Team #3 – Project Documentation

Overall Architecture:

- **Frontend -** Written with TypeScript and Next.js 14, Node 18, and TailwindCSS for styling.
- **Backend -** The backend runs on the Next.js server and is used mainly for routing and proprietary API calls, such as for cloud functions.
- **Database and Authentication -** Are implemented using Google Firebase Framework.
- **Google Cloud Functions -** were also used for overcoming resource bound functionalities such as LLM text generation tasks.
- **APIs -** Google's Places and Directions APIs, OpenMeteo API, and OpenAI's chat completion API.
- **Deployment -** The website is deployed using Vercel for hosting and CI/CD.

Routes:

Here is a list of all routes on the website with a short explanation about each page. All the pages are located under /src/app, inside a folder named with the corresponding route.

- **/:** The landing page, with the login form and link to sign up.
- **/sign-up:** Sign up form.
- **/sites:** The main page of the website, where you can explore and filter from the hundreds of natural sites in database, and choose a place to start planning a trip to. The MainScreen component is the main component of this route.
- **/plan:** After selecting a site in /sites, the user is routed to this page where they can plan their trip, add stops and have a visualization of the trip with google maps API.
- **/my-trips:** A page where the user can view a list of all trips they have planned
- **view-trip:** After selecting a trip in /my-trips, the user is routed to this page where he can view the itinerary with other information about the trip. Clicking on the edit button routes the user to /plan route and lets them edit the chosen trip.

Frontend:

Here is a list of the main pages and components used in the project with a short description (all located in /src):

- **components/LandingScreen.tsx:** The **landing page** containing the **Login/Sign Up** form.
- **components/SiteCard.tsx:** Summarizes all the information about a site in the database, when clicked it opens **dialog** with additional information about the site and a photo gallery.
- **components/PlaceCard.tsx:** Displays the information about a place near a site which can be a gas-station, parking lot or a restaurant.
- **components/MainScreen.tsx:** The view containing the searching & filtering flow, with a set of search results given as a list of SiteCard Components.
- **components/Navbar.tsx:** The navbar shown at the header of the website, with links to the routes: /my-trips and /sites.
- **components/SavedTripCard.tsx:** Shows information about a trip the user created, when clicked, routes to **/plan** with the corresponding trip in context to view and/or edit the trip.
- **components/ItineraryNode:** Represents a stop at a planned trip, used in **/plan** and **/view-trip** pages.
- **components/CommentBlock:** Displays a comment shown in each site's comments list.
- **components/usePaginate:** A custom hook to add pagination to the sites results shown in the /sites route.
- **components/ui:** A folder with UI components used along with all the other components, consisting of proprietary UI components developed by us and modified versions of open-source ones.
- **Context/SiteContext.tsx:** A React Context configuration for the Website. There are 2 context variables: **Site** (current natural site from the db) and **Trip** (current trip the user has planned).
- **components/TripTip.tsx:** Displays the tips and recommendations generated by the LLM prompt, dynamically renders only the subset of tips that are relevant to the specific trip itinerary.

APIs:

Here is a list of all the 3rd side APIs used in the project, with short explanations about their utilization:

- **Google Places:** Used for gathering information about nearby restaurants, parking and gas stations near each natural site.
- **Google Directions:** Used for displaying the route of the trip the user is creating/created.
- **Open Mateo:** Used for gathering live weather information.
- **OpenAI & Cloud Functions:** For generating customized tips & recommendations based on the trips itinerary and timing.

Data and Database:

Here is a list of all the Collections in the database with a short explanation:

- **Locations:** The sites' data was scraped with a Python script using the BeautifulSoup module for storing the sites':
    - Name
    - Short description
    - Difficulty
    - Length of track
    - Duration of track
    - District
    - Images of the site and surroundings
    - GPS coordinates
    - Tags
- Additional data was generated based on the existing model and external APIs –
    - Weather is stored and cached with timestamps, so when a web client's weather query detects the weather timestamp for a site is too old, it's updated on the spot.
    - Waze & Google Maps links are generated based on the GPS coordinates.
- **NearbyPlaces:** Contains information about places near the main sites (gas stations, parking lots, restaurants).
  The data was gathered with Google's Places API and is stored in the database in this collection.
  For each site the collection contains a document with information about the site's nearby gas stations, parking spots and restaurants.

In addition to the name, location, and other static properties of the **NearbyPlace**, the entry also contains the distance from the site, and the Google rating (in stars, out of 5) if applicable, as well as a Waze navigation link.

- **Trips:** Contains information about all the trips created on the website. Includes the fields:
    - ownerId (of the user that created the trip),
    - custom trip name,
    - date,
    - itinerary (array of all the stops in the trip)
    - googleRouteLink (a link to display the trip's route in the Google Maps native/web app).
- **Comments:** All the comments and rankings left by users on the website.

Authentication:

Achieved using Firebase Authentication services. There are two ways to authenticate:

1. Using Google account
2. Sign up with email and password to create a native account.

Utillities:

Scripts in src/lib, used across the web app:

- **handleWeather**: Includes a function which updates weather when stale using OpenMeteoAPI and caches it in the database, and a function which reads the weather information from the database.
- **Search-utils:** Includes the functions which are processing the queries from MainScreen and returns all the relevant sites.
- **Firebase-config:** Configuration of firebase service, alongside with helper functions to handle the database.

LLM Implementation:

To provide additional value for the users they wouldn't be able to get using traditional trip planning sites, and to make the experience of creating a trip even more personal and tailormade.

Using OpenAI's chat completion API, and their latest GPT 4o model for a quicker, multilingual prompt performance, we provide a set of tips and possible warnings to the users during the last step of planning their trip.

GPT 4o was chosen due to it being faster compared to the previous versions, especially the extremely slow GPT 4. And OpenAI also promises better multilingual performance (which we saw in testing compared to GPT 4 Turbo) however the GPT 4 Turbo set of models (4o included) is more susceptible to hallucinations, so prompt engineering techniques had to be implemented, specifically chain-of-thought prompting, serialization to JSON with examples, and tweaks to its Hebrew answering pattern, since no commercially available model is completely versed in Hebrew. (For example, the model tried to transliterate the word "Hydration" to Hebrew when reminding users to bring water to their planned trip in August, so the prompt was modified to include:

```
If necessary to keep the sentence coherent, use the opposite term, for example if you need to translate 'stay hydrated' and you can't find a proper translation for 'hydrated', use 'הימנעו מהתייבשות'. which means 'avoid dehydration'.
```

The prompts are fed with the information about the trip, and specifically the planned main attraction and date, while excluding the (current) weather information to prevent it from confusing the model with the actual weather at the time of the trip. Then the LLM is prompted with generating 4 pieces of information:

1. **Weather and conditions** – if the users should be informed about the weather conditions on the site, heatwaves in the summer, mudslides in the winter, vermin and bugs, etc.
2. **Possible inactivity or unavailability of the site's attractions** – for example if a reservoir is dry during the summer, if the site is closed during the period of the trip, or if the blooming flowers of the site are only showing during the spring, but the trip is planned for autumn.
3. **Reminders** – depending on the attraction and time of year, this part would remind the users to prepare and bring equipment and provisions that might be necessary for the trip. Hats and shoes for long walks, picnic equipment and snacks for rest areas if they plan on doing so, and swimsuits if there are water sources around the site.
4. **Summary** – A short, tailormade sentence of regard wishing the users a fun trip.

Since it’s not a running part of the codebase, the content of the Cloud Function, which includes the prompts, a modified copy that doesn’t produce errors due to dependencies on the firebase-functions library is stored in src\lib\tripRecommendation-cloud-function.ts