Atlas Itinerary Project Design

By Gabriel Taylor, Dustin Zook, Sorelle Djuissi, and Gabriyel Sorensen

# Project Parts

## Design Description

### The Front End

When the user accesses the page, they will be prompted to sign up using their Google account (Figure 3.2). Beneath the option to sign up using their Google account, returning users will be shown how they can sign in through a message with an embedded link that takes them to the sign-in page, which also requires the use of their Google Account.

Once users have finished signing up or signing in, they are greeted with the website’s Home Page (Figure 3.3). The Home Page’s main focus is the “About section”. This section will explain to users the ins and outs of the website. The background of the group behind it will be included in the About Section, as well as details on how to navigate the site. Each option in the navigation bar will be displayed as a label at the top of the page; each label will contain an embedded link that will take users to the respective page for that option. The Navigation Bar has two main purposes: to allow users to either start creating an itinerary or to allow users to view old itineraries.

Along the top of the webpage, there will be a navigation bar. The navigation bar will contain the names of each of the pages the application offers, the Home page, the Create Itineraries page, the My Trips page (View Itineraries), as well as an Account option. The Account option will not be its own page, but instead will be a drop down that users can access. The Account option that is presented to the users will contain their email address, their current location, and a sign-out option.

If the user chooses to select Create Itineraries, they will be presented with a page where they can create itineraries (Figure 3.4). There will be a text box for their Travel Destination. The text box for their destination will be blank, allowing users to input their destination’s name. The text box will also serve as a search bar, displaying predicted destinations based on what is already typed. As an example, this feature is similar to the Google search bar. If the user does not enter a destination, the application will be able to detect the user’s current location and generate suggestions for their Travel Destination. The user’s current location will be the default travel destination. The Travel destinations will be in a lighter color in the search bar to indicate to the user that they are different from the website's necessary text and are capable of being changed. If the user inputs a destination or has their location services turned off, the page will display a message to the effect of “Location not found.”

Once the destination searched for is found, the user will be presented with a sidebar containing several categories. These categories will be presented as three buttons. When a button is pressed, a series of cards relevant to the selection will be displayed. The default category button pressed will be lodging. Lodging will allow users to select locations where they can stay for the day, nap in the evening, or sleep for the night. Most users will probably select one location for all three of these things to happen. The second category will be dining. Dining will allow users to select a location to get food at. For breakfast, should the lodging provide it, the user will be able to select the lodging as their dining option for breakfast. The final category provided will be the attractions category, where users can search through the places that the area has that are frequented during visits.

The lodging category will contain subcategories that the users can choose from. The subcategories will be listed in a filter option that appears next to the Destination search bar. If a filter has not been selected, the default listing of the destination API will be shown. Once a filter option is selected, the options will appear as a series of cards that the user can select. Each card will contain a list of relevant details about the lodging. The most commonly chosen lodging will no doubt be hotels. Hotels offer the basics a traveler might need, things like a shower, a bed, maybe a fridge, a closet for clothes, and possibly even a TV. The next option will be a Bed and Breakfast. These are typically small, privately-owned lodgings in a personalized setting. They are very home-like. Finally, users may choose from specialty lodgings like ecolodges, hostels, or even mountain huts if the area contains such.

Similar to the lodging category, attractions will also be represented by an Attractions button. When selecting attractions, the filter will appear next to the search bar, unless it is already there from when one of the other categories was selected. This time, it will promote new subcategories related to attractions that users may want to visit. Examples of the subcategories include attractions such as boat tours and water sports, shopping, as well as nearby nature and parks.

Dining will be the third button users can click. This option performs almost identically to the first two. The filter option allows users to pick between a café, a sit-down restaurant, or fast food. Fast food restaurants are the simplest to understand. They are quick and concise, allowing users to enter and leave the establishment in a timely manner. Sit-down restaurants require more patience and should be planned around and not just dropped in with the hope of fitting into the schedule. Cafes are a little nuanced, providing the users with an atmosphere to take things at their own pace. These locations mainly offer coffee and other beverages meant to boost productivity during stationary work.

