IPKT WiSe 23/24
Technical Documentation
Group M — TripTip

TECHNISCHE UNIVERSITÄT DARMSTADT

Felix Bauer (2331324), David Henn (2399049), Mithusan Naguleswaran (2362856), Thai Binh Nguyen (2316080), Tim Carlo Päpke (2531454)

February 5, 2024

Table of Contents

1 Motivation

The genesis of our application stems from the tangible challenges encountered by a cohort of five enthusiasts during our collective voyages. Despite our shared passion for travel, we found ourselves frequently caught in uncomfortable predicaments due to deficient organizational acumen and disparate expectations regarding the trajectory of our excursions. Past travel experiences have demonstrated that conflicts regarding destination and planned activities significantly hinder the enjoyment of the journey for friends.

In response to these challenges, we resolved to develop an innovative application, aptly named "**TripTip**", aimed at surmounting these obstacles by establishing a user-friendly and secure platform designed to enhance collaborative travel experiences. This includes features such as creating surveys, uploading tickets, and managing financial aspects, thereby catering to the diverse interests of individuals.

Our motivation lies in extending a helping hand to fellow travelers, aiding them in crafting unforgettable experiences through seamless trip planning assistance.

2 Technical Overview

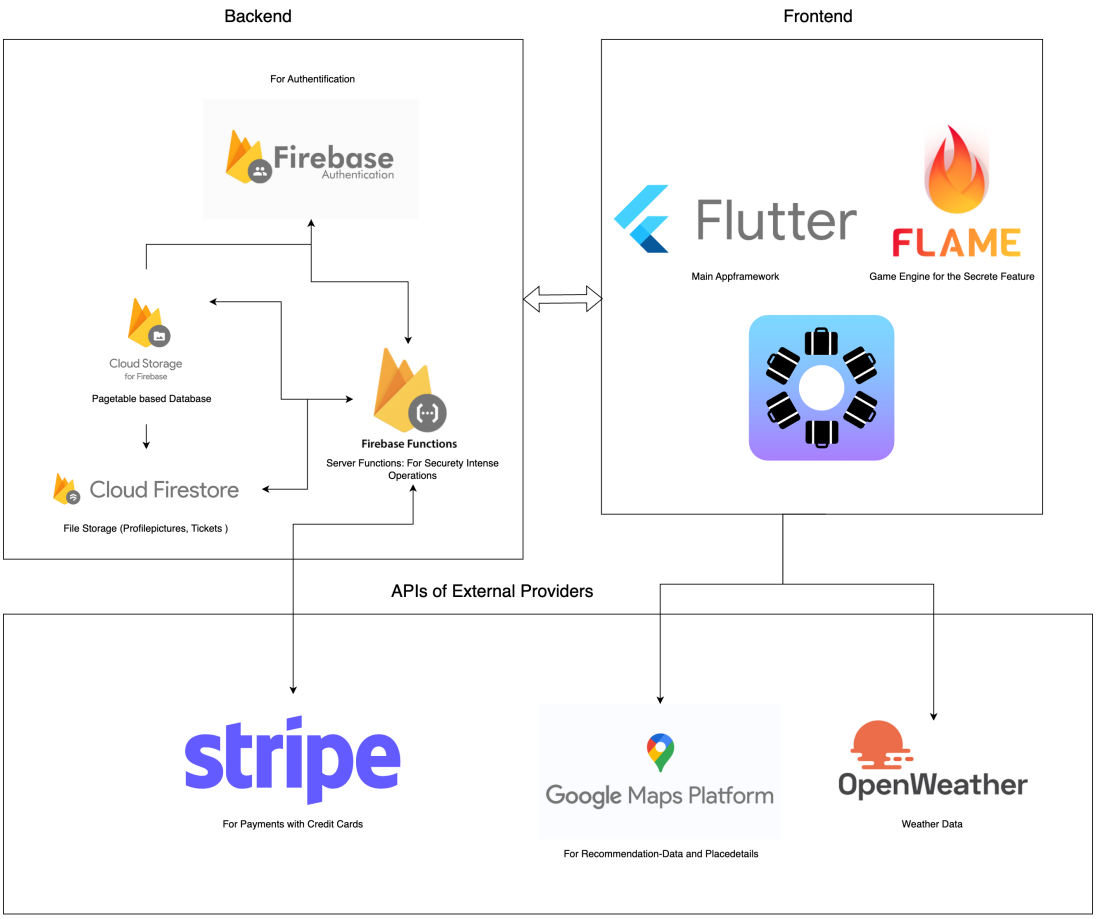

Figure 1: Tech Overview

Our app is developed in Flutter, as it provides us with a plethora of features to speed up development, namely highly customizable UI as well as hot-reload of the UI while the app is running. Furthermore, it has the capability to publish apps across platforms, giving us the freedom to build for other platforms, such as iOS or web browsers.

All changes occurring on the dashboard, tickets, payments and diary, are updated in real-time for all collaborating users. This is facilitated through the utilization of a Stream Builder in Flutter, which listens via Firebase to WebSocket.

We use Google's Firebase as our backend, mainly focusing on the real-time update and data storage portion of Firebase. We use Firebase Auth for user information and login credentials, Cloud Storage for file storage, mostly pictures, Cloud Firestore Firestore for non-file data, and Firebase Functions for restricted user capabilities, to protect user data from malicious intent.

In order to facilitate the implementation of the payment, maps, and weather information pages, we utilized Stripe to manage all of our payment transactions, Google Maps API to construct a functional map and its

functionalities, and OpenWeather to display the weather at the current location.

For the game, we used a Flutter-based game engine named Flame, which eases the development of games in Flutter by providing a game loop as well as a component-based system to handle a large amount of objects to render and compute.

By incorporating all of these toolkits, we created an app with plenty of features on a low-cost basis in a short time.

2.1 Frontend Architecture

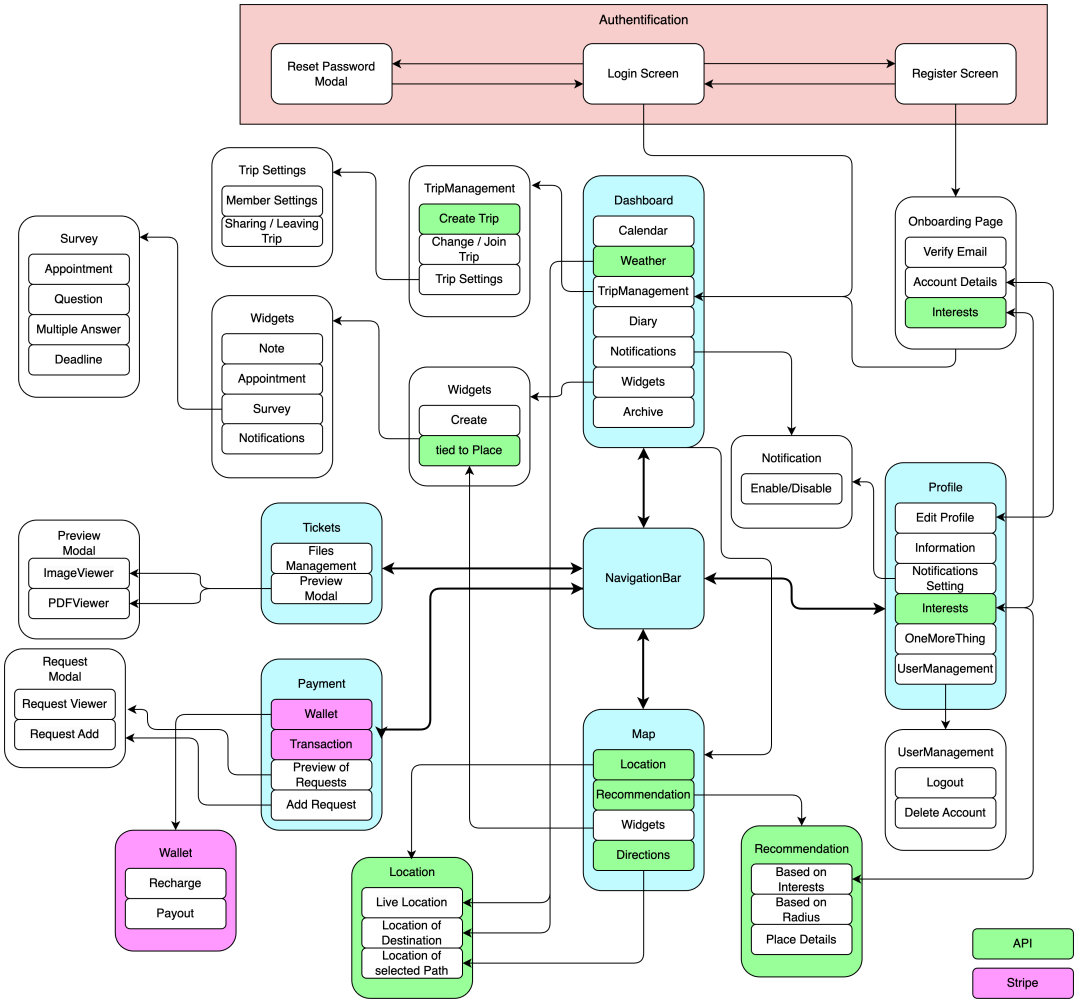

Figure 2: Frontend Architecture

As illustrated in Figure 2 the Frontend Architecture is comprehensive. The structure of the authentication is kept very simple. Users are provided with the option to create an account within the designated registration section. Should a user already possess credentials, they may avail themselves of the option to log in. Moreover, if users forget their passwords, the restoration procedure is also feasible.
The primary means of navigating through our application is facilitated by the navigation bar, which is linked

to the app's key features. Allowing users to move seamlessly between different pages and functionalities of the app with just one effortless click. Each distinct page associated with the navigation bar is distinguished by a highlighting in blue in Figure 2. Furthermore, each page and each specific function that uses an API are shown in green in the graph. Additionally, for features utilizing Stripe, a distinctive pink color has been assigned. We wanted to give the reader a short and simple impression of which features a third-party provider uses for their special functionality.

2.2 Backend Architecture

Firestore Architecture: For storing data, we use the Firebase Firestore database. Firestore is a table-based database that is divided into collections consisting of many documents. In the following, we illustrate the structure, but won't go deeper than surface level to explain the functionality.

- **Trips:** This collection stores every trip as a unique document with shared data for the trips, such as place details or members. Furthermore, every trip contains three sub-collections:
  - **Days:** which is utilized for storing each day and its widgets, either in an active widgets list or an archive list, as well as the daily diary. Due to performance reasons, we didn't use a sub-collection for the archive and active list.
  - **Payments:** This collection holds every money requests in a trip. Every request possesses several attributes, among which the most significant is the "to" list attribute, which stores the individual to whom the request is addressed.
  - **Tickets:** Each ticket is accompanied by a title and a reference to the stored document in the Firebase Storage.
- **Users:** Our users collection extends the given Firebase Authenticator collection, which stores user email, password and a unique ID. We add functionality like interests, a profile picture, payment information, a payouts subclass and the ability to store every open and closed payment of the users. To ensure the safety of someone's data, a user is only allowed to access his own documents.
- **Tasks:** This collection is utilized for the purpose of scheduling tasks. Every task is processed by the Firebase Cloud Function named "taskRunner". When the "taskRunners" sees that a document in the collection has the status pending and should be processed now, it will call the corresponding worker with the documents options attribute. Such workers are, for example, being used to open the diary function.

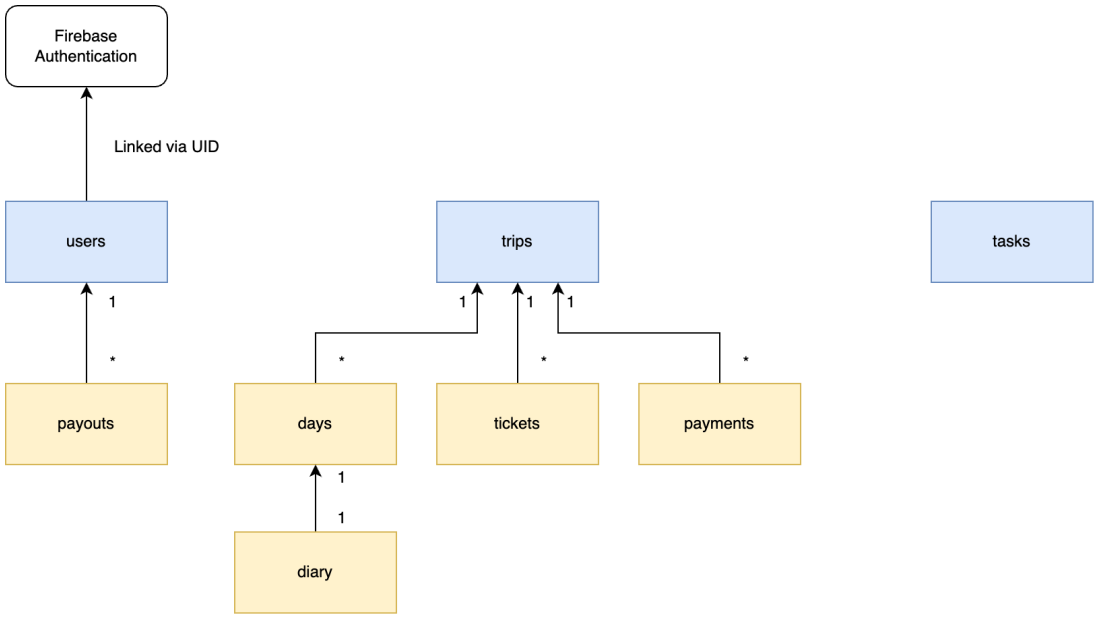

Figure 3: The Firestore Architecture

2.3 Storage

We use Firebase Cloud Storage as our storage solution, where profile pictures and tickets are stored.

2.4 Authentication

The Firebase Auth service was used for authentication. Firebase enables passwords to be stored securely and logins via OAuth service providers such as Google or Facebook. Furthermore, emails are sent for verification and users can request a link to reset their password.
The Firebase authentication service can also be easily integrated into Flutter. At the moment, users can authenticate themselves with an email and password and via Google. Unfortunately, we were not able to fully integrate a Facebook login, as Facebook would have to check our app manually. However, this check by Facebook would be the last necessary step, as we have already integrated the rest. After the user has authenticated themselves, the user data is saved in the User Collection in the Firestore. The profile picture is stored in the Firestore. If the user deletes their account, the Firebase Cloud function "deleteUser" is called, which deletes all payment requests and the user from all requests in all trips. Finally, the user is deleted from all trips. However, the user's widgets remain, as the other participants can decide for themselves whether they want to keep them.

2.5 Cloud Functions

Firebase Cloud Functions are small scripts that are executed on Google's servers. These can be written in either JavaScript or Python. However, as the support for Node.js(JavaScript) is significantly greater, we decided to use JavaScript. Firebase Functions allow executing security sensitive Operations like handling Payments Operations, where the End devices shouldn't handle the verification process. Firebase Functions are also essential when sending Push Notifications to the user's Devices. We have implemented the following functions:

- **taskRunner:** This function is used to execute various tasks in the task's collection at specific times. Every task is linked to a specific "worker", which is then executed at that point in time. If the worker has run without error, the tasks are deleted. Since workers are created via widgets, they are also deactivated, when a widget is moved to the archive. The following workers are available:
  - **Survey Conversion:** This worker is used to convert a survey widget on the dashboard into a widget corresponding to the survey result after a certain interval. A push notification is also sent to all Trip members.
  - **LastChanceSurvey:** This worker is used to remind all members to vote again before the survey expires or to check their vote.
  - **WriteDiaryNotification:** This worker randomly selects one of the group members to write a diary entry for the past trip day. Of course, only for days that are in the interval. It then opens the widget for this person by modifying the widget in the Firestore.
  - **AppoinmentNotification:** Whenever an event created via a widget occurs, a Push Notification is sent to all trip members.
- **joinTrip:** For security reasons, this worker is required for people to join trips, as people cannot write to trips in the Firestore that they are not members of. This worker checks whether the join code is correct and if so, the person is allowed to join.
- **leaveTrip:** This cloud function is required to send a push notification to all users when someone leaves a trip.
- **removeUser:** Used to delete accounts as described in the authentication part.
- **stripeRefund:** This function is used to create a claim as described in the payments section.
- **stripeEvent:** This cloud function is required for the webhook, as described in the payments section.
- **stripeAddPaymentsMethod:** This cloud function is used to create a new stripe user.
- **deleteRequest:** With this function, the user can delete their requests in a trip. If the request has already been paid, the corresponding amount is sent back to the user.
- **bookToBankAccount:** This cloud function is used to create payouts to users and create an entry in the payout's collection.
- **surveyCreated:** used to fire push Notifications when a Survey is added to Dashboard.
- **deleteOldArchivedWidgets:** Runs every Day to delete Archived Widgets, which are archived longer than 4 Days.

2.6 Notifications

Two plugins, *firebase_messaging* and *flutter_local_notifications*, are employed to integrate push notifications into the app. The *firebase_messaging* plugin establishes a link to the Firebase Cloud Messaging (FCM) service, allowing the reception of notifications. Cloud functions trigger the sending of push notifications based on specific conditions, such as the creation of a new post document. Once a push notification is received, the *flutter_local_notifications* plugin is utilized to showcase a banner featuring the title and content of the notification. All Notification which are bound to Dashboard Widgets are tappable, which will direct the user to the corresponding day on the dashboard.

2.7 Payments

Within the payments feature, users have the ability to submit requests to other users, who can then collectively reimburse them. The implementation proved to be a significant challenge, as there are few providers that offer peer-to-peer payments between accounts, especially ones suitable for our testing purposes. Ultimately, we opted for the Stripe API, which supports one-sided payments using credit cards in a testing environment. Outside the testing phase, stripe also provides additional payment options.

When a user fulfills a payment request from another user, the amount is solely recorded in a wallet shared by both users. If this wallet falls into a deficit, the user needs to recharge it by settling the outstanding amount using a stripe payment with their credit card. For this purpose, a stripe account is created first via the cloud function if the user does not already have one. The cloud function *stripeRefund* is used for this, taking only the user's UID as a parameter.

The function returns a Payment ID and a Security Code. With this information, a Payment Sheet contained in the Stripe Flutter library can be launched on the user's mobile device. This payment sheet allows the user to input their credit card information in a way that we cannot intercept, as the data is only sent to stripe. Stripe securely stores the information, allowing the user to reuse it for future transactions. The Stripe API allows the specification of Webhooks, which are called when a specific event occurs. We have developed the function *stripeEvent* for this purpose, with the URL address: *https://striperefund-j4qxu5zaoq-ey.a.run.app*, which can be invoked as required.
When a transaction is successfully processed, a corresponding related event occurs, prompting the cloud service to reset the wallet to zero. Since the stripe prohibits requests in excess of 0.50€, any request below this amount will not be processed. Therefore, such requests are increased to 1€. After the payment, the user can then withdraw the surplus from the wallet.

If the user has a positive balance in the wallet, they can initiate a withdrawal. This triggers a sheet where they can input their bank details. Once a user has filled in his Payments Information, it will be stored in Firebase, for Payouts Purpose. In the user subcollection, the withdrawal is added to the "payouts". It's important to note that this payout is only mocked, meaning that nothing further happens after adding it to the collection. For a functional withdrawal system, further integration with a bank or another payment service provider would be required. However, within the scope of such a project, achieving this would not be possible without the necessary legal legitimacy.

2.8 Basic Security

For security, we employ Firebase Rules [1] for both Firebase Cloud Storage and the Firebase Firestore database. In Firestore, we have restricted write access so that users can only write to trips in which they are members. The joining of a trip must be done through a Cloud Function. Users are restricted to editing their own user document, and within that document, they are permitted to modify all fields, except for the balance, in order to prevent any unauthorized modifications to their allocated amounts. Users are also prevented from creating a user subcollection named "Payouts" and writing to it. All these security measures are enforced through Cloud Functions.
For storage, the verification process only checks if the tickets are not larger than 5 MB and if the user is a member of the trip. For more Information about the Firebase Rules, look up at the Appendance, which is on the last Page of the Documentation.

[1] Firebase Rules

2.9 Recommendation System

For the recommendation system, we utilize the Google Maps Nearby Search API. The API allows specifying certain interest types, which are grouped into various categories found Google API [1] . Users can select from a set of categories we have deemed meaningful on the SetInterests page. Through an algorithm, subtypes are then determined and stored in the user document. As querying with more than 50 subcategories would be too much for Google, the types are divided into two lists and sent to Google. The suggestions are then presented on the Maps page.

2.10 Abuse Protections

If the login or registration encounters three unsuccessful attempts, the user is required to authenticate themselves through Google reCAPTCHA to verify their authenticity as a genuine individual and not a bot. In trips, users can create a maximum of 100 widgets per day. Trips have a maximum duration of 100 days. A maximum of 100 tickets can be uploaded per trip. There is always a person who is the trip administrator (the creator during creation). Users can be removed from the journey by the creator, as well as all their widgets. When a widget is removed, it is first moved to the archive. It will be deleted from the archive only 4 days after removal.

2.11 Navigation Bar

For traversing through our app we implemented a navigation bar at the bottom, to shorten the time to traverse to each page, as well as split our app into different pages with distinct functionalities, as it improves user experience in not handling everything on one big page. The Navigation Bar is split in 5 sections: Dashboard, Payments, Map, Tickets and Profile. The functionality of each page will be explained in the User Manual. While the user is on a page, the currently shown icon for the corresponding page will be white, while every other icon will be black, to signal which page the user is currently on.

3 User Manual

3.1 Login Page

When the user opens the app, they encounter the login page, as shown in screenshot (4). If the user has already created an account, they can enter their email address and password to log in, immediately accessing the dashboard page. If the user forgets their password, they can click on the "Forgot Password?" tab and enter their email address to receive a password reset link via email (5), allowing them to create a new password. If the user enters incorrect login information three times, they must solve a reCAPTCHA at the next login (6). Additionally, there is an option to log in with Google, unless the user has previously registered with Google. The capability to log in via Facebook has been not only conceptualized but also implemented within the system. However, due to the prolonged processing time required to obtain approval from Facebook Live for the login functionality, the option is currently suspended. They can be redirected to the register page by clicking the "Create a new Account" button to create an account.

[1] Google API

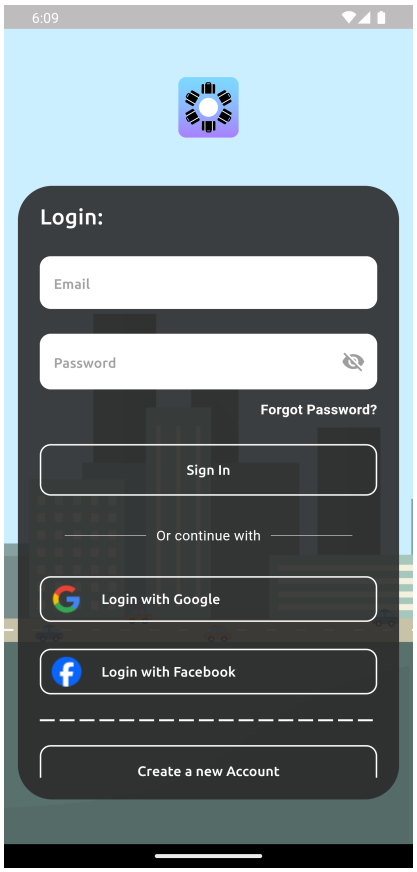

Figure 4: Login Page

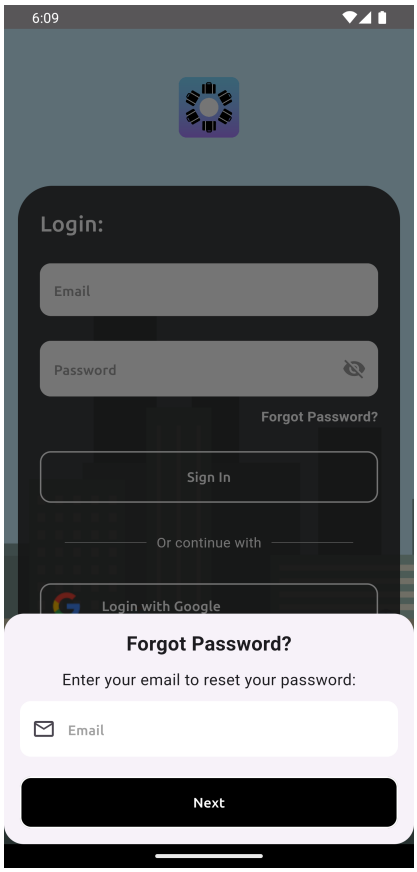

Figure 5: Reset Modal

Figure 6: reCAPTCHA

3.2 Register Page

Upon accessing the registration page (7), users input their email and password, validating the information through a subsequent entry to establish an account. Upon accurate data entry, users are seamlessly directed to the verification page. An alternative registration pathway via Google facilitates users in bypassing the verification step, granting direct access to the account details page. The capability to register via Facebook has been not only conceptualized but also implemented within the system. However, due to the prolonged processing time required to obtain approval from Facebook Live for the login functionality, the option is currently suspended. For existing users, a prompt to return to the login page is available via the "Already have an account?" button.

