Backend Architecture

As this is a modular application consisting of several module 'plugins', but Node.js uses its own 'modules' as well, for the purposes of this text I will refer to each as follows:

Module – Node.js module

Plugin – one of the application modules

The backend of the Simple++ system was developed in Node.js using the Express web application framework and ejs – a JavaScript template engine. There are several Node modules created to provide easy access to commonly used functions within plugins – retrieving user data, updating user data, connecting to the database etc. This reduces code duplication significantly and means a potential third-party developer does not need to know how the underlying code functions, they simply need to use Node.js native 'require' calls to be able to use the provided functions.
The provided functions (as of now) are:

- userDetails (provides access to user details)
- dbConnect (provides access to reading and updating the database)

The application uses primarily local storage in the form of various JSON files. There is also a connection established to a MongoDB database, if need be – however, this is subject to change, as it may create unnecessary security risks and problems with GDPR compliance.

Every plugin is self-contained within its own folder, with its own routing and ejs template files – if need be, these could even be hosted on an external server. In the main configuration file for the application, **server.js**, each plugin's Node code is imported using Node 'require' – meaning any changes in the plugin code will be automatically updated with no need for manual changes. When the user clicks/taps on a given plugin icon, the server queries the **server.js** file, which at that point imports the code from all the modules. Every plugin is tasked with handling its own routing requests. If clicking on an icon sends a GET request for e.g. '/firstaid', then the **firstaid** module Node file needs to have code that responds to this.

On user input, every page is rendered on demand by ejs using the an appropriate view file. When clicking/tapping any icon, it sends a GET request to the backend routing system, which in turn renders one of the set-up views on demand, allowing to pass data to dynamically insert into the template. An ejs template may have something like the following:

> *"Welcome <%= username %>"*

Where <%= username %> will be replaced by the value of variable ‘username’, so long as it is passed to the template at the point of rendering.

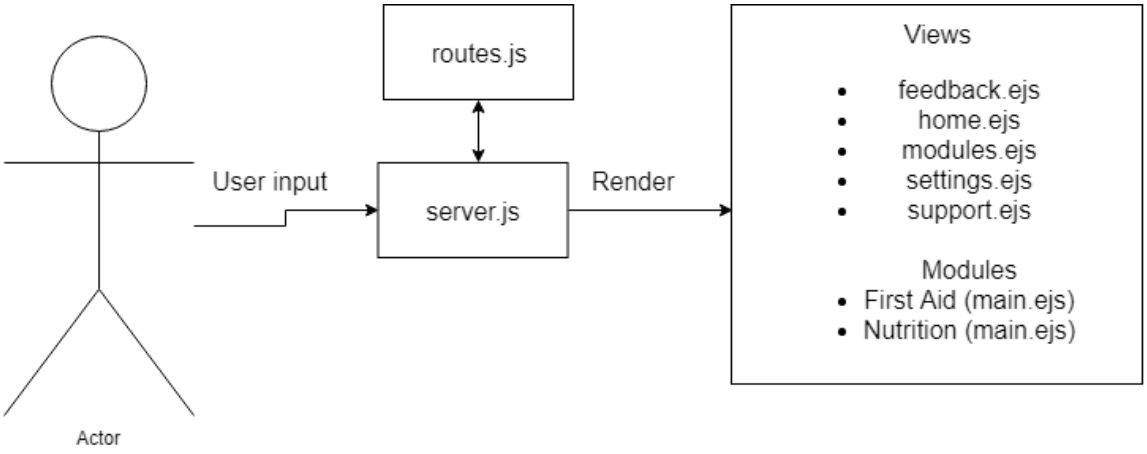
Partial views are provided to render the top bar and bottom bar; the only thing required to write for the individual plugins is the actual content of a page, to be inserted between the top and bottom partial views, for example:

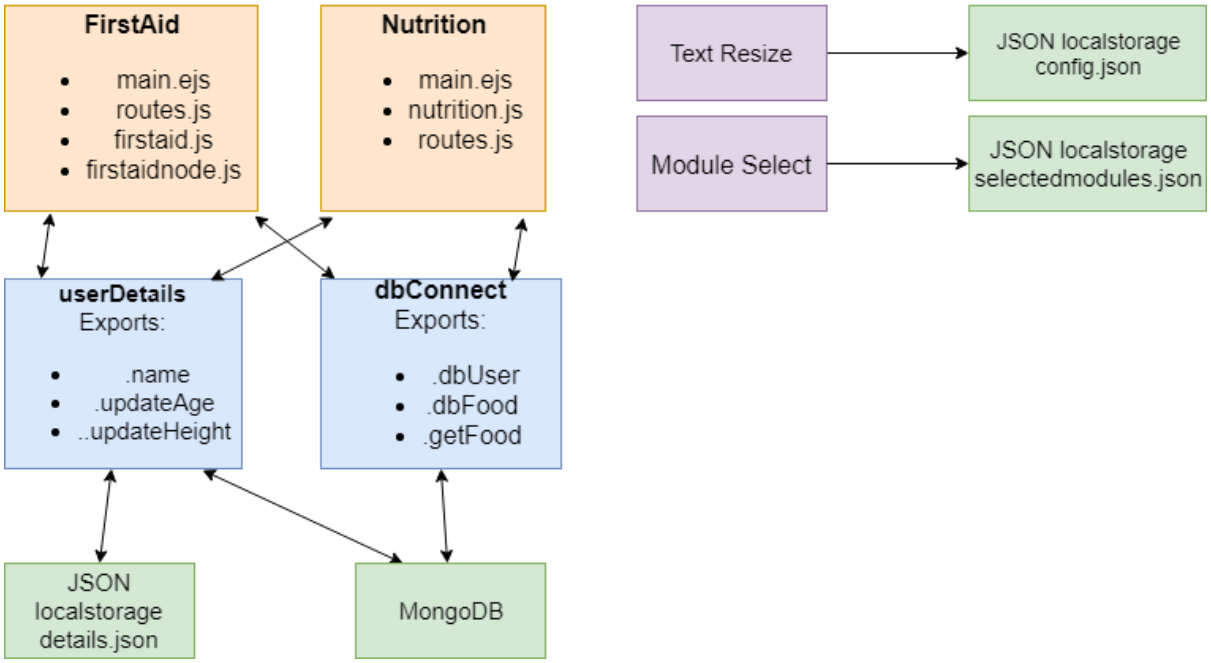
```
<%- include partials/moduletop %>

---- page content ---

<%- include partials/modulebottom %>
```

Once rendered, moduletop will be replaced with the HTML header and top navigation bar, while modulebottom will be replaced by the bottom navigation bar and HTML closing tags. The plugin can insert its own content between those two tags as a self-contained object.

The text resize function works by dynamically adjusting various CSS elements (such as font size, label size, box size) by a given value set by the user. This is primarily a vanilla Javascript/jQuery function.

The module selection feature works by saving the selected modules (read as checkboxes) into a JSON object in the form of an array of ‘modulename: state’ values, where the state is either true or false. This JSON object is then written to a file which is subsequently read by a JavaScript function which adjusts the frontend layout accordingly, hiding modules which have not been picked.


routes.js
Views
feedback.ejs
home.ejs
modules.ejs
settings.ejs
support.ejs
Modules
First Aid (main.ejs)
Nutrition (main.ejs)
User input
server.js
Render
Actor


FirstAid
main.ejs
routes.js
firstaid.js
firstaidnode.js
Nutrition
main.ejs
nutrition.js
routes.js
Text Resize
JSON localstorage config.json
Module Select
JSON localstorage selectedmodules.json
userDetails
Exports:
.name
.updateAge
..updateHeight
dbConnect
Exports:
.dbUser
.dbFood
.getFood
JSON localstorage details.json
MongoDB