Development Manual

Bionic Cloud User Interface

Spring 2018

Hallur Ólafsson
Haraldur Ingi Shoshan
Orri Axelsson

Local environment setup for Angular **3**
1. Code structure 3
1.1 ./src 3
1.2 ./app 4
1.3 ./components 4
1.4 ./interfaces 4
1.5 ./services 4
1.6 ./assets 5
1.7 ./environment 5
2. Imports 5
3. Coding rules 5
4. Continuous Delivery 6

Local environment setup for Angular

Prerequisites:

- Node.js and npm
- When Node and npm are installed, Angular must be installed
    - The following command needs to be entered in a terminal window
        - npm install -g @angular/cli@5.2.3
    - Clone the repo from Github
- When the steps above are completed, navigate into the Ossur-Bionic-Cloud-User-Interface folder and run npm install. This installs all the required dependencies for the project to run.
- When npm install is finished and this might take awhile.
    - Run ng serve -o to open a browser on localhost on port 4200
    - Run ng test to run Karma tests
- When the system is up and running the user needs to get a browser extension called CORS Toggle and turn it on.

1. Code structure

1.1 ./src

The source folder contains all the files and code that the system needs to function. It includes three folders, app, environments and assets. It contains global files like Index.html and styles.css. The index.html file is the root html file for the system where all other components are rendered into, it contains the body, head and the version of html we use and all CDN links. The style.css contains css code that all other components have access to globally.

1.2 ./app

The app folder contains three folders: components, interfaces and services. then we have the app component file, this is the root component. Then we have app.module.ts, this file links other modules to the project and provides services that are needed globally.

1.3 ./components

This folder contains all the components in the system that we create. Each component has its own separate folder. Each component consists of 4 files

- *.components.ts contains all the logic
- *.component.spec.ts contains all tests
- *.component.html contains all the HTML
- *.component.css contains all css

To create a new component the user has to navigate in the terminal to the component folder and type in:
ng g component <name of the component>
the angular cli automatically creates all the files

1.4 ./interfaces

This folder contains all the models of the system, when we acquire data from any of the services we create a model that maps the data gathered from the web service. When creating a new interface we can either do it in the editor and create the file or do it in the terminal. The name of the file is the same as the name of the interface we export from the file.

1.5 ./services

This folder contains all the services for the system, each service has its own separate folder. Each service consist of 2 files

- *.service.ts contains all service code
- *.service.spec.ts contains all service tests

To create a new service the user has to navigate to the services folder and type in:

ng g service <name of the service>

1.6 ./assets

This folder contains assets the system needs, such as images.

1.7 ./environment

This folder contains environment specifications for the system.

2. Imports

When importing components, services, interfaces or other modules into a file there is a certain rule to follow. All angular related modules go on top, other imports like service, interface or components have a comment above them describing what they are.

```
import { Component, OnInit } from '@angular/core';
import { Router } from "@angular/router";
import { NgModel } from '@angular/forms';

// Interfaces
import { User } from '../../services/auth/user';
import { variable } from '../../interfaces/variable';
import { event } from '../../interfaces/event';

// Services
import { AuthenticationService } from '../../services/auth/auth.service';
import { EventService } from '../../services/eventService/event.service';
```

3. Coding rules

1. All variable and function names should be camelCase, start with a small letter and then the next part of the name should have big letter.

```
var demoVariable: string;
```

2. Curly brackets
    a. The opening bracket should always open in the same line as the function, if-else and other statements are declared.
    b. The closing bracket should always close in a new line after the last line of code.

```
if(someBoolean) {
  console.log(demoVariable);
};
```

3. Always use double quotes (“”) when using quotes, not single (‘’)

```
var demoVariable: string = "double quotes";
```

4. Always use descriptive names for functions and variables

```
var todayLegStepCount: number = 23;
```

5. All functions, variables and objects should end with a semicolon (;)

```
var batteryLevel = 100;
```

4. Continuous Delivery

The system is set up with continuous delivery. For that setup, Github, Travis and Heroku are used. Whenever a pull request is made on Github, Travis takes over, builds the code and runs the tests. If all the tests are accepted then it’s possible to merge the pull request to the master branch on Github. When the code is merged with the master branch, Heroku takes over and pulls the code from Github, builds it and deploys online for the developers to see and test the system.