(
  stream: FirebaseFirestore.instance
      .collection("users")
      .doc(currentUserId)
      .collection("cars")
      .snapshots()
      .asBroadcastStream(),
  builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
    if (!snapshot.hasData) {
      return Center(
          child: buildText("No car added yet!", kBlack, 16, true));
    }
    var docs = snapshot.data!.docs;
```

Figure 3: StreamBuilder.

Calling cloud functions in the front-end was done by first storing the specific function in a variable, including information about the server region where the function is deployed. Afterward, the call method is called with the payload data on the function variable. Figure 4 shows an example of calling the cloud function for adding cars.

```dart
final HttpsCallable addCar =
    FirebaseFunctions.instanceFor(region: "europe-central2")
        .httpsCallable("addCar");

final HttpsCallableResult result = await addCar.call({
  "availability": true,
  "images": [],
  "lat": currentLocation.lat,
  "lng": currentLocation.lng,
  "price": price,
  "title": title,
});
```

Figure 4: Calling cloud functions in Flutter.

5.5 System Architecture

The system consists of an application and three services from Firebase: Authentication, Cloud Functions, Storage and the Firestore database.

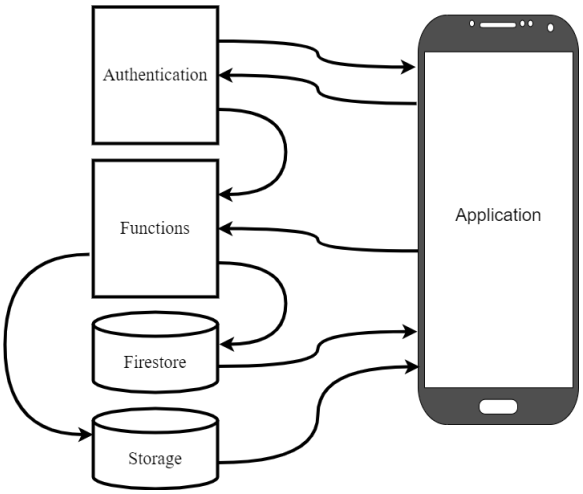


Figure 5: System structure

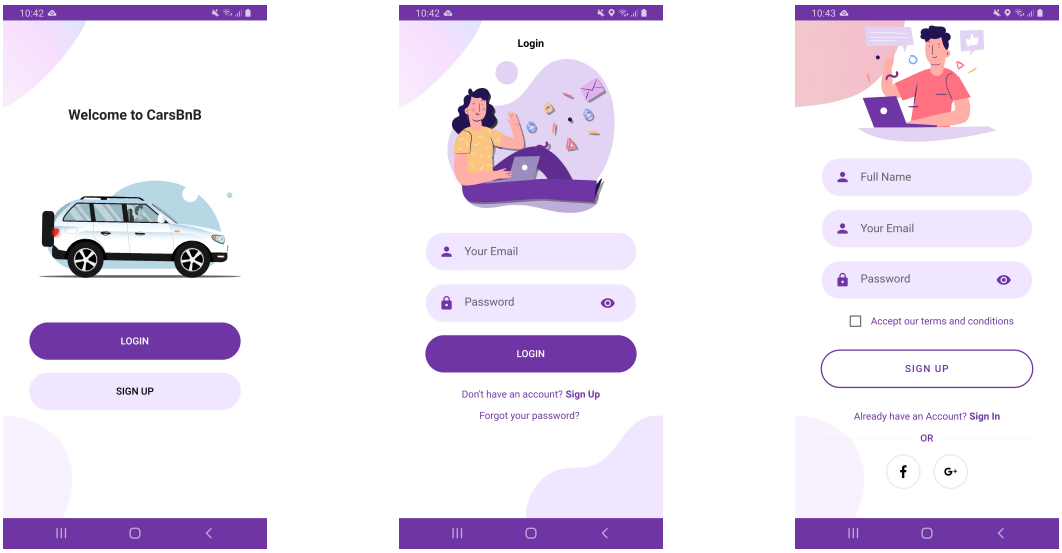
The architecture is illustrated in Figure 5. The arrows represent the direction that information can flow between components in the system. The application can for instance send a new user registration request to Firebase Authentication, which in turn can respond with a token. Firebase Authentication can then trigger an event that makes the Cloud Functions create the user document in Firestore.

