**Assignment 3: Client Side Development**

**This is what we did in Assignment 1, the basics of combining HTML and CSS.**

In the previous assignment you created a static set of pages that served as a prototype of the Website. The static pages allowed us to explore content related issues such as page layout, styling, navigation, and other presentation related issues.

**For Assignment 3, we’re implementing Angular to combine the pages into a single app, which is responsive and makes it so that we don’t have to open multiple HTML files piece-by-piece. It will also initialize the client services.**

In this assignment you will extend the prototype by making the pages dynamic. Angular will be used to implement data services, components, and single page navigation.

Angular uses the typescript programming language. <https://www.typescriptlang.org/docs/handbook/basic-types.html>

## Part 1: Initialize Angular Project and their Templates

In the previous assignment several static HTML pages were created. In this assignment pages will be put into an angular project.

**Put Assignment 1 Pages into an angular project:**

1. Enter your assignment folder
2. Command Prompt:
   1. Option 1: Hold shift and right click the folder, then choose open command prompt (or powershell) here.
   2. Option 2: You can **git bash** here.
3. **$ ng new my-app**

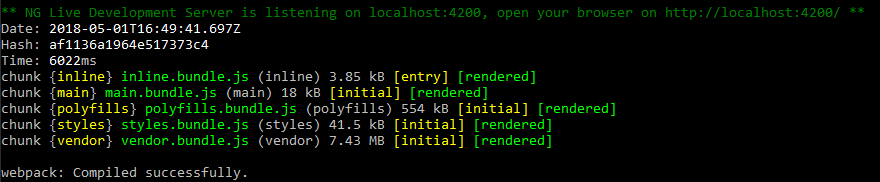
Wait for the initializing to be finished. You can see the following message in your command prompt:



Then we need to **run (compile) the project** by “npm start”.

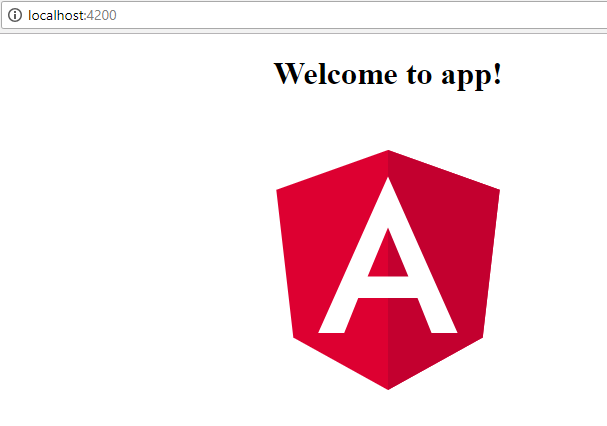
1. **$ cd my-app** navigates you to the created angular project
2. **$ ng serve**  compiles and starts the project

Wait for the compiling to be finished. You will see the following message in your command prompt:



Then you should be able to **find your angular project hosting** at localhost:4200

1. Open your browser
2. Type localhost:4200 in your browser to check it



If your page looks like this, your angular project is initialized.

\* more information can be found on <https://angular.io/guide/quickstart>

## Part 2: Create Angular Components and their Templates

In the previous assignment several static HTML pages were created. In this assignment pages will be refactored into angular components.

**Components** are a collection of assets that include typescript classes, html templates, and css stylesheets. We will create many different components to implement the various features of our Web application. We will implement each feature in its own dedicated directory. Follow the steps below to create the features and their components:

1. In **src/app** directory, create a components folder

**$ cd src/app**

**$ mkdir components**

1. In **src/app/components** directory, create folders for each of the main features of the application, e.g., create the following folders: **user**, **website**, **page**, and **widget**

**$ cd components**

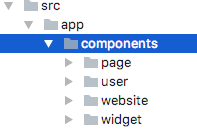
**$ mkdir page**

**$ mkdir widget**

**$ mkdir website**

**$ mkdir user**

**Note:** You can either use command above or create the folders by yourself in windows file explorer.

****

1. **Create Angular Components** for each of the HTML pages in their corresponding feature folder using the Angular CLI command, create Angular components:
2. In the user folder, create 3 Angular components for **login**, **profile** and **register** using below Angular CLI command, e.g.

