College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
Year Two - Special Topic
Level 6, Credits 15

Project 1: REST/GraphQL APIs

Assessment Overview

In this assessment, you will develop two types of **APIs** using **REST** & **GraphQL**. You will then deploy these **APIs** to **AWS Amplify**. You will choose the theme of your **API**. It could be on sport, culture, food or something else you are interested in. Your **API** data will be stored in a **MongoDB Atlas** database. The main purpose of this assessment is to demonstrate your ability to develop **APIs** using modern technologies as well as high-level concepts such as queries, authentication, relationships, validation, seeding, rate limits, filtering, sorting & pagination. In addition, marks will be allocated for code elegance, documentation & **Git** usage.

Learning Outcome

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

1. Design, create & deploy microservices using a range of industry-relevant technologies.
2. Critically reflect on & evaluate own learning to identify ways of further personal development.

Assessment Table

Assessment Activity	**Weighting**	**Learning Outcome**	**Assessment Grading Scheme**	**Completion Requirements**
Project 1: REST/GraphQL APIs	40%	1	CRA	Cumulative
Project 2: React	40%	1	CRA	Cumulative
Evaluative Conversation	20%	2	CRA	Cumulative

Conditions of Assessment

You will complete this assessment during your learner-managed time, however, there will be availability during the weekly meetings to discuss the requirements. This assessment will need to be completed by **Friday, 11**

February 2022 at 5:00 PM.

Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** across all assessments in **Year Two - Special Topic**.

Authenticity

All parts of your submitted assessment must be completely your work & any references must be cited appropriately. Provide your references in a **README.md** file. Failure to do this will result in a mark of **zero** for this assessment.

Policy on Submissions, Extensions, Resubmissions & Resits

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

Submissions

You must submit all program files via **GitHub**. You will need to create a new repository & add **grayson-orr** as a collaborator. The latest program files in the **main** branch will be used to mark against the **Functionality** criterion. Please test your **main** branch application before you submit. Partial marks **will not** be given for functionality in other branches. Late submissions will incur a **10% penalty per day**, rolling over at **5:00 PM**.

Extensions

Familiarise yourself with the assessment due date. If you need an extension, contact the course lecturer before the due date. If you require more than a week's extension, a medical certificate or support letter from your manager may be needed.

Resubmissions

Learners may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are to be completed within a negotiable short time frame & usually must be completed within the timing of the course to which the assessment relates. Resubmissions will be available to learners who have made a genuine attempt at the first assessment opportunity & achieved a **D grade (40-49%)**. The maximum grade awarded for resubmission will be **C-**.

Resits

Resits & reassessments are not applicable in **Year Two - Special Topic**.

Instructions

You will need to submit a **REST** & **GraphQL API**, & documentation that meet the following requirements:

Functionality - Learning Outcome 1 (45%)

- **APIs** are developed using **Node.js**.
- **APIs** can run locally without modification.
- **REST API**
  - Three **collections** containing at least four **fields** of data which you can interact with.
  - You must have a range of different data types, i.e., all **fields** of data can not be of type **string**.
  - Each **collection** must have a **controller** containing **CRUD** (create, read, write & delete) functionality.
  - Custom validation when creating & updating a **document**.
  - Each **collection** must be seeded with a **JSON** file. **Note:** It should be testing data.
  - API version set to v1. For example, an endpoint should look like - **/api/v1/institutions**
  - Return an appropriate status code & message when performing **CRUD** actions. For example, when a **document** is created, return 200 & **document** successfully created.
  - Return an appropriate message if a query does not return any data.
  - Filter & sort **API** data on all **fields** using query parameters. A user should be able to sort **API** data in ascending & descending order.
  - Paginate the **API** data so that 25 records are displayed per page.
  - **POST**, **PUT** & **DELETE** routes are protected using **JSON Web Tokens (JWT)**.
  - Set the **API** rate limit to 25 requests per minute.
  - **API** application deployed to **AWS Amplify**. The application must be usable i.e., a user should be able to perform requests to your **API**.
  - **API** data is stored in a **MongoDB Atlas** database.
- **GraphQL API**
  - Your **GraphQL API** must use the **REST API** above.
  - Provide five schemas that return different **API** data.

Code Elegance - Learning Outcome 1 (45%)

- Use of intermediate variables. No method calls as arguments.
- Idiomatic use of control flow, data structures & in-built functions.
- Functions & variables are named appropriately.
- Efficient algorithmic approach, i.e., using the appropriate function(s) when querying your **collections**.
- **API** resource groups named with a plural noun instead of a verb, i.e., **/api/students** not **/api/student**.
- Function header comments explain each **CRUD** action in a **controller**.
- In-line comments explaining complex logic, i.e., a line of code that may need additional explanation.
- Code files are formatted using **Prettier**.
- No dead or unused code.
- Databases configured for production environment, i.e., do not expose your database credentials.

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

- Provide the following in your repository **README.md** file:
  - URL to the **APIs** on **AWS Amplify**.
  - How do you setup the environment for development, i.e., after the repository is cloned, what do you need to run the **APIs** locally?
  - How do you deploy the **APIs** to **AWS Amplify**?
- Commit messages **must**:
  - Reflect the context of each functional requirement change.
  - Be formatted using the naming conventions outlined in the following:
    - **Resource:** https://dev.to/i5han3/git-commit-message-convention-that-you-can-follow-1709

Additional Information

- You **must** commit at least **five** times per week. By the end of this assessment, you should have at least **50** commits.
- **Do not** rewrite your **Git** history. It is important that the course lecturer can see how you worked on your assessment over time.