spaceXchange

P2P Parking Spot Rental android App

Internet Praktikum Summer Term 2023 - Group F
Nabel Jajeh
Marvin Heidinger
Thomas Hugo
Ali Khalaji
Georgi Totev
Haroun Abdelsamad
July 12, 2023

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

1 Introduction

1.1 Motivation

In today's fast-paced world, finding a convenient and affordable place to park your car can be a challenging and time-consuming task. Whether you're navigating a busy city, running errands, or exploring a new destination, finding a place to park is often a frustrating and time-consuming process. That's where spaceXchange comes in.

The motivation behind spaceXchange is to address the challenges and inefficiencies associated with traditional parking methods. We understand the struggles of finding an available space, dealing with high costs, and the lack of flexibility in options. Our aim is to revolutionize the parking experience by creating the first application for the rent of people's private parking spots.

One of the key challenges we want to address is the limited availability and high demand for parking. Traditional car parks and garages often fail to meet the needs of drivers, especially at peak times or during popular events. Our app connects car owners directly, allowing them to share their available parking spaces with others in need. This peer-to-peer approach creates a wider network of parking options, increasing the chances of finding a suitable space.

Another challenge we are addressing is the rising cost of parking. Traditional parking facilities can be expensive, especially in densely populated areas or popular tourist destinations. By connecting users directly, spaceXchange promotes competitive pricing and offers more affordable alternatives. Car owners can earn extra income by renting out their own parking spaces, while users looking for a place to park can find low-cost options that fit their budget.

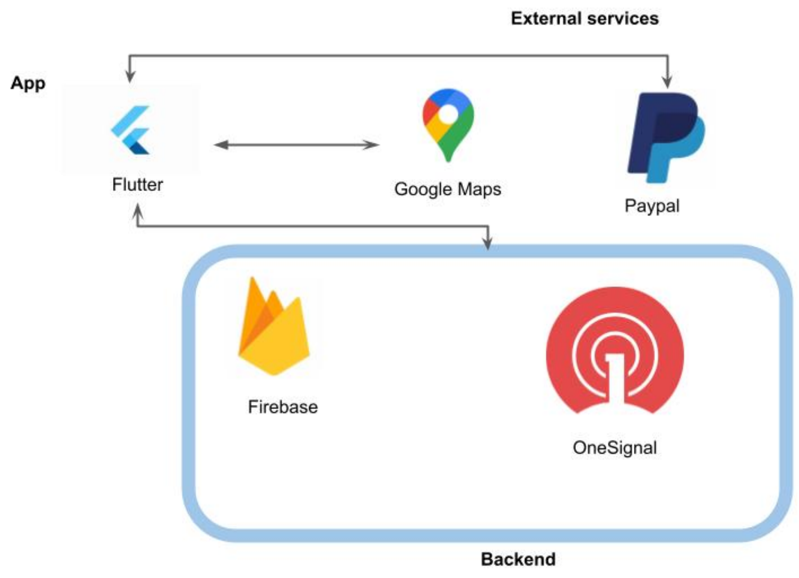
1.2 Concept

For this purpose we decided to create an android application that fulfills all of the desired functionalities. Users are able to see all the available parking spaces on the map with their corresponding prices. Users can also filter the displayed parking spots based on their attributes. When a parking spot is clicked, the corresponding information is displayed. Information like price per hour, area , electric charging station, whether the spot is covered, if there is video surveillance available and whether there is access for handicapped people. The rating by past renters is also shown among the displayed information. Afterwards the user can book the spot by choosing on of the available time slots displayed in a booking calendar. The user can also navigate to the parking space. Upon booking the user and the owner both get a notification with the booking confirmation and are able to start a conversation with our chat feature. Owners of parking spots could also put their parking spaces for rent and upload a picture of them. An optional survey will generate information about the user and allow the app to make recommendations for parking spots. After a booking is confirmed, you can see it in the history window, where they are separated in to ongoing bookings and closed bookings.

1.3 Feature List

- Book parking spots
- Upload and offer parking spots
- Reputation system in form of reviews
- Recommendation system by using the user's personal preferences collected via a survey
- Filtering parking spots by different features
- Notifications for bookings and chat messages
- Chat between owner and renter
- Payment option with PayPal
- Booking history to see your current and past bookings
- Navigation to the parking spot by redirecting you to Google Maps with the specific location

2 Architecture



3 Frontend

We used Flutter to build the user interface of our app. In the following subsections, we will delve into each single page and feature in more detail.

3.1 Account

In the following subsection everything regarding the account is explained, e.g. where and how the user can interact with his personal information.

