Flapjack Documentation – Frontend

Contents

1 Introduction **2**

2 Requirements, Building and Executing **2**

2.1 Requirements 2

2.2 Environment Variables 2

2.3 Building 3

2.4 Running 3

3 File Structure **3**

4 Redux Store **4**

4.1 Persistence 4

4.2 Store: Session 5

4.3 Store: Plots 5

4.4 Development Tools 5

5 Communication with API **5**

5.1 Authentication 6

5.2 HTTP Requests 6

5.3 Web Socket Communication 6

1 Introduction

The frontend application is written with JavaScript, using the frontend library React for an interactive design, the D3 visualization library via Plotly and Redux for global state management and data persistence. This application is intended for convenient data exploration and basic analysis, and, as well as the backend, can be deployed using docker for ease of setup.

This document aims to describe the application's key structures and functions for development purposes.

2 Requirements, Building and Executing

2.1 Requirements

This app is built with React, running under the JavaScript runtime environment Node.js, and deployed in containers via Docker. All dependencies are handled via the Dockerfile within the project source.

As such, the only pre-requisite for running the application is a modern version of Docker.

2.2 Environment Variables

To be able to protect secret keys or differentiate between development and production environments, an environment file must be created within the project root, and named `env.prod` for production environment and `env.dev` for development environment. These files must have the following fields:

```
REACT_APP_HTTP_API="<API HTTP url>"
```

```
REACT_APP_WS_API="<API WebSocket url>"
```

Future environment variables which are expected to be used within the code (via `process.env`) must have a name starting with `REACT_APP_`.

2.3 Building

As mentioned previously, dependencies are downloaded and installed in a docker container when it is built. To build the application, run:

```
docker-compose -f <docker-compose path> build
```

Where `<docker-compose path>` is `./docker-compose.dev.yml` for development environment and `./docker-compose.prod.yml` for production environment.

2.4 Running

After creating the environment variables and building the container, the application can be run with:

```
docker-compose -f <docker-compose path> up
```

Where `<docker-compose path>` is `./docker-compose.dev.yml` for development environment and `./docker-compose.prod.yml` for production environment.

3 File Structure

The application folders are organized as follows:

- `public`: Contains the entrypoint HTML file and other public files.

- `src`: Contains the application main source code, including React containers, helper and store files.
  - `api`: Contains files with classes that handle communication with the api. See section 5.
  - `Components`: Contains folders with react components for each different view or collection of components.
  - `redux`: Contains a folder for Redux reducers, another folder for Redux actions and `store.js` which configures the Redux store. See section 4
  - `routes.js`: File containing the web page's routes. It contains an array of routes with the following format:

```
{
    label: '<Name of route>',
    route: '<Path of route>', // E.g.: /browse
    viewRenderer: <Component to render view>
    navbarRenderer: <Component to render in navbar>, // Optional
    requiresAuth: <true or false>, // Whether view requires an
   authenticated user
}

```

Files are documented in the code as needed, using JSDoc's documentation syntax.

4 Redux Store

To maintain a global state within the React application, Redux is used for storing user data and user created plots.

4.1 Persistence

To persist relevant data between sessions, the library `redux-persist` is used, achieving a connection with the browser's local storage.

4.2 Store: Session

This store contains user information like his username, email, access token and refresh token (see section 5). All fields, except for the access token, are persisted between sessions within the browser's local storage.

4.3 Store: Plots

This store, also named `viewTabs`, persists the user created plots in the local storage, maintaining relevant information for reconstructing them between sessions. This relevant information includes:

- Tab title
- Tab id
- Plot Data: Containing the data and layout passed to a Plotly plot.

4.4 Development Tools

While in the development environment, the store is wrapped with a library called Redux Devtools Extension, which allows the developer to access the store using the appropriate extensions in Chrome or Firefox.

5 Communication with API

The folder `src/api` contains files that implement interfaces for communication with the Flapjack API.

5.1 Authentication

Authentication with the API is handled via JSON Web Tokens, using two different tokens:

- Access Token: Required for all authenticated endpoints. Expires after 5 minutes and is not stored in the browser's local storage.
- Refresh Token: Used for obtaining a new access token. Expires after 1 day and is stored in the browser's local storage. When the webpage starts up, this token is used for restoring the user's session.

5.2 HTTP Requests

The file `src/api/index.js` implements an interface for communication with the Flapjack API via HTTP requests, automatically handling access tokens, query string building and authentication.

This interface implements functions for `GET`, `POST`, `PATCH` and `DELETE` requests to the API, as well as other functions for registering a new user, authenticating a new user, refreshing the access token and signing out the current user.

5.3 Web Socket Communication

The file `src/api/apiWebSocket.js` implements an interface for communication with the Flapjack API via web sockets. This interface handles authenticated connection with the web socket endpoint and event listener asignation.