32.0 hours

Project 9

Single Page App

Why can't I submit this project yet?

You must first complete all prior Techdegree content before you can submit this project. Additionally, you cannot submit more than one project at a time.

Submit for Peer Review

Your current activity is HTML Tables.

- Instructions
- How you'll be graded

In this project, you'll create a recipe book single page application (SPA) using AngularJS. To complete the project, you'll use your knowledge of AngularJS to create controllers, update the provided HTML templates with ng attributes, and create a service that calls into the provided Node.js REST API.

Your AngularJS SPA will be a "front end" or "client" application that complements the provided Node.js "back end" or "server" application. The code for the Node.js application is included as part of this project, so you can run the back end server locally. You're welcome to explore that code, but you won't have to make any changes to it, so you can treat it as a "black box" (see https://en.wikipedia.org/wiki/Black_box).

This project will push you to go beyond what is taught in the Angular Basics course. To complete this project, you'll need to use other resources, including but not limited to the official AngularJS documentation (see https://docs.angularjs.org/api). Learning how to ask the right questions and find answers is a key skill that all developers need to know.

Your application will include two screens: a screen that displays a list of recipes and a detail screen that allows the user to add or edit recipes.

Before you start

To prepare for this project you'll need to make sure you complete and understand these steps.

5 steps

- **Have a GitHub account and know how to create a new repository and upload files to it. As with the previous projects, you'll submit your finished working using GitHub.**

- **Download the project files. We've supplied the following files for you to use:**

    - The `public` folder is where you'll be building your AngularJS application. We've provided some files to get you started.
    - The `images` folder contains a handful of images that are used in the application's design.
    - The `scripts` folder is where you'll put all of your JavaScript files. We've included a `route-config.js` file that contains the code to configure the routes for your AngularJS application.

- The `styles` folder contains the application's CSS files.
- The `templates` folder contains two HTML files that you'll use as AngularJS templates. The `recipe-detail.html` and `recipes.html` files are the templates for the "Recipe Detail" and "Recipes" screens. You'll be adding ng directive attributes to these templates as you add functionality to the application.
- The `index.html` file is the one and only HTML page in the application (this is a single page application after all!)
- The `src` folder contains the code files for the provided Node.js REST API. You won't be making any changes to any of the files in this folder, but you're welcome to (optionally) explore the code to see how the API functions.
- We've included a `.gitignore` file to ensure that the `node_modules` folder and the application's database files don't get pushed to your GitHub repo.
- The `nodemon.js` file configures the nodemon Node.js module, which we are using to run the provided REST API.
- The `package.json` file is the project's npm configuration, which includes the project's dependencies.
- The `RecipeAPI.json.postman_collection` file is a collection of Postman requests that you can use to test and explore the provided REST API.

- **When developing your AngularJS application, you'll need to run the provided Node.js application locally on your PC. This application has two important responsibilities. First, it serves the files in the `public` folder as the root of your application. Second, it provides the REST API that your application will use.**

  - Open a `Command Prompt` (on Windows) or `Terminal` (on Mac OS X and Linux) instance and browse to the root project folder.
  - Run the command `npm install` to install the required dependencies.
  - Run the command `npm run-script db` to build the database.
  - Run the command `npm start` to run the Node.js application.
  - To test your application, open your web browser and browse to `http://localhost:5000/`. You should see your application's "Home" or "Default" page, which is the `index.html` file located in the `public` folder.
  - To test the REST API, open your web browser and browse to `http://localhost:5000/api/recipes`. You should see a list of recipes displayed in JSON format.
  - You can press `Ctrl-C` to stop the Node.js application.
  - Important Note: When testing your AngularJS application, you must start the Node.js application and browse to your application using the URL `http://localhost:5000/`. Do not open the `index.html` file from the file system as this will produce errors and unexpected results.

- **The required AngularJS JavaScript files have already been included as part of this project via the npm `angular` and `angular-route` packages. Additionally, the provided `index.html` HTML page already includes the necessary `<script>` tags to load the `angular.js` and `angular-route.js` files.**

- **To learn more about Angular routing with ng-route, look at the example in the Project Resources section.**

Project Instructions

To complete this project, follow the instructions below. If you get stuck, ask a question in the community.

8 steps

- **Set up your angular application:**

    - Create an AngularJS module, save it into a variable called `app`, and add `ngRoute` as a dependency. * Important Note: It's important to use `app` for the name of your app, otherwise the provided Angular routes won't function properly.
    - Update the `index.html` file with the ng-app directive.
    - As you add JavaScript files to your project, remember to add the necessary script tags to the `index.html` page

- **Create a service called `dataService`. This service will make calls to the REST API we have provided for you. Remember that you'll need to use the built-in `$http` service as a dependency for your dataService.**

    - The base URL for the REST API is `http://localhost:5000/`.
    - To your service, add methods to call the following API endpoints:
        - GET `/api/recipes` - Gets all of the recipes.
        - GET `/api/categories` - Gets all of the categories.
        - GET `/api/fooditems` - Gets all of the food items.
        - GET `/api/recipes?category={category}` - Gets all of the recipes for the specified category.
        - GET `/api/recipes/{id}` - Gets the recipe for the specified ID.
        - PUT `/api/recipes/{id}` - Updates the recipe for the specified ID.
        - POST `/api/recipes` - Adds a recipe.
        - DELETE `/api/recipes/{id}` - Deletes the recipe for the specified ID.

- **Set up routes for the "Recipes" and "Recipe Detail" screens:**

    - Uncomment the commented-out code in the `route-config.js` file in the `scripts` folder
    - The `/` route will display the "Recipes" screen.
    - The `/edit/{id}` route will display the "Recipe Detail" screen for the specified recipe ID.
    - The `/add` route will display the "Recipe Detail" screen (with no data).
    - To create links to the "Recipes" and "Recipe Detail" screens, you can add HTML anchor elements with `href` attribute values of `/#/` and `/#/edit/{id}` (or `/#/add` if you want to add a recipe). Be sure to replace `{id}` with an actual recipe ID from your a record in your database.
    - From your controllers, you can browse to the "Recipes" and "Recipe Detail" screens using the built-in AngularJS `$location` service's `path` method. For instance, after a user has saved a recipe using the "Recipe Detail" screen, you can send the user back to the "Recipes" screen with: `$location.path('/')`.

- **Create a `RecipesController` AngularJS Controller that provides the business logic for the `recipes.html` template. Important Note: The name of the controller needs to match in order for the provided Angular routes to function properly. Update the `RecipesController` controller to satisfy the following requirements:**

    - The list of recipes can be filtered by the selected category
    - When a recipe "Edit" button is clicked, the user is taken to the "Recipe Detail" screen, where they can view and edit the details of the recipe.
    - Clicking the recipe "Delete" button deleted that recipes.
    - Clicking the recipe "Add" button adds a new recipe

- **Add built-in ng directives to `recipes.html`.**

- Add directives to the "Categories" select list element to:
  - Populate the list with the categories from the database.
  - Refresh the recipe list when the user has selected a new category
- Add a directive to the "Add Recipe" button to handle user clicks.
- Add a directive to the "No recipes found!" outermost `<div>` element so that the message only displays when there are no recipes to display.
- Add directives to the `<div>` element that represents each recipe row to:
  - Repeat the `<div>` element (and it's content) for each recipe to display.
  - Hide the `<div>` element when there are no recipes to display.
- Within the recipe row `<div>` element, make the following updates:
  - Add a directive to the first `<a>` element in order to allow users to click on a recipe row in order to display the details for that recipe using the "Recipe Detail" screen.
  - Add binding expressions to display the recipe information.
  - Add directives to the "Edit" and "Delete" links to handle user clicks.