Each Lodging, Attraction, or Dining location will be displayed as a list of cards. Each of these cards will display the address of the location and the name of the location. A paging system will allow the user the change the number of cards displayed; the default number of cards displayed will be 9. The cards will be organized according to their rating. When one of the cards is clicked, images of the location will be displayed, as well as a description of the location, the rating of the location, and an “add to itinerary” button(Figure 3.5). After selecting “add to the itinerary”, the user will be asked for the date and time for the location. Four text boxes will appear, one for the start and end date as well as the start and end time. Each of the text boxes will accept numerical values in accordance with time intervals. The first number must be any number between 1 and 9; however, if the first number is more than 2, the second spot must be occupied by a colon. After the colon, the next number can range between 0 and 5. The number that follows can range from 0 to 9. If the user attempts to input a letter or a number that does not follow these rules, the system will prevent the user from allowing that input.

After the location is successfully added to the itinerary, a sidebar will appear with the current itinerary being built. The itinerary will be displayed in a calendar view, with each saved location displayed as a bullet point on the day it is associated with. Each day may be expanded to include the locations and events for that day, which are listed in chronological order. Users can choose to select events and move them around or even delete them.

Lastly, there will be a Save As button (Figure 3.6). The Save As, when clicked, will prompt users to name their itinerary. Naming the itinerary will make it easier for the user to identify later in the View Itineraries option. Once the name is input into the system, the itinerary is saved. After the itinerary is saved, the user is taken back to the start of the Create Itineraries process, where they can begin the process of creating another itinerary.

After the user successfully creates an itinerary, most of the users will then choose to go to the View Itinerary page to look at or edit their itinerary. If the user chooses to visit View Itinerary before they have created an itinerary, they will be informed that they do not have any itineraries yet, and they will be prompted to visit the Create Itinerary page. On the View Itinerary page, the first thing the users will notice is that the itineraries they have built are presented to them in a card format (Figure 3.7). The card will consist of the name of the itinerary, the start date, and the end date of the itinerary. If the user clicks on the itinerary card, they will be able to see more of the details they included in the itinerary. This will be possible because, after clicking on the itinerary, the card will expand to a fuller view.

The view of the card will be calendar-like in the way that it is presented to the viewer. There will be bullet points for each event that will occur on each of the days that the itinerary spans. Clicking on a day has a similar behavior to clicking on an itinerary. The day will expand to a fuller view with the events for the day listed in chronological order. However, unlike in Create Itinerary, the user will first see the day in Viewing Mode. This will mean that the user will not be able to make changes at first when they expand the day. Near the top right corner of the expanded day card will be an Edit Mode button that will switch the user into Edit Mode. After the user enters Edit Mode, they will be able granted permission to make changes to the day as if they were on the Create Itineraries page.

In Edit Mode, users will be allowed to select events. Once the event is selected, the users will be able to access the delete button for the event, where they can remove it from their day. If the user should press the delete button for the event, they will be shown a warning message. The message will display a single line of text to the effect of, “Are you sure you want to delete this event?” and two options for proceeding will be offered. The first option will be to the effect of “No, I am not sure. KEEP THE EVENT.” While the second option will be, “yes. DELETE THE EVENT.” The important sections that users will be most interested in focusing on will be in all caps to help the user understand the decision more clearly.

Edit Mode will also allow users to move events. Once an event is selected, users can move those events around by editing the time the event will take place in the text box next to the event. After moving the event to a different time, the official displayed time of the event will be updated. If this event is not available at the new scheduled time, a warning alert will be displayed in red next to the event, notifying the user, “This event is not available at this time.”

