College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
ID608001: Intermediate Application Development Concepts
Level 6, Credits 15

Project

Assessment Overview

In this **individual** or **group** assessment, you will develop a **RESTful API** using **Express**, **Node.js** and the **OpenTDB API**, and deploy it as a **web service** on **Render**. Your data will be stored in a **PostgreSQL** database on **Render**. Also, you will develop **frontend applications** using **React** and **Vite**. These applications will consume the **RESTful API** mentioned above. The main purpose of this assessment is not just to build **full-stack applications**, but rather to demonstrate an ability to decouple your **RESTful API** from your **frontend applications**. In addition, marks will be allocated for code quality and best practices, documentation and **Git** usage.

Learning Outcomes

At the successful completion of this course, learners will be able to:

1. Apply design patterns and programming principles using software development best practices.
2. Design and implement full-stack applications using industry relevant programming languages.

Assessments

Assessment	Weighting	Due Date	Learning Outcome
Practical	20%	26-06-2024 (Wednesday at 4.59 PM)	1
Project	80%	21-06-2024 (Friday at 4.59 PM)	1 and 2

Conditions of Assessment

You will complete this assessment during your learner-managed time. However, there will be time during class to discuss the requirements and your progress on this assessment. This assessment will need to be completed by **Friday, 21 June 2024** at **4.59 PM**.

Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** over all assessments in **ID608001: Intermediate Application Development Concepts**.

Authenticity

All parts of your submitted assessment **must** be completely your work. Do your best to complete this assessment without using an **AI generative tool**. You need to demonstrate to the course lecturer that you can meet the learning outcome(s) for this assessment.

However, if you get stuck, you can use an **AI generative tool** to help you get unstuck, permitting you to acknowledge that you have used it. In the assessment's repository **README.md** file, please include what prompt(s) you provided to the **AI generative tool** and how you used the response(s) to help you with your work. It also applies to code snippets retrieved from **StackOverflow** and **GitHub**.

Failure to do this may result in a mark of **zero** for this assessment.

Policy on Submissions, Extensions, Resubmissions and Resits

The school's process concerning submissions, extensions, resubmissions and resits complies with **Otago Polytechnic | Te Pūkenga** policies. Learners can view policies on the **Otago Polytechnic | Te Pūkenga** website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Submission

You **must** submit all application files via **GitHub Classroom**. Here is the URL to the repository you will use for your submission – https://classroom.github.com/a/wlzE5yYo. If you do not have not one, create a **.gitignore** and add the ignored files in this resource - https://raw.githubusercontent.com/github/gitignore/main/Node.gitignore. The latest application files in the **main** branch will be used to mark against the **Functionality** criterion. Please test before you submit. Partial marks **will not** be given for incomplete functionality. Late submissions will incur a **10% penalty per day**, rolling over at **5:00 PM**.

Extensions

Familiarise yourself with the assessment due date. Extensions will **only** be granted if you are unable to complete the assessment by the due date because of **unforeseen circumstances outside your control**. The length of the extension granted will depend on the circumstances and **must** be negotiated with the course lecturer before the assessment due date. A medical certificate or support letter may be needed. Extensions will not be granted for poor time management or pressure of other assessments.

Resits

Resits and reassessments **are not** applicable in **ID608001: Intermediate Application Development Concepts**.

Instructions

Functionality - Learning Outcomes 1 and 2 (50%)

