**Capstone - Front-end (v1.4)**

This is a guide book on how to build the front-end, capstone project. It will not be a step-by-step tutorial but a general guide of which Angular objects to create in what order.

1. Create the project.
   * Starting in the repos folder
   * Use the ng tool to create the Angular project. Be sure to include the switch to include **routing**. Optionally you can include the switch to exclude the test files (--skip-tests) from being created.
   * This will create a new folder in the repos folder and create all the files and folders in that new folder.
2. Start up the project so it begins running.
3. Using Visual Studio Code, open the project folder.
4. Edit the **app.module.ts** file and add the imports and the class names for the HttpClientModule and FormsModule. You will need them later.
5. Edit the **app.component.html** file and clear all the template html that came with the generated file. Add the pseudo-html tag so that all routed components are rendered in the **app.component.html**.
6. Create subfolders under the app folder to stored all the classes, services, pipes, and components for the user, vendor, product, request, and requestline.
7. Create a folder under app called core to hold miscellaneous components, services, and pipes.
8. Using the **cli**, generate *components* in the core folder for: home, about, e404.

*> ng g c core/home > ng g c core/about > ng g c core/e404*

1. Using the **cli**, generate the **system.service.ts** service in the core folder. (*This will be used to share the logged in user with all components*)

*> ng g s core/system*

**Menu**

1. Create a folder called menu under app
2. To create the application menu, you'll need to create a **menu.class.ts** that defines the data for each menu item. The menu class will have properties for: display and route. Both are strings. display is what shows on the page to the user and route is a route path defined in the **app-routing.module.ts**.

*> menu.class.ts (hand-create new file and name it)*

1. There will be a menu component and (optionally) a menuitem component which will work together to create and render the menu on the page. *You may render the menu without the menuitem component if you wish.*

*> ng g c menu/menu*

1. In the **menu.component.ts**, create an array of **menu.class.ts** data with every item in the array representing a single item in the menu. Make sure to include a menu item for the home and about components.
2. In the **menu.component.html**, add the required HTML to render the menu horizontally on the page using the <nav>, <ul>, <li>, and <a> tags. Make sure to space the menu items apart evenly. If you will use the menuitem component, some of these HTML elements will go in that component.
3. You will need to apply some CSS styling to the HTML in order to make the menu display correctly. The general style should be:
   * It is pinned to the top of the page and span the entire width of the page
   * The height should be just one normal text line
   * The background color should be in contrast to the rest of the page's background color
   * The menu items text color should be in contrast to the menu background color
   * The menu items should be evenly spaced apart

**Routing**

1. To the **app-routing.module.ts**, add the imports and routes for the home, about, and e404 components.
2. Fill the routes array with the special first and last route paths. Navigate to these components with these routes:
   * The path /home navigates to the home component
   * The path /about navigates to the about component
   * The path \*\* navigates to the e404 component
   * Note: Not sure where to direct special first route – implemented as home for now

**Home, About and E404 Components**

1. Edit the home, about, and e404 components and add the pseudo-html tag to render the menu at the top of the html of each component

**Creating the basic user, vendor, product, and request objects**

In this section, the instructions will use the user as an example. Each of these four objects would be created similarly.

1. Create or generate the **user.class.ts** file
   * Make sure to set default values for all non-nullable properties like numbers and booleans.
2. Generate the **user.service.ts**. The basic functions needed are:
   * list(): get all rows
   * get(id): get a single row by primary key
   * create(user): insert
   * change(user): update
   * remove(user): delete
3. Generate the user-list component
   * Activated when the user clicks the Users menu item
   * Displays all the user instances in a table as soon as the component displays
   * Add the component to the routes array in the **app-routing.module.ts**
4. Generate the user-detail component
   * Activated when the user clicks the detail link on a user instance in the **user-list.component.html**
   * The component must read the id from the route
   * Using the id, call a function in the service to read by primary key
   * The user is rendered using a table with two columns where the left columns are the labels and the right columns are <input> tags that display data for the user.
   * The component includes a button named Delete which will delete the user instance and return to the user-list.
   * Add the component to the routes array in the **app-routing.module.ts**
5. Generate the user-create component
   * Activated when the user clicks the Create link at the top of the **user-list.component.html**
   * Display an empty user form containing the user properties
   * The user data is filled in by the user
   * The Save button is clicked and the user instance is added. When successful, the component navigates back to the user-list component
   * Add the component to the routes array in the **app-routing.module.ts**
6. Generate the user-edit component
   * Activated when the user clicks the edit link on a user instance in the **user-list.component.html**
   * The component reads the id from the route
   * Using the id, a read by primary key returns the instance of the user
   * The user is rendered using a table with two columns where the left columns are the labels and the right columns are <input> tags that display data from the user.
   * The user data is modified by the user
   * The Save button is clicked and the user instance is changed. When successful, the user-list is displayed
   * Add the component to the routes array in the **app-routing.module.ts**

**Additional service and component functions required**

This section displays additional objects beyond the standard objects that need to be added to complete the capstone

**Sort**

1. Generate the **sort.pipe.ts**.
   * This pipe should be generated in the core folder and will providing sorting of the list pages by clicking on some of the column headings. It can be used for all list components.
   * The first time a column is clicked, that column data is sorted in ascending sequence.
   * If the same column is clicked again, the same column data is sorted but in descending sequence. Every time the same column is clicked, the order of the sort on the column is reversed
   * When a different column is clicked, the data is sorted by that column value is ascending sequence.

**User**

1. Add a login(username, password) function to the **user.service.ts**
   * Pass in a username and password
   * The function reads for a single user with the specified username and password.
2. Generate the user-login component
   * Displays a textbox for username and a textbox for password and include a Login button
   * The username and password is entered and the Login button is clicked
   * Call the Login service function passing the username and password
     + If found, returns the user instance which is stored in a property in the **system.service.ts** then navigates to the request-list component.
     + If not found, returns a 404 error and the the page remains on the **user-login.component**.
3. Generate the user-search pipe
   * This will be generated in the user folder.
   * This pipe will be added to the **user-list.component.html** to allow the user to do an incremental search on the list of users.

**Vendor**

No additional methods are needed

**Product**

No additional methods are needed

**Request**

1. Create a requests(id) function in the service to retrieve all requests in review status but not owned by the user whose primary key is id.
2. Create a review(req) function in the service to set the request req to review or approved based on the request total
3. Create a approve(req) function in the service to set the request req to approved
4. Create a reject(req) function in the service to set the request req to rejected. When rejected, a request must include text in the rejectionReason property
5. Create a request-lines component.
   * This component will selected from an action on the request-list component and it will display some properties from the selected request. At a minimum, the request properties id, description, status, total, and the username from the user that owns the request.
   * At the top of this component, there should be a Create link that navigates to a requestline-create component.
   * On the component, there should be a Review button. It can be placed in the area of the request properties or at the bottom of the page. When clicked, the component should call the review(req) function in the service.
   * In addition, all the requestlines or lineitems attached to the request are displayed in a list format with each line having actions: edit and delete.
     + Clicking edit will call the component that edits a requestline or lineitem
     + Clicking delete will allow the user to confirm the delete then remove the requestline or lineitem from the request
6. Create a request-review-list component.
   * This component will display a list of requests in REVIEW status
   * Each request will have an action called 'Review' which when clicked will navigate to the request-review-item component.
7. Create a request-review-item component
   * This component will display a request and its lines in the same display as the request-lines component except that no maintenance links are allowed. In essence, the changes are permitted to the request data or the lines on the request. There is no Review button on this component.
   * Two additional buttons are displayed at the bottom of the page: Approve and Reject
     + The Approve button will call the approve(res) function causing the status of the request to change to APPROVED
     + The Reject button will call the reject(res) function causing the status of the request to change to REJECTED. If a request is rejected, the reviewer is REQUIRED to enter some text in the rejectionReason property. A textbox for this data must be provided to the reviewer either displaying constantly on the page or can be revealed dynamically when the Reject button is clicked.

**Requestitem**

The Requestline does NOT need a list component that displays *every* Requestline on a page. Requestlines will always be display for the request they are attached to. The request-lines component fulfills the roll of displaying a list of requestline items

1. Create the requestline-create component.
   * This component will create a new requestline and attach it to the request from where it was called. The component will display only a select list of products and a textbox quantity to the user along with a Save button.
   * *Note: Because the new requestline is attached to the request automatically and is not selected by the user, the requestId must be passed to this component via the route. This requires that the route to this component include a variable for the****request id****. The requestId foreign key in the requestline must be set using the value passed in via the route. Without doing so, the requestId will be zero and the create will fail.*
   * When the requestline is created successfully, navigate back to the request-lines component.
2. Create the requestline-edit component
   * This is a standard edit component which displays the requestline, allows the user to change the product and/or quantity, click the Save button, and, when successfully, navigate back to the request-lines component.