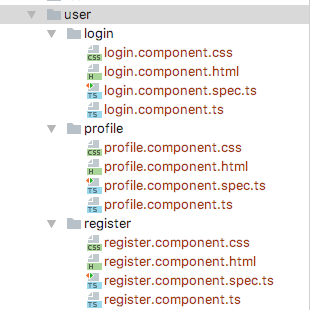
**$ cd user 🡨 Specifying Folder Name**

**$ ng g c login 🡨 Creating the component for the HTML page named within the folder**

**$ ng g c profile 🡨 Creating the component for the HTML page named within the folder**

**$ ng g c register 🡨 Creating the component for the HTML page named within the folder**

Your files and directories should look as follows

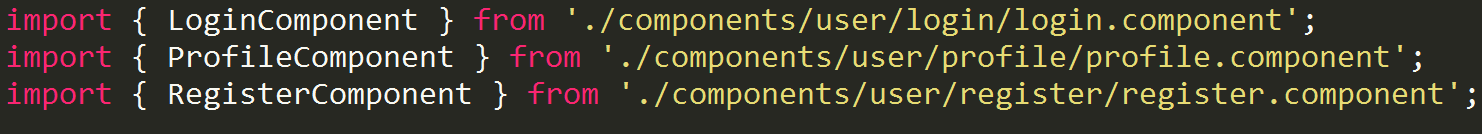
******

**Note:** When a component is created, Angular will automatically import the component in app.module.ts.

**IMPORTANT:** Make sure the (pink) import statement in **app.module.ts** (seen below) has the right component path (in yellow).

**Question:** What exactly is the purpose of the **app.module.ts** page?

The component name will be specified in the declarations section in **app.module.ts**.



1. Repeat for the website folder:

**$ cd website**

**$ ng g c website-new**

**$ ng g c website-edit**

**$ ng g c website-list**

1. Repeat for the page folder:

**$ cd page**

**$ ng g c page-new**

**$ ng g c page-edit**

**$ ng g c page-list**

1. Repeat for the widget folder:

**$ cd widget**

**$ ng g c widget-chooser**

**$ ng g c widget-edit**

**$ ng g c widget-list**

1. Repeat inside the widget-edit folder:

**$ cd widget-edit**

**$ ng g c widget-header**

**$ ng g c widget-image**

**$ ng g c widget-youtube**

1. **Refactor the old HTML files into angular fragments**, by removing the following tags: html, head, meta, title, link, script, body. **Keep only the main content of the pages** (the content held between the <body>**content**</body> tags).
2. **Copy code from old HTML files to corresponding new HTML files** present in each component folders

For example, copy code from login.html to login.component.html.

**Note:** If you’re starting from scratch with no content in the HTML files, like you will for the Final Project, this is where you will enter your HTML content, rather than having to copy and paste it like we did from Assignment 1 (HTML/CSS) content to Assignment 3 (Angular) content.

1. **Delete all old HTML files**.

**Note:** I have created empty HTML documents for my project before we started learning Angular. Should I write down the file and HTML doc names and delete them? This might be the route.

**Keep in Mind:** These new files are not meant to be viewed independently. They are instead meant to be dynamically included as part of a *single page* that already provides these tags. The **index.html** file will serve as the single page container of the Website.

## Part 3: Configure Angular Routing

In the previous assignment we created several static web pages linked together using hyperlinks. In this assignment we will refactor navigation using Angular's routing module and implement a ***single page application* (SPA)**.

1. Create an **app.routing.ts** file inside **app** folder.

This configures the angular application routing and implements the navigation described in the page flow diagram.

Use the routing provider to map the ***routerLink*** to the components. For instance, the following snippet of code binds the **/login** route to its component **LoginComponent** in the **app.routing.ts** file.

**Note:** This takes the components we just created, states their names in the **app.routing.ts** file, and essentially makes them all acknowledge and communicate with each other the way that they did when we first created them in the HTML files in Assignment 1.