Similarly to events, users will also be able to select days. Selecting a day will enable the user to delete the day through the newly appeared delete button. Attempting to delete the day will result in a similar message of, “Are you sure you want to delete this day and all of its events?” Again, a set of options, “no, I am not sure. KEEP THE DAY,” and “yes. DELETE THE DAY,” will be provided. Selecting the day will also allow users to move the day’s events to another day. Similarly to events, if a day’s events are not all available on the new day they are moved to, the user will be notified with a red message next to every event that is not available on the new day.

Behind the beautiful and fun cards and bullet points, systems like React are working to make sure everything stays in one piece. React operates as a kind of merging of HTML and JavaScript. In React, each piece on a page is a component. Starting on the Sign Up page, the Google sign-up option will be a component that links to the Google api. This will reduce the number of complicated measures the developers will have to take to ensure user security, as Google is a larger multi-million-dollar company that has been at this for a long time. In JavaScript, developers are required to follow a first-letter, capital-letter naming convention. JSX code requires the same kind of naming convention, with the first letter of component names capitalized. In the case of the Sign Up page, the name for the overarching component might be “function SignUpPage{}”.

When React is running, the functions will have other, more specific functions embedded in the main function component. For instance, the function for the component on the sign-up page responsible for the message “Already have an account? Sign in here” will contain a function in it that accesses a URL link to the sign-in page, while also being a function inside of the function SignUpPage{}. Because of this nested method of creating each component, some actions can be repeated. The Sign Up and Sign In pages will both have a function component that accesses the api for Google accounts. So instead of writing this twice, making a component call will be more effective.

The project will be comprised of five main pages (Figure 1). The sign-up page, the sign-in page, the home/about page, the create itineraries page, and the view itineraries page. Then, each page will be granted access to the components based on their need for them. The main components are the Google Authentication, Navigation Bar, Destination Search, Sidebar, Filter Drop Down, Card Grid, User Menu, Location Modal, Location Card, Itinerary Calendar, and Itinerary Side Bar Calendar. Organizing the components in this way will minimize the amount of repeat code in React. It will also help to minimize the amount of time spent coding.

The sign-up page and the sign-in page will both be given access to the Google Authentication component. The User Menu will also have access to the Google Authentication component, as this component is where the Account option will be stored, and the sign-out feature will be implemented. The Create and View Itinerary page will both share the Itinerary Side Bar Calendar, Itinerary Calendar, Card Grid, and Location Modal. The Home Page, the Create Itinerary, and the View Itinerary will all share access to the Navigation Bar.

Atlas Itinerary is not just going to be about the flashy cards and functioning links. Users will build itineraries for themselves, which means the development team will need a way to organize user itineraries into separate spaces, specifically for each user. This will require the use of a database capable of storing an enormous amount of data from multiple users without misallocating the data to the wrong user. The database manager that the project has implemented to tackle this problem is Supabase.

Supabase boasts the ability to be easily connected to by Software Development Kits like React. Unlike the stereotypical database, Supabase does not use SQL, but instead it uses JavaDoc api code. The project will be tedious as it is. With the requirement of learning new technology and the time required to do so. Any advantages offered, like easy connections, make the project run more smoothly. In the case of Supabase and React, being able to use similar functions and coding languages between the software systems will help make the process more streamlined and reduce the potential for careless mistakes.

The Supabase database also boasts integrability with user authentication, like the Google API that this project utilizes. One of the team’s main concerns was that the security risks associated with creating a stack or organizing account names from scratch were that it would be time-consuming and that careless mistakes or unforeseen errors would be commonplace. However, Supabase integrates with Google’s api so well that this concern should be trivial in the long run.

Within the database (Figure 2), there will be a series of entities. These entities will be the main variables when it comes time to write the code. For the users, each user is assigned an ID. That ID is linked to their Google sign-in information. It will also record their first and last name. The Itinerary will also have access to the User ID. This will be important for displaying information that is attached to the user. The Itinerary will store the name of the itinerary, the travel destination, as well as the start and end dates that indicate the duration of the itinerary. The locations of the itinerary are stored under a location ID. Also stored in the location is a trip ID representing the ID in the geolocation data, as well as the start date, end date, and the starting and ending times. This information is also attached to the itinerary ID.

