College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
IN608: Intermediate Application Development Concepts
Level 6, Credits 15

Django & OpenTDB API

Assessment Overview

For this assessment, you will design, develop & deploy a quiz tournament application using Django, the OpenTDB API & Heroku. The main purpose of this assessment is not just to build a full-stack application, rather to demonstrate sound programming by following the Model, View, Template architectural design pattern & SOLID principles. Marks will be allocated for functionality & best practices such as application robustness, code elegance, documentation & git usage.

Due to a nation-wide lockdown, your local pub is no longer able to run their weekly quiz tournament onsite. Your local pub owners know you are an IT student & ask if you want create an online quiz tournament application for them. The pub owners want an application that allows users to signup, login, participate in various quiz tournaments & keep track of scores so that they can give away prizes at the end of each quiz tournament. In addition to the basic features, you suggest a variety of features which will enhance the quiz experience.

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 by Friday, 11 October 2020 at 5pm.

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/uQeihzqX.

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

Functionality & Robustness - Learning Outcomes 1, 2

- Dependencies are correctly managed using **Pipenv** & **Pipfile**.
- User (applies to both admin & player users) features:
  - Log into the application using an email/username & password.
  - Logout of the application.
  - Incorrect formatted input values handled gracefully using validation error messages, for example, username input field is blank.
  - Update user profile using an HTML form. Updatable fields include username, first name, last name, email & profile picture.
  - View scores for each quiz tournament. Display the total players participated, average score, number of likes, player's name, completed date & player's score. Sort the scores in descending order by player's score.
  - Request a password reset if user's password is forgotten. If an email exists, the user will be emailed instructions for resetting their password.
- Admin user specific features:
  - Create a new admin user using **python manage.py createsuperuser**. Fields must be username, first name, last name & email. Email must be unique. By default, the admin user's profile picture is set to a placeholder image using a signal. For ease of marking, please create an admin user with the username: **admin** & password: **P@ssw0rd123**
    - **Resource:** Django Signals
  - Create a new quiz tournament. Fields include name, category, difficulty, start date & end date.
  - A quiz tournament consists of 10 questions dynamically fetched from the **OpenTDB API**. Categories must be dynamically fetched from the following URL https://opentdb.com/api_category.php. Choices can not be hardcoded. Questions can be multi-choice, true/false or a mixture of both. Handle gracefully if there are no questions in the response.
  - Receive an email notification when a new quiz tournament has been created.
    - **Resource:** Django Sending Email
  - View all quiz tournaments in an HTML table.
  - Update a quiz tournament. Updatable fields include name, start date & end date.
  - Delete a quiz tournament. Prompt the user for deletion.
  - View the number of likes for each quiz tournament. Display this information in an appropriate chart, i.e., bar or pie chart using **Chart.js**.
    - **Resource:** Chart.js
  - For each quiz tournament, download a PDF containing a formatted table of scores using **ReportLab**. Only display the download button if a quiz tournament has one or more scores.
    - **Resource:** Django Outputting PDFs
- Player user specific features:
  - Create a new player user using Django's user authentication system & an HTML form. Fields & constraints are the same as the admin user.
  - Display ongoing, upcoming, past & participated quiz tournaments. Paginate data across several pages with **Next/Previous** links.
    - **Resource:** Django Pagination
  - Player user should not be able to participate in upcoming, past or participated quiz tournaments.

- Participate in ongoing quiz tournaments. All player users that enter the same quiz tournament will be presented with the same 10 questions.
- Questions must presented on separate pages.
- Display appropriate feedback for correct & incorrect answers. If a question is answered incorrectly, display the correct answer.
  * **Resource:** Django Messages
- Player user should be able to leave an ongoing quiz tournament at any time & return to the last presented question. For example, a player user answers the first five questions then logs out of the application. The player user returns to the quiz tournament three hours later & is presented with question six.
- When the player user's quiz tournament is completed, display their score out of 10.
- Like & unlike quiz tournaments.

• Display error pages for the following HTTP status codes:
  - 400 - Bad Request
  - 403 - Forbidden
  - 404 - Not Found
  - 500 - Internal Server Error
  - **Resource:** Django Error Views

• Visually attractive & responsive user-interface with a coherent graphical theme & style using **Bootstrap** or **TailWind CSS**.

•
  - **Resources:**
    * Bootstrap
    * TailWind CSS

• Application deployed to **Heroku** with **Gunicorn**.
  - **Resources:**
    * Deploying Python and Django Apps on Heroku
    * Gunicorn

• Data is persistently stored in **Heroku PostgreSQL**.
  - **Resource:** Heroku PostgreSQL

• Unit & integration tests cover models, views, forms & API.