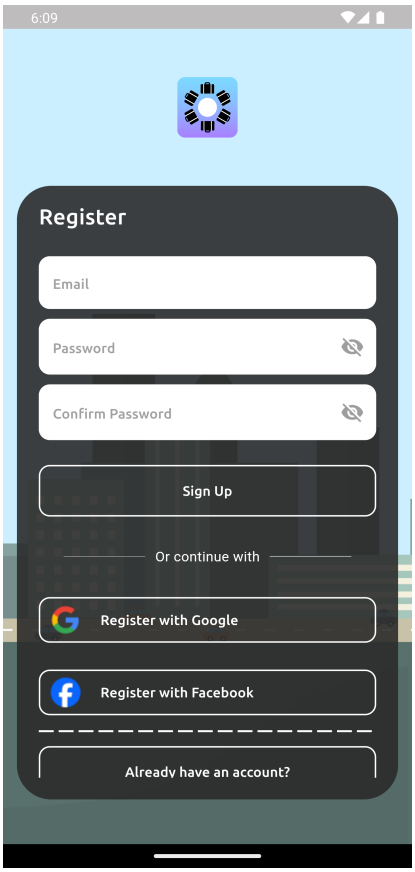

Figure 7: Register Page

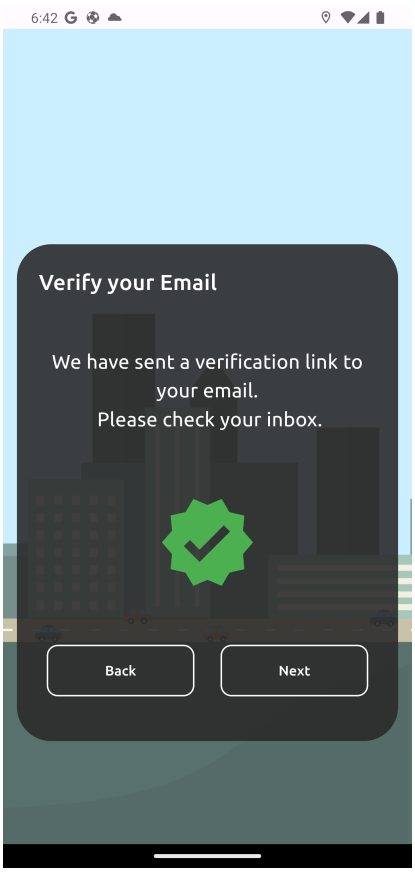

Figure 8: Verify Page

3.3 Verification Page

On the verification page, if the user incorrectly enters their email address, they can use a back button to return to the login/register page. Otherwise, the user receives a verification link via email, which they must confirm. Clicking "Resend Link" sends another email in case the first one wasn't received. If the verification is successful, the verification symbol turns green, and the "Resend Link" button becomes a "Next" button (8), leading to the account details page.

3.4 Account Details Page

On the Account Details Page (9), the user can set their profile picture, enter their first and last name, as well as their date of birth. When a user registers via Google, their first and last name are automatically extracted from their Google account. However, their birthday information is not retrieved during this process. Additionally, the user can view their registered page and copy their user ID. Clicking the "Select your interests" button takes them to the next page.

3.5 Interests Page

On the interests Page(10) the user can freely select their interests. The selected interests are then transmitted to the Google Maps Nearby Search API. Those not selected will not be displayed on the map. The chosen interests will be shown on the map. Clicking "Finish" takes the user to the trip management page.

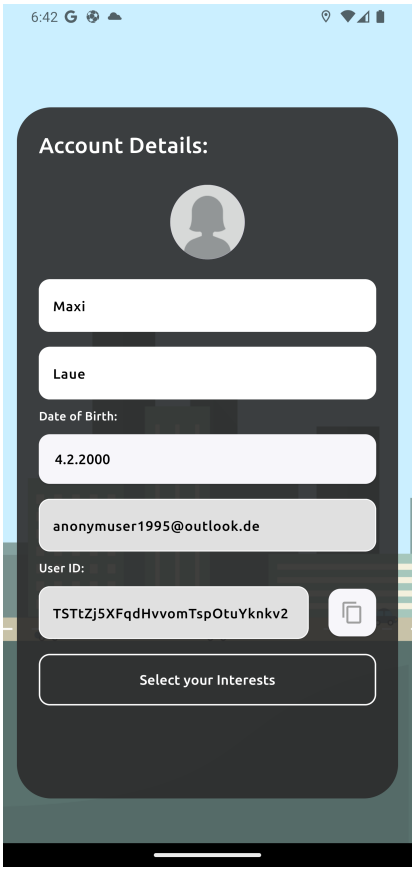

Figure 9: Account Details

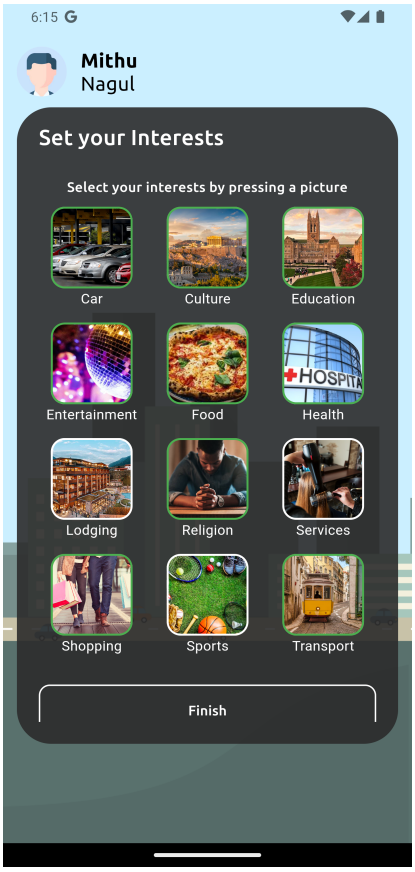

Figure 10: Interests

3.6 Trip Management

In the Trip Management interface, as illustrated in the provided image (11), the top bar features a back arrow button on the far left, allowing users to return to the Dashboard page. By selecting the plus icon, users navigate to the selected page (14) where they can either create a new trip or join an existing one. Additionally, users have the option to cancel these actions, returning them to the management page. An icon depicting a crown denotes the user's admin status. The current trip the user is engaged in is highlighted by a blue border. Furthermore, users can easily switch between trips by simply clicking on the desired trip. Swiping left (11) enables users to remove trips from their list, indicated by a red delete icon. Tapping the blue share icon directs users to a page where they can copy the TripID and add members. The middle icon within the swipe functionality reveals a bottom sheet displaying group members. If the user is an admin, they can further modify members status, delete widgets, or remove members from the trip by swiping left again(13).