The application can also call Cloud Functions directly. If the user changes their profile picture, the picture is sent from the application to the Cloud Functions, which then saves

the picture in Firebase Storage and writes the URL of the picture to the user's document in Firestore. After that, the application can read the URL in Firestore and then access the picture from Firebase Storage. The application cannot write directly to Firestore or Firebase Storage by design, due to security concerns.

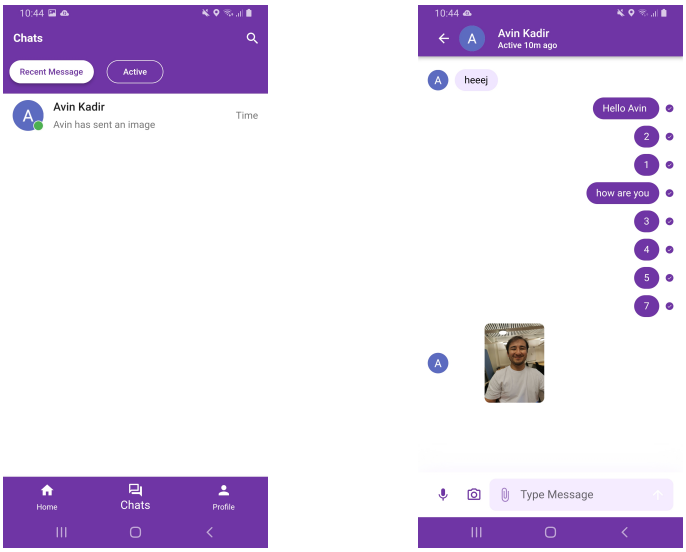
5.6 Results

This section will describe what was achieved in the project.



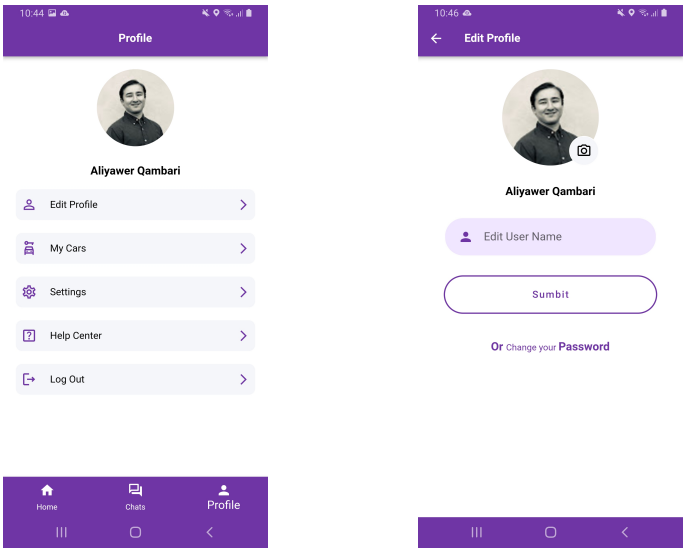
(a) Start (b) Login (c) Sign up

Figure 6: Authentication screens



(a) Chat list (b) User chat

Figure 7: Chat screens



(a) User profile (b) Edit profile

Figure 8: Profile screens

The application begins with a start screen which is shown in Figure 6a. Depending on if the user wants to login or sign up, they are redirected to either the login screen in Figure 6b or the sign up screen in Figure 6c.

When the user is authenticated, he/she automatically enters the map screen where all the currently available cars are shown in Figure **??**. If the user happens to press on a car icon on the map, he/she enters the car information screen shown in Figure **??**, where information about the car and the user is presented. The user can start a chat with the owner by pressing on the white button in the bottom right corner, if he/she happens to be interested in renting the car. An example of a chat is shown in Figure 7b.

