Dashboard developer manual

Table des matières

Glossary

Widget: A small component, capable of visualizing or editing small data

JWS: JSON Web token, JavaScript object authentication token to access or refresh access to an application

OAuth2: Industry standard authorization protocol to secure API data

Modal: Window within a webpage that takes control of the input

CRUD: Create, update, read, delete, a way of managing data through four possible actions.

The dashboard's concept

The dashboard's principle

This web application consists in an interface composed of multiple panels, each representing a service that is linked to a web application. Those panels are what we call widgets. A service is an API of another application or website such as Spotify, Google, Github, Facebook and any other service whose API might be open to developers.

In other to use a widget, the user is asked to link his Dashboard account to a service we might work with, using the OAuth2 protocol to get an access token to the specified service and a refresh token to maintain the link to the OAuth2 account.

Once the user has done so, he will be able to add any of the widgets related to the service he linked to. The widgets may have two main functionalities:

- Search: The user has access to a text input within the widget which allows him to search a specific document or data through the service
- Display: The user has no form of input whatsoever and get some form of content from the service directly displayed in the widget

The objective of the application is to centralize the content of many more services so the user might access more information within a single webpage.

The dashboard is also modulable visually speaking, the user might be able to expand or reduce a widget, allowing to choose the amount of information to be displayed or even move widgets and saving the layout they prefer.

The main features of the web application

Now that we properly understand the functioning of a generic dashboard, we'll see here how the Autosuffisant's Dashboard performs and what are its key features.

Authentication system

When the user connects to the application for the first time, he must create an account to access the dashboard. Here are the two possible ways of registering

Manual form registering

The most common way of creating an account is via a classical form. The user will be asked to put some personal information as well as a password with some main security features, such as:

- A minimum of eight UNICODE characters
- At least one digit number
- At least both an uppercase and lowercase alphabetical character

After putting every detail such as his username, email and password, the user will have it's account registered and will be promptly redirected to the dashboard, the API will generate a JWS refresh token, allowing the user to come back later in another session without having to log again.

OAuth2 registering

A simpler way of connecting, allowing the user to use an already existing account from another service and allowing the web application to already have access to a myriad of data that the user will not have to give or complete by himself.

The user has the choice to login with the Google authentication service, allowing him to sync his account with the Dashboard. His does not have to give a password but simply authorize the dashboard to use his Google data. This method is preferred since the user does not need to input anything more than consenting to share his google personal data.

Navigation system

The navigation system which rules the user's location within the application is the masterpiece of the site.

Routing

The application is composed of a router whose behavior will be explained further in the classes section. The router simply allows, through a button interface situated in the top left corner to travel through different pages of the application

The dashboard has three main routes, Login route, which is the unauthenticated route, the Dashboard route which contains the widget panel and the About route which displays some client and server's attributes

Some more routes are to be added following the roadmap which are the admin panel, allowing to edit any user's data without having to directly edit within the database, so a non-developer individual might easily administrate the website. Another possible route could be a link route so the applications linking could be properly separated from the dashboard. Another route might be a profile route, allowing an user to edit his preferences and some sensitive data.

Authentication checking

The Login route has a specific functionality, preventing an unauthenticated user to access the dashboard's authenticated routes. It has a wrapper, checking its login credentials and whether or not the user has a JWT refresh token. This system allows us to not have to redirect in each route and manually check each time if the user is logged in or not.

Widget layout

Adding a widget

The widget layout contains a grid of cards, which can be added, moved, or deleted. They are saved within the application's database each time they are modified. When the user wants to add a widget, he can by accessing the widget adding panel through the 'Add widget' button, located on the bottom left corner of the page. This will open a modal, offering a range of linking buttons

Moving a widget

The user might change his visual layout by accessing the 'edit layout' button right next to the previously mentioned button, this will trigger an edit mode which allows the user to move each card through a drag and drop system

Removing a widget

Through the 'edit widgets' system, the user can also delete a widget by clicking the red cross on the top right corner of a widget. This will open a confirmation modal asking if the user really wants to remove this widget.

About.json

The about.json route displays two kind of data from the API's server.