- **RESTful API:**

- **Note:** You are required to use the provided **schema.prisma** file. You can find this in the **assessments** directory of the **course materials** repository.
- **User:**
  - You will have **two** types of users - **admin** and **basic** user.
  - An **admin** and **basic** user will have the following information: first name, last name, username, email address, profile picture, password, confirm password and role. The confirm password does not need to be stored in the database. The users' profile picture will be from the following **API** - https://avatars.dicebear.com/docs/http-api. Each profile picture should be, in most cases, different. I suggest using a random seed when setting the user's profile picture.
  - An **admin** and **basic** user can login, get their information and update their information but not delete themselves. An **admin** user can get all users' information, update all **basic** users' information but not **admin** user's information and delete all **basic** users but not **admin** users. A **basic** user can register.
  - When performing a **POST** request for registering a **basic** user, the following error checking needs to be implemented using **Joi** and/or conditional statements:
    - First name has a minimum length of two characters, a maximum length of 50 characters and alpha characters only.
    - Last name has the same error checking as first name above.
    - Username is unique, has a minimum length of five characters, maximum length of ten characters and alphanumeric characters only, i.e., johndoe123.
    - Email address is unique, contains the username above, an @ special character and a second-level domain, i.e., johndoe123@email.com.
    - Password has a minimum length of eight characters, maximum length of 16 characters and contains one numeric character and one special character.
    - Confirm password is the same as the password above. **Note:** Confirm password will not be a field in the **User** table. Rather, it will be used to validate the user's password.

    For each error check, a status code and response message is returned, i.e., "First name must have a minimum length of two characters".
  - When performing a **POST** request for logging in a user using either username/password or email address/password, return a status code, a response message, i.e., "<User's username> has successfully logged in" and the user's **JWT**.
  - When performing a **PUT** and **DELETE** request, return a status code and a response message, i.e., "<User's username>'s information has successfully updated" or "<User's username> has successfully deleted".
  - Five **admin** users are seeded via a **script** in the **package.json** file. The **admin** users' data will be fetched from a local file and inserted into the **User** table using **Prisma**.
  - Five **basic** users are seeded via an **admin** user. The **basic** users' data will be fetched from a private **GitHub Gist** using **Axios** and inserted into the **User** table using **Prisma**.
- **Quiz:**
  - Each quiz will have the following information: name, start date, end date, category, difficulty, type, number of questions, list of questions, list of correct answers, list of incorrect answers, list of scores and average score. The category, list of questions, list of correct answers and list of incorrect answers will be fetched from the following **API** - https://opentdb.com/api_config.php. The difficulties will be easy, medium and hard. The types will be multiple choice or true/false. An **admin** user can create and delete a quiz. A **basic** user can participate in a quiz only once. An **admin**, **basic** and **unauthenticated** user can get all quizzes, get all past quizzes, get all present quizzes and get all future quizzes, get a list of scores for a quiz and average score for a quiz.
  - When performing a **POST** request for creating a quiz, the following error checking needs to be implemented using **Joi** and/or conditional statements:
    - Name has a minimum length of five characters, a maximum length of 30 characters and alpha characters only.
    - Start date has to be greater than or equal to today's date.
    - End date has to be greater than the start date and no longer than five days.
    - Number of questions has to be ten.

For each error check, a status code and response message is returned, i.e., ”Name must have a minimum length of five characters”.

  - When performing a **POST** request for a **basic** user who is participating in a quiz, the following error checking needs to be implemented using **Joi** and/or conditional statements:
    - Can not participate if quiz has not started or has ended.
    - Answered all ten questions.
  - When performing a **POST** request for a **basic** user who has participated in a quiz, return a status code, a response message, i.e., ”<User's username> has successfully participated in <Quiz's name>”, user's score and quiz's average score.
- Implement versioning, **Helmet**, **CORS**, **rate limiting**, **caching** and **compression**