**Question:** How do we know what/how to name the **Route** and **Component** as listed below?

|  |
| --- |
| **Note: This is the app.routing.ts file.**  import {Routes, RouterModule} from "@angular/router";  import {ModuleWithProviders} from "@angular/core";  import {LoginComponent} from "./components/user/login/login.component";  import {RegisterComponent} from "./components/user/register/register.component";  import { PageEditComponent } from './components/page/page-edit/page-edit.component';  import { PageListComponent } from './components/page/page-list/page-list.component';  import { PageNewComponent } from './components/page/page-new/page-new.component';  import { WebsiteEditComponent } from './components/website/website-edit/website-edit.component';  import { WebsiteListComponent } from './components/website/website-list/website-list.component';  import { WebsiteNewComponent } from './components/website/website-new/website-new.component';  import { WidgetChooserComponent } from './components/widget/widget-chooser/widget-chooser.component';  import { WidgetListComponent } from './components/widget/widget-list/widget-list.component';  import { WidgetEditComponent } from './components/widget/widget-edit/widget-edit.component';  import { WidgetHeadingComponent } from './components/widget/widget-edit/widget-heading/widget-heading.component';  import { WidgetImageComponent } from './components/widget/widget-edit/widget-image/widget-image.component';  import { WidgetYoutubeComponent } from './components/widget/widget-edit/widget-youtube/widget-youtube.component';  const APP\_ROUTES : Routes = [  { path : '', component: LoginComponent},  { path : 'login' , component: LoginComponent},  { path : 'register' , component: RegisterComponent},  { path : 'user/:userId' , component: ProfileComponent},  { path : 'user/:userId/website' , component: WebsiteListComponent},  { path : 'user/:uid/website/new', component : WebsiteNewComponent},  { path : 'user/:uid/website/:wid' , component: WebsiteEditComponent},  { path : 'user/:uid/website/:wid/page' , component: PageListComponent},  { path : 'user/:uid/website/:wid/page/new' , component: PageNewComponent},  { path : 'user/:uid/website/:wid/page/:pid' , component: PageEditComponent},  { path : 'user/:uid/website/:wid/page/:pid/widget', component : WidgetListComponent},  { path : 'user/:uid/website/:wid/page/:pid/widget/new' , component: WidgetChooserComponent},  { path : 'user/:uid/website/:wid/page/:pid/widget/:wgid' , component: WidgetEditComponent},  ];  // Export the routes as module providers  export const Routing: ModuleWithProviders = RouterModule.forRoot(APP\_ROUTES); |

Use the example above to implement navigation as described in page flow diagram and the following routes

**Route Component**

1. **login, , LoginComponent**
2. **register RegisterComponent**
3. **user/:uid ProfileComponent**
4. **user/:uid/website WebsiteListComponent**
5. **user/:uid/website/new WebsiteNewComponent**
6. **user/:uid/website/:wid WebsiteEditComponent**
7. **user/:uid/website/:wid/page PageListComponent**
8. **user/:uid/website/:wid/page/new PageNewComponent**
9. **user/:uid/website/:wid/page/:pid PageEditComponent**
10. **user/:uid/website/:wid/page/:pid/widget WidgetListComponent**
11. **user/:uid/website/:wid/page/:pid/widget/new WidgetChooserComponent**
12. **user/:uid/website/:wid/page/:pid/widget/:wgid WidgetEditComponent**

Where **:uid**, **:wid**, **:pid**, and **:wgid** are path parameters encoding the IDs of particular users, websites, pages, and widgets. We will work on that in next assignment.

1. Import the **app.routing.ts** file into the imports array in the **app.module.ts** file. Notice the imports array in following sample **app.module.ts file.**

