38883171 – Reflective Report Draft

lu-lvb-JourneyPlanner

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

Requirements:

https://github.com/JRamirezDD/lu-lvb-JourneyPlanner/issues?q=is%3Aissue%20assignee%3AJRamirezDD

Originally Assigned Requirement

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	nearbysearchsearchService 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

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

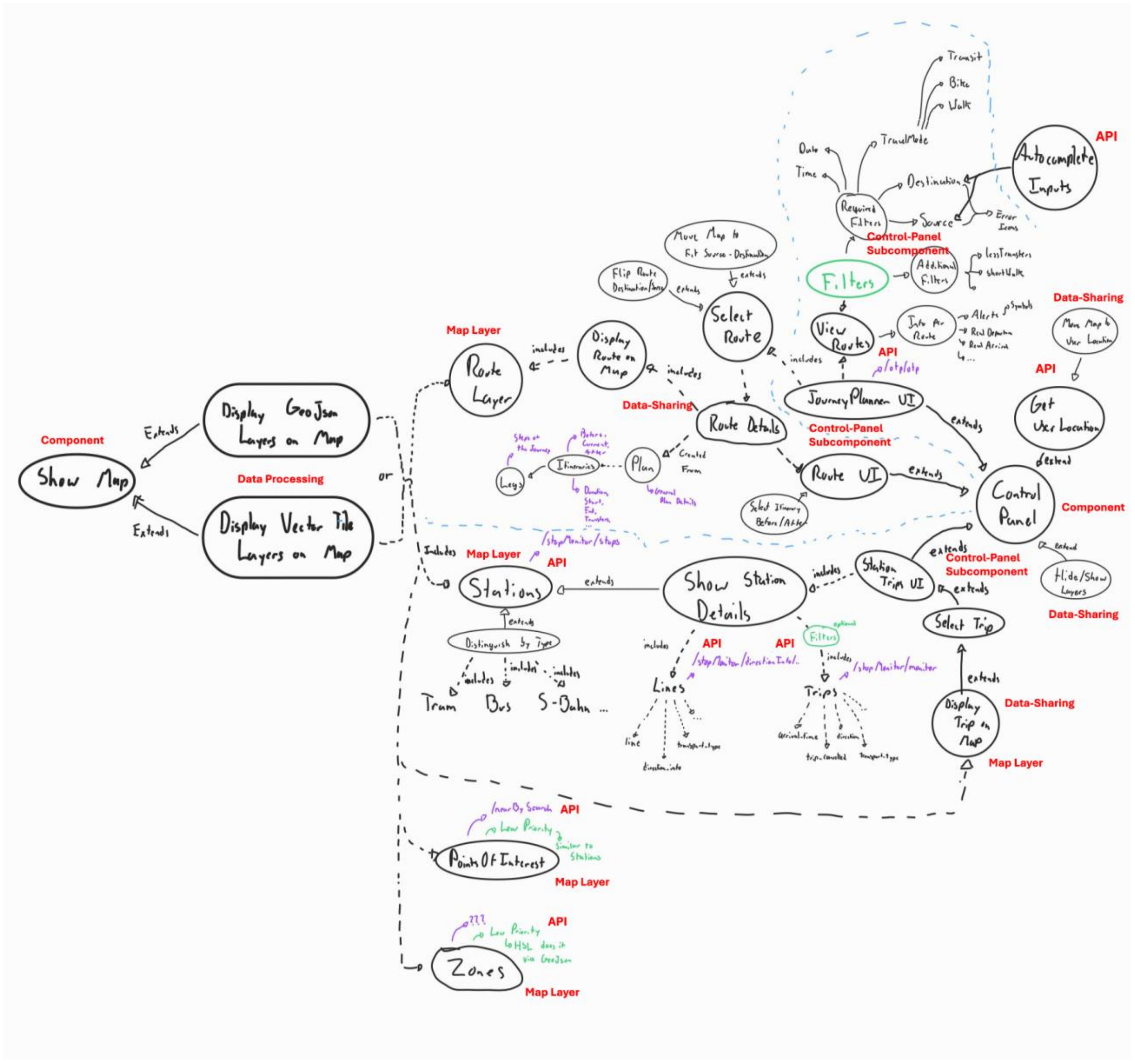
#42	Create routingDataContext	S	COMPLETED
#43	Create stopmonitorDataContext	S	COMPLETED

Architecture:

Reasoning about chosen architecture and potential alternatives

Identifying architectural components

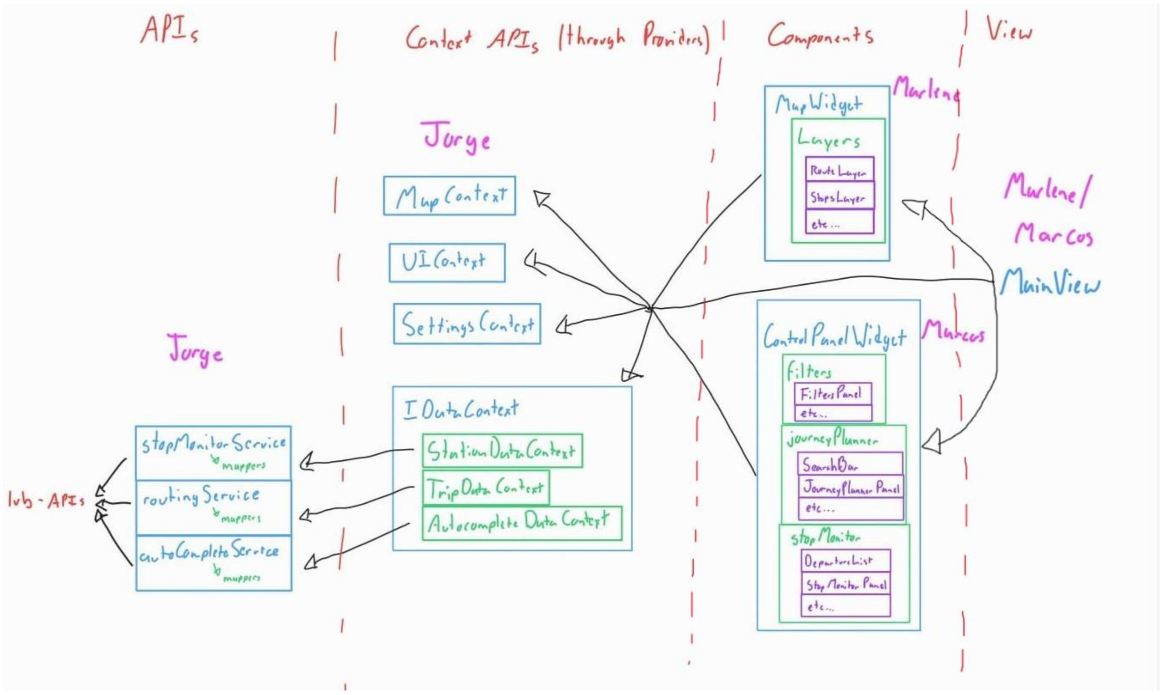
An unrefined Use Case diagram was used for the initial definition of the required components, interactions, and external sources of data.



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 and simple extensibility. **Component-Communication Design Decision and Implementation**

Context APIs can naturally be integrated with the rest of the components in the layered architectural pattern.



Testing

Description of testing approach and (automated) tests

Application tests are implemented via Jest.

Tests can be manually ran using 'npm test' when in the app's folder.

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 provided time, this is very ambitious, so consider us satisfied if non-ui components are fully tested.

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 test that the CI process of the application runs as expected via the following steps:

1. Install dependencies
2. Run Jest tests
3. Build project

If a commit fails one of these steps, it will not be allowed into the main branch.

Impediments and plans to address them

Reflection on why something does not work (yet) and how this can be addressed

- Context APIs
    - Goal was to hand over task to team member to give me time to work on supporting less capable team members, but he fell ill during the critical development time. 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
    - 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 in the back-end, and then share it to 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.

Contributions

- Contributions
    - Component definition.
    - Architectural approach.
    - Requirement definition.
    - Task distribution.
    - Answering questions and helping team members implement their components.
    - Implementation of HTTP Client.
    - Implementation of Automated Tests on GitHub.
    - Implementation of Environment Variables.
    - Implementation of API-services to communicate with LVB-APIs.
        - sampleService
        - stopMonitorService
        - routingService
        - autocompleteService
    - Conversion of Raw API Data Objects into application Entities with tests.
    - Entity Enums, Interfaces, Data Objects.
    - Implementation of Context Interface and Contexts.
        - sampleContext
        - Context Interface
        - settingsContext
        - uiContext
        - mapContext
    - Implementation of Data Context Interface, standardized API access hook, and Data Contexts.
        - Data Context Interface
        - routingDataContext

        - stopmonitorDataContext
        - autocompleteDataContext
    - Example components and pages for other team members to reference.
        - Components (under /components/_sample)
            - autocompleteComponent
            - counterComponent
            - settingsComponent
        - Pages (under /app/examples/contexts)
            - autocomplete
            - counter
            - multi-component
            - trigger-function
- Shared Contributions
    - Map Layers
        - Interface Layer
        - ItineraryLayer
    - Map Widget