- **Frontend Application - Quiz:** Create a new application using **Create Vite App**. In this application:
  - **User:**
    - Create a page that allows a **basic** user to register.
    - Create a page that allows an **admin** and **basic** user to login and logout.
    - Create pages that allow:
      - An **admin** user to get their information, update their information but not delete themselves, get all user's information, update all **basic** users' information but not **admin** user's information and delete all **basic** users but not **admin** users.
      - A **basic** user to get their information and update their information but not delete themselves.
  - **Quiz:**
    - Create pages that allow:
      - An **admin** user to create and delete a quiz.
      - A **basic** user to participate in a quiz only once.
      - An **admin**, **basic** and **unauthenticated** user to get all quizzes, get all past quizzes, get all present quizzes, get all future quizzes, list of scores for a quiz and average score for a quiz.
  - Create a navigation bar that allows an **admin**, **basic** and **unauthenticated** user to navigate to the pages above.
  - Incorrectly formatted form field values handled gracefully using validation error messages.
  - Create a footer that displays the current year and the name of the application.
  - UI is visually attractive with a coherent graphical theme and style using **Tailwind CSS** and **Shadcn UI**.

- **Frontend Application - RESTful API:** Create a new application using **Create Vite App**. In this application:
  - Create a page that allows an **unauthenticated** user to get **RESTful API** information regarding the following:
    - Logging in a user
    - Registering a user
    - Creating a quiz

    The information to display includes:
    - Name of the endpoint. For example, **Register User**.
    - HTTP method. For example, **POST**.
    - Endpoint URL. For example, **/api/v1/auth/register**.
    - Request body.
    - Response body.
  - UI is visually attractive with a coherent graphical theme and style using **Tailwind CSS** and **Shadcn UI**.

- **Testing:**
  - Implement the following **end-to-end tests** using **Cypress**:
    - Registering a new user.
    - Logging in as an **admin** user and creating a quiz.
    - Logging in as a **basic** user and participating in a quiz.

- **Scripts:**
  - Run your **RESTful API** and **frontend applications** locally.
  - Run your **end-to-end tests** using **Cypress**.
  - Create and apply a migration using **Prisma**.
  - Reset your database using **Prisma**.
  - Seed **admin** users.
  - Open **Prisma Studio**.
  - Lint your code using **ESLint**.
  - Format your code using **Prettier**.

Code Quality and Best Practices - Learning Outcome 1 (45%)

- A **Node.js .gitignore** file is used.
- Environment variables' key is stored in the **.env.example** file.
- Appropriate naming of files, variables, functions and resource groups.
  - Resource groups are named with a plural noun instead of a noun or verb, i.e., **/api/v1/items** not **/api/v1/item**.
- Idiomatic use of control flow, data structures, in-built functions, data fetching and state management design patterns.
- Efficient algorithmic approach.
- Sufficient modularity.
- Each **controller**, **route** and **component** file has a **JSDoc** header comment located at the top of the file.
- Code is formatted using **Prettier**.
- Code is linted using **ESLint**.
- **Prettier**, **ESLint** and **Cypress** are installed as a **development dependency**.
- No dead or unused code.

Documentation and Git Usage - Learning Outcomes 1 (5%)

- A **GitHub** project board or issues to help you organise and prioritise your development work. The course lecturer needs to see consistent use of the **GitHub** project board or issues for the duration of the assessment.
- Provide the following in your repository **README.md** file:
  - A URL to your **RESTful API** as a **web service** on **Render**.
  - A URL to your published **RESTful API** documentation. Each route needs to be documented. Include a description, example request and example response.
  - How do you setup the environments, i.e., after the repository is cloned?
  - How do you run your **RESTful API** and **frontend applications** locally?
  - How do you run your **End-to-end tests**?
  - How do you create and apply a migration?
  - How do you reset your database?
  - How do you seed **admin** users?
  - How do you open **Prisma Studio**?
  - How do you format your code?
  - How do you lint your code?

- Use of **Markdown**, i.e., headings, bold text, code blocks, etc.
- Correct spelling and grammar.
- Your **Git commit messages** should:
  - Reflect the context of each functional requirement change.
  - Be formatted using an appropriate naming convention style.

Additional Information

- **Do not** rewrite your **Git** history. It is important that the course lecturer can see how you worked on your assessment over time.
- You need to show the course lecturer the initial **GitHub** project board or issues before you start your development work. Following this, you need to show the course lecturer your **GitHub** project board or issues at the end of each week.