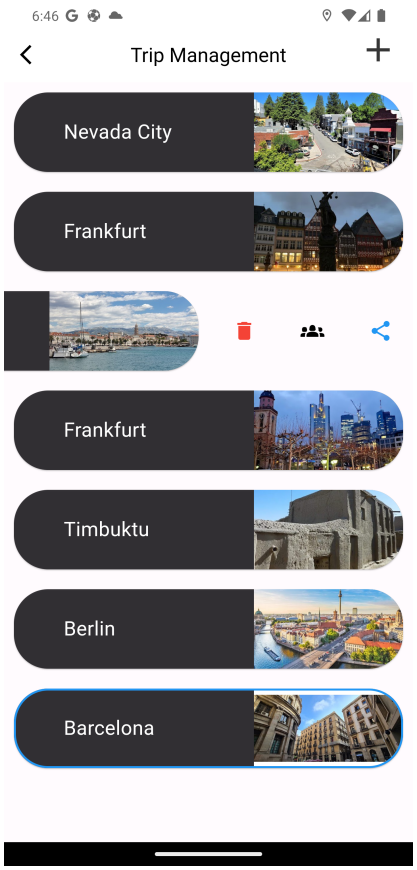

Figure 11: Trip Management Page

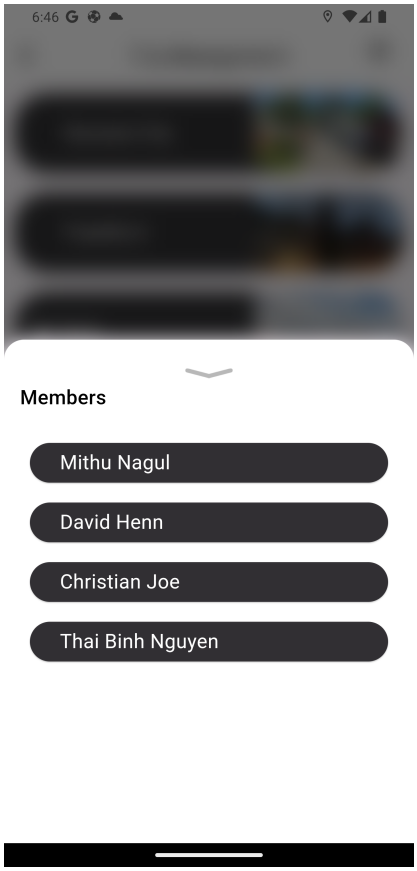

Figure 12: Member of the Trip

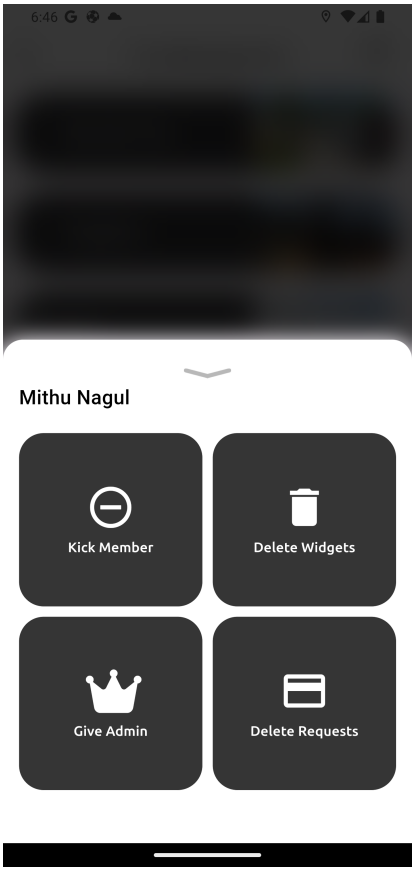

Figure 13: Member Feature

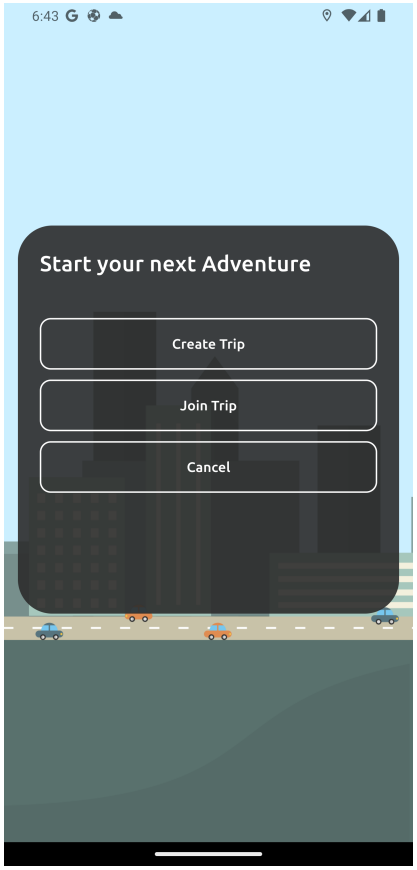

Figure 14: Select Trip

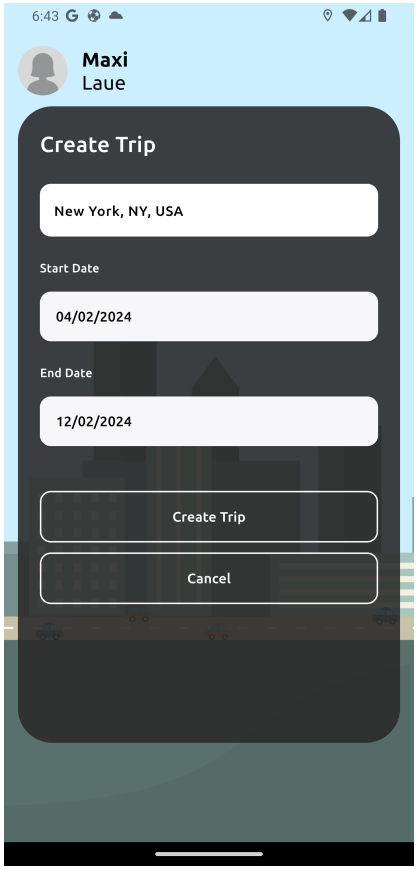

Figure 15: Create Trip

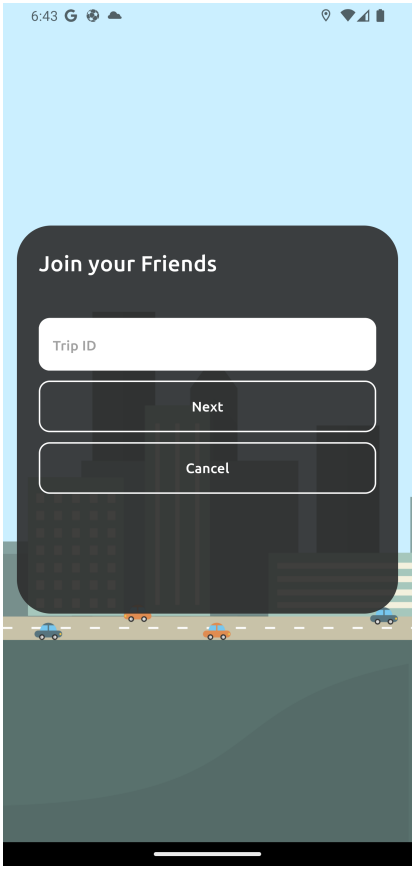

Figure 16: Join Trip

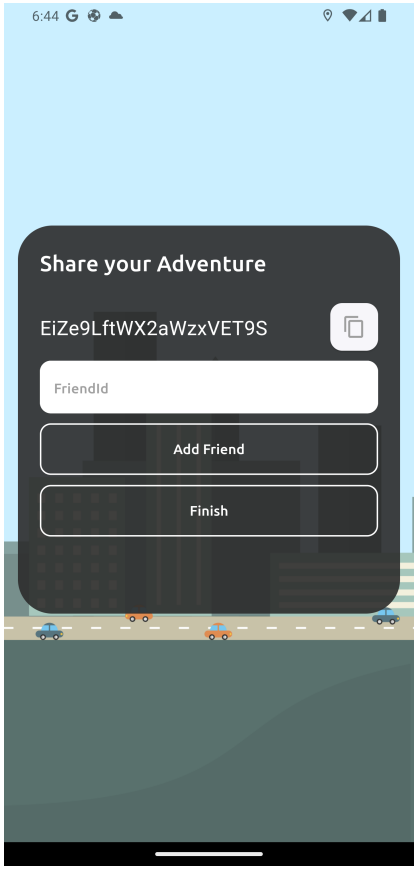

Figure 17: Share Trip

3.7 Create Trip

Within the Create Trip page (15), users view their first and last names along with their profile picture. Additionally, users select the trip's location, facilitated by an autocomplete feature. In our App, Trip Locations are restricted to Locations. Start and end dates for the trip must also be specified. However, the trip duration must not exceed 100 days. Upon clicking "Create Trip," the trip is generated and the user can either add their Friend with their UID or copy the TripID, while the "Cancel" button returns users to the previous page.

3.8 Join Trip

In the Join Trip page (16), users input the TripID and confirm their selection using the "Next" button, which redirects them to the Trip Management page. Alternatively, selecting "Cancel" navigates users back to the previous page.

3.9 Share Trip

The Share Trip page (17) is used to copy the TripID, as well as enter a friend's UserID to add them to the trip with which the page was opened. When tapping on the button on the right of the ID, the whole TripID is copied to the users' keyboard clipboard. Tapping the Add Friend button will add the person with the given UserID, if it exists. The back button lets the user return to the trip management page.

3.10 Dashboard

When you open the app and are already logged in, the dashboard always appears. The dashboard can be divided into three main sections: The top bar, the widget list and the navigation bar. As already mentioned, trips in our app have a certain duration, which can be defined when the trip is created. On the dashboard, the trips are divided into their individual days. Firstly, we will look at the first part: the Top bar.
At the top left of the Top bar is a weather Icon (18)(1), which shows the weather for the selected trip location. In the center of this is a label showing the trip location (18)(2). By pressing on this Icon, you are lead to a weather page (24), where you can get more detailed Information for the corresponding Location and your current Location.
To the right of this is a hamburger menu, which takes you to the trip management page. You can also use this menu to add new widgets (23) to the currently viewed day as well as go to the archive page where you can restore widgets that have already been deleted (19)(7). Below this is a slider which you can use to determine which day is currently displayed.
This slider has different functionalities:
If you are currently in the trip interval, the current day is selected. Otherwise, the first day of the trip is selected. In the first case, Today is also displayed above the center circle and yesterday or tomorrow above the circle on the left and the circle on the right. If the interval displayed above the three circles is outside the current day for the user of the trip, three consecutive dates are displayed above it. You can move forwards or backwards in time by clicking on the left and right circles. In doing so, one day is always added to or subtracted from the selected day. By clicking on the center circle, a date picker appears which allows you to freely select a day from the trip interval. (18)(5) The left button is consistently utilized to choose the trip's start date when the user is not currently within the trip interval. However, if the user clicks the button while within the trip interval, they will be redirected to today's date.(18)(4) The right button can be used to modify the interval in which the trip takes place; this takes you to a corresponding page.(18)(6)

Now we turn to the second part, the Widgets List view: This is the largest part of the page. All widgets for the currently selected day are displayed here. If the selected day is in the Past, the list view is locked and can't be edited. By default, every day, has a Diary widget. which is also the only not deletable widget. If you swipe left on a widget, users have the option to delete or edit their own created widgets or those created by their group members(22). Additionally, by dragging them, users can adjust the order of all widgets except the diary Widgets and Survey results.
In addition to the diary widget, there are three different widgets: a normal note that is stored on the dashboard, which has a title and a description. Appointments, these have a time within the selected day, a title and a description. When the appointment occurs, a push notification is also sent to all travelers. Last but not least, there is a survey widget: This has a title, a description and the survey points for which each travel group participant can decide. When creating polls, you can decide whether you want to allow multiple votes for each participant or just a single vote. Polls can be created for a regular question or a time point. You can also set a deadline, after which the poll should be closed for everyone. After the deadline occurs, a corresponding widget is created for the polls, an appointment widget for time polls and a note widget for polls with regular questions. Furthermore, a push notification is sent to all group participants informing them about the survey results.
When Appointment widgets or Survey widgets are associated with a place, a map-symbol is shown on the right side. If someone presses on that symbol, he or she will be redirected to the Map Page showing the Place he or she pressed on. On the Bottom of the page (Part 3), one can see the navigation bar.

