PlanParrot
DAT076

Eimer Ahlstedt, Alexander Brunnegård, Sebastian Kvaldén

March 2023

1 Introduction

This report details the creation and usage of a web application called *PlanParrot*. PlanParrot is meant to be a simple, one-stop site to plan your trips. The idea is that instead of going to a few different sites to book flights and hotels and then researching activities, restaurants and more for your destination, a PlanParrot user is simply taken through a wizard where they are presented with all the options and bookings they need. After planning your trip through PlanParrot, you should have all your bookings made for the whole trip.

There was not time to implement all ideas during the course however. For example the wizard contains no step to book your flight or other method or travel. Other possible features that were not implemented are complete trips, where a user could simply browse different already planned trips and choose one they like, and sharing trips between users. This means that features of planning a trip are implemented, and the structure to add more features exists.

To view the code for the application, visit this repo:
https://github.com/Kvalle99/WebAppProject

2 Use Cases

These are the use cases currently implemented in PlanParrot:

1. Log in.
2. Log out.
3. Create a new trip.
4. Switch between different trips on the same user.
5. Select destination for the trip.
6. Search for a destination.
7. View more information about a destination. Currently this means viewing all activities at a destination instead of just a few sample activities.
8. Select a start and end date for the trip.
9. Select accommodation for the trip. Only accommodations in the selected destination can be chosen.
10. Search for an accommodation. Again, only accommodations in the selected destination appear.
11. Add multiple activities from the destination to the trip.
12. Search for activities.
13. See a live updated overview of the trip.
14. Browse the site without logging in. This enables first time users to see the features of the site and potential destinations/hotels/activities.

3 User Manual

This chapter will show how to setup and use PlanParrot.

3.1 Setting up PlanParrot

PlanParrot is currently not hosted on a server. To view the application a user must first clone the previously linked repo. NodeJS is also required to run the project.

Once the files are on your computer, open one console to [repo]/client and one to [repo]/server. Run the command "npm i" in both consoles. Once the command is done, run the command "npm run dev" in the server console, and "npm start" in the client console. After loading, the application should be accessible by navigating to localhost:3000 in a browser.

3.2 Using PlanParrot

In this section a walkthrough of using the features of PlanParrot is given.

3.2.1 Logging in

When first landing on PlanParrot, a user is prompted to either log in or browse without logging in. The log-in page can be seen in Figure 1.

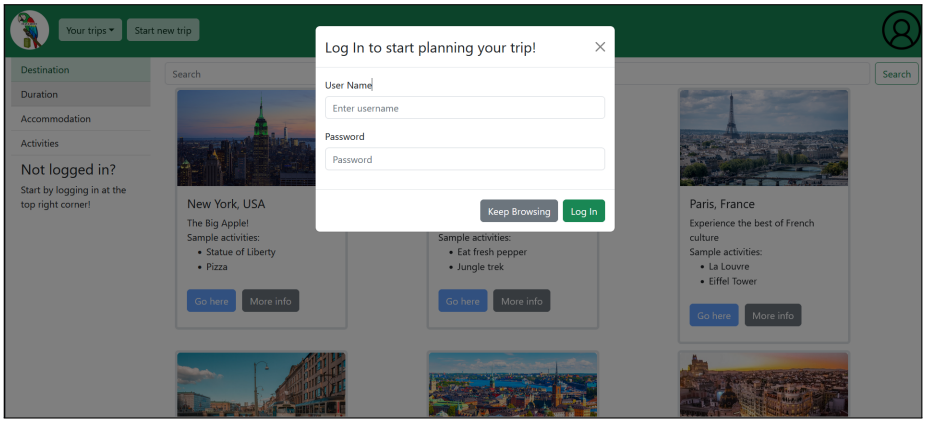

Figure 1: The log-in page of PlanParrot

If the user chooses to browse without logging in, they are unable to create trips, and can only browse the different destinations, accommodations and activities. Currently there is no way of creating a user, so to log in use one of the pre-existing users, for example **User Name:** *robin*, **Password:** *robin*.

3.2.2 Browsing Without a Trip

If a user chooses not to log in, or a logged in user has no currently chosen trip, they can still see all options available in each category. However, the buttons for adding elements to a trip are deactivated and greyed out. The "Duration" page is also disabled. An example of this mode of browsing can be seen in Figure 2.