- Client data, including solely his IP

- Server data, it represents all the possible services that the user can link with. Each service includes some widgets, characterized by their name, description, options and parameters.

Summary of the chosen functionalities and packages

To design the application some technological choices were made for the interface, the database and the backend server.

Language and framework

Language

Many web languages were available during this project, we chose JavaScript which happen to offer a huge range of scripting capabilities as well as a lot of web integrations, JS has a lot of libraries and frameworks to abstract the code and quickly build powerful applications. Since it was explicitly asked that we do not uselessly recreate code. This language seemed to be the best choice.

Framework

Frontend

JavaScript offers a huge lot of frameworks, we chose ReactJS which can render on both client and server, and which can be powerful when coupled to some interface frameworks and some top-level middleware. The interface was done with Material UI v4, offering many ready components to easily construct an application's user interface. Redux was chosen as our main middleware for an easier abstraction of the code between the front and the backend.

Backend

NodeJS as a lower framework is useful enough for a web server using JavaScript, we couple it with expressJS which allows to easily create a powerful routing system using the CRUD method.

To handle data, we used Mongoose to comply with our database choice and it was all we needed for an efficient server.

Database

For an easily controllable database, we used MongoDB since it's light, recent, and quite easily scalable. Mongo offers an incredibly good administration interface for developers with Mongo express, for higher tier servers, developers might use Mongo compass which is an even more powerful software for data visualization and manipulation. Finally mongo is extremely easy to manipulate through Mongoose in the backend server as it is easy to create database models and edit them with high abstracted code.