The project also utilizes Vercel. Vercel enables the team to monitor the application in various ways once it is live. Once the program is live, Vircel enables the team to monitor the application’s web analytics. Web Analytics tells the team the number of visitors the application has, as well as the number of times the page was viewed. The standard Page Views are cumulative from all the pages the application provides; however, Vircel also displays analytics for separate pages. In the instance of this project, the application as a whole might have 300 views, but the Create Itinerary and View Itinerary pages might have 100 views each, with the remaining 100 views spread across other pages.

The project will require a large amount of front-end data to be displayed for the user to see, and presenting this data in an absorbable manner would be difficult to do. To tackle this problem, this project will implement its front-end data through React. This decision was made in part due to the familiarity some of the group members had with the software. React uses JavaScript, which is a coding language the group is highly familiar with. This means that even though some of the members are seeing React for the first time, they understand JavaScript to a degree that learning how to use it in React will be relatively simple.

Supabase was not the project's first choice. In fact, Firebase was the project's first choice. Due to the familiarity some of the members had with using Firebase, the group felt confident that those members could help to educate the other team members who did not know how to use Firebase. However, after some time and consideration, the members realized that if they chose to use Supabase instead of Firebase they could save time on building the project. This is because Supabase boasts the ability to be easily integrated with software like React. This is exemplified by the fact that Supabase does not use SQL code but instead uses Javidoc code, similar to React. Any level of familiarity between software choices accelerates the productivity of a project by leaps and bounds. So, in choosing to use Supabase over Firebase, the group is choosing to go with a software they originally knew almost nothing about in the pursuit of a more easily integrated system.

Vercel was chosen for the information that it provides. With the provided Web Analytics, the team can monitor the application and deduce from the data if it is performing at acceptable levels. As a server-side software program, it possesses the bare necessities that our group needs without requiring the group to empty their wallets. The simplicity and cost-effectiveness make it ideal for this project, given its lifespan that it will have, seeing as it will probably die off by the end of the semester.