3.1.1 Account creation

When opening the app for the first time, the users are first asked to allow the notifications for the app as seen in figure 1.a and then redirected to the page where they either sign up a new account, as it is shown in figure 1.d, or log in, as it is shown in figure 1.b. After signing up the information is sent to Firebase and a new account is created with the provided personal information. After signing up, the user gets redirected to a survey where he or she is asked to answer several questions about his or her personal parking preferences, i.e.: preference for parking spots with a charging station for electric cars, covered parking spots, parking spots with handicapped access and parking spots with video surveillance. After filling out the survey, the user gets asked to give permission to the location and afterwards gets redirected to the map view.

3.1.2 Log in

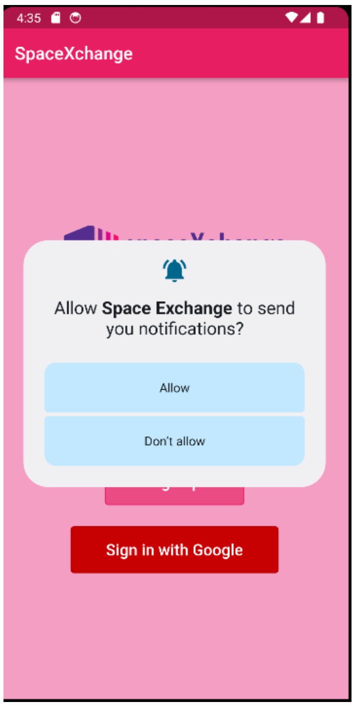
If the user already has an account, they can simply log in by entering their credentials which are sent to the database and get verified. If the login procedure is successful, the user gets redirected to the map view. After logging in once, the user stays logged in even after closing the app and he or she doesn't need to log in again.

3.1.3 Account Details

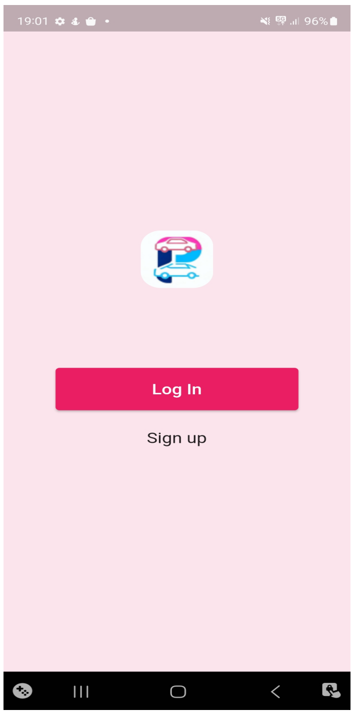
The Account page is accessible through the navigation bar on the bottom. By clicking on the account icon, the user is redirected to a page with two options, i.e. one button that redirects the user to the account details page and another button called ”My parking spots” that redirects the user to an overview of their parking spots. By clicking on ”Account Details” the user gets redirected to the account detail page. There, they will get an overview of their personal information and are able to edit them. Furthermore, at this page the user is able to log out, delete their account or fill out the survey again, which is important for the recommendation. Figure 1.f shows what the account page looks like.

3.2 Map View

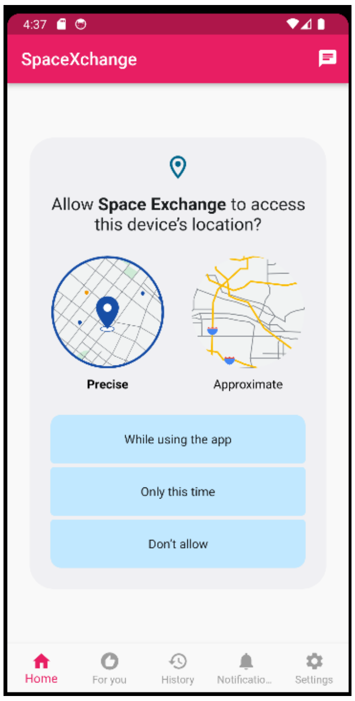
Logged-in users can enter the map view by click on the home button which is located at the navigation bar on the bottom of the screen. The map shows the user’s current location with a blue marker. Furthermore, all available parking spots with their prices are displayed as markers on the map. These parking spots can be filtered by clicking on the filter button. You can also search for specific locations with the input field at the top of the screen and you can also go back to your location by click the location button next to the filter button. By clicking on a marker, a small window will open from below with more information about the selected location, i.e. a little picture which can be expanded by clicking on it, the rating of the parking spot, the name, a description and its features. The user has the option to either navigate to the location by clicking on the ”Navigate” button or book this parking spot by clicking on ”Book now”. A picture of the map can be seen in figure 2.a.



