Full Stack Programming Project III

PROG3017 Full Stack Programming

Evaluation : 30% of Final Grade

Due Date : Dec 6

Assignment Description

In a previous lesson we explored developing a true RESTful Web API using Node.js / Express. The Web API enabled us to add, edit, and delete *technologies* from the Tech Roster's MongoDB database. It is time we put this Web API to use!

Full Stack Programming Project III will be to develop a full administration web app for the Tech Roster.

The requirements are:

- Using /techRoster-Admin (Unit 4 / Lesson 1) as a starting point, develop an administration full stack web app (aka an administration end) that will enable the user to add / edit / delete technologies AND courses from the Tech Roster's MongoDB.
- Like all full stack web apps, the requirements can be split up into:

  Database Requirements:

  - Add a new collection called courses to the dbTechs database that contains course documents (_id / course code / course name) for all programming / technical courses of the IT Web Program
  - The technologies collection remains the same as developed during our lesson. While the content of this collection will change as you test, its structure must remain the same (no added fields are allowed!)

  Server-Side Requirements:

  - The current Web API only supports working with technology documents - you will need to add post / put / delete support to work with course documents.
    - Your existing get request should fetch *all* the data in JSON format (technologies and courses)
    - Remember that success and error status codes should be returned with a response for it to be RESTful
  - The handling of the requests to your RESTful Web API (via Express) *must* be done using promises or async / await
  - Your MongoDB database will contain two collections – and these are NOT relational, but it doesn't mean you don't have to worry about keeping the data consistent!

  Client-Side Requirements:

  - The web app must start with a main page that lists all technologies and courses with buttons beside each to edit / delete as well as a single add button for each.
  - The add new technology page must provide inputs for name, description, difficulty dropdown, and a series of checkboxes to select which courses use the technology
    - The name and description fields are required

  - The edit technology page will look the same as the add new technology page except all content will be populated according to the technology the user wants to edit.
    - This population includes checking off the corresponding course checkboxes that use the current technology
    - The name and description fields are required
  - The delete technology page will provide a warning with the technology's name
  - The add new course page must provide inputs for course code and name
    - The course code and name fields are required
  - The edit course page will look the same as the add new course page except all content will be populated according to the course the user wants to edit.
    - The user can only modify the course name on this page. While the course code is displayed, it is greyed out. This limitation is to simplify this already large web app!
  - The delete course page will provide a warning with the course's name
  - All pages (except the main page) will include an ok and cancel button
    - Clicking either button will ultimately take the user back to the main page

  - You may notice that the main component's template already contains a commented out <router-outlet> directive. You are required to explore routed components for this project (Unit 1 / Lesson 7). No nested components allowed.
  - A loading screen with an animated spinner should be displayed while all data communication is being carried out to prohibit the user from tinkering with the controls of the web app until it is ready
  - While the /techRoster-Admin project already contains some Sass and flexbox styling, you will need to ensure this web app is indeed responsive
  - It is very easy to develop this web app with A LOT of redundancy in your angular code – try to come up with clean and efficient ways to avoid this – for example, creating a new technology document is *almost* the same process as creating a new course document!

  - See sample screen captures for references of above pages

- Although it *should* be included, there is no need to worry about adding a user login for this administration web app
- As a final step, publish the working Tech Roster Administration web app to AWS. Be sure to not leave it live after code review since there is no login protection!

Requirements (Marks breakdown)

Admin Full Stack Web App (MEAN Stack)	
MongoDB Courses Collection	1
Application Test Cases *Various test cases that test most functionality requirements above*	19
Server Side Code / Structure *Node.js / Express RESTful Web API Code*	4
Client Side Code / Structure *Angular Client Side Code and Structure*	6
Responsive with Flexbox / Sass Styling	1
Publishing web app to AWS	4
TOTAL MARK	35

Other Notes

- This project will be marked on the due date through code review.