|  |
| --- |
| **Note: This is the app.module.ts file.**  import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import {FormsModule} from '@angular/forms';  import {HttpModule} from '@angular/http';  import { AppComponent } from './app.component';  import { LoginComponent} from "./components/user/login/login.component";  import { ProfileComponent } from './components/user/profile/profile.component';  import { RegisterComponent } from './components/user/register/register.component';  import { PageEditComponent } from './components/page/page-edit/page-edit.component';  import { PageListComponent } from './components/page/page-list/page-list.component';  import { PageNewComponent } from './components/page/page-new/page-new.component';  import { WebsiteEditComponent } from './components/website/website-edit/website-edit.component';  import { WebsiteListComponent } from './components/website/website-list/website-list.component';  import { WebsiteNewComponent } from './components/website/website-new/website-new.component';  import { WidgetChooserComponent } from './components/widget/widget-chooser/widget-chooser.component';  import { WidgetListComponent } from './components/widget/widget-list/widget-list.component';  import { WidgetEditComponent } from './components/widget/widget-edit/widget-edit.component';  import { WidgetHeadingComponent } from './components/widget/widget-edit/widget-heading/widget-heading.component';  import { WidgetImageComponent } from './components/widget/widget-edit/widget-image/widget-image.component';  import { WidgetYoutubeComponent } from './components/widget/widget-edit/widget-youtube/widget-youtube.component';  import { Routing } from './app.routing'; **🡨 App Routing File**  **Question:** What is NgModule? What does it do, exactly? What is a declaration?  **Question:** What is a declaration?  @NgModule({  declarations: [  AppComponent,  LoginComponent,  ProfileComponent,  RegisterComponent,  PageEditComponent,  PageListComponent,  PageNewComponent,  WebsiteEditComponent,  WebsiteListComponent,  WebsiteNewComponent,  WidgetChooserComponent,  WidgetHeadingComponent,  WidgetImageComponent,  WidgetListComponent,  WidgetYoutubeComponent,  WidgetEditComponent  ],  **Question:** What is the purpose of the imports array?  imports: [ 🡨**imports file array**  BrowserModule,  FormsModule,  HttpModule,  Routing **🡨 App Routing File**  ],  **Question:** What is the purpose of the providers array?  **Question:** What is the purpose of the bootstrap array?  providers: [TestService],  bootstrap: [AppComponent]  })  export class AppModule { }  **Question:** What is the purpose of export class AppModule? |

In **index.html**, include **<app-root>Loading...</app-root>** selector where all components will be dynamically included

1. Delete all the content in **app.component.html** and add **<router-outlet></router-outlet>** in it.
2. Refactor (replace) all hyperlinks in the html files of the components to use ***routerLink***directive instead of referring to actual files. For instance, change the hyperlink from the login page to the register page, **<a href="register.html">**, to use hash fragment: **<a routerLink="/register">**. Use the routes listed in the **app.routing.ts**.

## Part 4: Create Client side Angular Services

Create angular services to provide a central place to access/update data. Create a separate service for each type of entity: **user**, **website**, **page**, and **widget**. Each service must provide **CRUD operations** to manipulate the corresponding entity: **create**, **read**, **update**, and **delete**. Create the following services with the listed CRUD operations. Implement all services in **src/app/services** folder.

* 1. **src/app/services/user.service.client.ts**
  2. **src/app/services/website.service.client.ts**
  3. **src/app/services/page.service.client.ts**
  4. **src/app/services/widget.service.client.ts**

### UserService

Implement **UserService** in a file called **user.service.client.ts**. Declare the service in a Typescript class of the same name. In the service declare a local array called **users** that will be used to simulate data from a database. The local **users** array is only temporary and will be removed in the next assignment where data will be fetched from the server. Use the following data to initialize the **users** array

**[**

**{\_id: "123", username: "alice", password: "alice", firstName: "Alice", lastName: "Wonder", email: "**[**alice@gmail.com**](mailto:alice@gmail.com)**"},**

**{\_id: "234", username: "bob", password: "bob", firstName: "Bob", lastName: "Marley", email: "**[**bob@whatever.com**](mailto:bob@whatever.com)**"},**

**{\_id: "345", username: "charly", password: "charly", firstName: "Charly", lastName: "Garcia", email: "**[**charly@ulem.com**](mailto:charly@ulem.com)**"},**

**{\_id: "456", username: "shiyu", password: "shiyu", firstName: "Shiyu", lastName: "Wang", email: "swang@ulem.org"}**

**]**

Implement the following API (functions) in the **UserService** service:

**CRUD**

**Create:**

1. **createUser(user)** - adds the **user** parameter instance to the local **users** array

**Read:**

1. **findUserById(userId)** - returns the user in local **users** array whose **\_id** matches the **userId** parameter
2. **findUserByUsername(username)** - returns the user in local **users** array whose **username** matches the parameter **username**
3. **findUserByCredentials(username, password)** - returns the user whose **username** and **password** match the **username** and **password** parameters

**Update:**

1. **updateUser(userId, user)** - updates the user in local **users** array whose **\_id** matches the **userId** parameter

**Delete:**

1. **deleteUser(userId)** - removes the user whose **\_id** matches the **userId** parameter

Here's an example snippet of code that declares the **UserService** in **user.service.client.ts** and a couple of API functions. Follow the same pattern for all other services.