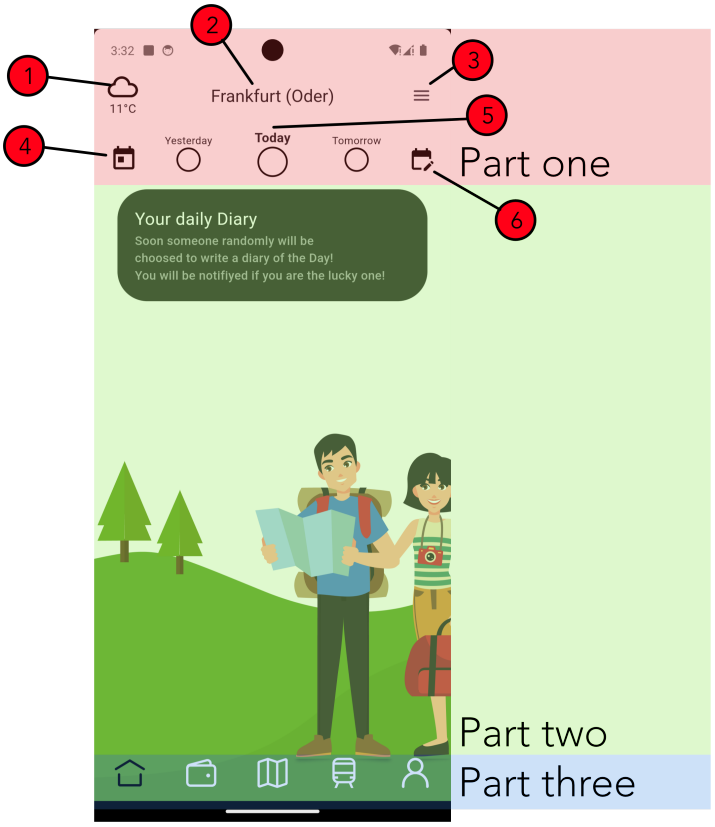

Figure 18: Dashboard

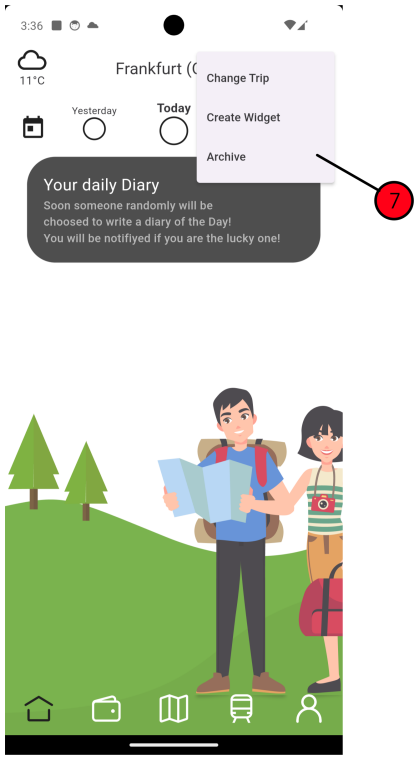

Figure 19: Menu

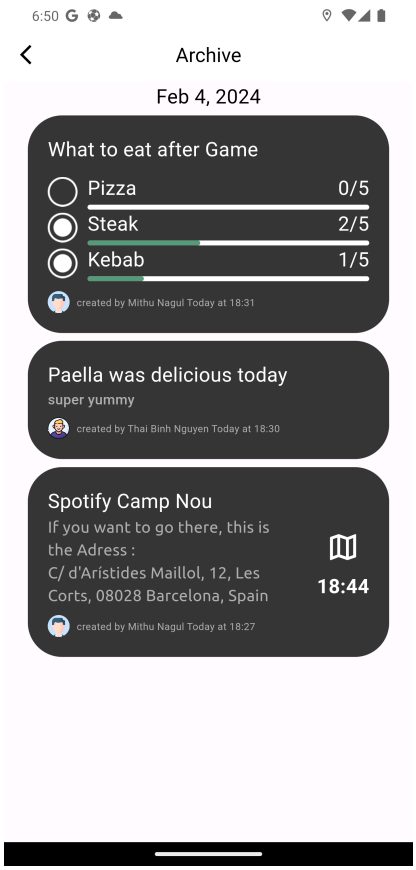

Figure 20: Widgets Archive

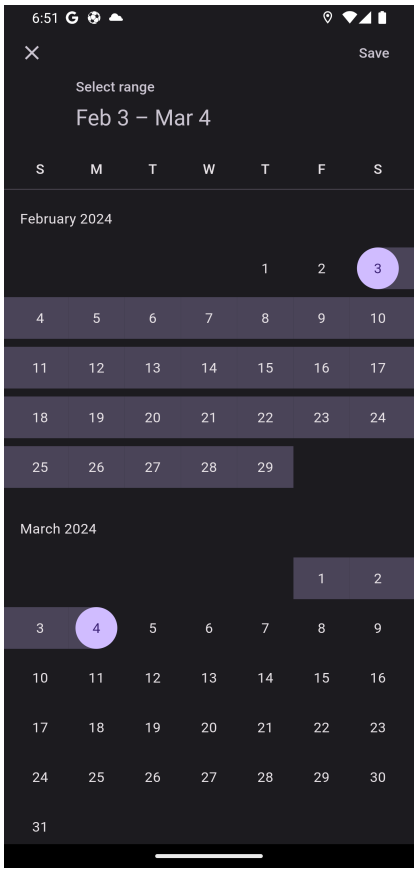

Figure 21: Calendar

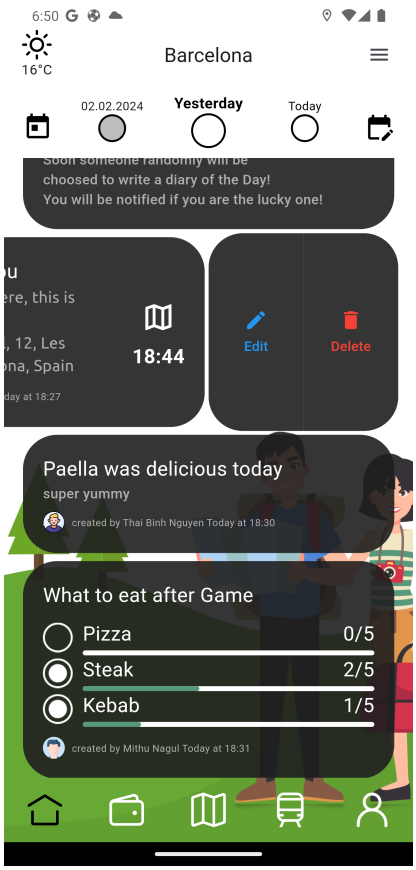

Figure 22: Editable Widgets

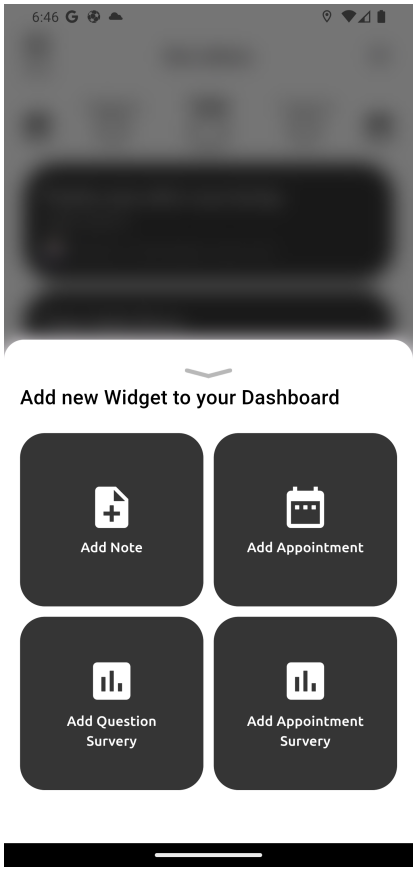

Figure 23: Dashboard Widgets

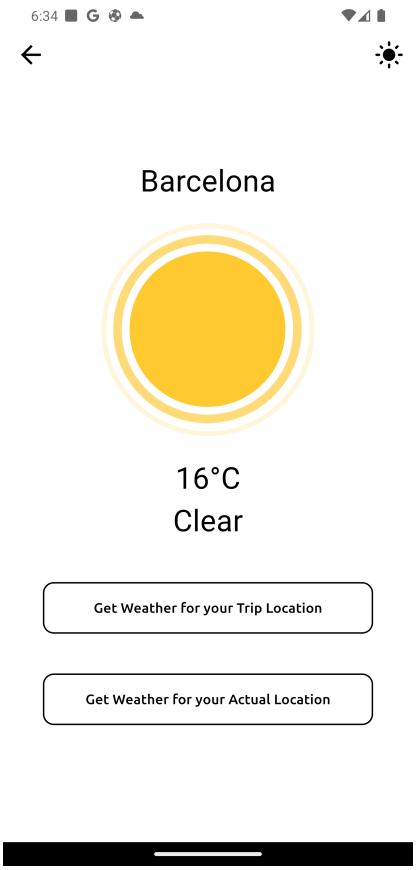

Figure 24: Weather Page

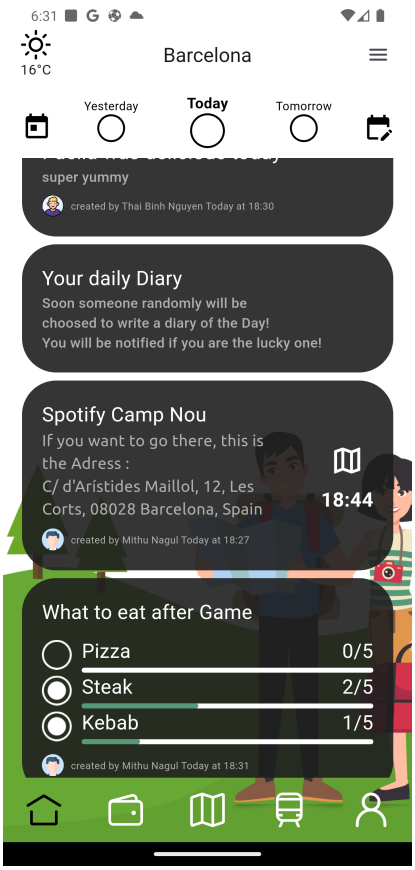

Figure 25: Dashboard Page

3.11 Diary

The Diary Widget opens for each day only on the following day. From the group lists, a user is randomly selected to write the diary for the selected day. For this user, a specific 2-hour time window is randomly chosen between 8 AM and 10 PM throughout the day. The user has access to a rich text editor(26). Meanwhile, all other users can track changes in real-time while the user writes the diary. After the time window closes, every User can read the diary(27).

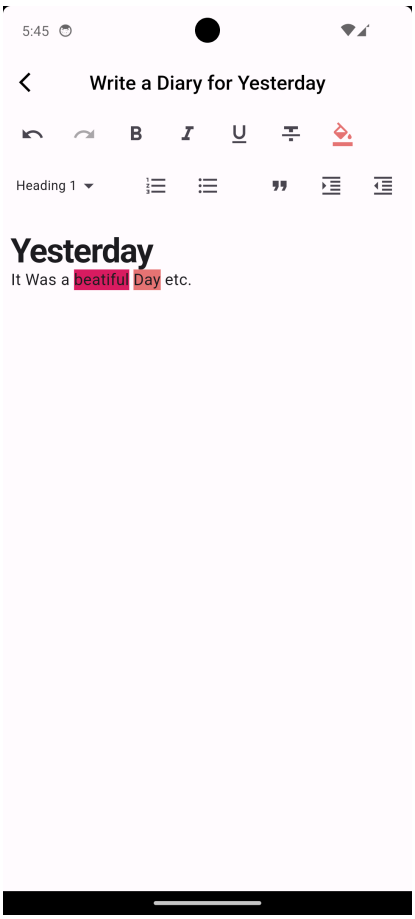

Figure 26: Write a Diary

Figure 27: Read a Diary

3.12 Payments System

On the Payment Page, as illustrated in the attached screenshot(28), users can click the "+" icon in the upper right corner to create a financial request for other members within the trip. This action opens a Bottom Sheet (29), allowing the user to input a title and optionally a description for the request. Additionally, the user is prompted to set a total amount. Upon selecting the "Next" button, the user proceeds to the next page of the Bottom Sheet.
Additionally, (30) users can individually select members under the "Select a member" tab. Upon selecting a member and tapping on the "+", the member is added to the list, allowing users to assign them a specific amount. Furthermore, users have the option to select a member to take over the total amount using the game feature, identified by the distinctive purple icon. This will be elaborated further in the feature section later on. Members can be deleted by swiping left, with amounts updating instantly. The user can use the "Back" button to modify the previous Bottom Sheet, and upon clicking the "Finish" button, the request is created.
The user can either choose to select all trip members and evenly distribute the total amount among them, including themselves, by checking the "Share Equally All" checkbox (31). The amounts can be distributed evenly among the chosen members by checking the "Share Equally"(30) checkbox or manually entered.
Upon returning to the Payment Page, the user will observe a "Card icon" (32) positioned in the top left corner. By clicking on it, a Bottom Sheet will appear, providing the user with an explanation that recharging can only be done with a credit card, whereas refunds can be processed using bank details.
Below the top bar of the app, the user sees their current balance. If positive, the "Book to my Bank-Account"(28) button allows them to input their account holder information, IBAN, and bank code for the amount to be transferred to their account. If the balance is negative, the button becomes "Recharge" allowing the user to reset their balance to zero by entering their credit card details. Under the "Your Requests"(33 and 34) tab, the user views created requests with titles and amounts. Clicking a container reveals the created request, but text fields and Checkboxes are not editable.
Under the "You owe"(35) tab, users can view requests from other members, which include names and amounts. By clicking on a container, it expands to reveal titles and amounts created by the respective member. To settle the debt, users can utilize the "Slide to Pay" feature. It’s important to note that users can only settle all debts at once; there is no option to settle individual debts selectively.

Hint: You can test the Stripe with the Visa Testcard Number: 4242424242424242 and a random number for the other fields. For the Payouts you need to fill in a valid IBAN, which is checked via the Checksum: Just fill in this IBAN for test purpose: DE02370502990000684712