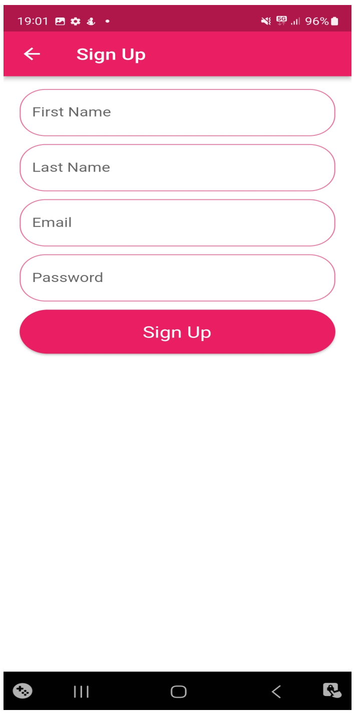
(a) Notification access



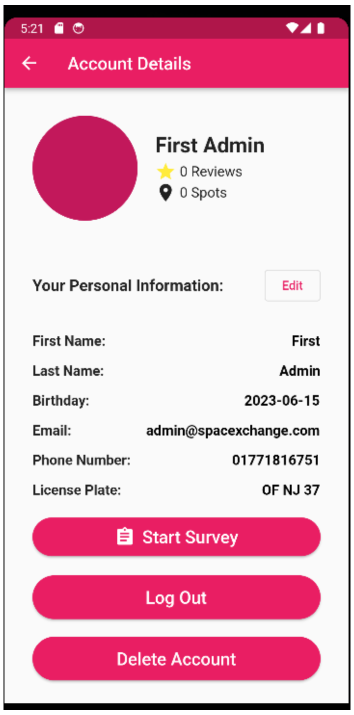
(b) Start page



(c) Location access



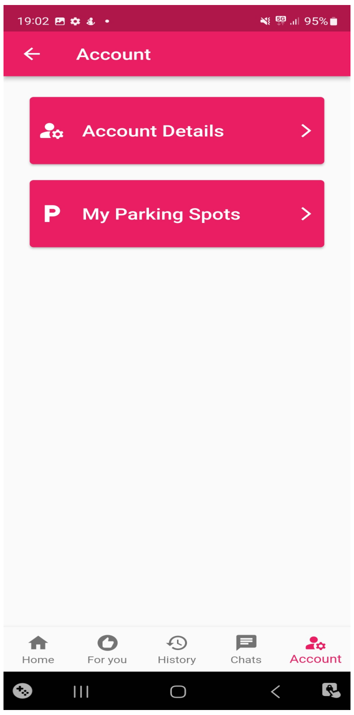
(d) Signing Up Info



Your Personal Information:

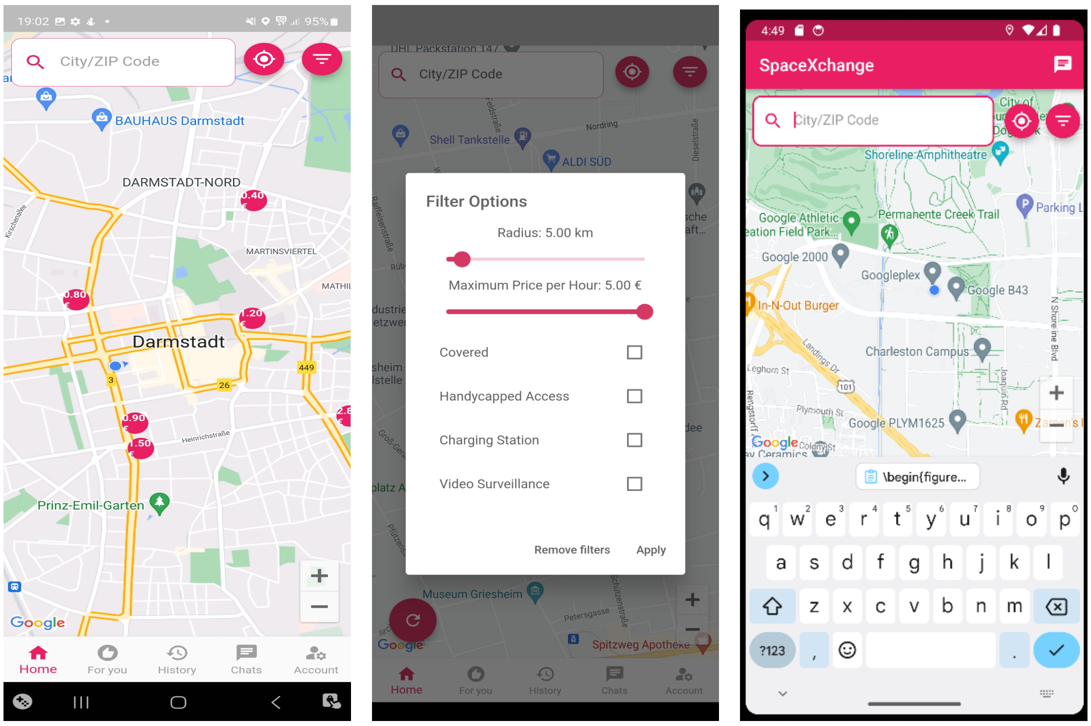
First Name:	First
Last Name:	Admin
Birthday:	2023-06-15
Email:	admin@spacexchange.com
Phone Number:	01771816751
License Plate:	OF NJ 37