|  |
| --- |
| **Note: This is the user.service.client.ts file.**  **Question:** What exactly does this import statement do?  import { Injectable } from '@angular/core';  **Note:** The @Injectable() injects the service into the module.  **Question:** How exactly? Is it a function?  // injecting service into module  @Injectable()  export class UserService {  constructor() { } **🡨Question:** What exactly does this do?  users = [**🡨** This is the array that stores the information that users enter.  {\_id: "123", username: "alice", password: "alice", firstName: "Alice", lastName: "Wonder", email: "alice@gmail.com"},  {\_id: "234", username: "bob", password: "bob", firstName: "Bob", lastName: "Marley", email: "bob@whatever.com"},  {\_id: "345", username: "charly", password: "charly", firstName: "Charly", lastName: "Garcia", email: "charly@hotmail.com"},  {\_id: "456", username: "shiyu", password: "shiyu", firstName: "Shiyu", lastName: "Wang", email: "swang@ulem.org"}  ];  **Question:** Can this array be blank to start? If so, would I have to include the object names like \_id:, username, password, etc.?  createUser(user: any) {  user.\_id = Math.random();  this.users.push(user);  return user;  }  findUserById(userId: string) {  for (let x = 0; x < this.users.length; x++) {  if (this.users[x].\_id === userId) { return this.users[x]; }  }  }  findUserByUsername(username: string) { … }  findUserByCredentials(username: string, password: string) { … }  updateUser(userId, user) { … }  deleteUser(userId) { … }  } |

### WebsiteService

Implement **WebsiteService** in a file called **website.service.client.ts**. Declare a service called **Website Service** implemented in a class of the same name. In the service, declare a local array called **websites** that will be used to simulate data from a database. The local **websites** array is only temporary and will be removed in the next assignment where data will be fetched from the server. Use the following data to initialize the **websites** array:

**[🡨** This is the array that stores the information that users enter about the websites they create.

**{ \_id: "123", name: "Facebook", developerId: "456", description: "Lorem" },**

**{ \_id: "234", name: "Tweeter", developerId: "456", description: "Lorem" },**

**{ \_id: "456", name: "Gizmodo", developerId: "456", description: "Lorem" },**

**{ \_id: "890", name: "Go", developerId: "123", description: "Lorem" },**

**{ \_id: "567", name: "Tic Tac Toe", developerId: "123", description: "Lorem" },**

**{ \_id: "678", name: "Checkers", developerId: "123", description: "Lorem" },**

**{ \_id: "789", name: "Chess", developerId: "234", description: "Lorem" }**

**]**

Implement the following API in the **WebsiteService** service

**CRUD**

**Create**

1. **createWebsite(userId, website)** - adds the **website** parameter instance to the local **websites** array. The new website's **developerId** is set to the **userId** parameter

**Read**

1. **findWebsitesByUser(userId)** - retrieves the websites in local **websites** array whose **developerId** matches the parameter **userId**
2. **findWebsiteById(websiteId)** - retrieves the website in local **websites** array whose **\_id** matches the **websiteId** parameter

**Update**

1. **updateWebsite(websiteId, website)** - updates the website in local **websites** array whose **\_id** matches the **websiteId** parameter

**Delete**

1. **deleteWebsite(websiteId)** - removes the website from local **websites** array whose **\_id** matches the **websiteId** parameter

### PageService

Implement **PageService** in a file called **page.service.client.ts**. Declare the service in a class of the same name. In the service declare a local array called **pages** that will be used to simulate data from a database. The local **pages** array is only temporary and will be removed in the next assignment where data will be fetched from the server. Use the following data to initialize the **pages** array:

**[🡨** This is the array that stores the information that users enter about the pages they create.

**{ \_id: "321", name: "Post 1", websiteId: "456", description: "Lorem" },**

**{ \_id: "432", name: "Post 2", websiteId: "456", description: "Lorem" },**

**{ \_id: "543", name: "Post 3", websiteId: "456", description: "Lorem" }**

**]**

Implement the following API in the **PageService** service

**CRUD**

**Create**

1. **createPage(websiteId, page)** - adds the **page** parameter instance to the local **pages** array. The new page's **websiteId** is set to the **websiteId** parameter