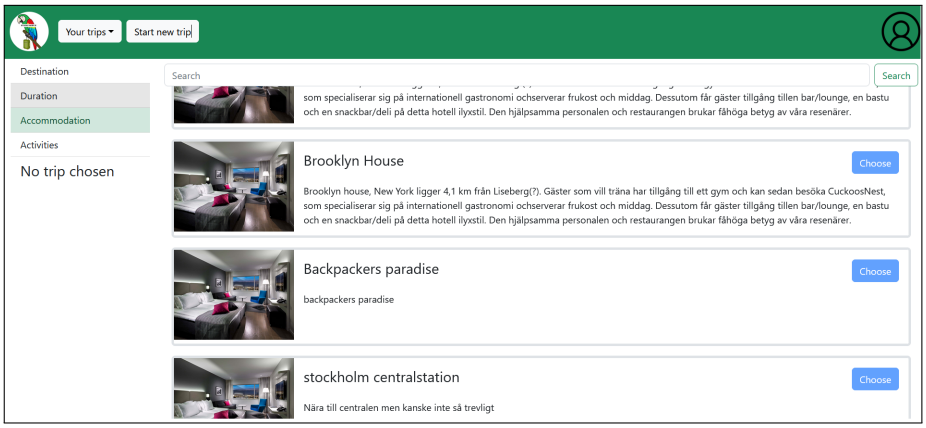

Figure 2: A who is not logged in/has not chosen a trip browsing accommodations

3.2.3 Creating and Changing Trips

To be able to plan a trip, a user needs to have a trip selected. This is done by using the drop-down menu "Your trips". If the logged in user has no trips, they can create a new one by clicking "Start new trip". They are then prompted to give the new trip a name, after which it is selected and added to users list of trips. An example of this flow is shown in Figure 3.

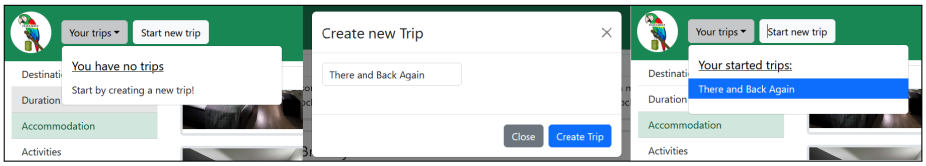

Figure 3: The flow of creating and changing trips

3.2.4 Planning your Trip

The main feature of PlanParrot is planning trips. This is done through a wizard-like flow. First, users can select their destination. The choice of destination

comes first since it dictates both what accommodations and activities will be available. After choosing destination the user can select their accommodation and add any amount of activities they like. The duration of the trip can be chosen at any point. The start date of the trip must be before its end date. Figure 4 shows the Destination Page, which is the main page of the application, with a fully planned trip in the overview.

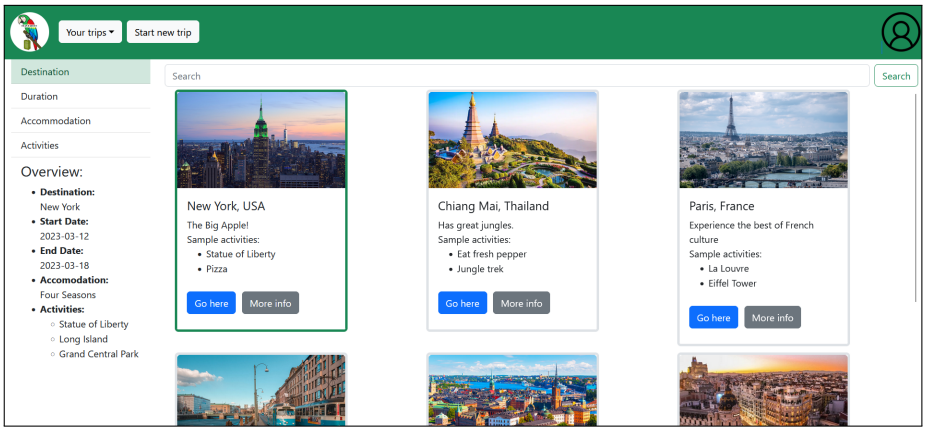

Figure 4: Destination Page and a fully planned trip

Since they have not been shown so far, here are the Duration and Activities pages:

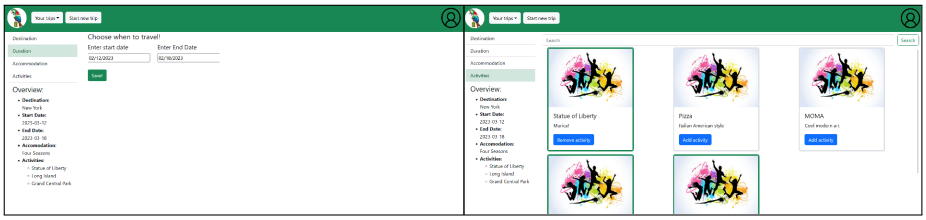

Figure 5: The Duration and Activities pages

4 Design

PlanParrot is made with React, Express and NodeJS. As the last assignment was optional, and we were a bit short on time, we decided to not incorporate a database into the project. We decided that the integration of the database would diminish the time spent on bug fixes, test suites and finalising the functionality of the use cases. Thus, no database is present in the final product.

4.1 TypeScript, HTML and CSS

TypeScript is a superset of JavaScript, meaning that it has additional features as compared to the standard JavaScript programming languages. A notable feature is its typing, enabling compile-time errors. This means that you should explicitly define the type of your variables set return types, making the outcome more deterministic.

HTML is a markup language used for creating structure structure and content of web pages. HTML code can be modified using JavaScript (or TypeScript) via DOM-manipulation, to change the structure of the webpage dependant on logical states.

CSS is a styling sheet language to customize HTML elements. It is partly used to separate the styling to a separate file, making the HTML files less cluttered, more managable and easier to read. It also supports various selectors, such as hover, making simple changes.

4.2 React

React is a library for JavaScript, made by Facebook, used for frontend development. It is oftentimes used to make single-page applications via multiple digestible components, which in turn lowers coupling. These components can be reused whenever necessary, of course depending on how they are made by the developer. The components have a .jsx (or .tsx) file type, which unify both the usual divided HTML and JavaScript (or TypeScript) files to a cohesive component.

4.2.1 React-bootstrap

React-bootstrap was using for easily implementing the frontend components with pre-made layouts, styling and basic logic. It was used primarily to save time, by not having to implement all CSS and HTML from scratch.

4.3 NodeJS

NodeJS is a event-driven runtime environment, which allows efficiency in concurrent connections. It supports asynchronous programming, using callbacks, promises and async/await declarations of these asynchronous portions. By using NodeJS, you can send a request to a database, await the response, and then

act according to the response gotten. While waiting for the response, the thread does not get blocked from handling other calls, thus more calls can be handled concurrently.

4.4 Express

Express is a framework of NodeJS, hence most of the code is already written, to just implement into your own code. The framework can manage routing, sessions, HTTP requests and error handling, per example. In our program it is used especially in the routing process, receiving the calls from the frontend.

4.5 External Libraries

4.5.1 Axios

Axios is a library used for making HTTP requests, whilst allowing asynchronous responses via promises. This enables us to send an HTTP request to the back-end, wait for the response, and then operate on it, using the *.then()*-method.

4.6 Code Design

The code of the application was designed to follow good principles as far as was possible with the given development time and experience. This subchapter discusses a few aspects of the code quality.

4.6.1 Separation of Concerns

In the back-end, separation of concerns is achieved through a layered design. At the core are the model classes in the model layer, representing the logical elements of the application. To work with the back-end, a separate router-layer exists that handles incoming API calls. This router then uses a service layer that accesses and manipulates the model classes. To follow the Dependency Inversion Principle, the router layers depends on interfaces that the service layers implements. There are two service classes that do not implement interfaces. This is because they only export a single function each.

Another aspect of Separation of Concerns is separating between the front-end and back-end. This means that the client code should only contain the logic necessary to display information correctly. We achieve this by making sure front-end classes always make API calls to the back-end to change the logical representation of the application.

