College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
ID607001: Introductory Application Development Concepts
Level 6, Credits 15

Project

Assessment Overview

In this **individual** assessment, you will develop two **REST APIs** using **Express** and **Node.js**, and deploy them as a **web service** on **Render**. Your data will be stored in a **PostgreSQL** database on **Render**. In addition, marks will be allocated for code quality and best practices, documentation and **Git** usage.

Learning Outcome

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

1. Design and build secure applications with dynamic database functionality following an appropriate software development methodology.

Assessments

Assessment	Weighting	Due Date	Learning Outcome
Practical	20%	13-11-2024 (Wednesday at 4.59 PM)	1
Project	80%	13-11-2024 (Wednesday at 4.59 PM)	1

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 **Wednesday, 13 November 2024** at **4.59 PM**.

Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** across all assessments in **ID607001: Introductory Application Development Concepts**.

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/WBzw8fEH. 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. Create a branch called **project**. The latest application files in the **project** 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**.

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 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.

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 on the due date and for poor time management or pressure of other assessments.

Resits

Resits and reassessments are not applicable in **ID607001: Introductory Application Development Concepts**.

Instructions

Functionality - Learning Outcome 1 (50%)

- **Your choice REST API (25%):**
  - Create a new directory called **your-choice-rest-api** and create a new **Express** application using **Node.js**.
  - Can run in development and production without modification.
  - **Four models**. Each **model** contains a **minimum** of **three fields** excluding the **id**, **createdAt** and **updatedAt fields**.
  - A range of different data types, i.e., all **fields** in a **model** can not be of a single type.
  - **Four relationships** between **models**.
  - **One model** has an **enum field**.
  - A **repository**, **controller** and **route** file for each **model**. Each **controller** file needs to contain operations for **POST**, **GET all**, **GET one**, **PUT** and **DELETE**. **Note:** You can create a generic **repository** file for all **models** if you wish.
  - Return an appropriate success or failure message, and status code when performing the operations, i.e., **”Successfully created an institution”** or **”No institutions found”**, and **200** or **404**.
  - **Filter** and **sort** your data using **query parameters**. All **fields** should be filterable and sortable (in ascending and descending order).
  - **Paginate** your data using **query parameters**. The default number of data per page is 25.
  - Return an appropriate message if an endpoint does not exist.
  - When creating and updating, validate each **field** using **Joi**.
  - Scripts for running your **REST API** locally, creating and applying a migration, resetting the **PostgreSQL** database, opening **Prisma Studio**, checking your code and formatting your code are included in the **package.json** file.
- **OpenTDB REST API (25%):**
  - Create a new directory called **opentdb-rest-api** and create a new **Express** application using **Node.js**.
  - Can run in development and production without modification.
  - In your **schema.prisma** file, implement the following **enums**:
    - **Role** - BASIC and ADMIN.
    - **Type** - multiple and boolean.
    - **Difficulty** - easy, medium and hard.
  - In addition, implement the following **models**:
    - **User** - id, emailAddress, firstName, lastName, password, loginAttempts, lastLoginAttempt and role.
    - **Category** - id and name. **Note:** id is an **Int**.
    - **Question** - id, quizId, question, correctAnswer and incorrectAnswers.
    - **Quiz** - id, categoryId, name, type, difficulty, startDate and endDate.
    - **UserQuestionAnswer** - id, userId, quizId, questionId, answer and isCorrect.
    - **UserQuizScore** - id, userId, quizId and score.
  - The category, list of questions, list of correct answers and list of incorrect answers will be fetched from the https://opentdb.com/api_config.php.
  - An **ADMIN** user can:
    - Login.
    - Create (POST) a quiz, retrieve all (GET) quizzes, retrieve (GET) a quiz, update (PUT) a quiz and delete (DELETE) a quiz.

    * Retrieve all **ADMIN** and **BASIC** user information excluding password.
    * Retrieve all scores.
  - A **BASIC** user can:
    * Register and login.
    * Retrieve all (GET) quizzes and retrieve (GET) a quiz.
    * Play a quiz.
    * Retrieve all scores.
  - When 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.
  - When playing 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.
  - Two **ADMIN** and three **BASIC** users are seeded via a **script** in the **package.json** file. The **ADMIN** and **BASIC** users' data will be fetched from a local file and inserted into the **User** table using **Prisma**.
  - Implement **Helmet**, **rate limiting** and **compression**.
  - Scripts for running your **REST API** locally, creating and applying a migration, resetting the **PostgreSQL** database, seeding **ADMIN** and **BASIC** users, opening **Prisma Studio**, checking your code and formatting your code are included in the **package.json** file.

- In addition, each **REST API** will:
  - Have an endpoint for Swagger documentation. Each route needs to be documented.
  - Store their data in a **PostgreSQL** database on **Render**. **Note:** You may have to use the same database for both **REST APIs**.
  - Be deployed as a **web service** on **Render**.

Code Quality and Best Practices - Learning Outcome 1 (40%)

- 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.
  - API endpoints are versioned, i.e., **/api/v1**.
  - 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 and in-built functions.
- Efficient algorithmic approach.
- Sufficient modularity.
- Each file has a **JSDoc** header comment located at the top of the file.
- Code is formatted.
- No dead or unused code.

Documentation and Git Usage - Learning Outcome 1 (10%)

- 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. **Note:** If you cannot create a **GitHub** project board, please use **GitHub** issues.
- Provide the following in your repository **README.md** file:
  - A URL to your **REST APIs** as a **web service** on **Render**.
  - How do you setup the environments, i.e., after the repository is cloned?
  - How do you run your **REST APIs** locally?
  - How do you create and apply a migration?
  - How do you reset the **PostgreSQL** database?
  - How do you seed **ADMIN** and **BASIC** users?
  - How do you open **Prisma Studio**?
  - How do you check your code?
  - How do you format your code?
  - ERD of both **REST APIs**.

  Note: You can find a **README.md** template in the **assessments** directory on **GitHub**.
- 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

- Your choice **REST API** must be signed off by the course lecturer before you start your development work.
- **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.