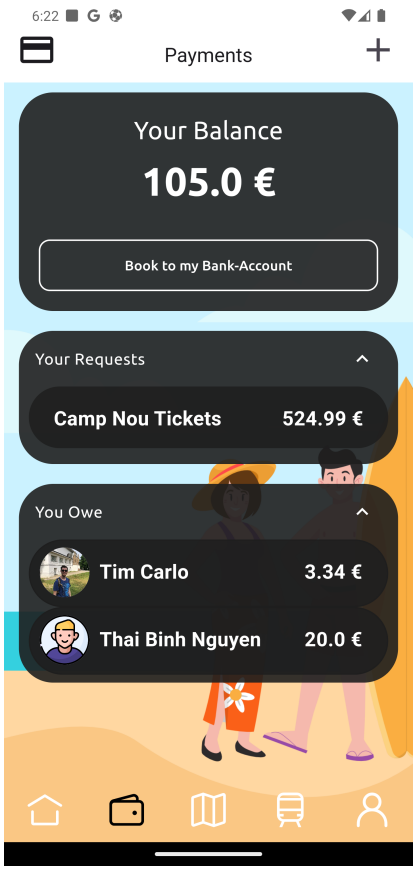

Figure 28: Payment Page

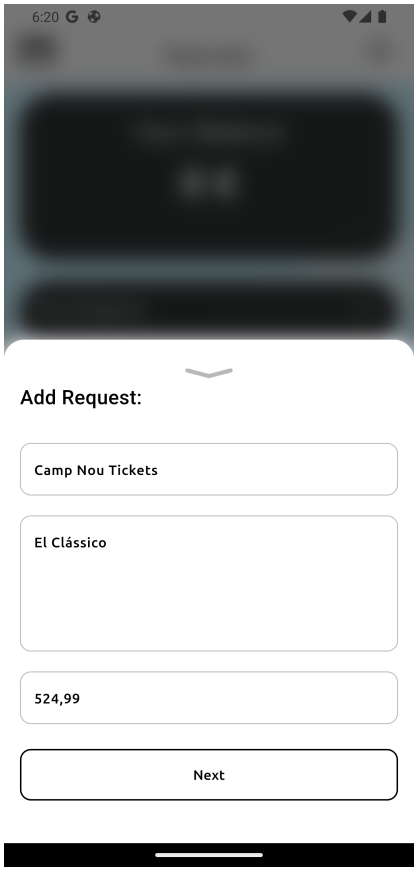

Figure 29: Request Modal

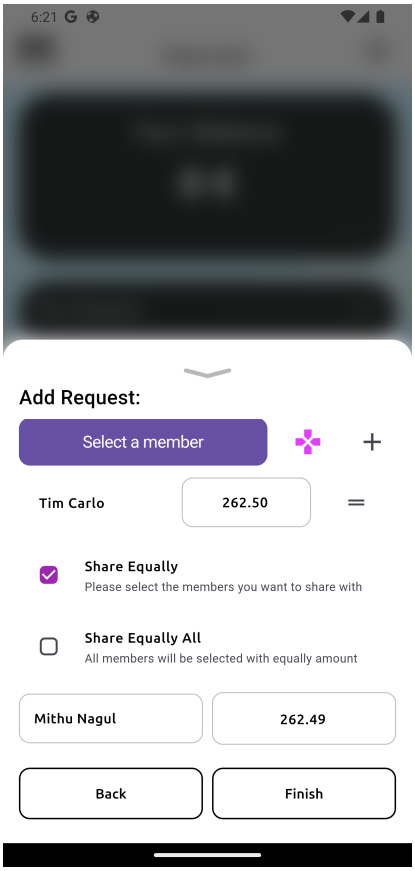

Figure 30: Share Equally

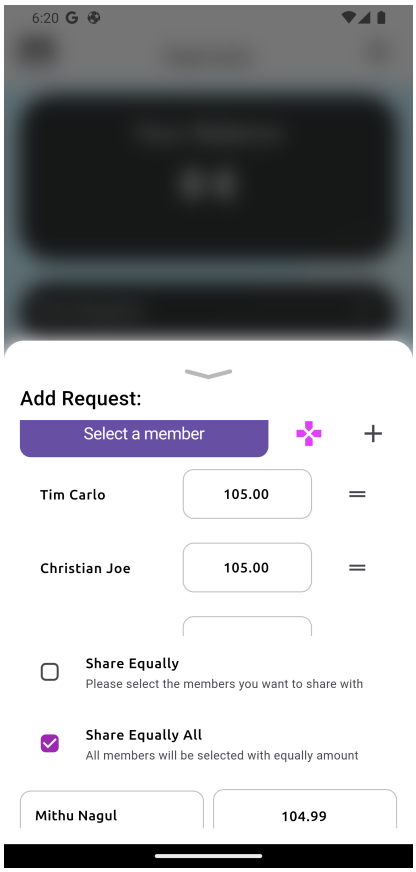

Figure 31: Share Equally All

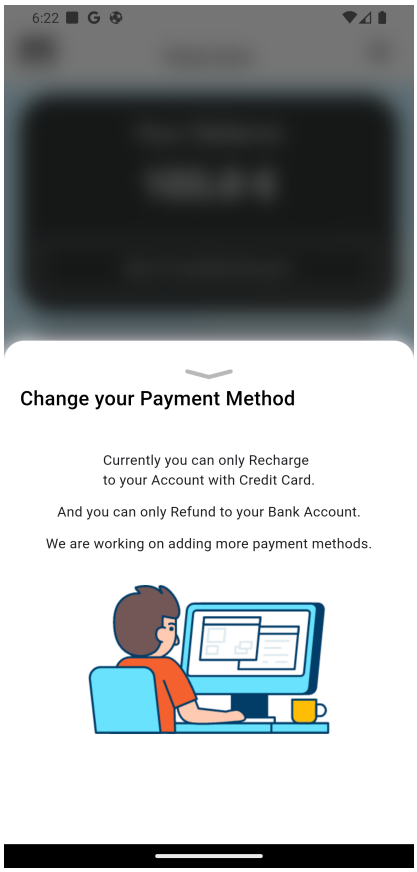

Figure 32: Icon Card

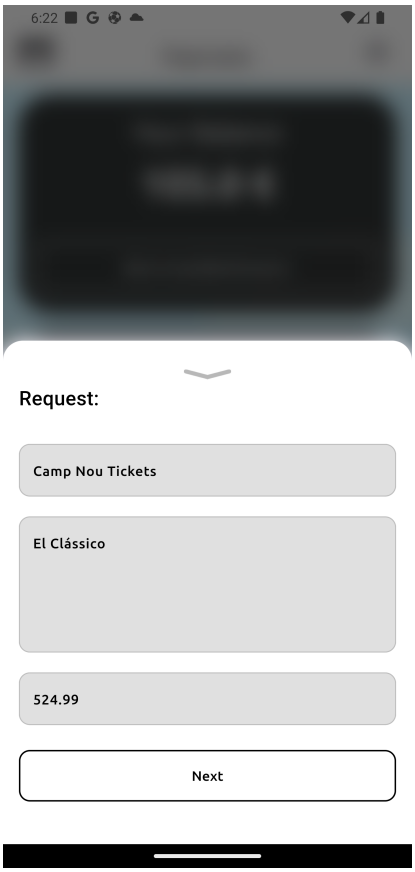

Figure 33: First Preview

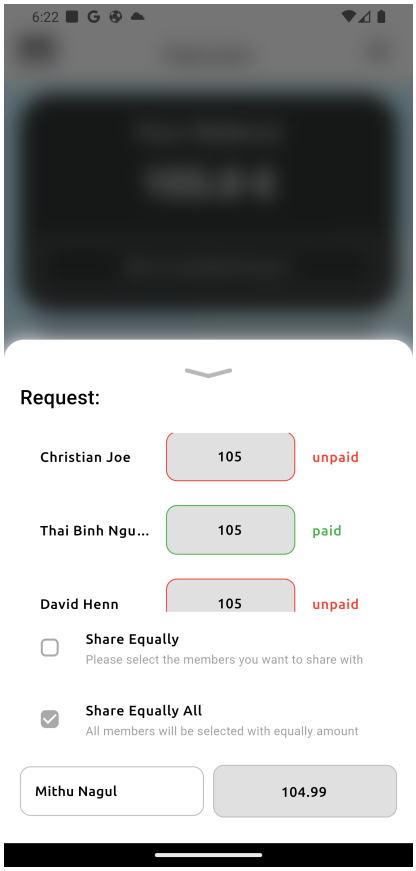

Figure 34: Second Preview

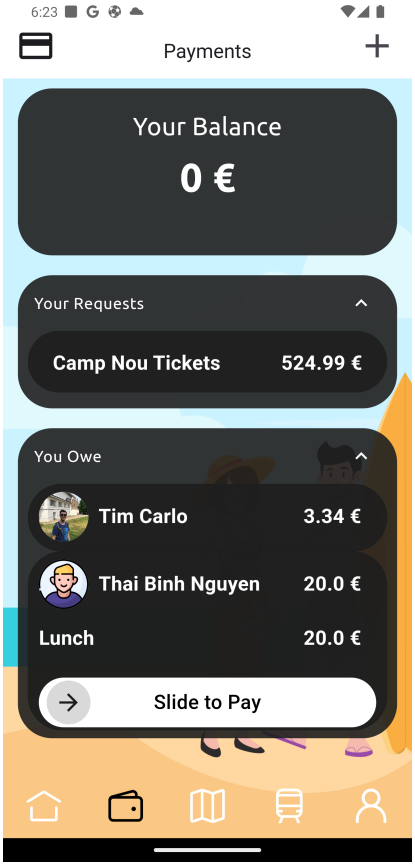

Figure 35: Slide to Pay

3.13 Map Page

On the Map Page, as shown in the attached screenshot (36), the user observes a "Location" button on the top bar's leftmost side. By clicking on this icon, the user navigates to their current location. Additionally, at the center of the top bar, the page title is visible. Finally, on the far right, the "Vacation" button allows the user to access their specified trip location upon clicking the icon.

Furthermore, the user encounters the text "Tap to see personalized recommendation. Long press to set origin and destination," introducing two scenarios.

Firstly, when the user clicks on the map, a blue circle appears, and the camera position of the map shifts to focus on the circle (36).

A slider on the far right of the screen allows adjustment of the radius of the blue circle. Just below it, a white icon reveals recommendations based on user-selected interests from the "Interest Page" upon clicking. Up to 10 interests are displayed, each marked with a pin icon. Clicking the icon again hides the pins of the user's recommendations, and it transforms into a different icon representing non-recommendations.

The small container initially displays the place name and current Google star ratings. Clicking on the container expands it(37), providing the user with more information about the location, such as address, phone number, and opening status. Clicking the arrow-down button at the top of the container reduces its size again.

Clicking the "Add to Dashboard" button presents the user with a bottom sheet (38), prompting the selection of a date, which has to be in the interval of the trip and the type of widget, whether it be an appointment, question survey, or appointment survey. All widgets created from the map to the dashboard already have the title of the location and are bound to that specific location.

On the "Appointment" (39) Widget, the user can specify the time to make the appointment with the group and in the description, it is already prefilled with a question if the members want to go there, with the address of the location, where the member is free to add some text into the field if they want. By pressing the "Add Appointment to Dashboard" button, the widget is going to be added on the dashboard. If the user clicks on the map icon of the widget in the dashboard (44), they are redirected to the map, focusing solely on the location associated with the widget. The functionalities of the other scenario, which will be explained later, can also be applied here. By clicking on the Container and expanding it, the camera position zooms to the bounded location(45) and can also look up the reviews and photos again. The same functionality applies to the other widgets as well.
On the "Question Survey" (40) Widget, the user can set the deadline of the survey, and can allow multiple answers by checking the box. Also, yes or no is already predefined which can't be deleted with a left swipe and the user cant add his own answer or question with the add icon. With the "Add Survey to Dashboard" it appears in the dashboard so all the members of the trip can vote. After the deadline ends, the widget is going to transform into an appointment widget, while retaining the same functionality as described above.,

On the "Appointment Survey" (41) Widget, the user can set the deadline of the survey, and can allow multiple answers by checking the box. And can select the time of the Appointment, through an add icon. If there are more than 2 options added to the list, then the "Add Survey to Dashboard" Button appears, and the Widget can be added to the Dashboard, so members can interact with the widget. After the deadline ends, the widget is going to transform into an appointment widget, while retaining the same functionality as described above.

Clicking the center container again flips it (42), allowing the user to read Google reviews or view images under the "Photos" button (43) by swiping up or down. The number of photos for the location is also displayed below. The "Review" button allows the user to read reviews.

Swiping right or left navigates the user to the previous or next place recommended by the API.

Clicking the red cross icon reveals two scenarios. If the current location is active, indicated by the blue button slider, the camera position navigates to the current location. Conversely, if the current location is inactive, the camera position navigates to the trip location.

In the alternate scenario, a long press by the user displays a green pin, facilitating determination of the origin. Another long press sets the destination using a red pin. The "Location" button remains situated on the leftmost side of the top bar and retains its original functionality. However, the "Vacation" button transforms into a "Zoom to Route" button, enabling users to zoom into the route they've created. Positioned on the left side, users can view the distance and duration of the specified route, which dynamically adjusts between kilometers or miles based on the region, utilizing the API. It's important to note that the duration assumes the user is traveling by vehicle(46).
Both scenarios work together when interacting from the map. However, when navigating from the dashboard to the map, only the second scenario functions. The location associated with the widget is displayed there only as a container, with the top bar adjusting with different functionalities, as described above.
Directly below, green, red, and blue slider buttons are visible. It is important to note that if the current location is not active, the blue slider will not be displayed. Clicking the green slider expands it, and clicking "Ori" shifts the camera to the position where the user created the origin with a long press. Clicking the red cross deletes the green pin, while clicking the left arrow reduces the slider. These functions are applicable to both the red and blue sliders. For the red slider, clicking "Des" shifts the camera to the destination. Whereas, clicking "Cur" shifts the camera to the current location of the user. Additional long presses enable the user to set new origin and destination pins.