Revised requirements

# Appendix

## Component Diagram

Figure 1.1 Pages

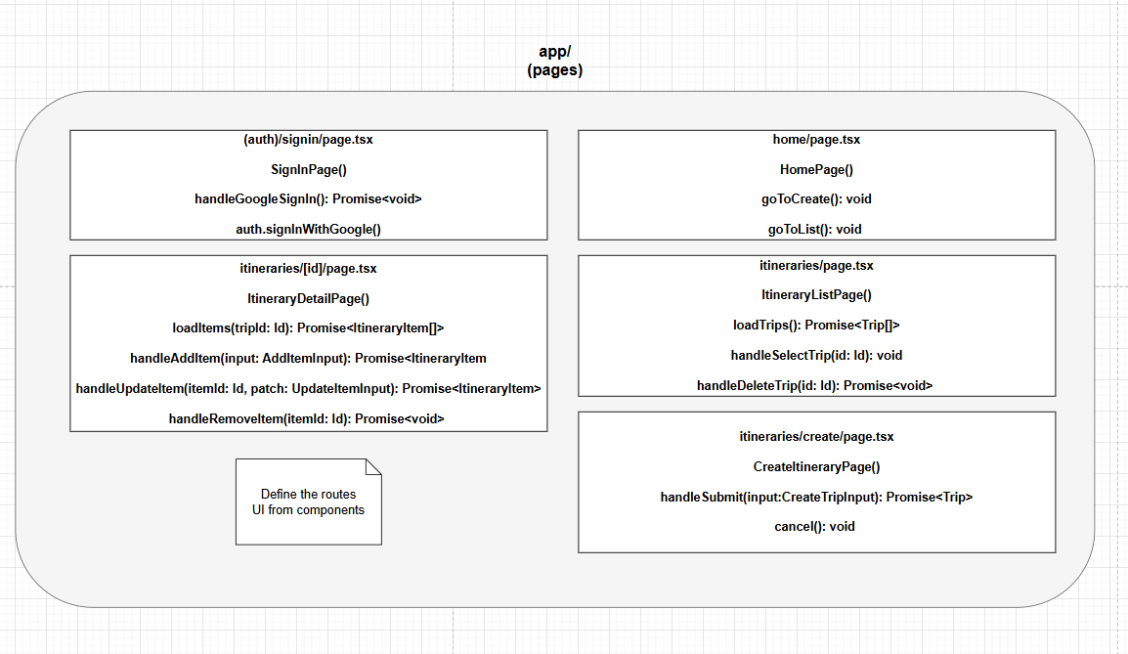


Figure 1.2 Components

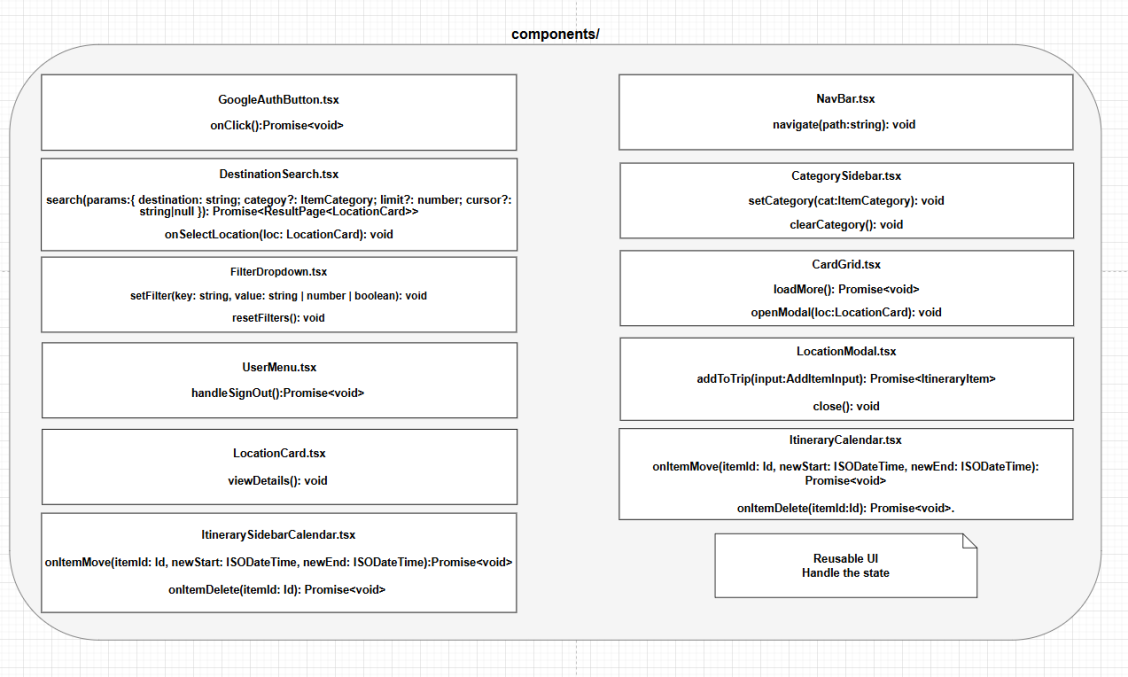


