38883171 – Reflective Report Final

lu-lvb-JourneyPlanner

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

Requirements:

My responsibilities for this sprint mainly revolved around preparing access to the lvb APIs from the UI components, providing the automated testing framework, and coordinating and organizing the application's development. As the project moved forward, I concluded that developing the component-communication framework would also fall into my responsibility. For a more detailed overview on the specific tasks that I performed, refer to *Appendix 1.1 – Originally Assigned Requirements* and *Appendix 1.2 – Additional Completed Requirements*.

In addition, I was responsible for preparing access to the LVB APIs, setting up the automated testing framework, and coordinating development. To support development, I created example components and pages that exemplify the use of the Context API implementations. Additional to my assigned tasks, aiding group members in their implementations also took up a significant amount of time. Additional contributions to the Map Component and the Internationalization framework was required on my end.

Architecture:

An unrefined Use Case diagram was used for the initial definition of the required components, interactions, and external sources of data. Refer to *Appendix 2.1 – Use-Case Diagram Draft*.

The main architectural choice came when deciding how the UI Components would communicate with each other (marked as Data-Transfer in the Use-case diagram).

When looking at other OTP-UI Implementations, they all use a methodology named Props. Nevertheless, we believe that Context APIs are the superior approach, as they allow for increased modularity, easier extensibility, and simpler communication of objects via a "subscription" model. Refer to **Component-Communication Design Decision and Implementation** and *Appendix 2.2 – Data-Flow Diagram* for more details.

Our implementation of Context APIs consist of 2 parts: The *InterfaceContext*, which is used for defining which attributes and functions should be exposed to the subscriber, and the *ContextProvider*, which details the actual implementation of the specified methods. For a component to utilize the *ContextProvider*, first, a subscribing component will declare its intention use the *InterfaceContext* via accessing the *useContext()* hook, whose goal is to provide type safety and availability checks. The parent component (ideally the view page) will then instantiate the *ContextProvider*, which replaces the value that is returned by the *useContext()* hook.

Testing

Application tests are implemented via Jest. Ideally, we aim to create unit tests of all the minor parts, but core, of the application, such as data format conversion, layer generation, reactions to context changes, among others. With the time provided, this is very ambitious, so consider us satisfied if non-ui components are fully tested.

This sprint I implemented tests that ensure that incoming data from the LVB is correctly mapped into in-application objects. Considering that JavaScript objects are naturally transformable into JSON format, I set up a direct comparison between the data before and after transformation, which aims to ensure that there's no data malformation.

Automated Tests

GitHub job (https://github.com/JRamirezDD/lu-lvb-JourneyPlanner/blob/main/.github/workflows/node.CI.yml) was created to ensure that commits to the main branch are compliant. Its goal is to ensure that the CI process of the application runs as expected via the following steps: Install dependencies, Run Jest tests, Build project

Impediments and plans to address them

- Context APIs
  - Goal was to hand over tasks to team member to give me time to work on supporting less capable team members, but he fell ill during the critical development time. I proceeded to develop the component-communication framework myself, and although some shared attributes are likely missing, it's finished for the most part.
- Map Layers
  - I had to involve myself a lot more than expected for the implementation of the Map Component. Team members are having issues standardizing the OTP data import into the MapBox component. Will create a framework with interfaces to simplify approach, and team members will just have to set-up filters for the imported OTP data.
- Map rendered by client
  - I'm not fully familiar with the correct approach to using and loading layers into MapBox, but currently the raw data is provided to the client, and then it renders the map. This might be an issue further on, and we might have to render the map first at the back-end, and then share it with the client.
- API-Key used by client
  - API-Key needed to access the LVB APIs is currently exposed to the client. This is not ideal, and it poses a security risk. We'll have to create a back-end API or proxy that injects the API-Key for accessing these resources.
- CORS
  - LVB APIs block requests from non-public IP addresses through CORS. Hopefully the same proxy as implemented for the API-Key injection will solve the issue. Currently using a public proxy for development.

Appendices

Appendix 1. - Requirements

Appendix 1.1 – Originally Assigned Requirements

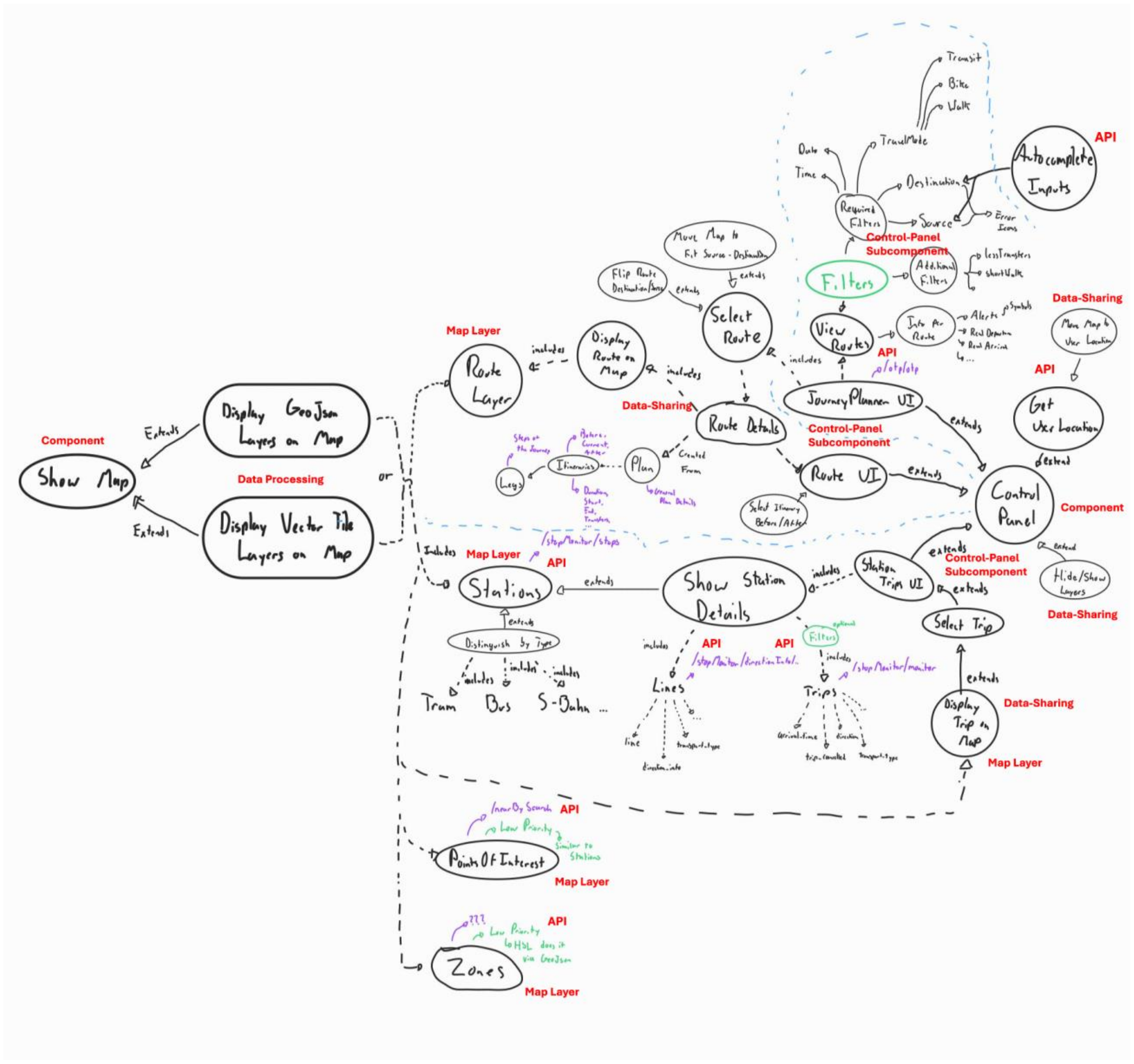
GH Issue #	Title	Estimated Size	Status
#1	Document APIs in shared PostMan Workspace	XS	COMPLETED
#2	Create Use Case Diagram for Requirement Interaction Overview	S	COMPLETED
#4	Create Repository Skeleton	XS	COMPLETED
#5	Create App Skeleton with Plugins	S	COMPLETED
#6	ESLint Formatting Rules	XS	PENDING Basic rules implemented.
#7	Prettier Formatting Rules	XS	PENDING
#8	autocompleteService API Implementation	XS	COMPLETED
#11	Create Sprint 1 Requirement Tasks and Distribution	S	COMPLETED
#12	nearbysearchService API Implementation	XS	CANCELLED Extra effort, not demanded in requirements.
#13	routingService API Implementation	XS	COMPLETED
#14	stopmonitorService API Implementation	XS	COMPLETED
#28	BaseMap Implementation	L	PENDING

Appendix 1.2 – Additional Completed Requirements

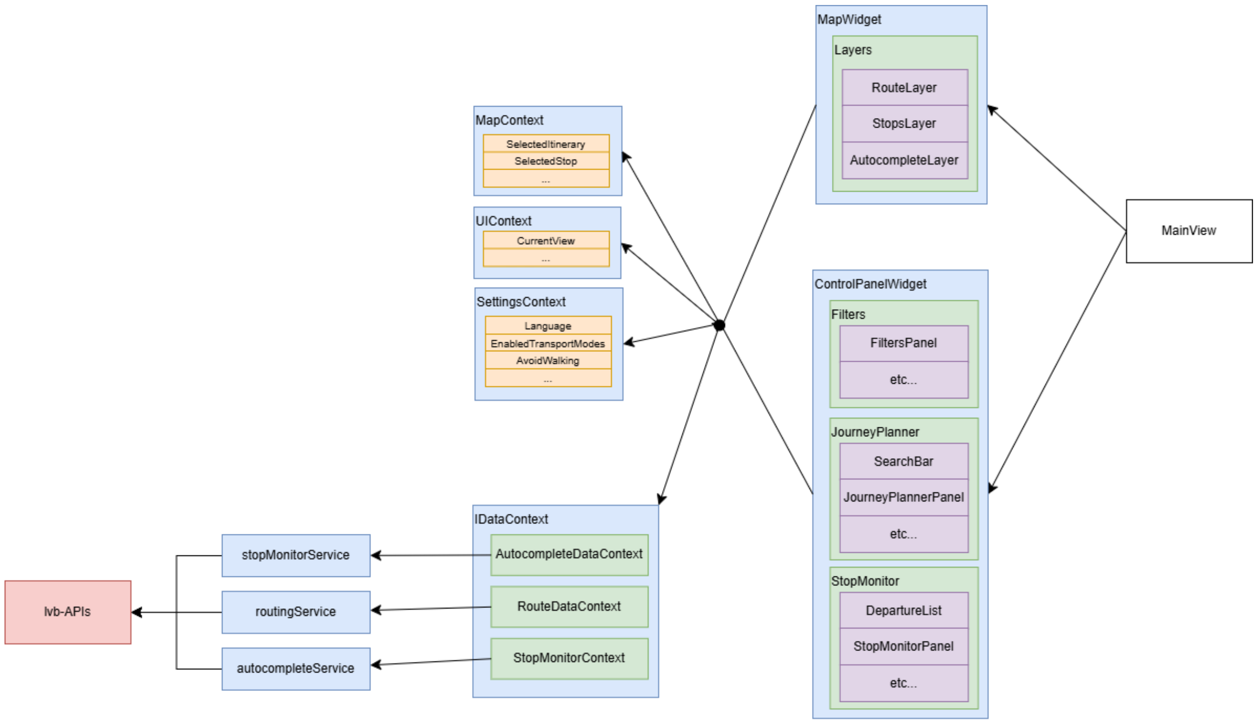
GH Issue #	Title	Estimated Size	Status
#15	Component-Communication Design Decision and Implementation	L	COMPLETED
#18	Implement automated unit test execution for PRs	M	COMPLETED
#19	Add interface toGeoJson for DTO objects	XS	COMPLETED
#30	Create MapContext	XS	COMPLETED
#31	Create UIContext	XS	COMPLETED
#32	Create TripContext	XS	CANCELLED
#33	Create DataContext Interface	M	COMPLETED
#34	Create SettingsContext	XS	COMPLETED
#35	Context API Interface	M	COMPLETED
#40	Removing API-Key from client	XS	PENDING
#41	Create autocompleteDataContext	S	COMPLETED
#42	Create routingDataContext	S	COMPLETED
#43	Create stopmonitorDataContext	S	COMPLETED

Appendix 2. – Requirements

Appendix 2.1 – Use-Case Diagram Draft



Appendix 2.2 – Data-Flow Diagram


MapWidget
Layers
RouteLayer
StopsLayer
AutocompleteLayer
MapContext
SelectedItinerary
SelectedStop
...
UIContext
CurrentView
...
SettingsContext
Language
EnabledTransportModes
AvoidWalking
...
MainView
ControlPanelWidget
Filters
FiltersPanel
etc...
JourneyPlanner
SearchBar
JourneyPlannerPanel
etc...
StopMonitor
DepartureList
StopMonitorPanel
etc...
IDataContext
AutocompleteDataContext
RouteDataContext
StopMonitorContext
stopMonitorService
routingService
autocompleteService
lvb-APIs