(e) Account Details page



(f) Account page

Figure 1: Screenshots of Welcome page



Filter Options

Radius: 5.00 km

Maximum Price per Hour: 5.00 €

Covered ☐

Handycapped Access ☐

Charging Station ☐

Video Surveillance ☐

Remove filters Apply

(a) Map (b) Filters (c) Search

Figure 2: Screenshots of Home page

3.2.1 Filter

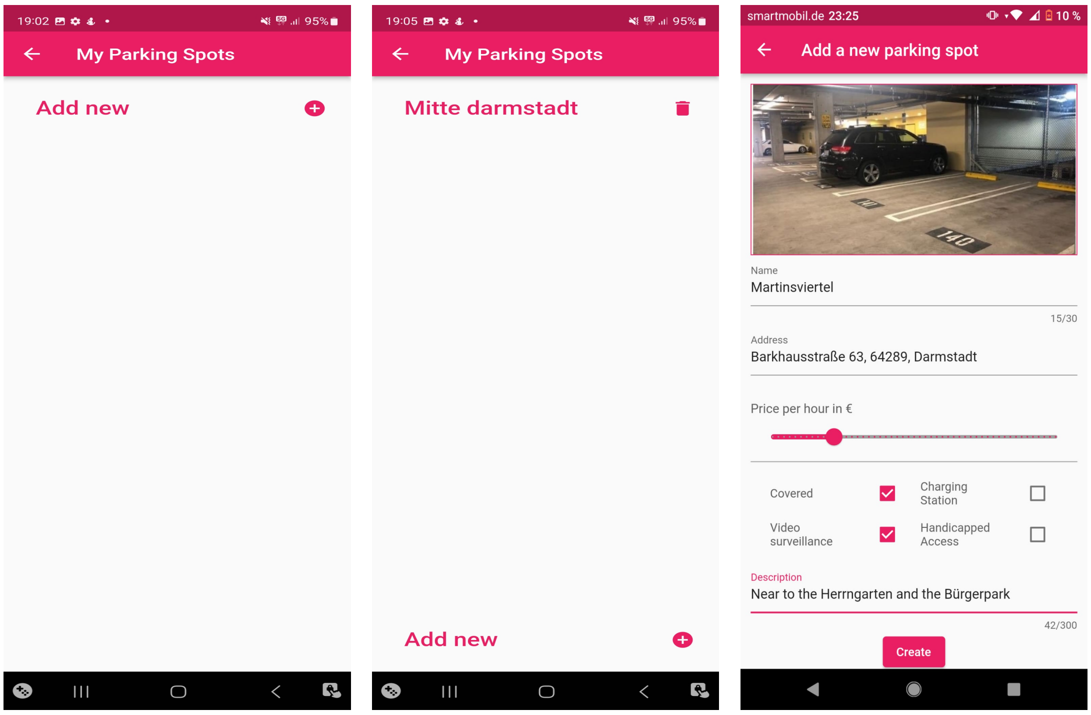
By clicking on the filter icon, the filter options will appear. There, the user is able to filter for different features, i.e. handicapped access, charging station for electric cars, covered parking spots and whether the parking spot is under video surveillance. The filter also offers the user to search parking spots based on the price. Figure 2.b offers a better visualization for the filtering tool. Each feature is displayed with a checkbox next to it in order to enable or disable it. The price is controlled through the slider. The filter is then either applied or reset. Users can also filter the spots based on a radius which is controlled through a slider in the filtering window.

3.3 Upload parking spots

Owners of parking spots can get an overview of their uploaded parking spots by clicking on the button with the caption ”My parking spots”, whose location is described in section 2.1. They can add another parking spot by clicking on ”Add new” (Figure 3). At this page, owners can create new parking spots with all necessary informations such as address, picture and description.