4.6.2 Testing

To ensure the code works as expected, tests are implemented. Currently most tests are front-end tests. This is due to time constraints. The back-end contains

tests as well, but with significantly less coverage. Adding more back-end tests is an area of improvement, since it can for example prevent unnoticed bugs.

4.6.3 Cohesion of Model Layer

Currently the model layer does not "use itself" properly. For example, there are classes for destinations and accommodations, but the Trip.ts class simply uses strings instead of these implemented classes. This is far from optimal for many reasons. It makes most comparisons entirely string-based and can create much confusion from the perspective of an external developers, especially on the front-end. This is a major refactor, and carrying it out would be made a lot simpler with a database, which is why we have not done it.

4.6.4 Code Reuse

Some aspects of the application are very similar in both logic and implementation. For example, very little differs between the logic of selecting destination, accommodation and activities. Yet there is very little in the code that would signify this, since all three aspects have completely separate, yet similar, code. An abstraction could be made here, which would make extending the code with more features, such as selecting a method of travel, significantly easier. This is done to some extent, for example with the search functionality, but future development would benefit from such an abstraction.

4.7 Backend APIs

In the backend, we have a few different routers for the varying pages and functionality. Mainly, we have one router for accommodation, destination and activity, which currently are being used to fetch available elements according to the appropriate page. The other two routers are the router for user information, as well as the one for fetching entire trips.

4.7.1 userRouter

UserRouter.get("/login") - gets a token if the user is an appropriate user.
Body - { params: { userName: string; password: string } }
Success - 200
Fail - 401

4.7.2 tripRouter

TripRouter.get("/getTrip") - Fetches a trip depending on ID and userID.
Body - { params: { userToken: string; myId: number; tridId: number } }
Success - 200
Fail - 406

TripRouter.post("/handleActivity") - Adds an activity to the current trip or removes it if it is already added.
Body - { activity: string; dest: string; id: number }
Success - 200
Fail - 406

TripRouter.get("/getActivities") - Fetches all activities on the trip.
Body - { params: { id: number } }
Success - 200
Fail - 500

TripRouter.post("/saveDates") - Saves dates chosen to the trip.
Body - { userId: number; tripId: number; startDate: number; endDate: number }
Success - 200
Fail - 406

TripRouter.post("/changeAccommodations") - Changes the accommodation to the one chosen.
Body - { userId: number; tripId: number; accommodationName: string; accommodationCity: string }
Success - 200
Fail - 500

TripRouter.post("/changeDestination") - Changes the destination to the one chosen.
Body - { userId: number; tripId: number; destinationName: string }
Success - 200
Fail - 500

TripRouter.get("/getMyTrips") - Fetches all trips available for the user.
Body - { userToken: string; uId: number }
Success - 200
Fail - 401

TripRouter.post("/createTrip") - Creates a new, empty, trip.
Body - { userToken: string; userId: number; tripName: string }
Success - 200
Fail - 406

4.7.3 destinationRouter

TripRouter.get("/getDestinations") - Fetches all destinations dependant on the search string input by the user.
Body - { params: { searchText: string } }
Success - 200
Fail - 500

4.7.4 activityRouter

TripRouter.get("/getAllActivities") - Fetches all activities dependant on the current destination of the trip search string input by the user. If no destination is chosen, all activities matching the search string is returned.
Body - { params: { dest: string; searchText: string } }
Success - 200
Fail - 500

4.7.5 accommodationRouter

TripRouter.get("/getAccommodations") - Fetches all accommodations dependant on the current destination of the trip search string input by the user. If no destination is chosen, all accommodations matching the search string is returned.
Body - { params: { destination: string; searchText: string } }
Success - 200
Fail - 500

5 Responsibilities

While most of the website has been worked on by everyone at some point and everyone has done bug fixing, design adjustments and many other minor tasks, these are the main responsibilities each group member has taken:

- **Eimer** has mainly worked on the Activities page, the Destination page and the overview in the sidebar.
- **Alexander** has been responsible for the search functionality, the creation of new trips, making the UI change dependant on states (i.e if a user is logged in or not, if no trips have yet been created).
- **Sebastian** Has mostly focused on the log-in, navigation bar and calendar page functionality. As well as overall structure of the components and a lot of bug fixes all over the application.