The application contains a navigation bar in the bottom of the screen. If the button called *Chats* is pressed, the user is redirected to the chat list screen in Figure 7a, where all of the user's current chats are listed. The *Profile* button redirects the user to the profile screen, shown in Figure 8a. There, the user can press *My Cars* to add new cars and change availability of already added cars, as shown in Figure **??**. The availability decides if the car is shown on the map or not. The *Add car* button leads to Figure **??**, where the user needs to input information and pictures of the car. It is also possible to change the profile picture and name by pressing *Edit profile* in the user's profile screen. This leads to the screen in Figure 8b. Lastly, the user can also log out of the application by pressing the *Log out* button, which redirects the user to the start screen.

5.7 Security

Security is achieved through the use of rules restricting read and write access. Only authenticated users are able to invoke Cloud Functions. Functions that modify data pertaining to a certain user also verify that the invoker is the same user that is the target of the function. All writes to the database are done through Cloud Functions to prevent malicious modification of Firestore or Cloud Storage. Cloud functions can sanitize inputs by ensuring they do not contain potentially harmful data. By performing sanitization of inputs in the back-end, no trust is placed in the client.

6 Testing

6.1 Front-end

Testing the UI in Flutter can be done with various methods:

- Manual testing or user testing is done by having a person testing the respective functionalities of the application. The person presses buttons and tries different things to see if the application functions correctly. This can be very tricky because bugs can be overlooked or not appear during the tests.
- Automated testing is done by writing tests that are performed automatically and can

be used for future builds. If the application changes over time, tests ensure that the main functionality of the application is kept intact and nothing is broken by the change.

Categories for automated testing:

- Unit tests
- Widget tests for Flutter applications
- Integration tests

	Unit	Widget	Integration
Confidence	Low	Higher	Highest
Maintenance cost	Low	Higher	Highest
Dependencies	Few	More	Most
Execution speed	Quick	Quick	Slow

We refer to [30] for details of Flutter testing.

6.1.1 Unit tests

Unit tests are used to test a specific and single function. They are written to make sure the function that is tested does what it is supposed to do or produces the right thing. Dependencies from outside need to be mocked for this kind of test to work. In UI testing, this kind of test does not help that much because the only thing you can test with this is functions and not widgets in Flutter[30].

6.1.2 Widget tests

Widget tests are used to test individual widgets and their functionality and location. The goal of the widget test is to make sure that the widget looks and functions as intended when an actor interacts with the widget. A widget test should be able to function like in the real environment but similar to the unit tests, the dependencies should be mocked up. Therefore more implementation needs to be done before being able to test the widgets in a test environment[30].

6.2 Integration tests

Integration tests are the most complete test because it tries to test part of the application and does not mock up dependencies. It tries to see how different components work together and how they communicate. Generally, integration tests work on a real environment OS or emulator[30].

6.3 Back-end

In the back-end, testing was conducted on the Firebase Cloud Functions.

6.3.1 Manual testing

Testing was done by using manual testing, which was deemed to be the best approach. As soon as an HTTP callable function was written, it was integrated with the front-end and tested thoroughly manually. At first, the most fundamental functionalities of the function ware tested to ensure that it was working. After that stage, edge cases were tested to ensure that there were no hidden bugs in the function. The background function that is triggered by user registration, was also tested manually by creating a user in the application and observing the result in the database.

This approach worked but it was not ideal, since the functions could not be tested without help from the front-end team. Ideally, the functions would have been thoroughly tested by the back-end team before they were handed to the front-end team, to make the integration as smooth as possible.

6.3.2 Unit tests

Unit testing was disregarded because the functions do not return a value that can be compared with an expected value in a meaningful way. Instead, they return 0 when finishing without errors or 1 including an error message in the log if an error occurred. The reason for this is that the only purpose of the functions is to do operations in the database.

