College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
IN710: Object-Oriented Systems Development
Level 7, Credits 15

Assessment 02: MVT (Model, View, Template)

Assessment Overview

For this assessment, you will use Django with a text editor, i.e, Vim, Visual Studio Code, etc to build a trivia quiz application that allows users to participate in **tournaments**. As well as implementing the core functionality, you will be required to **independently** research & implement four components. In addition, marks will also be given for code elegance, robustness & git usage.

Assessment Table

Assessment Activity	Weighting	Learning Outcomes	Assessment Grading Scheme	Completion Requirements
Exams 1-5	30%	1, 2	CRA	Cumulative
Practicals	20%	2, 3	CRA	Cumulative
Design Patterns	25%	2, 3	CRA	Cumulative
MVT	25%	2, 3	CRA	Cumulative

Conditions of Assessment

This assessment will need to be completed by Friday, 19 June 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/MKLNTR0q. For ease of marking, please submit the marking sheet with your name & student id number via **Microsoft Teams** under the **Assignments** tab.

Group Contribution

All git commit messages must identify which member(s) participated in the associated work session. Proportional contribution will be determined by inspection of the commit logs. If the commit logs show evidence of significantly uneven contribution proportion, the lecturer may choose to adjust the mark of the lesser contributor downward by an amount derived from the individual contributions.

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. Discuss theoretical and pragmatic issues surrounding design and implementation of enterprise software systems.
2. Analyse a problem statement for a complex software system and design an appropriate class architecture for the problem solution.
3. Design and implement components of large software systems following industry standard software engineering methodologies and producing industry-quality code.

Instructions

Application Requirements - Learning Outcomes 2, 3

The trivia quiz application **must** have the following functional requirements:

- System:
  - Run without modification in Google Chrome or Mozilla Firefox.
  - Correct management of application dependencies. Use Pipenv & Pipfile.
- Features:
  - User features - applies to both admin & player users:
    - Login using a username & password.
    - Incorrect formatted input values handled gracefully using validation error messages.
    - View highscores for each tournament. Display total taken, the player's name, completion date, player's score & average score. Descending order by player's score.
  - Admin specific features:
    - Create a new admin using the Django's admin interface. For ease of marking, please provide an admin with the user name **admin** & password **P@ssw0rd123**
    - Create a tournament. A tournament **must** have a name, category, difficulty, start date & end date. Do not use Django's admin interface. Tournaments must be created via an HTML template.
    - A tournament consists of 10 questions fetched dynamically fetched from the OpenTDB API. This API provides a list of categories & difficulties.
    - View & delete tournaments. Display the tournament's questions.
    - **Research:** API endpoints for each model using Django REST Framework or Swagger. **Note:** player **should not** have access.
  - Player specific features:
    - **Research:** Create a new player using the Django's authentication system.
    - Display ongoing, upcoming, past & taken tournaments.
    - Participate in ongoing tournaments. All players that enter the same tournament will be presented with the same 10 questions. One attempt per player.
    - Player should not be able to immediately participate in upcoming, past & taken tournaments.
    - Questions presented separately. Do not display multiple questions on a single screen.
    - Provided feedback after each answer is submitted.
    - Allow the player to decide when to proceed to the next question.
    - When the player's tournament attempt is finished, display their score out of 10.
- User-Interface:
  - Visually attractive user-interface with a coherent graphical theme & style. Application does not need to be mobile responsive.
  - Clear & well-structured navigation.
  - Unknown URLs handled correctly.
- Database Management:
  - Data persistently stored in MariaDB or other database management systems. This is not limited to relational databases. SQLite will not be accepted.
  - Custom Django admin command that populates at least five players.
- Deployment:
  - **Research:** Application deployed on Heroku or PythonAnywhere.
  - Provide a URL to the deployed application in the repository README.md

Automation Testing - Learning Outcomes 2, 3

- Coverage of models, views & APIs via unit & integration testing.
- **Research:** At least five end-to-end tests using Selenium WebDriver.

Git Usage - Learning Outcomes 2, 3

The language translator repository must have the following git requirements:

- At least five feature branches excluding master.
- Commit messages reflect the context of each functional requirement change.

