38883171 – Reflective Report – Sprint 2 – Final

lu-lvb-JourneyPlanner

https://github.com/JRamirezDD/lu-lvb-JourneyPlanner

Requirements:

For this sprint I was responsible for making the application accessible by the end user. This meant putting the components together into a single, coherent page.

The internationalization implementation had to be re-done to be compliant with the requirements proposed by LVB, which meant localizing the translation logic.

Setting up the hosting and CD jobs were also part of this sprint's tasks. The page has been made available at: https://jramirezdd.github.io/lu-lvb-JourneyPlanner/de

I also took over the task of migrating from MapBox to MapLibre, as well as setting up the MapWidget component to enable displaying layers and reacting to source data change and interactions observed in the contexts.

Architecture:

When looking at the implementation of the main page, multiple UI Patterns and Design Patterns were followed. The main ones include:

Observer pattern:

useEffect hook listens for windows resize events and modifies the behavior of the component after certain threshold.

Conditional Rendering:

Different versions of the page is rendered based on a condition, in this case resolution.

With the goal of hiding away the API keys away from the client, a proxy solution that injects the API keys into the client's requests was needed. This proxy solution is hosted server-side, and it receives the user request to certain endpoint, injects the API Keys, and forwards the request to the API. This design pattern is called *API Gateway*, and it's commonly used in web applications. Additional benefits include rate limiting, logging, request validation, etc, protecting the API from abusive requests by the clients. These endpoints are accessible under /api/proxy/lvb/...
This *API Gateway* pattern also solves any CORS issues by giving the client a valid origin address.

Considering the hard requirement of hosting the application in a static manner, we provide 3 options for the application building.

- By running npm run build:frontend the application is built in the next.js 'export' mode, which only builds the components required to serve the application as a static site. This also has the advantage of being preprepared and optimized for client rendering.

- By running npm run build:backend, the application is built excluding the /src directory, which includes the front-end components. Only the api routing functionality is included in this build mode. This part of the application can be replaced with an external component.
- By running npm run build:standalone, the entire application is built in the next.js standalone mode, which bundles all components into a self-contained directory with all the next.js functionality.

As part of the hosting solution, a 2-part CD automated CD Job has been created.
https://github.com/JRamirezDD/lu-lvb-JourneyPlanner/actions/runs/13570018874
This is ran whenever there is a push to the main branch, ensuring that the latest version of the application is always available for the customer. The build:backend output is hosted on Vercel, and the build:frontend output is hosted on GitHub Pages.

Testing

The following tests were developed:

MainView Integration Test -
Ensure that MainView and the children components are rendering and are accessible as expected.

1. Create a mock maplibre-gl instance.
2. Render the MainView component with the accompanying required contexts.
3. Verify that control panel buttons were rendered as expected.
4. Verify that the mock mapbox-gl instance was rendered.

Unit Test of nearbysearch mappers –

1. Create mock json objects of the API objects
2. Individually test that the raw JSON object directly compares with the mapping result.
3. Attempt to map an entire sample response, and validate that the outcome is as expected.

Impediments and plans to address them

- GitHub pages does not support implemented proxy solution.
    - GitHub Pages is designed to host only static websites. This means HTML, CSS, Javascript, and other client-rendered functionality. Server-side functionality such as API routes, which require a server to handle dynamic content and execute server-side code, are not supported.
    - Solution was to split hosting in 2 platforms, as described earlier.
- Miscommunication regarding map implementation from customer to developers
    - It was not specified which platform to utilize for the implementation of the map. Developers assumed that MapBox would be expected, as it's what is used on the LeipzigMOVE mobile app. Nevertheless, the client intended for us to utilize OSM/Leaflet.
    - Given that both MapBox and Leaflet allow for data importing via GeoJSON, an agreement was reached with the customer that the development with an intermediary package called 'MapLibre' would continue, and if the project is used by the client, further efforts after the final delivery will be done to transition into OSM/Leaflet, which should not be very complicated.