Figure 1.3 Library

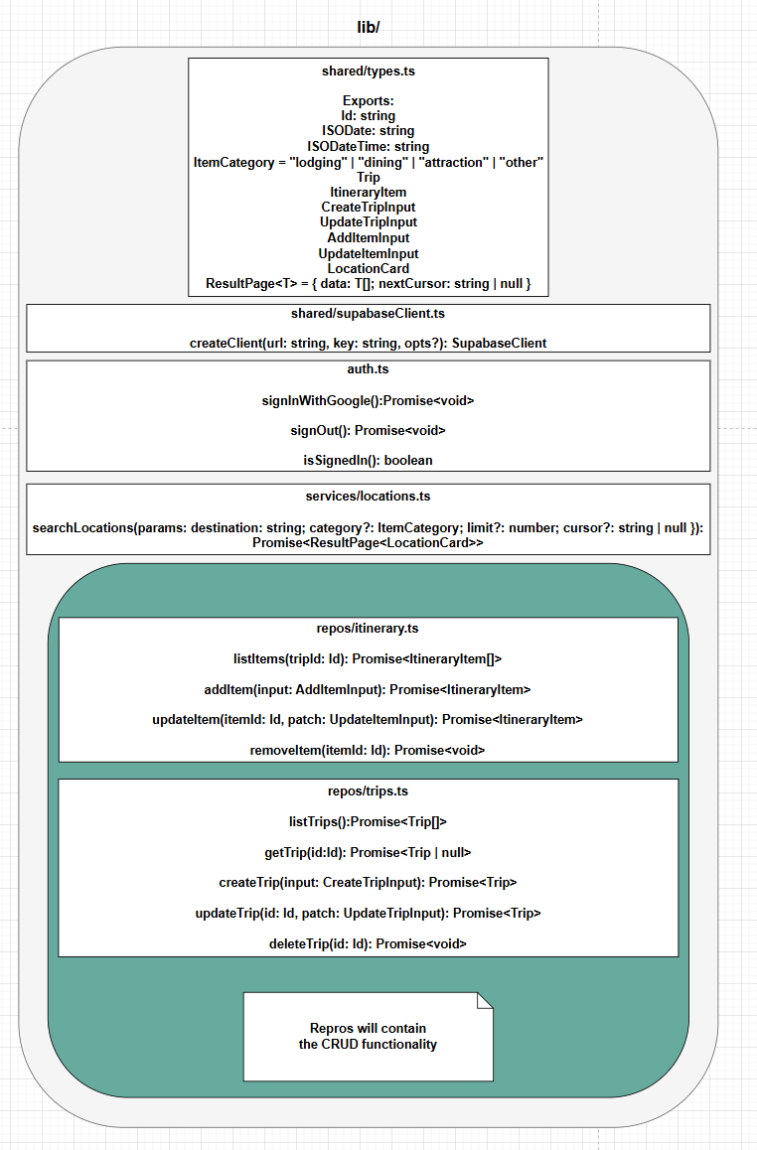
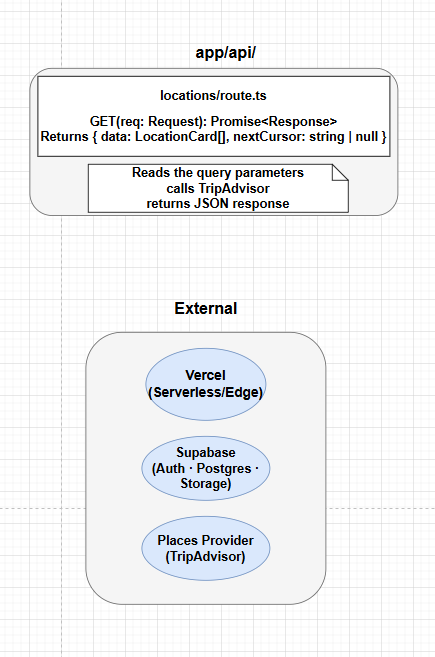
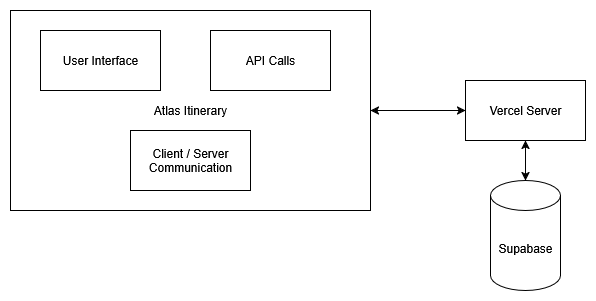


