Architecture Document

Sharing logistics simulation

Client: Astone Shi
Version 1.0
Team:
Antonin Thioux
Bjar Karim
Gheorghe Pojoga
Lonneke Pulles
Lorenzo La Rocca

Under supervision of Alex Tutea
2020

1 Introduction

The product will be a web application that is simulating the effect that sharing space for goods would have. The application's main goal is to determine if it would be a worthwhile idea to set up a platform on which companies or truck owners could offer empty space to transport other people's goods.

The user can set multiple parameters for the simulation:

1. the start and end locations,
2. the number of trucks for each of three predefined types, and
3. the types of goods, which each have a volume, a weight and a quantity.

To simplify the simulation in first instance, our client only needed to be able to enter the start and end location of a trip. In the future, this may be extended to include locations in between those points.

The three metrics that will show the result of the simulation are

- transport time
- CO2 emissions
- number of trucks used

2 Architectural overview

The project does not need a persistent database, because all simulated data will be calculated on the spot every time. This web application is designed without a database or back end and so the front end makes up the entire application. We thought that the calculations for the simulation are simple enough to be able to do them in the front-end application. Adding a back end and thus a necessary API to connect the two applications would make the project unnecessarily complicated. If the application should be extended to store the results of the simulation in the future, a database and back-end application can be added to the project without having to change too much in the already existing front-end application.

The application is mainly created with Vue.js. This is a lightweight and open source front-end Javascript framework that mainly focuses on simplifying the creation of a dynamic Single Page Application. A Vue project is split into two folders: a folder for static content called 'public' and a folder for reactive content called 'src'. This last folder contains javascript files and so-called Vue (view) components, which can be composed of other Vue components. Each component consists of three parts: an html template, a Vue part (where local data is stored and methods are created) and a css style part. These types of components are called single-file components and this is what largely characterises .vue files.

In a more abstract way, the Vue app can split into three parts: the state, actions and the view.

The state consists of a store with the data, the actions are defined as functions in the .vue files and indirectly called by the user and the view consists of Vue components, which can be visualised as a tree. For our project, the view is composed of the following Vue components:

The root component of the view is called App.vue. This App.vue file is linked to by a file called main.js, which first renders the app and then mounts it.

The top-level components are the two views, or web pages, called Home.vue and OutputPage.vue.

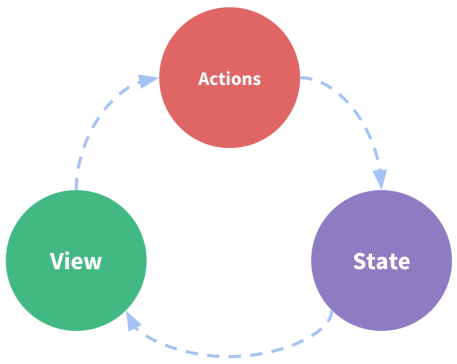


Figure 1: Three main parts of a Vue application.

