Data Hub - Architectural Description

Constraints

- Server should run on Node LTS (12.x)
- Server should provide a RESTful API
- Client should support Chrome and Firefox
- No JS framework can be used, only JQuery is permitted

Server Architecture

The server is divided into two modules: Routes and Repository.

The Routes module manages the routing of the RESTful requests coming from clients. Thanks to the small scope of the project, it also handles the Business Logic for the server.

The Repository module handles persistency. It is based on a NoSQL DBMS, given the dynamic structure of the data to be stored. To keep the server as simple as possible, the persistence is implemented through LokiJS, a lightweight document-oriented database.

The Repository module is independent from the Routes one and can be switched without the latter noticing it. This way, it is easier to change or upgrade how persistency is implemented.

Client Architecture

Since most of the business logic is handled on the server, the client can be considered a thin client.

The client too is divided into two modules: UI and API.

The API module can be used to perform RESTful requests to the server. It is kept separate from the rest so it can be easily updated to reflect changes of the server's RESTful API.

The UI module is used to dynamically build views. It follows a reactive pattern and leverages JQuery to easily interact with the HTML document.

These two modules allow for the creation of a complex single page application.

Authentication

Authentication is obtained through a token system: in order to use the API, clients need to attach a valid token as a cookie in each of their HTTP request to the server.

To obtain a new valid token, clients must provide either an already valid token or valid user credentials. The server then generates a new token based on user credentials and current date, stores it in the database and sends it to the client.

Future improvements:

- Caching on Client:
  - Cache small data that appears frequently (user/project names)
- Caching on Server:
  - Cache project status
- Switch to more performant DBMS, like MongoDB.
- Switch to Reactive UI framework like React or Flutter since the client architecture is loosely based on them.

- Client and Server functionalities can easily be expanded by adding respectively views and routes.