Figure 1.4 API/External



## Block diagram

Figure 2

## User Interface Storyboard

Figure 3.1

## 

## Figure 3.2 Sign-up page

## 

Figure 3.3 Home page

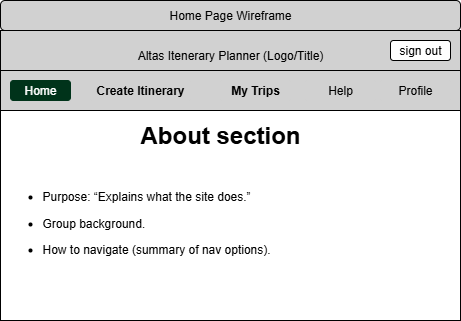


Figure 3.4 Create Itinerary

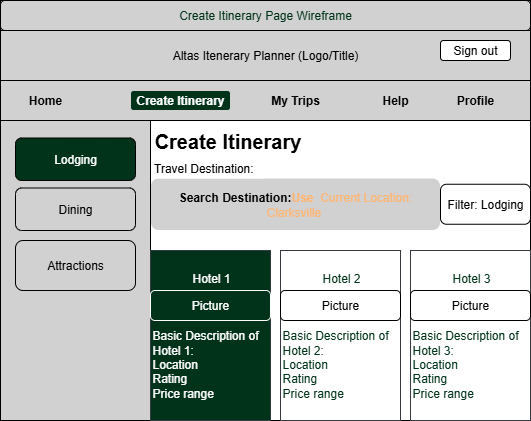


Figure 3.5 Create Itinerary(add to Itinerary)

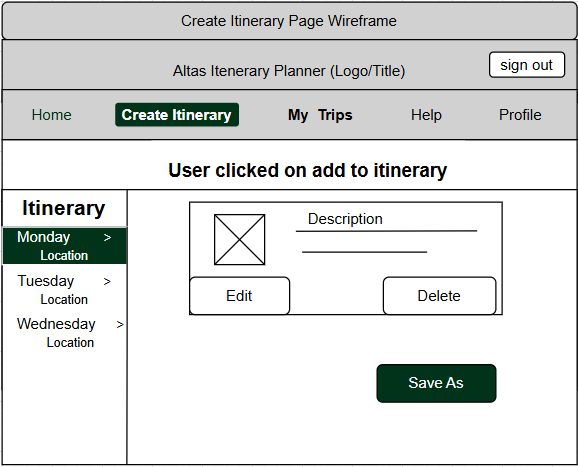


Figure 3.6 Save As itinerary

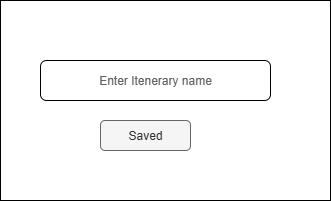
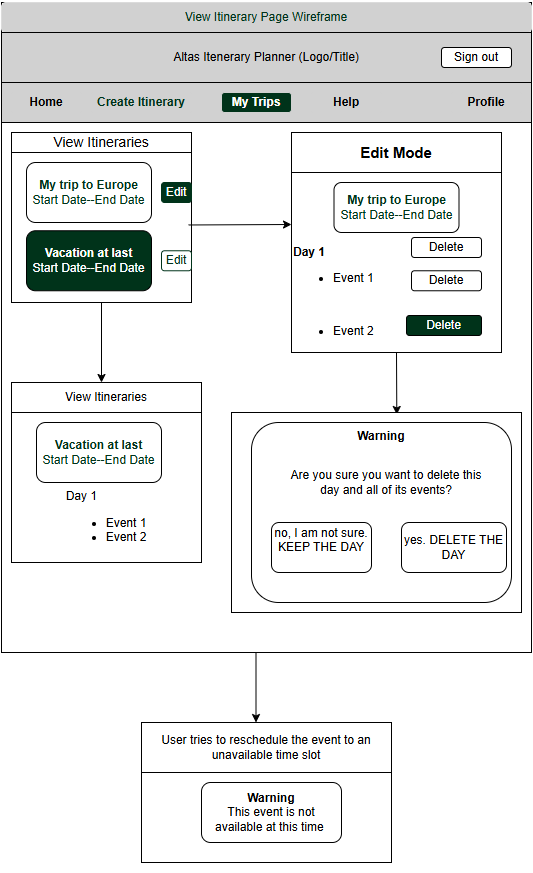