- **Create a `RecipeDetailController` AngularJS Controller that provides the business logic for the `recipe-detail.html` template. Important Note: The name of the controller needs to match in order for the provided Angular routes to function properly. Update the `RecipeDetailController` controller to add the following features:**

  - Add or update a recipe. Allow the user to provide the following values::
    - Name (text box)
    - Description (multi-line text box)
    - Category (select list)
    - Prep Time (text box)
    - Cook Time (text box)
  - Add one or more ingredients and steps to a recipe. The user should be able to provide the following values:
    - Item (select list)
    - Condition (text box)
    - Quantity (text box)
  - Add a step to a recipe. The user should be able to provide the following values:
    - Description (text box)

- **Add built-in Angular ng directives to `recipe-detail.html`:**

  - Replace the static text "Add/Edit Recipe" with a binding expression that displays the recipe name when editing a recipe.
  - When adding a recipe, display the static text 'Add New Recipe'.
  - Add directives to both the "Save Recipe" and "Cancel" buttons in order to handle user clicks.
  - Add a directive to the following properties to bind their value or contents to the corresponding properties on the recipe model:
    - Name, Prep Time and Cook Time `<input>` elements
    - Description (textarea)
    - Category (select element)
    - Prep Time (input)
    - Cook Time (input)
  - Add a directive to the "Ingredient" `<div>` element so that it repeats for each recipe ingredient to display.
  - Add directives to the "Item" `<select>` element to bind its value to the recipe ingredient model's `foodItem` property and to populate the list with the food items from the database.
  - Add a directive to the "Condition" `<input>` element to bind its value to the recipe ingredient model's `condition` property.

- Add a directive to the "Amount" `<input>` element to bind its value to the recipe ingredient model's `amount` property.
- Add a directive to the "Delete" `<a>` element so that you can delete the recipe ingredient when the user clicks on the link.
- Add a directive to the "Add Another Ingredient" `<button>` element in order to handle when the user clicks to add a new recipe ingredient.
- Add a directive to the "Step" `<div>` element so that it repeats for each recipe step to display.
- Add a directive to the "Description" `<input>` element to bind its value to the recipe step model's `description` property.
- Add a directive to the "Delete" `<a>` element so that you can delete the recipe step when the user clicks on the link.
- Add a directive to the "Add Another Step" `<button>` element in order to handle when the user clicks to add a new recipe step.

- **Wrap all of your JavaScript application, controller, and services code in immediately invoked functions in order to prevent from polluting the global namespace.**

Extra Credit

To get an "exceeds" rating, you can expand on the project in the following ways:

1 step

- **Write code that asks the user to confirm a recipe deletion before it is deleted.**

NOTE:

- To get an "Exceeds Expectations" grade for this project, you'll need to complete **each** of the items in this section. See the rubric in the "**How You'll Be Graded**" tab above for details on how you'll be graded.
- If you're shooting for the "Exceeds Expectations" grade, it is recommended that you mention so in your submission notes.
- Passing grades are final. If you try for the "Exceeds Expectations" grade, but miss an item and receive a "Meets Expectations" grade, you won't get a second chance. Exceptions can be made for items that have been misgraded in review.

Download files

Zip file

Project Resources

Course

Angular Basics

External Link

AngularJS Documentation

http://www.journaldev.com/6225/angularjs-routing-example-ngroute-routeprovider

Angular Routing Example

External Link

AngularJS Router Documentation

External Link

AngularJS `ng-options` Directive Documentation

External Link

AngularJS `ng-href` Directive Documentation

External Link

AngularJS `$location` Service Documentation

External Link

AngularJS `$http` Service Documentation (`get`, `post`, `put`, `delete` methods)

External Link

ngInspector

External Link

AngularJS Batarang

External Link

John Papa's AngularJS Style Guide

Workshop

The Module Pattern in JavaScript

External Link

Immediately Invoked Function Expressions

External Link

Bookmarklet to Detect Global Variables

Need Help?

Have questions about this project? Start a discussion with the community and Treehouse staff.

Get Help