Classes of the program

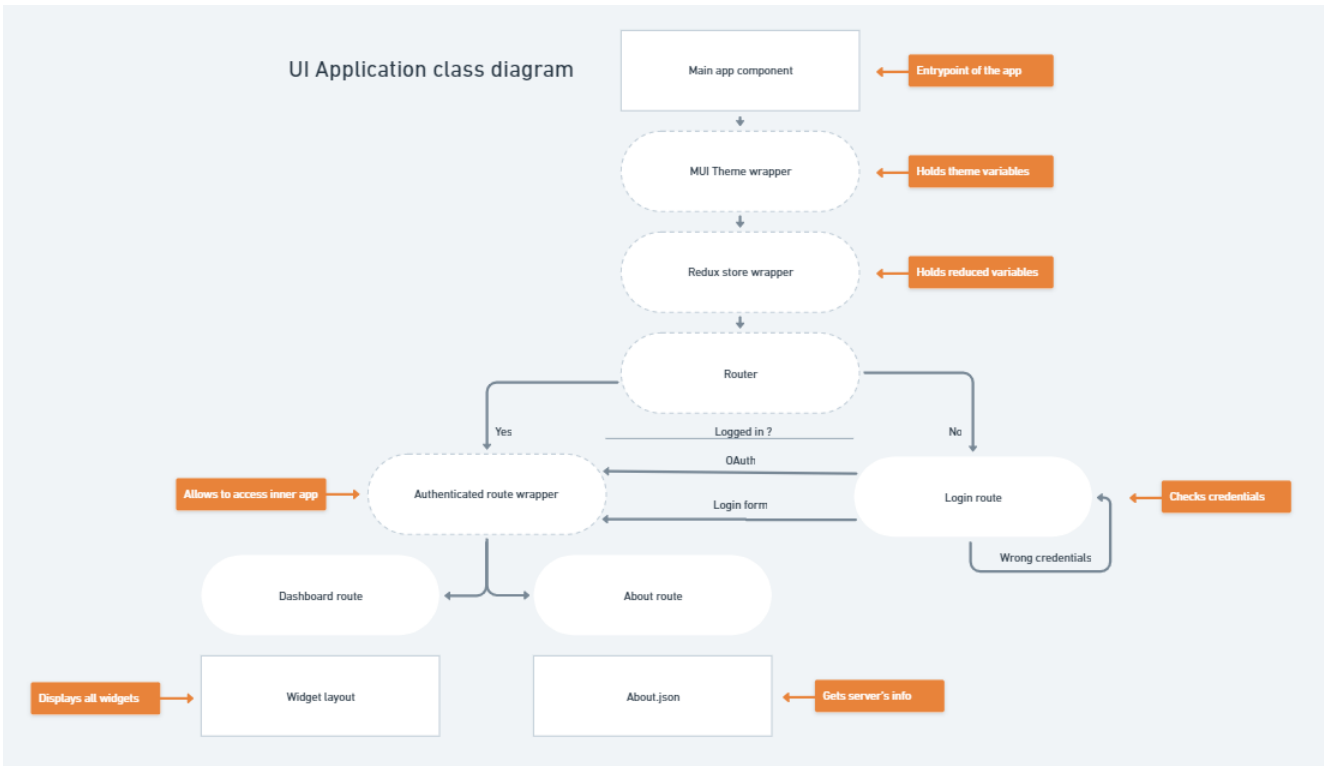
Hierarchy of the classes

The hierarchy is quite simple and is as follows:

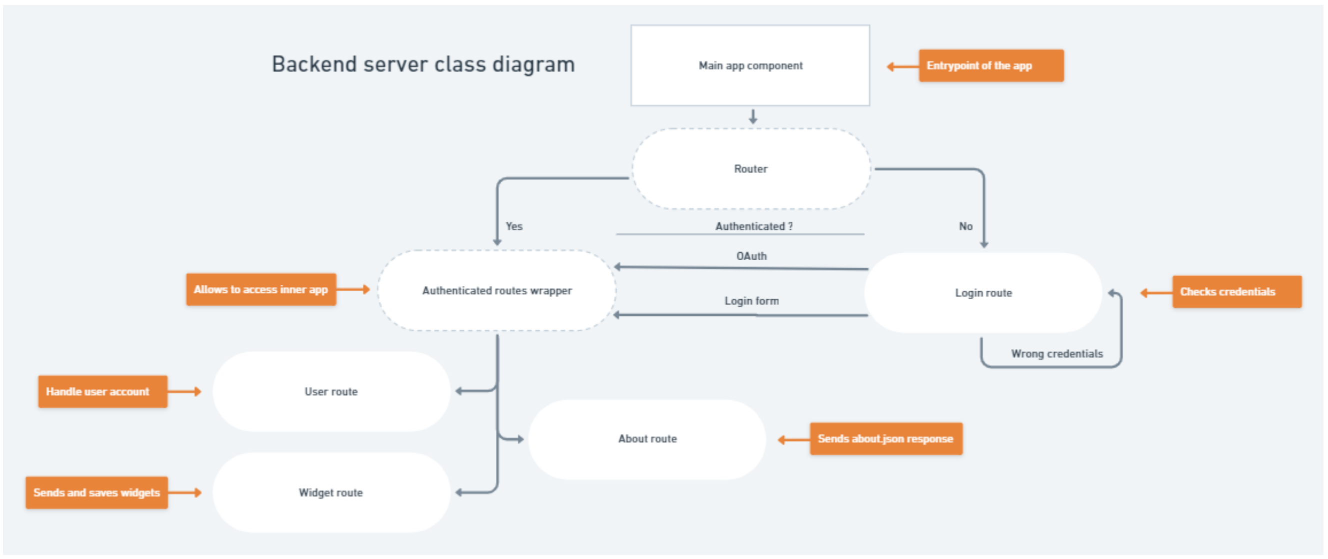
The app is ruled by the React-router router system, which is wrapped by the Material UI theme component, it doesn't rule other components but solely passes the application theme so all subcomponents from the Material UI library. The router holds each route and its associated component. The router wraps the Redux main component that allows reduced data to be retrieved from any route component of the application. By then it is easy. Each route has a main component called index holding its subcomponents that are rules by the state variables of both Redux and the index component.

Class diagram

UI App class diagram



Backend server class diagram


Backend server class diagram
Main app component
Entrypoint of the app
Router
Yes
Authenticated ?
No
OAuth
Allows to access inner app
Authenticated routes wrapper
Login form
Login route
Checks credentials
Wrong credentials
Handle user account
User route
About route
Sends about.json response
Sends and saves widgets
Widget route