3.4 Booking

When the user clicks on a parking spot in the map, they will be shown the information of the parking spot in a pop up and at the bottom of the pop up the user can book the spot by tapping on ”Book now”, as it is displayed in figure 5.a. After clicking the ”Book now” button the user will be redirected to the booking page, where they will see a calendar with the available time slots marked as green and the booked slots with the color grey. When they select a time slot, it will be marked yellow until the booking process is done (see figure 5.b). This calendar widget was taken from https://pub.dev/packages/booking_calendar and was modified for specific needs.



Name
Martinsviertel
15/30

Address
Barkhausstraße 63, 64289, Darmstadt

Price per hour in €

Covered ✓
Charging Station ☐
Video surveillance ✓
Handicapped Access ☐

Description
Near to the Herrngarten and the Bürgerpark
42/300

(a) Empty list (b) Parking spots list (c) New spots

Figure 3: Screenshots of Parking spots page

3.4.1 Payment

After confirming the selected time slot, the user will be redirected to a PayPal page where they can insert their credentials and pay the amount directly. However, for demonstration purposes we inserted another button where you can skip the payment to try it out yourself. This is not a feature which is supposed to persist.

3.4.2 Notifications and Chat

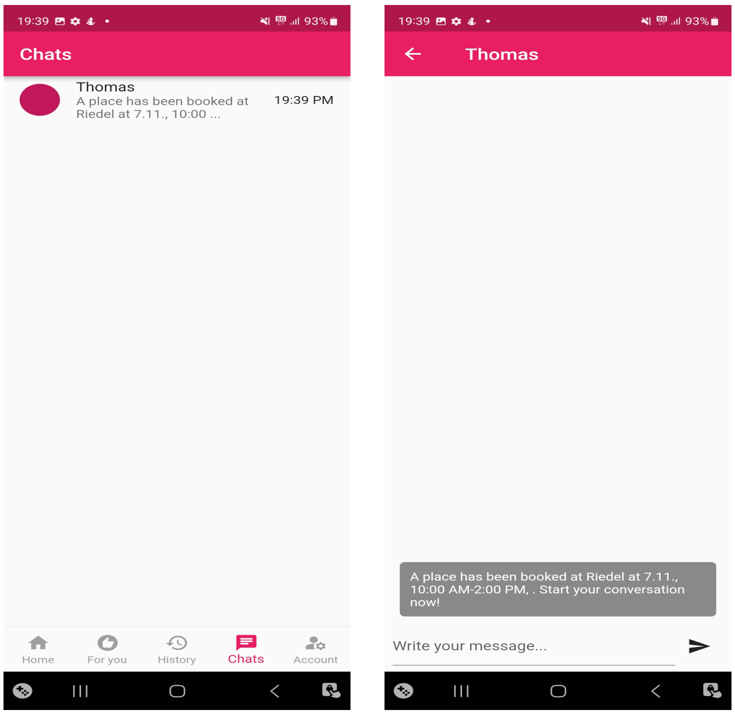
After confirming the booking process through the confirmation pop up shown in figure 5.c, the owner will receive a push notification and a chatroom will be opened for the user and owner to discuss anything regarding the booking, e.g. sending the license plate or car model to the owner for verification purposes. For every message that is sent, a push notification will be sent to the person that receives the message.

3.5 History

By clicking on the history icon at the navigation bar on the bottom of the screen, the user gets redirected to the history page, where the user has an overview about all current and past bookings. When a booking is confirmed, it is placed under the current booking section until the time expires, by which then it is placed in the past bookings section, as shown in figure 6.

3.6 Recommendation

After signing up, users have the opportunity to take part in a survey that allows them to get recommendations regarding parking spots based on their answers to the questions. Figure 7.a shows the pop up that appears upon signing up. Another option to take part in this survey is to access it via the account management page, as shown in figure 7.b. After completing the survey, users can see recommendations