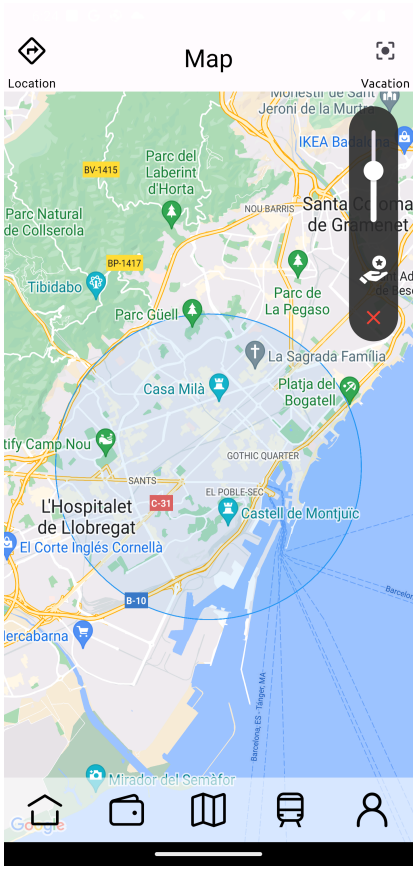

Figure 36: Map Page

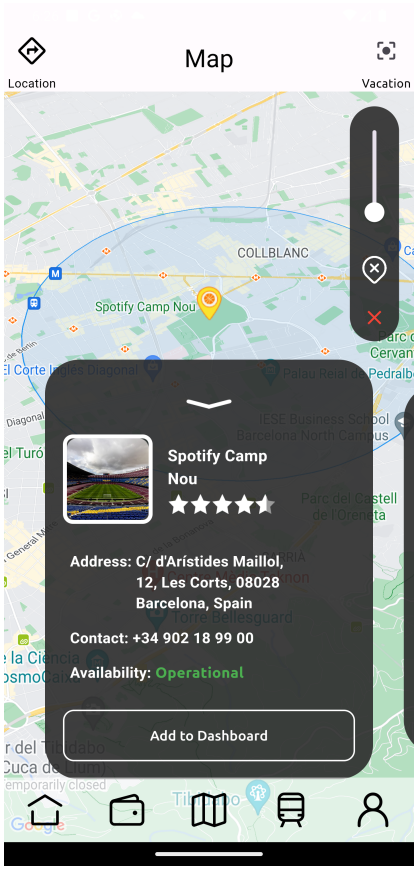

Figure 37: Front View

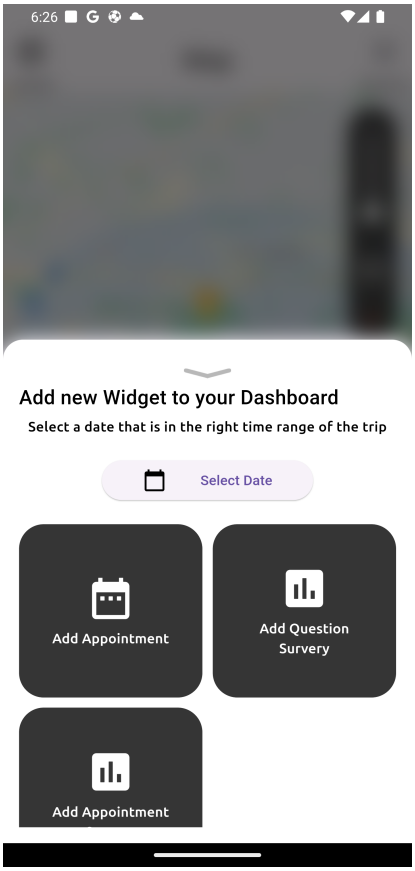

Figure 38: Map Widgets

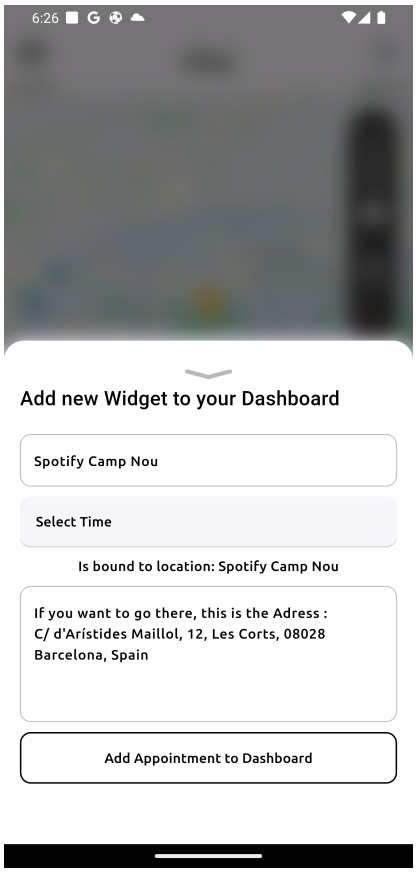

Figure 39: Appointment

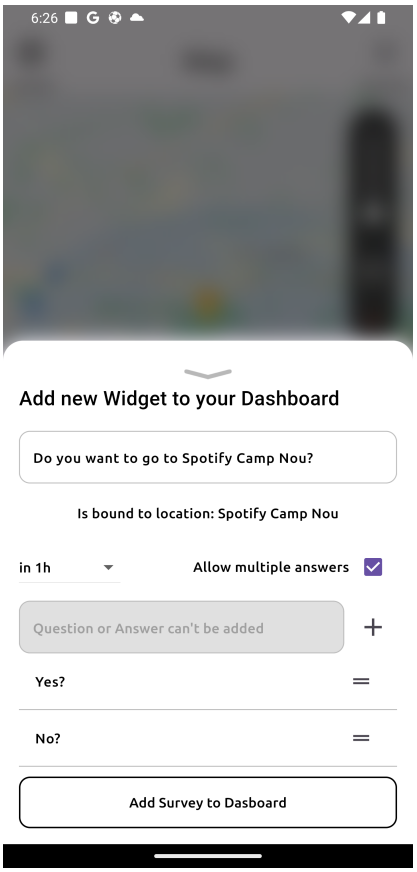

Figure 40: Question Survey

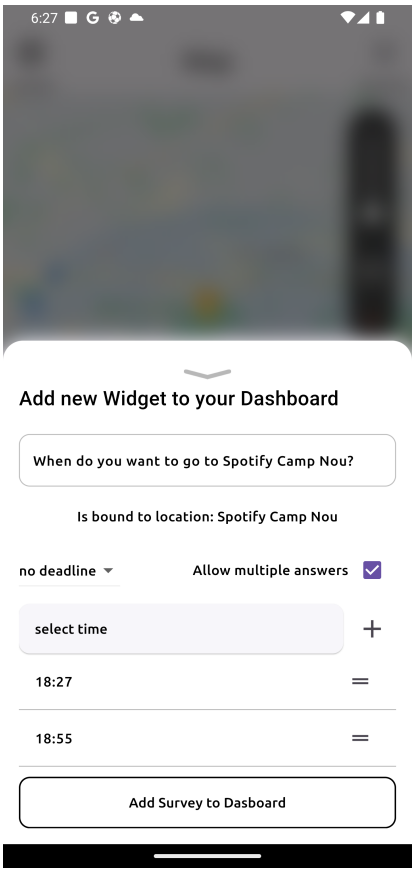

Figure 41: Appointment Survey

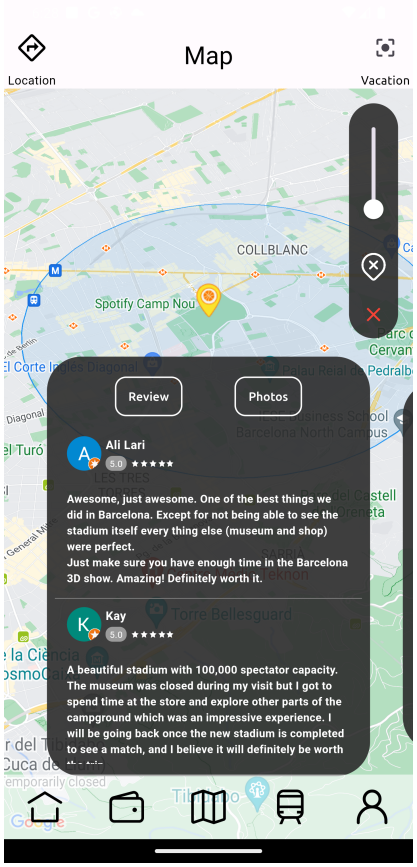

Figure 42: Back Card Reviews

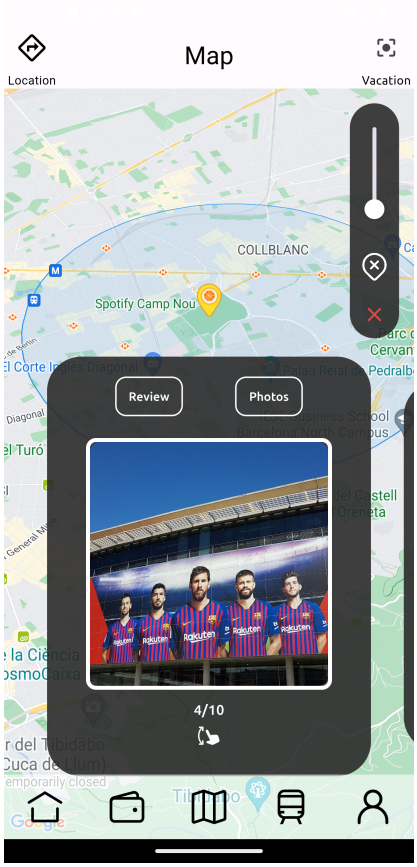

Figure 43: Back Card Photos

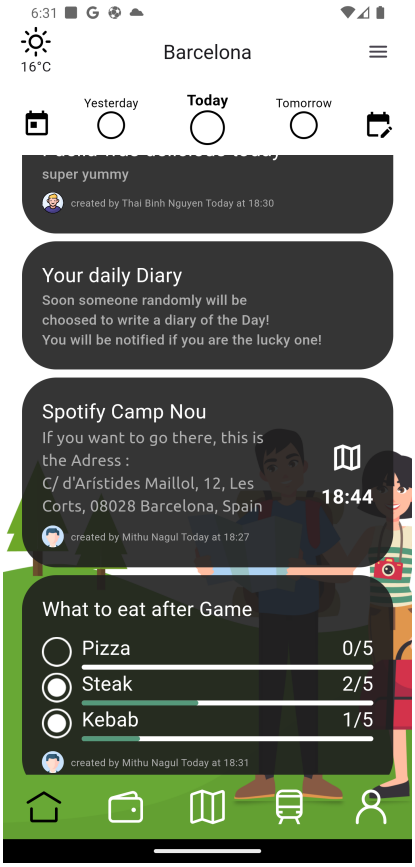

Figure 44: Map Widget in Dashboard

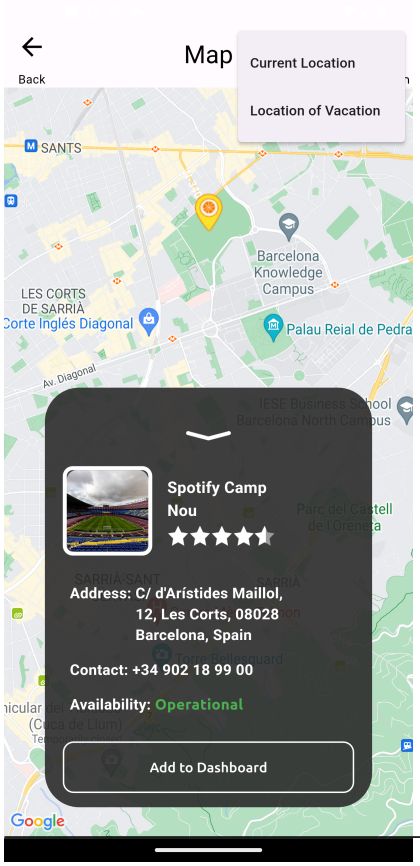

Figure 45: Front View

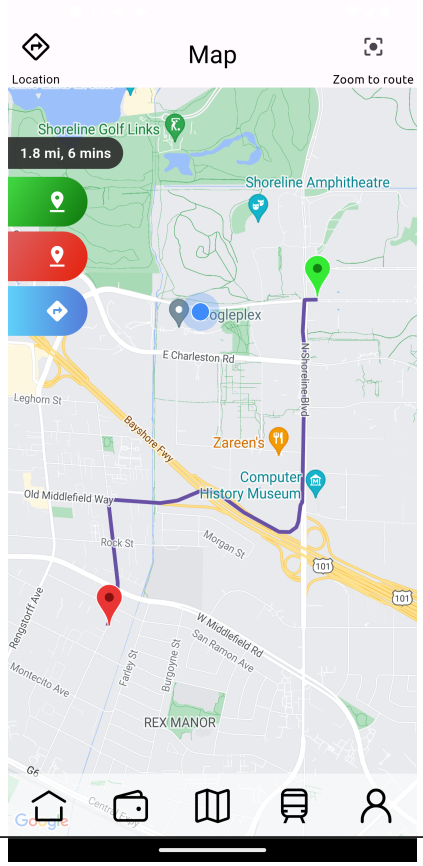

Figure 46: Map Widgets

3.14 Tickets Page

Within the Tickets Page, as depicted in the attached screenshot (47), users have the option to click the "+" icon located in the upper right corner. This action triggers the appearance of a Bottom Sheet (48), allowing the user to specify a title according to their preferences. Furthermore, the user can either capture an image of the ticket or upload a picture or a PDF file. By selecting the "Upload Ticket" button, the user initiates the process of uploading the ticket. Upon returning to the Tickets Page, the user will observe the newly created ticket represented as a container with the title. Clicking on the container provides a preview of the generated document. By selecting the document, the user is redirected to a separate page, enabling them to zoom in and out if it's an image (49), but they can also navigate through the document seamlessly using the top bar on the right side if it's a PDF file. On the left side, users can return to the container where they see the preview of the document. It is also possible to delete the ticket by performing a Left Swipe.