6.3.3 Interactive tests

Interactive testing was deemed useful. It is done through the Cloud Functions shell, which provides tools for invoking functions with test data. This would have allowed for thorough testing without front-end integration and would therefore have solved the issue of needing front-end team help with testing. The problem was that it is currently impossible to invoke HTTP callable functions using context.auth, which all of the functions are using to receive information about the authorized user. Therefore, interactive testing had to be disregarded as well[31].

6.3.4 Test quality

Manual testing was most likely not enough to catch all bugs in the back-end code. There is a high possibility that there exist some edge cases that have not been tested that could cause problems in the future.

7 User documentation

7.1 Installation

7.1.1 Android installation

- Go to this link:
  https://drive.google.com/drive/folders/1Kb
  4jSB-syxcu82P-ZD7AoWUdUJjxBohb?usp=sharing
- Download app.apk
- Open it on your Android Device.
- On some phones you may be asked to enable installation from unknown sources.

7.1.2 iOS installation

iOS installation instructions cannot be provided due to the lack of a way to distribute a pre-built binary for iOS devices without an Apple Developer account.

8 Project management approach

In this section, the tools used in the project management approach are first described, followed by the approach itself and how the tools were incorporated.

8.1 Tools

8.1.1 Kanban

The Kanban method in software development was introduced in 2004. The method pushes project teams to visualize the workflow, limit work in progress (WIP) at each workflow stage, and measure cycle time. Kanban consists of a board that provides visibility to the software process since it shows the assigned work of each developer, clearly communicates priorities, and emphasizes bottlenecks. Additionally, its goal is to minimize WIP by only developing the requested items. This produces a constant flow of released work items to the customers since then the developers focus on those few items at once. The Kanban method aims to quickly adapt the process by using shorter feedback loops. The main incentives for the usage of Kanban are the focus on flow and the absence of obligatory iterations[32].

8.1.2 Communication

Discord is a free instant messaging platform available on Windows, Mac OS, Android, iOS, and web browsers. It allows users to create private servers that can be joined through invitations. The servers themselves can be divided into themed channels for specific subjects. It is also possible to add multiple voice channels that are made for both voice communication and screen sharing[33].

Facebook Messenger is also an instant messaging application available on multiple platforms. The application lets users create individual chats between two users as well as group chats. Additional features are media sharing and voice chats among other things[34].

8.1.3 Other

GitHub is a web-based version-control and collaboration platform for software developers. It was founded on Git, which is an open-source code management system to make software build faster. Git is used for storing the source code for a project and tracking the complete history of all changes to that code. It allows developers to collaborate on a project more effectively by providing tools for managing conflicting changes from multiple developers. GitHub lets developers create repositories, which contain all of a project's files, as well as each file's revision history. Repositories can have multiple collaborators and can be either public or private[35].

Github also has a project board feature that can be integrated with repositories. Project boards consist of issues, pull requests, and notes categorized as cards in columns added by the user. The project board cards contain relevant metadata for issues and pull requests, such as labels, assignees, the status, and who opened them. In addition, the cards can be moved between the columns by the project collaborators[36].

Google Drive is a free cloud-based storage service that enables users to store and access files online. The service syncs stored documents, photos et cetera across all of the user's devices, including mobile devices, tablets, and PCs[37].

8.2 Approach

Using the described tools in the previous section, our project management approach was a straightforward one. We used Kanban to visualize our progress of the project and make sure we do not overlap tasks. In the beginning, we set up the board using the Github project board feature and created tasks for the most relevant problems we had at that time. After adding all the tasks to the backlog we started assigning ourselves to the tasks we were responsible for. A slight deviation from Kanban is that we did not limit how many tasks can be in a specific column. Using this Kanban method helped us see the progress of the project

on a board and how far we are into the project and more importantly what remains in the backlog. There are always problems that appear during a project; for those, we created a task card and added it to the backlog to be solved.

We mostly worked in person in the room designated for the project. We used the communication tools to make planning and communication easier when we were not at the university. Discord was used to store important information such as links and meeting times. We also used the voice channels when we needed to work together from home. Messenger was mainly used for communication in the group and for giving updates on our work from home.

We worked three days per week in person and two days alone at home. After a few weeks, we adjusted this by having a quick update through the chat of what everyone was doing during those two days when we were not at the school.

We created a Google Drive folder which was used to organize all the artifacts/deliverables that were needed in the project. We also created a Github repository that hosted our code and let us have different branches where we all could work separately from each other.

During the project our team was split in three main areas of responsibilities. The group had: a project manager (Aliyawer Qambari), a team that was responsible for the front-end (Aliyawer Qambari, Umer Farooq), a team that was responsible for the back-end (Avin Kadir, Alexander Sellström), and a responsible for the report (Florin Perpelea).

Even though the group was split into multiple teams, we were working together and sometimes worked on different tasks that were not in our area of responsibilities.

9 Lessons learned

We found out that testing is very hard to do when you have the project in the end stages and that it is much better to start testing as early as possible. Since we did not do this in this project, we encountered bugs that could have been avoided by starting testing early.

During the project, we found out that a project is much more than writing code for the application. We did not take into account the time it takes for organizational purposes (meetings, writing diaries, teacher meetings, project presentations). Because of our misjudgment of the time, we filled in the project plan without leaving room for these mandatory activities: because of this, we were behind the schedule during certain weeks.

We also learned that we should have put effort into researching if the technology we planned to use was testable. As mentioned in Section 6.3, it was impossible to do any testing on the Firebase Cloud Functions except manual testing. This put extra pressure on the front-end team since they had to help with testing while they already had a lot of work to do. While Firebase has a lot of benefits, it would still have been a better idea to select a

testable technology. This would ensure that the back-end is functional before integrating it with the front-end, thus making the process a lot smoother and quicker.

The group should have spent time explaining our code to each other regularly. In this project, we all mostly worked on our tasks separately. This led to us having a situation where only the one who had written the code knew how it worked. This caused problems at the end of the project when one project group member got ill and needed help with finishing the tasks that were left.

10 Future work

In this section, we discuss features that we did not have time to implement.

10.1 Identity verification

Adding a means of verifying the identity of users would make it harder for would-be criminals to abuse the service and get away with it. This can be achieved by using an API provided by for instance BankID[38] or Freja eID[39].

10.2 Web application

Making our app available on the web would improve the ease of access for users. Flutter allows the application to be exported for the web, but doing so would break some functionality due to certain components requiring API keys not being configured.

10.3 Rating system

A rating system can be used by users to like or dislike other users. This rating system if used correctly can have a positive impact on the application ecosystem. Problematic users can be disliked by other users and they will not be contacted in the future. This system can be abused by liking or disliking as a mass to influence someone else's rating but if used properly it can help the application.

10.4 Legal agreements

Terms and conditions for using the application will need to be created by a lawyer before public release. There may also be a need for legal documents regarding the exchange of services between two users.

10.5 Connectivity issue

One bug that we found and needs to be fixed is a connectivity issue. Currently, if you lose internet connection the application will simply show a lot of null pointers due to the lack of connection to the server.

References

[1] *Can we make cities car free?* Retrieved 2021-12-10. URL: https://youtu.be/g9-9CxCxrVE.

[2] *Airbnb*. Retrieved 2021-12-09. URL: https://www.airbnb.com/.

[3] *How Airbnb builds trust between Hosts and guests*. Retrieved 2021-12-09. URL: https://www.airbnb.com/help/article/4/.

[4] *Reviews for stays*. Retrieved 2021-12-09. URL: https://www.airbnb.com/help/article/13/.

[5] *GoMore*. Retrieved 2021-12-09. URL: https://gomore.se/.

[6] *carsbnb*. Retrieved 2021-12-10. URL: https://github.com/harishankar0301/carsbnb.

[7] *Figma prototyping*. Retrieved 2021-12-09. URL: https://www.figma.com/prototyping/.

[8] *Figma education*. Retrieved 2021-12-09. URL: https://www.figma.com/education/.

[9] *Flutter*. Retrieved 2021-12-10. URL: https://flutter.dev/.

[10] *What is widgets in Flutter?* Retrieved 2022-01-11. URL: https://www.geeksforgeeks.org/what-is-widgets-in-flutter/.

[11] *Dart*. Retrieved 2021-12-10. URL: https://dart.dev/.

[12] *React Native*. Retrieved 2022-01-11. URL: https://reactnative.dev/.

[13] *Fuchsia*. Retrieved 2022-01-11. URL: https://fuchsia.dev.

[14] *5 Reasons Why Flutter Is Better Than React Native*. Retrieved 2022-01-11. URL: https://betterprogramming.pub/5-reasons-why-flutter-is-better-than-react-native-cf2e9b077f66.

[15] *Google Maps Platform FAQ*. Retrieved 2021-12-10. URL: https://developers.google.com/maps/faq#whatis.

[16] *The 3 Most Important Reasons Why You Should Choose Google Maps*. Retrieved 2022-01-11. URL: https://blog.mapspeople.com/why-choose-google-maps.

[17] *What is Firebase?* Retrieved 2022-01-03. URL: https://www.educative.io/edpresso/what-is-firebase.

[18] *FlutterFire Cloud Firestore*. Retrieved 2022-01-04. URL: https://firebase.flutter.dev/docs/firestore/usage.

[19] *Using Cloud Functions for Firebase*. Retrieved 2022-01-04. URL: https://firebase.flutter.dev/docs/functions/usage.

[20] *Firebase Products*. Retrieved 2022-01-03. URL: https://firebase.google.com/products-build.

[21] *Comparing Firebase vs MongoDB*. Retrieved 2022-01-04. URL: https://www.mongodb.com/firebase-vs-mongodb.

[22] *How to install Django*. Retrieved 2022-01-04. URL: https://docs.djangoproject.com/en/4.0/topics/install/#get-your-database-running.

[23] *Django*. Retrieved 2022-01-04. URL: https://www.djangoproject.com/.

[24] *Firebase Authentication*. Retrieved 2021-12-10. URL: https://firebase.google.com/products/auth?gclid=CjwKCAiAksyNBhAPEiwAlDBeLKa35szgBpIRis0WH6bUXdAAYYv9RP-yhU4iDSl9nj-StGJwVHZHmxoCAaYQAvD_BwE&gclsrc=aw.ds.

[25] *JWT*. Retrieved 2021-12-10. URL: https://jwt.io/introduction.

[26] *Firebase Authentication docs*. Retrieved 2021-12-10. URL: https://firebase.google.com/docs/auth/where-to-start.

[27] *Cloud Firestore*. Retrieved 2021-12-10. URL: https://firebase.google.com/docs/firestore.

[28] *Firebase Storage docs*. Retrieved 2022-01-11. URL: https://firebase.google.com/docs/storage.

[29] *StreamBuilder*. Retrieved 2022-01-12. URL: https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html.

[30] *FlutterTesting*. Retrieved 2021-12-10. URL: https://docs.flutter.dev/testing.

[31] *Test functions interactively*. Retrieved 2022-01-05. URL: https://firebase.google.com/docs/functions/local-shell.

[32] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. "Kanban in software development: A systematic literature review". In: *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. 2013, pp. 9–16. DOI: 10.1109/SEAA.2013.28.

[33] *Discord*. Retrieved 2021-12-28. URL: https://discord.com/.

[34] *Messenger Features*. Retrieved 2022-01-02. URL: https://www.messenger.com/features.

[35] *Github*. Retrieved 2022-01-02. URL: https://searchitoperations.techtarget.com/definition/GitHub.

[36] *About project boards*. Retrieved 2022-01-02. URL: https://docs.github.com/en/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards.

[37] *Google Drive*. Retrieved 2022-01-02. URL: https://searchmobilecomputing.techtarget.com/definition/Google-Drive.

[38] *BankID*. Retrieved 2022-01-11. URL: https://www.bankid.com/en.

[39] *Freja eID*. Retrieved 2022-01-11. URL: https://frejaeid.com/en/home/.