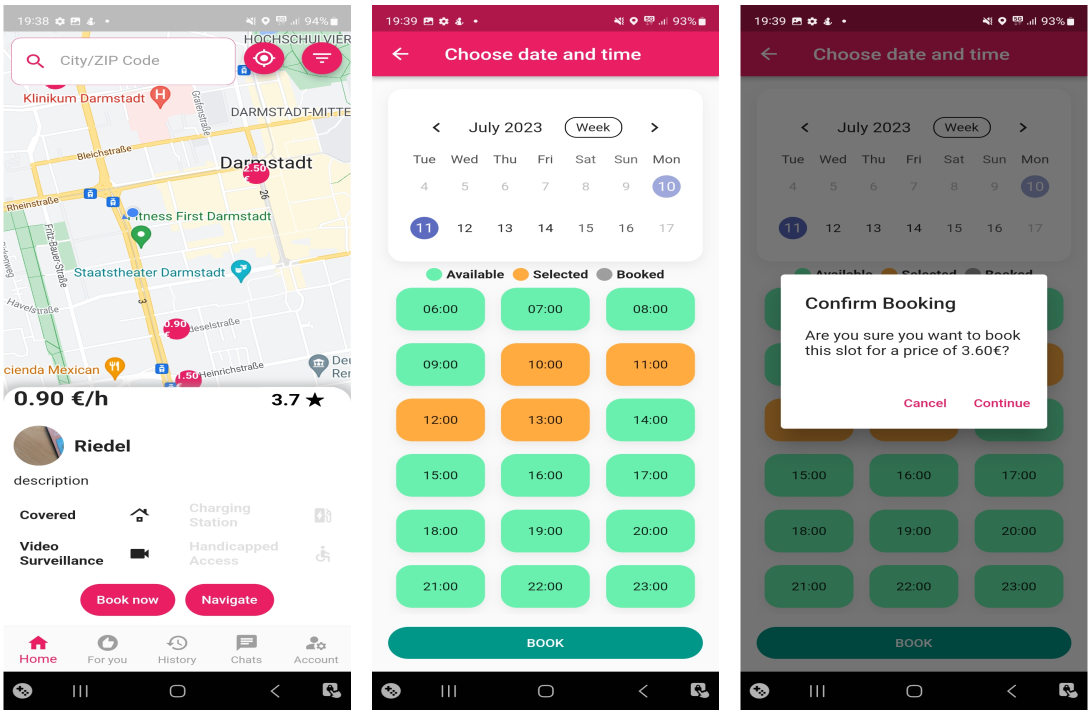


Thomas
A place has been booked at Riedel at 7.11., 10:00 ... 19:39 PM

A place has been booked at Riedel at 7.11., 10:00 AM-2:00 PM, . Start your conversation now!

(a) Chat notification (b) Chat conversation

Figure 4: Screenshots of Chats

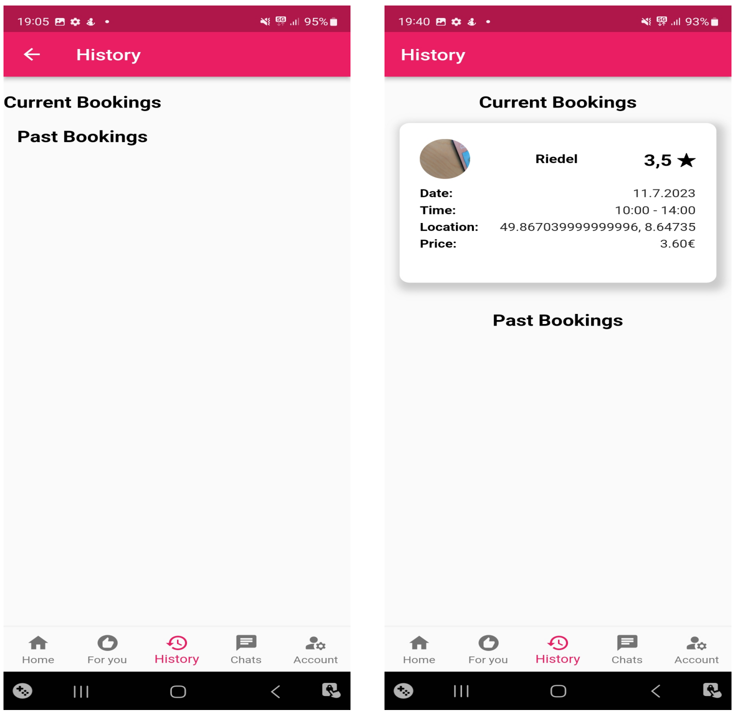


Confirm Booking

Are you sure you want to book this slot for a price of 3.60€?

(a) Parking spot view (b) Calendar (c) Confirmation pop up

Figure 5: Screenshots of Booking process



Riedel 3,5 ★

Date: 11.7.2023
Time: 10:00 - 14:00
Location: 49.867039999999996, 8.64735
Price: 3.60€