• End-to-end tests cover signup, login, logout, creating, updating & deleting a quiz tournament & participating in a quiz tournament.

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 application.

• At least 15 feature branches excluding the **master** 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 **master** branch.

• Commit messages must reflect the context of each functional requirement change.

Assessment 01: Django & OpenTDB API Assessment Rubric

	10-9	**8-7**	**6-5**	**4-0**
Functionality & Robustness	Application thoroughly demonstrates functionality & robustness. Unit & integration tests thoroughly demonstrate coverage of models, views , forms & API. End-to-end tests thoroughly demonstrate coverage of signup, login, logout, creating, updating & deleting a quiz tournament & participating in a quiz tournament.	Application mostly demonstrates functionality & robustness. Unit & integration tests mostly demonstrate coverage of models, views , forms & API. End-to-end tests mostly demonstrate coverage of signup, login, logout, creating, updating & deleting a quiz tournament & participating in a quiz tournament.	Application demonstrates some functionality & robustness. Unit & integration tests demonstrate some coverage of models, views , forms & API. End-to-end tests demonstrate some coverage of signup, login, logout, creating, updating & deleting a quiz tournament & participating in a quiz tournament.	Application does not or does not fully demonstrate functionality & robustness. Unit & integration tests do not or do not fully demonstrate coverage of models, views , forms & API. End-to-end tests do not or do not fully demonstrate coverage of signup, login, logout, creating, updating & deleting a quiz tournament & participating in a quiz tournament.

Code Elegance	Application code thoroughly demonstrates code elegance on the following: • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, i.e., code adheres to SOLID principles. • Adhere to an OO & Model, View, Template architecture. • Adhere to pycodestyle (formally PEP8) style guide. • Efficient algorithmic approach. • Handling of API response codes. • Handling of HTML entities. • Header comments explain each class & method. • In-line comments explain complex logic. • Well-designed models containing fields & behaviours. • Flexible URL design. Not coupled to the underlying code.	Application code mostly demonstrates code elegance on the following: • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, i.e., code adheres to SOLID principles. • Adhere to an OO & Model, View, Template architecture. • Adhere to pycodestyle (formally PEP8) style guide. • Efficient algorithmic approach. • Handling of API response codes. • Handling of HTML entities. • Header comments explain each class & method. • In-line comments explain complex logic. • Well-designed models containing fields & behaviours. • Flexible URL design. Not coupled to the underlying code.	Application code demonstrates some code elegance on the following: • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, i.e., code adheres to SOLID principles. • Adhere to an OO & Model, View, Template architecture. • Adhere to pycodestyle (formally PEP8) style guide. • Efficient algorithmic approach. • Handling of API response codes. • Handling of HTML entities. • Header comments explain each class & method. • In-line comments explain complex logic. • Well-designed models containing fields & behaviours. • Flexible URL design. Not coupled to the underlying code.	Application code does not does not fully demonstrate code elegance on the following: • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, i.e., code adheres to SOLID principles. • Adhere to an OO & Model, View, Template architecture. • Adhere to pycodestyle (formally PEP8) style guide. • Efficient algorithmic approach. • Handling of API response codes. • Handling of HTML entities. • Header comments explain each class & method. • In-line comments explain complex logic. • Well-designed models containing fields & behaviours. • Flexible URL design. Not coupled to the underlying code.

Documentation & Git Usage	README thoroughly describes how to set the environment for development, run tests & deploy the application. Git branches thoroughly named with convention & contain the correct code relating to the functional requirement. Git commit messages thoroughly reflect the functional requirement changes.	README mostly describes how to set the environment for development, run tests & deploy the application. Git branches mostly named with convention & contain the correct code relating to the functional requirement. Git commit messages mostly reflect the functional requirement changes.	README briefly describes how to set the environment for development, run tests & deploy the application. Some git branches named with convention & contain the correct code relating to the functional requirement. Some git commit messages reflect the functional requirement changes.	README does not or does not fully describe how to set the environment for development, run tests & deploy the application. Git branches are not or are not fully named with convention & do not or do not fully contain the correct code relating to the functional requirement. Git commit messages do not or do not fully reflect the functional requirement changes.

Django & OpenTDB API Marking Cover Sheet

Name:

Date:

Learner ID:

Assessor's Name:

Assessor's Signature:

Criteria	Out Of	Weighting	Final Result
Functionality & Robustness	10	45	
Code Elegance	10	45	
Documentation & Git Usage	10	10	
Final Result			/100
This assessment is worth 50% of the final mark for the Intermediate Application Development course.			

Feedback: