**Capstone Project**

**Building a Full-Stack Application to  
Explore the Offerings in a Simple Bookstore**

**Capstone Overview**

In this capstone, you will create a full-stack application that allows bookstore employees to internally manage the books carried by a small bookstore.

You will begin by designing and implementing a back-end system for the bookstore. This will involve creating a relational database to store information about books, authors, and genres, and building a REST API to interact with the database. NOTE: This application will NOT include any ability to purchase books.

You will also design a front-end application that allows users to make queries about books that are carried, as well as add/delete/edit data.

**Part 1: Building the Node.js Back-end**

**Database Requirements**

Create a MySQL database for your bookstore with the following requirements:

1. Books Table:
   * book\_id (Primary Key)
   * title (VARCHAR)
   * author\_id (Foreign Key referencing the Authors table)
   * genre\_id (Foreign Key referencing the Genres table)
   * price (DECIMAL)
   * publication\_date (DATE)
2. Authors Table:
   * author\_id (Primary Key)
   * name (VARCHAR)
   * biography (TEXT)
3. Genres Table:
   * genre\_id (Primary Key)
   * genre\_name (VARCHAR)

Create a script to clear all data in all tables and then seed the tables with sample data to support API testing.

**REST API Requirements**

Create your API server and the following endpoints:

1. Books:
   * GET /books : Retrieve a list of all books
   * GET /books/{book\_id} : Retrieve details of a specific book
   * POST /books : Add a new book
   * PUT /books/{book\_id} : Update details of an existing book
   * DELETE /books/{book\_id} : Delete a specific book
2. Authors:
   * GET /authors : Retrieve a list of all authors
   * GET /authors/{author\_id} : Retrieve details of a specific author
   * POST /authors : Add a new author
   * PUT /authors/{author\_id} : Update details of an existing author
   * DELETE /authors/{author\_id} : Delete a specific author
3. Genres:
   * GET /genres : Retrieve a list of all genres
   * GET /genres/{genre\_id} : Retrieve details of a specific genre
   * POST /genres : Add a new genre

**API considerations:**

* Implement error handling for scenarios such as invalid input, non-existent records, and database connectivity issues.
* Ensure data validation for all inputs to the API to maintain data integrity.

**Back-end Repository Requirements**

* Provide a Git repository in either GitHub or GitLab
* Include in your repo's README file instructions on how to set up and run your project, including:
  + how to create the database and inserting sample data
  + how to run the server
  + list of endpoints
* Your repo should also include:
  + SQL scripts for creating the database and inserting sample data
  + source code for the REST API

**Part 2: Building a Front-end**

**Front-end Requirements**

Create a React front-end application that interacts with the REST API you built in Part 1. The application should include the following features:

**Home Page:**

* Display a welcome message and a brief description of the bookstore application

**Books Page:**

* Display a list of all books retrieved from the /books endpoint
* Allow users to view details of a specific book by clicking on a book title, which fetches data from the /books/{book\_id} endpoint
* Provide a form to add a new book, making a POST request to the /books endpoint
* Enable editing of book details through a form that sends a PUT request to the /books/{book\_id} endpoint
* Allow deletion of a book with a button that sends a DELETE request to the /books/{book\_id} endpoint

**Authors Page:**

* Display a list of all authors retrieved from the /authors endpoint
* Allow users to view details of a specific author by clicking on an author’s name, which fetches data from the /authors/{author\_id} endpoint
* Provide a form to add a new author, making a POST request to the /authors endpoint
* Enable editing of author details through a form that sends a PUT request to the /authors/{author\_id} endpoint
* Allow deletion of an author with a button that sends a DELETE request to the /authors/{author\_id} endpoint

**General Considerations:**

* Ensure the front-end application has proper error handling for failed API requests and invalid inputs
* Implement loading indicators for data-fetching operations
* Include basic styling to make the application user-friendly and visually appealing

**Front-end Repository Requirements**

* Provide a Git repository in either GitHub or GitLab
* Include in your repo's README file:
  + an overview of the project
  + instructions on how to set up and run your web application
  + sample screen shots of interesting features