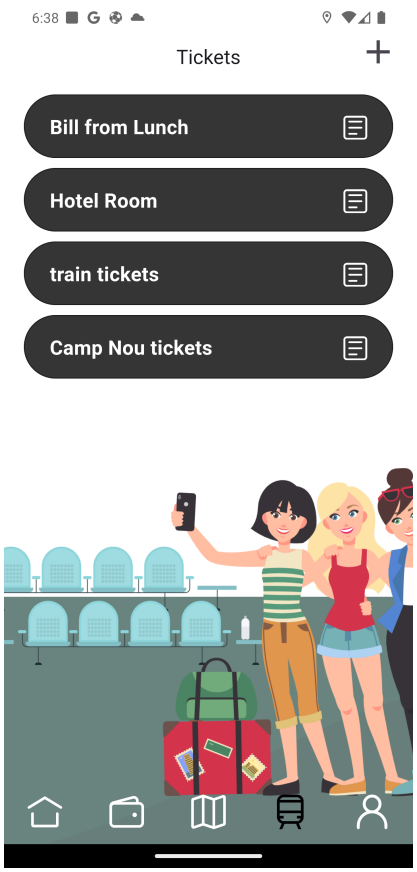

Figure 47: Ticket Page

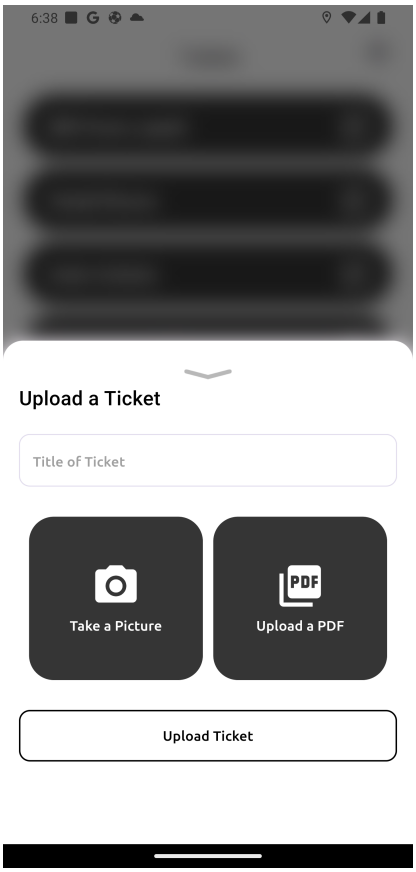

Figure 48: Ticket upload

Figure 49: Ticket Viewer

3.15 Settings Page

On the Settings Page, as depicted in the provided screenshot (50), users initially encounter their profile picture. By clicking the "Edit Profile" button, users can modify their profile picture, redirecting them to the Account Details page where the avatar circle allows for profile picture adjustments.

Furthermore, selecting the "Information" button navigates the user to the designated page, as illustrated in screenshot (51), where they can view and contact the creators along with their respective links.

Additionally, there is an "Enable/Disable Push Notifications" button, providing users with the autonomy to choose whether they wish to receive push notifications.

Subsequently, users can revisit and update their interests by clicking on the "Your Interests" button, facilitating adjustments based on changing preferences, where the user is directed to the "Interests Page".

A noteworthy hidden feature within our application is the "Game: Choose a Loser" button, leading the user to a page where they can engage in a game and select a loser.

The Logout button seamlessly guides the user back to the Login/Register Page, effectively logging them out of their account. With the ”Delete Account” Button, allows the user to delete their account as long as their wallet balance is neutral.

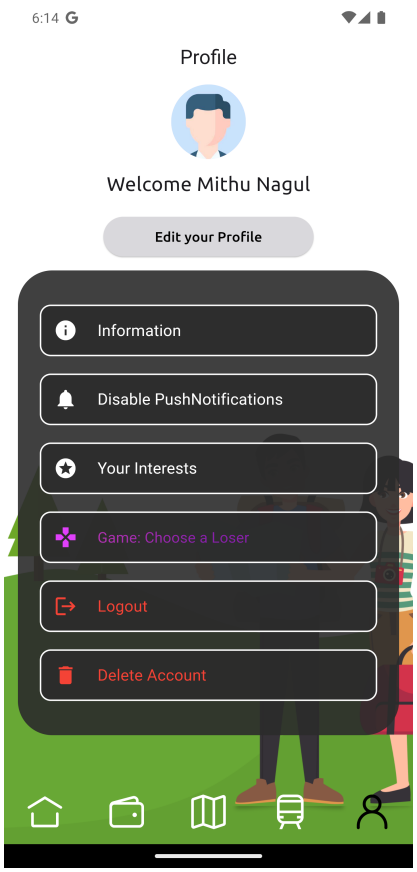

Figure 50: Settings Page

Figure 51: Info Trips

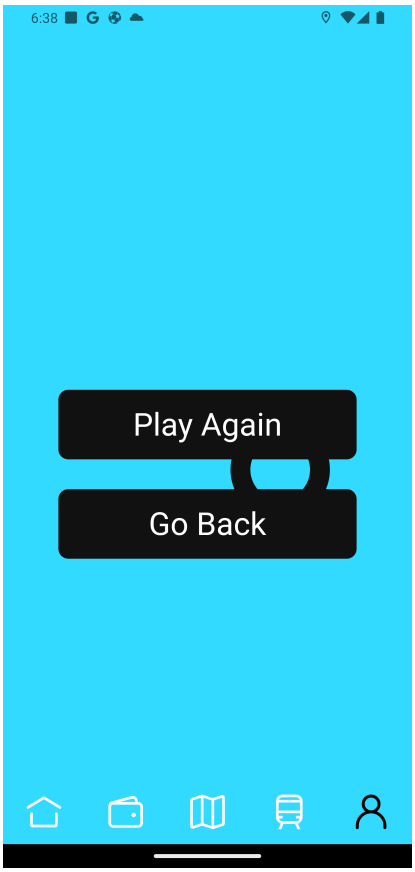

Figure 52: One More Thing

3.16 One More Thing

In the "One More Thing" (52), players on a mobile device can move their fingers simultaneously, creating circles. After a certain time, the remaining finger is marked, and the color of the loser is displayed. Subsequently, the game can be replayed, or by using a back button, the user is redirected to the Settings Page.

4 Main Features

The TripTip app incorporates a range of main features that are essential to its functionality. These features include:

- **Authentication:** By integrating Firebase Authentication, the app ensures secure login and account creation.
- **Profile Editing:** Within the application, users have the capability to tailor their profiles, including the option to modify profile pictures, update first and last names, and adjust their birthdates.
- **Notifications also when App is closed:** Utilizing a push notification system, the application ensures users remain informed about Diary expiration times and relevant updates such as widget creations, wallet disbursements, or other pertinent information, even when they are not actively engaged with the app. These notifications are transmitted as Firebase Cloud Messaging (FCM) notifications.
- **Creating Trips with autocomplete Place Suggestions:** When a user initiates the creation of a new trip and begins typing the first letters of their desired location, the autocomplete feature assists in completing the location entry process.
- **Joining Trips via a Code:** Users can join another trip by entering the TripID generated during the trip's creation.
- **Create Trip via a Code adding Friends** When creating a trip, users are provided with a TripID. Additionally, users can add other individuals to the trip by entering their User ID, which can be found on the Account Details Page.
- **A real-time collaborative Dashboard:** Through real-time, users can immediately visualize widget management on their dashboard.
- **A collaborative polling Tool:** The widget polling feature enables users to configure multiple-choice responses, set deadlines, and create polls that specifically involve time constraints alongside conventional surveys.
- **A real-time collaborative Tickets Manager:** Within the tickets section of the application, trip participants can capture images, upload files, and subsequently review the uploaded materials.
- **A Map showing your Destination:**The user can set a route by performing long presses, with the length unit varying based on the region where the pins are placed.
- **A recommendation System for suggesting places on a Map:** Leveraging a recommendation system, the map suggests locations to users based on their interests, accompanied by photos and reviews.
- **A Page to Select your interests:** Users can select their interests, which inform the recommendation system.

- **A Weather Information Page for your Trip location:** This page not only displays the current weather at the user's location but also provides weather updates for their trip destination.
- **A Game to pick a random Group member:**Trip members participate in a finger-placing game on their mobile devices, during which circles disappear over time until only one remains, determining the loser of the game.
- **A Payments System to share Bills with others:** The payment page offers a system for users to split expenses among trip members.
- **An App Information Page:** Users can contact app developers for inquiries, issues, or other matters through the application.
- **A Storage for keeping Appointments together with your Friends:** In the dashboards, appointments can be created, which can be viewed by not only me but also my group members.
- **A Storage for keeping Place Suggestions together with your Friends:**Furthermore, recommendations from the map can be transferred to the dashboard using widgets.
- **A Settings Page:** Where the user can access the account details page again to adjust your information, contact the creator of the map, enable/disable notifications, play a game, or simply logout/delete the account.

5 Conclusion

In summary, the project can be deemed a resounding success from our perspective. The development of the "TripTip" app aims to assist travelers, whether solo or in groups, in planning their journeys. Leveraging Firebase as the backend and the Flutter framework, numerous essential features have been implemented alongside a user-friendly UI/UX design, enhancing the user experience and facilitating seamless interactions among group members. Even though we won't publish the app, we have gained the skills to develop an app, front- and backend, as well as work on a larger scale project as a team.

6 Testing

Due to an unexpected health incident involving one of our team members, we sought clarification from the tutors regarding the necessity of the mandatory "Testing" feature. Recognizing the unique circumstances and understanding the situation, the tutors have shown their support. Given the absence of the affected team member, we have been graciously granted the opportunity to temporarily suspend the planned testing.

7 Outlook

We are considering the prospect of launching our application on the Google Play Store, thereby enabling easy installation for users. Following this, we also contemplate the option of releasing the app on platforms such as the web or iOS, leveraging the seamless compatibility afforded by the cross-platform Flutter framework. This multi-platform approach ensures broad accessibility and user convenience across diverse devices and operating systems.

Furthermore, we envision expanding the functionality of our application to serve as a comprehensive gaming catalog, offering users a diverse array of gaming experiences. This expansion aligns with our commitment to enhancing user engagement and satisfaction by providing a wide selection of entertainment options.

In addition to our existing registration and login methods, which include email, password, and Google authentication, we are currently awaiting confirmation from Facebook to go live so that users can log in and register via Facebook. However, we would also like to offer additional options such as GitHub or X.

Overall, our strategic approach involves not only leveraging the capabilities of the Google Play Store but also extending our reach to other platforms, enhancing the features and usability of our application, and diversifying authentication options to accommodate varying user preferences and needs.

8 Appendance

```
rules_version = '2';

function isSignedIn() {
  return request.auth != null;
}
function isMember(database,trip) {
  return /databases/$(database)/documents/users/$(request.auth.uid) in
     trip.data.members;
}

function isAdminOfTrip() {
  return resource.data.createdBy == request.auth.uid;
}

service cloud.firestore {
  match /{document=**} {
      allow read, write: if false;
  }
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read: if isSignedIn();
      allow create: if isSignedIn();
      allow update: if request.auth.uid == userId;
      allow delete: if request.auth.uid == userId;
    }
    match /trips/{tripId} {
      allow read: if
         /databases/$(database)/documents/users/$(request.auth.uid) in
         resource.data.members;
      allow create: if isSignedIn();
      allow update: if
         /databases/$(database)/documents/users/$(request.auth.uid) in
         resource.data.members;
      allow delete: if isAdminOfTrip();
      match /days/{dayId} {
        allow read: if
           isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
        allow create: if
           isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
        allow update: if isMember(database,
           get(/databases/$(database)/documents/trips/$(tripId)));
        allow delete: if isMember(database,
           get(/databases/$(database)/documents/trips/$(tripId)));
        match /diary/{diaryId} {
```

```
        allow read: if isMember(database,
           get(/databases/$(database)/documents/trips/$(tripId)));
        allow create: if isMember(database,
           get(/databases/$(database)/documents/trips/$(tripId)));
        allow update: if isMember(database,
           get(/databases/$(database)/documents/trips/$(tripId)));
        allow delete: if isMember(database,
           get(/databases/$(database)/documents/trips/$(tripId)));
      }
    }
    match /payments/{paymentId} {
      allow read: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
      allow create: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
      allow update: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
      allow delete: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
    }
    match /tickets/{ticketId} {
      allow read: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
      allow create: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
      allow update: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
      allow delete: if
         isMember(database,get(/databases/$(database)/documents/trips/$(tripId)));
    }
  }
  match /tasks/{taskId} {
    allow read: if isSignedIn();
    allow create: if isSignedIn();
    allow update: if isSignedIn();
    allow delete: if isSignedIn();
  }

 }
}
```