**Read**

1. **findPageByWebsiteId(websiteId)** - retrieves the pages in local **pages** array whose **websiteId** matches the parameter **websiteId**
2. **findPageById(pageId)** - retrieves the page in local **pages** array whose **\_id** matches the **pageId** parameter

**Update**

1. **updatePage(pageId, page)** - updates the page in local **pages** array whose **\_id** matches the **pageId** parameter

**Delete**

1. **deletePage(pageId)** - removes the page from local **pages** array whose **\_id** matches the **pageId** parameter

### WidgetService

Implement **WidgetService** in a file called **widget.service.client.ts**. Declare the service in a class of the same name. In the service declare a local array called **widgets** that will be used to simulate data from a database. The local **widgets** array is only temporary and will be removed in the next assignment where data will be fetched from the server. Use the following data to initialize the **widgets** array:

**[🡨** This is the array that stores the information that users enter about the widgets they create.

**{ \_id: "123", widgetType: "HEADING", pageId: "321", size: 2, text: "GIZMODO"},**

**{ \_id: "234", widgetType: "HEADING", pageId: "321", size: 4, text: "Lorem ipsum"},**

**{ \_id: "345", widgetType: "IMAGE", pageId: "321", width: "100%", url: "http://lorempixel.com/400/200/"},**

**{ \_id: "456", widgetType: "HTML", pageId: "321", text: "<p>Lorem ipsum</p>"},**

**{ \_id: "567", widgetType: "HEADING", pageId: "321", size: 4, text: "Lorem ipsum"},**

**{ \_id: "678", widgetType: "YOUTUBE", pageId: "321", width: "100%", url: "https://youtu.be/AM2Ivdi9c4E" },**

**{ \_id: "789", widgetType: "HTML", pageId: "321", text: "<p>Lorem ipsum</p>"}**

**]**

Implement the following API in the **WidgetService** service

**CRUD**

**Create**

1. **createWidget(pageId, widget)** - adds the **widget** parameter instance to the local **widgets** array. The new widget's **pageId** is set to the **pageId** parameter

**Read**

1. **findWidgetsByPageId(pageId)** - retrieves the widgets in local **widgets** array whose **pageId** matches the parameter **pageId**
2. **findWidgetById(widgetId)** - retrieves the widget in local **widgets** array whose **\_id** matches the **widgetId** parameter

**Update**

1. **updateWidget(widgetId, widget)** - updates the widget in local **widgets** array whose **\_id** matches the **widgetId** parameter

**Delete**

1. **deleteWidget(widgetId)** - removes the widget from local **widgets** array whose **\_id** matches the **widgetId** parameter

## Part 5: Import the client services to App Module

Remember to import all new service files from the **app.module.ts** page in **providers** array as shown in below sample **app.module.ts** file.

|  |
| --- |
| import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms';  import { HttpModule } from '@angular/http';  import { AppComponent } from './app.component';  import { LoginComponent } from './components/user/login/login.component';  import { Routing } from './app.routing';  import { UserService } from './services/user.service.client';  import { WebsiteService } from './services/website.service.client';  import { PageService } from './services/page.service.client';  import { WidgetService } from './services/widget.service.client';  @NgModule({  declarations: [  AppComponent,  LoginComponent,  ],  imports: [  BrowserModule,  FormsModule,  HttpModule,  Routing  ],  providers: [UserService, WebsiteService, PageService, WidgetService],  bootstrap: [AppComponent]  })  export class AppModule { } |

## Verify Files and Directory Structure

With all these changes from the previous assignment, below is a list of all the files worked on this assignment. Verify the name and location of the files. All files should be under the **assignment** directory

* **src**
  + **app**
    - **components**
      * **page**
        + **page-edit**

**page-edit.component.css**

**page-edit.component.html**

**page-edit.component.spec.ts**

**page-edit.component.ts**

* + - * + **page-list**

***similar files as above folder***

* + - * + **page-new**
      * **user**
        + **login**

**login.component.css**

**login.component.html**

**login.component.spec.ts**

**login.component.ts**

* + - * + **profile**

***similar files as above folder***

* + - * + **register**

***similar files as above folder***

* + - * **website**
        + **website-edit**

**website-edit.component.css**

**website-edit.component.html**

**website-edit.component.spec.ts**

**website-edit.component.ts**

* + - * + **website-new**

***similar files as above folder***

* + - * + **website-list**

***similar files as above folder***

* + - * **widget**
        + **Widget-edit**

**widget-edit.component.css**

**widget-edit.component.html**

**widget-edit.component.spec.ts**

**widget-edit.component.ts**

**widget-header**

**widget-header.component.css**

**widget-header.component.html**

**widget-header.component.spec.ts**

**widget-header.component.ts**

**widget-image**

***similar files as above folder***

**widget-youtube**

***similar files as above folder***

* + - * + **widget-chooser**

**widget-chooser.component.css**

**widget-chooser.component.html**

**widget-chooser.component.spec.ts**

**widget-chooser.component.ts**

* + - * + **widget-list**

***similar files as above folder***

* + - **services**
      * **page.service.client.ts**
      * **user.service.client.ts**
      * **website.service.client.ts**
      * **widget.service.client.ts**
    - **app.module.ts**
    - **app.routing.ts**

Implement navigation as shown in the page flow diagram and table below. Ignore links and buttons not listed here. Other links and buttons will be addressed in subsequent assignments.

### flow.png

# Deliverables

## GitHub and Hosted Environment Deliverables

To allow instructor to see your changes, please frequently commit and push your work to GitHub and OpenShift repositories. Below is an example of the commands you will use. The example assumes your project is located in **~/MSIMBO/webdev**:

**$ cd ~/MSIMBO/webdev**

**$ git add .**

**$ git commit -m 'A comment describing your work'**

**$ git push**

If using Heroku, you might need to deploy from within the Heroku dashboard. If you configured Heroku to auto deploy when the repo was updated, then you should be all set.

Verify that the files have copied to the github repository. Also visit your hosted environment website and verify that your changes are reflected on the remote server.

### Tagging a Release

We will be using code repository tags (or releases) to "submit" assignments. When you consider your work complete and ready for evaluation (ready for release), go to your code repository in GitHub and generate a release by navigating to "releases". Then click on "Create a new release" and type the name of the tag in the input field labeled "Tag version". We will be using the following tags for the various assignments:

assignment1 (previous assignment)

**assignment3 (this assignment)**

If you need to resubmit the assignment then create a new tag by adding a version number, e.g.,

assignment3.1, assignment3.2, etc...

I will grade the very last release.

# Potential Bugs

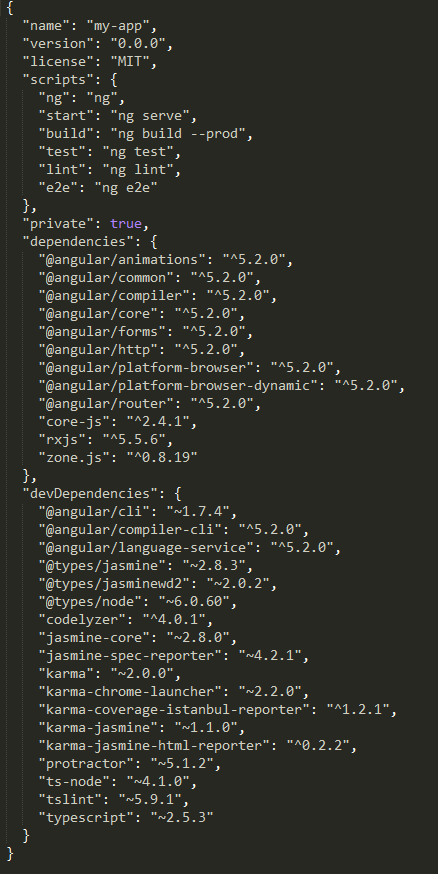
## Angular cli doesn’t generate components when you use “ng g c component-name”

This happens when your global angular version and the one in your project are not matching.

Home Terminal



Package.json



Note that both the versions of anglular/cli are 1.7.4

### Fix

If your versions are different, you have to open your command prompt at your project and do

npm install @angular/cli@**<terminal-version>** --save

## Create component (ng g c) will not import correct path of the created component in app module

### Fix

Just rename the import folder in app.module.ts file

Video: https://youtu.be/38xiRm5nS98

# Resources for Angular

Architecture: <https://angular.io/guide/architecture>

<https://angular.io/tutorial/toh-pt4>

<https://www.typescriptlang.org/docs/handbook/basic-types.html>