College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
IN607: Introductory Application Development Concepts
Level 6, Credits 15

Practical

Assessment Overview

In this assessment, you will develop & deploy an API of any theme using **Laravel** & **Heroku**. Your could be something to do with sport, culture, food...the world is your oyster. Your API data will be stored in a **PostgreSQL** database on **Heroku** & in a **MySQL** database locally. The main purpose of the assessment is to demonstrate your ability to develop a complex API using advanced features such as filtering, sorting & paging. In addition, marks will be allocated for code elegance, documentation & **Git** usage.

Learning Outcomes

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

1. Design & build usable, secure & attractive applications with dynamic database functionality following an appropriate software development methodology.

Assessment Table

Assessment Activity	Weighting	Learning Outcomes	Assessment Grading Scheme	Completion Requirements
Practical	20%	1	CRA	Cumulative
Project	80%	1	CRA	Cumulative

Conditions of Assessment

You will complete this assessment during your learner managed time, however, there will be availability during the teaching sessions to discuss the requirements & your progress of this assessment. This assessment will need to be completed by **Friday, 07 June 2021 at 5:00 PM**.

Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** over all assessments in **IN607: Introductory Application Development Concepts**.

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 Classroom**. Here is the URL to the repository you will use for your submission – https://classroom.github.com/a/ww3bvOnY. The latest program files in the **main** branch will be used to run your application. 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 **IN607: Introductory Application Development Concepts**.

Instructions

Note: You are not allowed to submit code examples given to you in the teaching sessions, i.e., **Student** & **Institution**.

You will need to submit an application & documentation that meet the following requirements:

Functionality - Learning Outcomes 1 (40%)

- **API** can run & display data locally without modification.
- Three **Models** which have create, read, write & delete functionality. For deletion, if the child table has a foreign key, you must use an on cascade delete, i.e., if data from the parent table is deleted, then data from the child is automatically deleted.
- Filter, sort & page **API** data from two or more **Models** using query parameters.
- Custom validation rules & messages applied to each **Model** where appropriate, i.e., **first name** is required. Return a message if a value for **first name** is not provided.
- HTTP error handling, i.e., if a student does not exists, return a **404** HTTP status code.
- At least **20 API** tests which verify the correctness of the **API**.
- Deployed to **Heroku**. The application must be usable i.e., a user should be able to interact with your **API**.
- **API** data is stored in **MySQL** for **development** & **PostgreSQL** for **production**.
- Each database table is seeded with their own **JSON** file.

Code Elegance - Learning Outcomes 1 (45%)

- Use of intermediate variables. No method calls as arguments.
- Idiomatic use of control flow, data structures & in-built functions.
- Sufficient code modularity, i.e., each **Model** should have their own **Controller** class.
- Adheres to an **OO** architecture, i.e., classes, methods & variable names are named appropriately. Methods are assigned to the correct class.
- Efficient algorithmic approach. Use the appropriate **Eloquent** function when querying your **Models**.
- **API** resource groups named with a plural noun instead of a verb, i.e., **/api/students** not **/api/student**.
- If necessary, in-line comments explaining complex logic, i.e., an **Eloquent** function may need an additional explanation.
- Code files are code formatted.
- No dead or unused code.
- Models contain the appropriate fields, behaviours & relationships.
- Database configured for development & production environments.

Documentation & Git/GitHub Usage - Learning Outcomes 1 (15%)

- Provide the following in your repository **README.md** file:
  - URL to your **API** on **Heroku**.
  - **API** documentation on **Postman**.
  - How do you setup the environment for development, i.e., after the repository is cloned, what do I need to run the **API** locally?
  - How do you run the **API** tests?
  - How do you deploy the application to **Heroku**?
- **API** documented using **Postman**.
  - **Resource:** https://learning.postman.com/docs/publishing-your-api/documenting-your-api
- Commit messages must reflect the context of each functional requirement change. **Do not** rewrite your **Git** history. It is important that the course lecturer can see how you worked on your assessment over time.
  - **Resource:** https://freecodecamp.org/news/writing-good-commit-messages-a-practical-guide