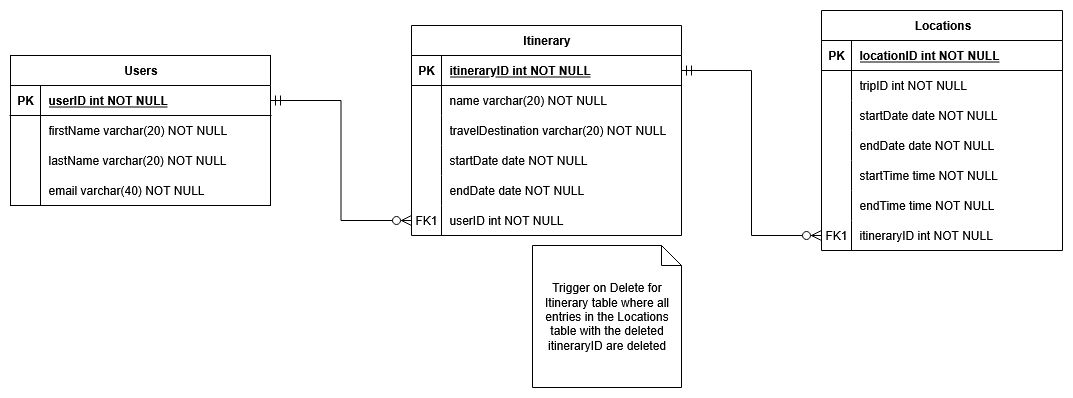
Figure 3.7 View Itinerary



## 

## Storage documentation

Figure 4 ER DIAGRAM



## Message documentation

Figure 5.1 Save Itinerary JSON

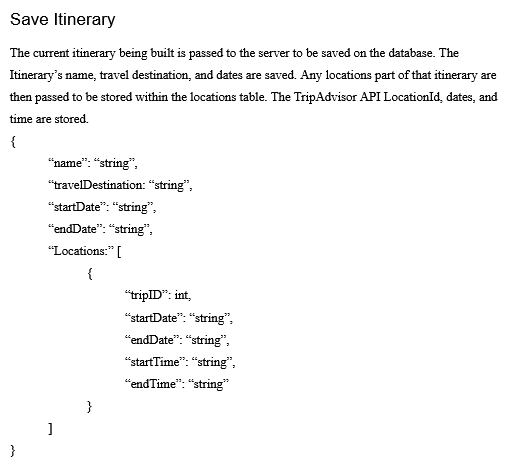
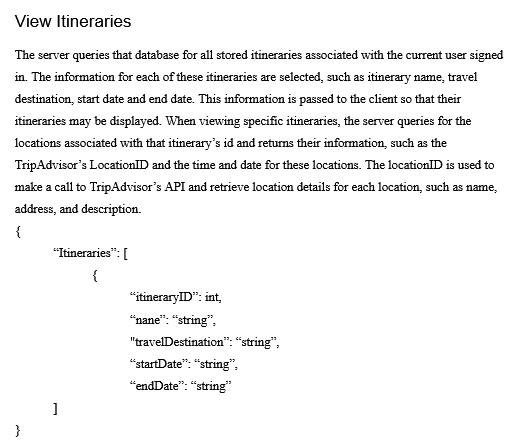


Figure 5.2 View Itineraries JSON



## Misc documentation