



College of Engineering, Construction and Living Sciences

Bachelor of Information Technology

IN608: Intermediate Application Development Concepts

Level 6, Credits 15

## Django REST Framework, React & OpenTDB API

### Assessment Overview

For this assessment, you will design, develop & deploy a quiz tournament API using Django REST Framework, React, OpenTDB API & Heroku. The main purpose of this assessment is not just to build a full-stack application, rather to demonstrate an ability to decouple the back-end from the front-end by creating two separate applications which interact with each other. Marks will be allocated for functionality & best practices such as application robustness, code elegance, documentation & git usage.

With the nation-wide lockdown over, your local pub is now able to run their weekly quiz tournament onsite. The online quiz tournament application proved to be a huge success & the pub owners ask if you want to create a public API which allows users to create their own quiz tournaments.

### Assessment Table

Assessment Activity	Weighting	Learning Outcomes	Assessment Grading Scheme	Completion Requirements
Practicals	20%	1	CRA	Cumulative
Django & OpenTDB API	50%	1, 2	CRA	Cumulative
Django REST Framework, React & OpenTDB API	30%	1, 2	CRA	Cumulative

### Conditions of Assessment

This assessment will need to be completed outside by Friday, 20 November 2020 at 5pm. There will be availability during the teaching sessions to discuss the requirements & progress of this assessment.

### Pass Criteria

This assessment is criterion-referenced with a cumulative pass mark of 50%.

## Submission Details

You must submit your program files via **GitHub Classroom**. Here is the link to the repository you will be using for your submission – <https://classroom.github.com/a/sSA9csHf>.

## Authenticity

All parts of your submitted assessment must be completely your work and any references must be cited appropriately.

## Policy on Submissions, Extensions, Resubmissions & Resits

The school's process concerning **Submissions, Extensions, Resubmissions and Resits** complies with Otago Polytechnic policies. Students can view policies on the Otago Polytechnic website located at <https://www.op.ac.nz/about-us/governance-and-management/policies>.

## Extensions

Please familiarise yourself with the assessment due dates. If you need an extension, please contact your 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

Students may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are completed within a short time frame (usually no more than 5 working days) and usually must be completed within the timing of the course to which the assessment relates. Resubmissions will be available to students who have made a genuine attempt at the first assessment opportunity. The maximum grade awarded for resubmission will be C-.

## Learning Outcomes

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

1. Demonstrate sound programming by following design patterns and best practices.
2. Design and implement full-stack applications using industry relevant programming languages.

## Instructions

This is a project-based assessment. Within your project you will need to implement the following:

### Functionality & Robustness - Learning Outcomes 1, 2

- Dependencies are correctly managed using **Pipenv/Pipfile** & **npm/package.json**.
- Deploy both applications as one to Heroku.
- Django REST Framework application (back-end):
  - Create model classes which store the following quiz tournament data: creator, name, category, difficulty, question, correct answer & incorrect answers.
  - Dynamically fetch **all** categories from the following URL [https://opentdb.com/api\\_category.php](https://opentdb.com/api_category.php) & store as choices in the appropriate model class.
  - For each model class:
    - \* Create a serializer class.
    - \* Create an **APIView** class or **api.view** function which reads, inserts, updates & deletes quiz tournament data. **Hint:** use the **GET, POST, PUT & DELETE** HTTP methods.
      - **Resource:** [Django REST Framework Views](#)
  - Quiz tournament data is persistently stored in **Heroku PostgreSQL**.
    - \* **Resource:** [Heroku PostgreSQL](#)
  - Unit tests cover models, views & OpenTDB API.
- React application (front-end):
  - Request quiz tournament data via **Django REST Framework** end-points using **Axios**. Data includes creator, name, category, difficulty, question, correct answer & incorrect answers.
  - Create a new quiz tournament. Display a form in a modal. Form fields include creator, name, category & difficulty. You **must** use the **select** input type for categories & difficulties.
  - Incorrect formatted form field values handled gracefully using validation error messages, for example, creator form field is blank.
  - View quiz tournaments in a table. Table data includes creator, name, category, difficulty. Paginate quiz tournament data across several pages with **Next/Previous** links.
  - Update a quiz tournament. Display a form in a modal. Form fields include creator, name, category & difficulty.
  - Delete a quiz tournament. Prompt the user for deletion.
  - View a quiz tournament's question, correct answer & incorrect answers when a quiz tournament name is clicked.
  - Visually attractive user-interface with a coherent graphical theme & style using **Reactstrap**.
    - \* **Resource:** [Reactstrap](#)
    - \* **Resource:** [React Buildpack](#)
  - Unit tests cover components.
  - End-to-end tests cover creating, updating & deleting a quiz tournament & viewing a quiz tournament's questions.

## Documentation & Git Usage - Learning Outcome 1

- Provide the following information in the repository **README** file:
  - How do you set up the environment for development, i.e., after the repository is cloned, what do I need to start coding?
  - How to run tests.
  - How to deploy the applications.
- At least 10 feature branches excluding the **main** branch.
  - Your branches must be prefix with **feature**, for example, **feature-`<name of functional requirement>`**.
  - For each branch, merge your own pull request to the **main** branch.
- Commit messages must reflect the context of each functional requirement change.