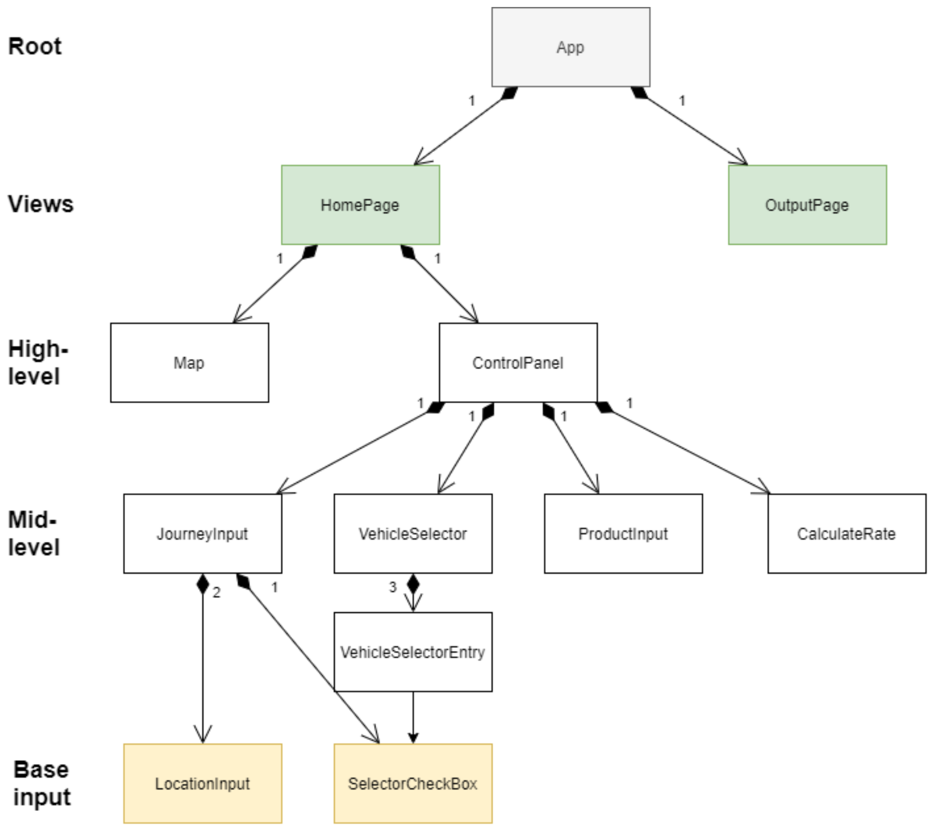


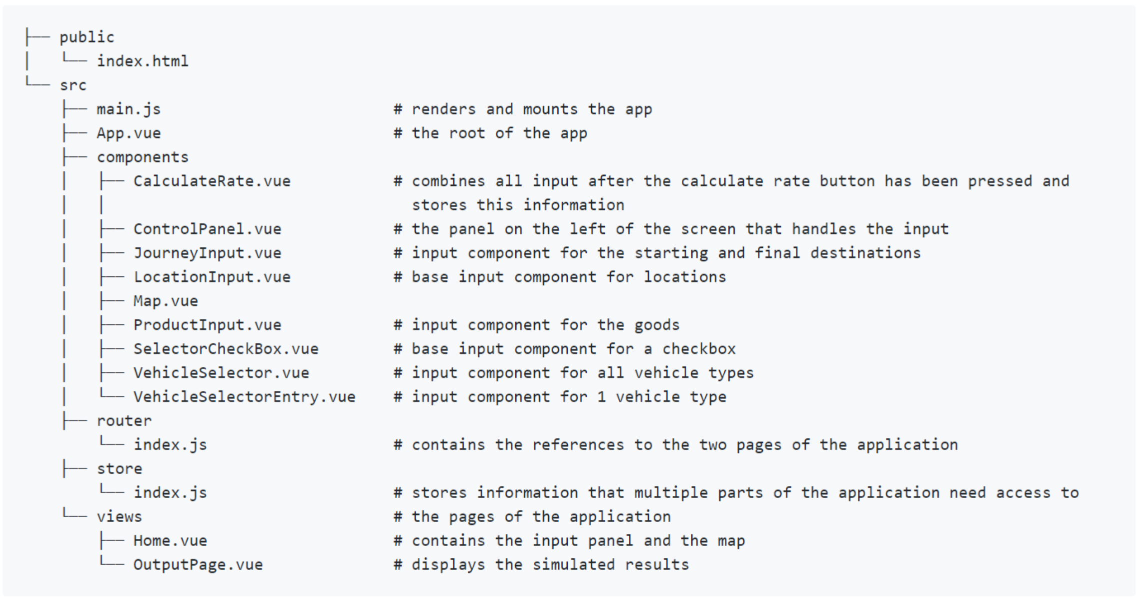
Figure 2: The tree structure of our application's view.

The Home view consists of two high-level components, ControlPanel.vue and Map.vue. In the ControlPanel the user can enter their input and choose values for the given parameters that will determine the simulation.

These high-level components use the middle-level components JourneyInput, ProductInput, VehicleSelector and CalculateRate.

These middle-level components, on their turn, may use the base components: LocationInput and SelectorCheckBox.

The exact file structure and explanations of what the responsibility of each file is, can be found below.



```
├── public
│   └── index.html
└── src
    ├── main.js                     # renders and mounts the app
    ├── App.vue                     # the root of the app
    ├── components
    │   ├── CalculateRate.vue       # combines all input after the calculate rate button has been pressed and
    │   │                             stores this information
    │   ├── ControlPanel.vue        # the panel on the left of the screen that handles the input
    │   ├── JourneyInput.vue        # input component for the starting and final destinations
    │   ├── LocationInput.vue       # base input component for locations
    │   ├── Map.vue
    │   ├── ProductInput.vue        # input component for the goods
    │   ├── SelectorCheckBox.vue    # base input component for a checkbox
    │   ├── VehicleSelector.vue     # input component for all vehicle types
    │   └── VehicleSelectorEntry.vue  # input component for 1 vehicle type
    ├── router
    │   └── index.js                # contains the references to the two pages of the application
    ├── store
    │   └── index.js                # stores information that multiple parts of the application need access to
    └── views                       # the pages of the application
        ├── Home.vue                # contains the input panel and the map
        └── OutputPage.vue          # displays the simulated results
```

Figure 3: Main file structure of the application.

Note that each element of the view can access the application's store.

Since Vue is meant to create Single Page Applications, we added a router to the project. What this router essentially does is mounting one of the two views (Home or OutputPage) to App.vue.

3 Technology Stack

The following is a list of used technologies and frameworks:

- HTML
- CSS
- JavaScript
- Vue.js
  - A Progressive JavaScript Framework
  - Libraries used:
    - Vuex: a state management pattern and library for Vue.js applications. Enables us to create a store which can be accessed anywhere in the application

        * Vue Router: enables us to map Vue components to routes and Vue Router renders them when needed. Makes it seem as if the application has multiple pages.

- npm
    - Node Package Manager. Required to install and run Vue.js
- Node.js
    - Required for npm. A back-end Javascript framework.

For the map, we use:

- Leaflet, an open source Javascript mapping library.
- Open source alternative to Google Maps API. Javascript mapping library.
- Plugins we used:
    - vue2-leaflet - it provides wrapping constructs of the basic Leaflet objects, which are designed for the Vue framework.
    - leaflet-routing-machine - this library is used for all the functionality that is related to routing e.g., finding the route,displaying the route, determining the distance and time necessary to complete the route etc.
- leaflet GeoSearch
    - To enable searching based on addresses and names instead of coordinates.
    - Though it is called leaflet GeoSearch, it has no dependencies on Leaflet.

3.1 Why we chose these technologies

3.1.1 Vue.js

Since nobody in our team had any experience with creating web applications, we couldn't make the decisions based on our own experience. We wanted to make a choice between the three most popular front-end Javascript frameworks, Angular, React and Vue.js. These frameworks allow Javascript code to be structured into components.

Since is Vue.js is said to have the lowest learning curve out of the three frameworks, we decided to go with Vue.js. Our TA also recommended this and on Stackshare.io it can be seen that Vue.js has the highest approval ratings.

Some other advantages of Vue.js are that it

- is lightweight and therefore quite fast, especially in comparison with Angular.
- is an MVC framework.
- is the newest technology. It is meant as an improvement over Angular and React.

Some disadvantages are that it

- may have a smaller community and less documentation. However, we considered the documentation to be more than good enough.
- has fewer libraries than Angular and React. However, the framework offers more than enough libraries to suit our project's needs.

3.2 Leaflet

Initially, we wanted to use the Google Maps API to show the map in the application, to show markers on the map, to find the route between the start and end locations and to calculate the time and distance needed for that route. Later on in the project we found out that we had to fill in payment details for the routing functionality. Though it would initially be free, we would have to pay if the application made more than a certain amount of requests per month. Since we didn't want to have the risk of one of us or our client having to pay a lot of money unintentionally, we decided to use an open source map API instead.

Leaflet is the most-used Javascript library for interactive maps. After some searching, we found that it had all features we needed if we used some extra plugins. It allows us to connect to OpenStreetMap, a free wiki world map.

4 Team organization

There is one team in our project. Each member of this team has their own Vue component to work on, however. We created the architecture together and then divided the components. When two components needed to be connected, the two team members worked on it together. Moreover, if anyone needed help, we of course helped our team mate with their part.

The division was as follows: Antonin created the VehicleSelector, Bjar created the ProductInput, Gheorghe created the Map and the store, Lorenzo created the ControlPanelOutput and Lonneke created the JourneyInput.

5 Change log

Who	**When**	**What**
Lonneke	11-3-2020	First draft of the architecture document.
Lonneke	24-3-2020	Updated technologies and added introduction, file structure and image of architecture.
Lonneke	4-4-2020	Added why we chose technologies and blackbox description in introduction.
Gheorghe	6-4-2020	Added the plugins necessary for the map