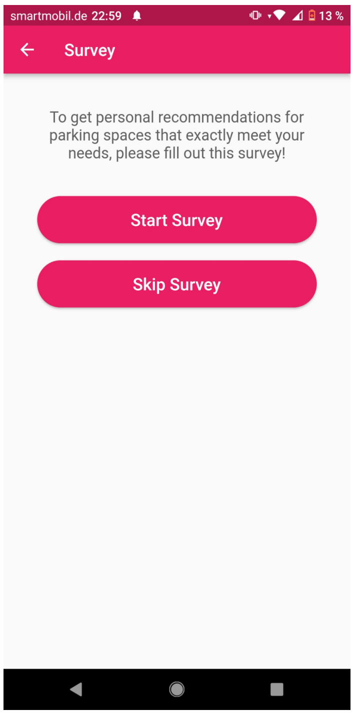
(a) Empty history (b) Current history booking

Figure 6: Screenshots of History

in the ”For you page” which, is accessible via the navigation bar at the bottom of the screen. The entries within the recommendation page are additionally sorted by preference. At the top, every parking spot is displayed that perfectly fits all of the user’s preferences. Afterwards, other parking spots are displayed that only cover a part of the user’s preferences.

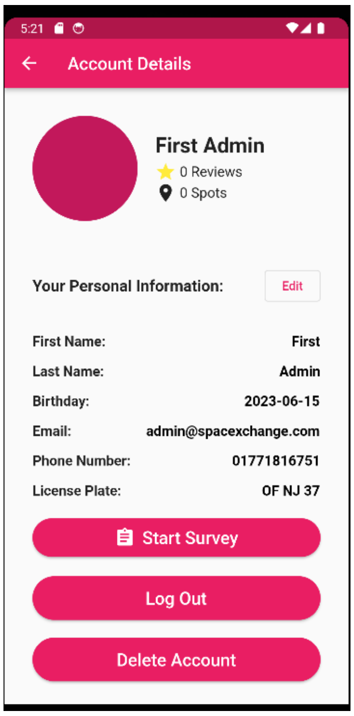
3.7 Review

Users are able to rate parking spots and write reviews about their experiences, so that other users have an idea and owners have the chance to improve their service based on the feedback. The rating feature is accessible by clicking on any parking spot and tap on the rating, or just a star, if there is no rating yet. The user will be redirected to a review page where he or she can see all of the past reviews and ratings. The user can add an own rating and review by clicking on the plus icon.



To get personal recommendations for parking spaces that exactly meet your needs, please fill out this survey!

(a) Survey



First Admin
0 Reviews
0 Spots

Your Personal Information:

First Name:	First
Last Name:	Admin
Birthday:	2023-06-15
Email:	admin@spacexchange.com
Phone Number:	01771816751
License Plate:	OF NJ 37

(b) Survey in the account page

Figure 7: Screenshots of Survey

3.8 Automated Testing

3.8.1 Testing Setup

To ensure the quality and reliability of our Android app developed with Flutter, we implemented automated testing. This allows us to verify that the app builds correctly and run unit tests to validate its functionality. We use the cirrusci/flutter:stable Docker image as the base image for our pipeline jobs. This image provides a stable version of Flutter for consistent and reliable builds and tests. Figure 8 depicts our CI/CD pipeline.

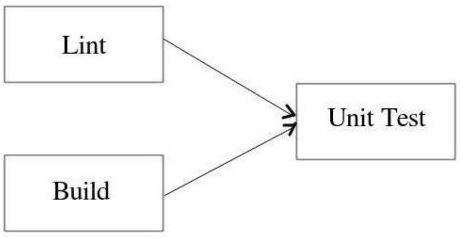


Figure 8: CI/CD Pipeline

3.8.2 Lint

Linting is an essential part of our development process, aiding in code quality and adherence to coding standards. By leveraging the Flutter Analyzer tool, we perform static code analysis on our Flutter app. The dedicated lint_job in our CI/CD pipeline executes linting checks using the cirrusciflutter:stable Docker image. This allows us to identify potential issues such as unused imports, style violations, and bugs early on. By incorporating linting into our automated testing, we maintain a high standard of code quality, enhance readability, and foster better collaboration among team members.

3.8.3 Build

The build_job is responsible for ensuring that our Flutter application builds correctly. This job is an essential part of our automated testing process. It performs the necessary steps to fetch dependencies and compile the Flutter app. By running this job, we verify that the app has been successfully built, which helps us identify any build related issues early in the development cycle. It runs on the gitlab shared-runner tagged with iptk and backup, ensuring that the job is running on appropriate resources. Once completed, the build artifacts are available for further stages in our pipeline.

3.8.4 Unit Tests

Unit tests are useful for verifying the behavior of individual functions, methods, or classes. The test package provides the necessary framework for writing unit tests, while the flutter_test package offers additional utilities for testing widgets. Unit tests that rely on live web services or databases can be problematic due to slow execution, inconsistent results and difficulty in testing different scenarios. To address these issues, we use the Mockito framework to simulate the behaviour of dependencies. By creating mock objects, we define expected behaviour, verify interactions, and perform reliable unit tests that do not rely on external dependencies. This approach enables faster execution, predictable results and comprehensive testing across different scenarios.

3.8.5 Integration Testing

Integration tests are used to test whole user experiences, e.g. uploading a parking spot. Two integration tests were implemented. The first one tests the user creation and deletion. The other one tests the upload of a new parking spot and its deletion afterwards. Recordings for both tests are added to the appendix.

4 Backend

For the database, we use Firebase as it offers many advantages such as real-time updates, analytics and crash reporting, easy integration and many more. Firebase also offers Cloud Firestore, which is a NoSQL document database that provides powerful querying capabilities and offline support. It allows storing structured data and retrieve it efficiently. In addition to Firestore, there is also Cloud storage that stores and serves user-generated content like images, videos, and files. It offers secure and scalable file storage with simple integration into the Flutter app.

4.1 Structure

Our application's backend primarily consists of five collections:

Users: This collection comprises documents containing three distinct groups of data. Personal information: This includes the user's last name, first name, email, etc. App optimization data: Additional keys are created post user's completion of a survey in our app. These keys store user behavior patterns, allowing us to provide an optimal user experience. Hardware data: Specific identifiers such as OneSignal Player Id, Token Id, etc., are stored here. Upon a user's first spot booking, a sub-collection called 'bookingsList' is created, where documents containing all necessary booking data are stored. Another sub-collection, 'parkingSpots', maintains the information about the parking spots offered by the user. The documents within this collection contain all the details about these spots, such as location, price, description, and unique technical data about the spot.

Chats: The documents here are kept simple, containing three keys - ReceiverId, SenderId, and a Map-Array. The Map-Array further contains the content of a message, senderId, and a timestamp. A new chat-document is automatically created in this collection following each successful booking unless one already exists.

ParkingSpots: The documents in this collection are identical to those in the 'parkingSpots' sub-collection under 'users'. The distinction here has another sub-collection, 'reviews', which houses review documents. Another sub-collection is responsible for populating our booking calendar with available time slots.

4.2 Authentication

Every signed in user is logged in using its ID, which is saved and managed through the Authentication extension in Firebase. Whenever users are logged in, they are connected to their ID in Firebase and all their data is then loaded to the app.

4.3 Security

We've built our app with user safety as a priority. To keep our user data safe and private, we used Firebase's security rules. We based our rules on rules_version '2' because it's easy to use, supports more detailed pattern matching, and works well with the Firebase Command Line Interface (CLI) for easy rule management.

A main rule we follow is that only users who have signed in can see or change data in our Firestore. This rule is shown in all our security rules, where a user must be signed in to read or write data.

To make our data even safer, we made extra rules for some collections. For example, in the users and bookingList collections, a user must be signed in and their user ID (given by Firebase Authentication) must match the document ID they're trying to see or change. This makes sure users can only see and change their own data, keeping other users' data safe.

We used a similar rule for the messages collection. Here, a user can only read or write a message if they are the sender or the receiver. This keeps users' private chats safe from other users.

Even though we've limited writing access for most documents to their owners, we've kept reading access for users who have signed in. This lets users see available parking spots, for example, but they can't change them.

We also added an extra security step by blocking writing access to any unspecified documents with the rule allow write: if false. This makes sure that no writing can happen outside of our defined collections and documents.

These security steps help to lower the risk of data breaches or unauthorized data changes. Users can use the app freely while owners keep control over their data, making our app safe and reliable for everyone

4.4 Notifications

For the notifications we used OneSignal for multiple reasons, the first being that its a cross-platform support for iOS, android and web. OneSignal has a rich notification feature list that allows to notify users using text, images, deep links and buttons. Furthermore it offers a powerful targeting option that allows targeting specific users based on their actions or features.

5 Possible Improvements

Possible Improvements for future developing could be firstly to enable the users to upload their profile picture. The current status of the profile picture is that a dummy picture is being used as the profile picture of the users. Enabling the users to manage their chats would also improve the functionality of our app. Another area where improvements would be beneficial is to empower the owners of the parking spots to alter the availability of their parking spots. Another improvement would be to develop an app compatible for IOS.

6 Outlook

At the end we learned as a group new technologies like Flutter and Firebase and how to utilize them to build our application. We transformed the requirements and wishes of the client into a functioning product. In the process we practiced the SCRUM process and applied it each step of the way, enabling us to successfully implement all features required from us. The next step would be to publish our application to Google Play and allow users to install it. We could also upload it to different platforms like iOS, which will not be hard, given that Flutter is a cross-platform framework.