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

Practical: Node.js REST API Testing Research

Assessment Overview

In this **individual** assessment, you will be given a **Node.js REST API** to **API test**. You will be required to independently research & write at least **50 API tests** using **Mocha** & **Chai**. You will use these dependencies to verify the correctness of the given **Node.js REST API**. It includes **CRUD** operations, authentication, query parameters, status codes & the shape of response data. You will also check the coverage of your **API tests** using **nyc**. 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 & build secure applications with dynamic database functionality following an appropriate software development methodology.

Assessment Table

Assessment Activity	**Weighting**	**Learning Outcome**	**Assessment Grading Scheme**	**Completion Requirements**
Practical: Node.js REST API Testing Research	20%	1	CRA	Cumulative
Project 1: Node.js REST API	30%	1	CRA	Cumulative
Project 2: React CRUD	50%	1	CRA	Cumulative

Conditions of Assessment

You will complete this assessment during your learner-managed time. However, there will be time to discuss the requirements & your assessment progress during the teaching sessions. This assessment will need to be completed by **Friday, 13 May 2022 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 program files via **GitHub Classroom**. Here is the URL to the repository you will use for your submission – https://classroom.github.com/a/Anc_bYhn. The latest program files in the **master** or **main** branch will be used to mark against the **Functionality** criterion. Please test your **master** or **main** branch application 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. If you use code snippets from **GitHub**, **StackOverflow** or other online resource, you **must** reference it appropriately using **APA 7th edition**. Provide your references in the **README.md** file in your repository. 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.

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 **seven days** 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 **ID607001: Introductory Application Development Concepts**.

Instructions

You will need to submit a **suite of tests** & documentation that meet the following requirements:

Functionality - Learning Outcome 1 (60%)

- **API tests** are written using **Mocha** & **Chai**.
- At least **50 API tests** verifying the correctness of the following:
  - CRUD (create, read, update & delete) operations.
  - Authentication (register, login & logout).
  - Validation rules, i.e., checking if field is required, etc.
  - Query parameters, i.e., filtering & sorting data.
  - Status codes, i.e., checking if a response returns 200, 404, etc.
  - Shape of the data, i.e., does the response data contain a specific column?
- **Code coverage** using **nyc**.

Note: Example test cases are provided at the end of this document.

Code Elegance - Learning Outcome 1 (30%)

- Use of intermediate variables. No method calls as arguments.
- Idiomatic use of control flow, data structures & in-built functions.
- Sufficient modularity, i.e., setup method at the beginning of each test case.
- Functions & variables are named appropriately.
- File header comments using **JSDoc**. You **need** to explain the purpose of each each **API test** file.
- In-line comments using **JSDoc**. You **need** to explain complex logic that is not obvious.
- Test files are stored a directory called **tests** located in the root directory & have the extension **.test.js**.
- Code files are formatted using **Prettier** & a **.prettierrc** file. You **need** to declare a **npm** script in your application's **package.json** file which automates this process. Rules **should** include:
  - Single quote is set to **true**.
  - Semi-colon is set to **false**.
  - Tab-width is set to **2**.
- Declare a **npm** script in your application's **package.json** file that runs the **API test** & **code coverage** sequentially.
- **Prettier**, **Chai**, **Mocha** & **nyc** are installed as development dependencies.
- No dead or unused code.
- Database configured for testing environment.
- Application's environment variables are stored in a **.env** file. Create **example.env** file containing all of the application's environment variables' keys. Do not include the environment variables' value.

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

- Provide the following in your repository **README.md** file:
  - How do you setup the environment for development, i.e., after the repository is cloned, what do you need to run the the **API tests** locally?
  - How do you run the **API tests**?
  - How do you run the **code coverage**?
  - How do you format the code?
  - How do you seed the **collections**?
- Use of **Markdown**, i.e., bold text, code blocks, etc.
- Correct spelling & grammar.
- Your **Git commit messages** should:
  - 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

- Attempt to commit at least **10** times per week.
- **Do not** rewrite your **Git** history. It is important that the course lecturer can see how you worked on your assessment over time.

Example Test Cases

- Reading a resource => three tests
- Invalid inputs when creating or updating a resource => at least nine tests
- Deleting a resource => three tests
- Valid & invalid inputs when registering & logging in a user => at least four tests
- Logging out a user => one test
- Endpoint not found => one test
- The number of available base endpoints => three tests
- Query parameters for each resource => at least 15 tests
- Status codes for each resource => at least six tests
- Shape of the data => at least five tests