Additional Resources

- OpenTDB API - https://opentdb.com/
- Django REST Framework - https://www.django-rest-framework.org/
- Swagger Docs - https://swagger.io/
- Deploying Django Apps on Heroku - https://devcenter.heroku.com/articles/deploying-python
- Deploying Django Apps on PythonAnywhere - https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/
- Coverage Module - https://coverage.readthedocs.io/en/coverage-5.0.4/
- Class-Based Views - https://docs.djangoproject.com/en/3.0/topics/class-based-views/intro/
- Performance & Optimization https://docs.djangoproject.com/en/3.0/topics/performance/

IN710 Object-Oriented Systems Development
Semester 1, 2020

Assessment 02: MVT (Model, View, Template)
Assessment Rubric

	10-9	8-7	6-5	4-0
Functionality & Robustness	Application thoroughly demonstrates functionality & robustness on the following: • Run without modification in Google Chrome or Mozilla Firefox. • Correct management of application dependencies. • Login using a username & password. • Incorrect formatted input values handled gracefully using validation error messages. • View highscores for each tournament. • Create a new admin using the Django's admin interface. • Create a tournament. • A tournament consists of 10 questions fetched dynamically fetched from the OpenTDB API. • View & delete tournaments. Display the tournament's questions. • API endpoints for each model using Django REST Framework or Swagger.	Application mostly demonstrates functionality & robustness on the following: • Run without modification in Google Chrome or Mozilla Firefox. • Correct management of application dependencies. • Login using a username & password. • Incorrect formatted input values handled gracefully using validation error messages. • View highscores for each tournament. • Create a new admin using the Django's admin interface. • Create a tournament. • A tournament consists of 10 questions fetched dynamically fetched from the OpenTDB API. • View & delete tournaments. Display the tournament's questions. • API endpoints for each model using Django REST Framework or Swagger.	Application demonstrates some functionality & robustness on the following: • Run without modification in Google Chrome or Mozilla Firefox. • Correct management of application dependencies. • Login using a username & password. • Incorrect formatted input values handled gracefully using validation error messages. • View highscores for each tournament. • Create a new admin using the Django's admin interface. • Create a tournament. • A tournament consists of 10 questions fetched dynamically fetched from the OpenTDB API. • View & delete tournaments. Display the tournament's questions. • API endpoints for each model using Django REST Framework or Swagger.	Application does not or does not fully demonstrate functionality & robustness on the following: • Run without modification in Google Chrome or Mozilla Firefox. • Correct management of application dependencies. • Login using a username & password. • Incorrect formatted input values handled gracefully using validation error messages. • View highscores for each tournament. • Create a new admin using the Django's admin interface. • Create a tournament. • A tournament consists of 10 questions fetched dynamically fetched from the OpenTDB API. • View & delete tournaments. Display the tournament's questions. • API endpoints for each model using Django REST Framework or Swagger.

	• Create a new player using the Django's authentication system. • Display ongoing, upcoming, past & taken tournaments. • Participate in ongoing tournaments. • Player should not be able to participate in upcoming, past & taken tournaments. • Questions presented separately. • Provided feedback after each answer is submitted. • Allow the player to decide when to proceed to the next question. • When the player's tournament attempt is finished, display their score out of 10. • Visually attractive user-interface with a coherent graphical theme and style. • Clear & well-structured navigation. • Unknown URLs handled correctly. • Data persistently stored in MariaDB or other database management systems. • Application deployed on Heroku or PythonAnywhere. • Custom Django admin command that populates at least five players. • URL to the deployed application in the repository	• Create a new player using the Django's authentication system. • Display ongoing, upcoming, past & taken tournaments. • Participate in ongoing tournaments. • Player should not be able to participate in upcoming, past & taken tournaments. • Questions presented separately. • Provided feedback after each answer is submitted. • Allow the player to decide when to proceed to the next question. • When the player's tournament attempt is finished, display their score out of 10. • Visually attractive user-interface with a coherent graphical theme and style. • Clear & well-structured navigation. • Unknown URLs handled correctly. • Data persistently stored in MariaDB or other database management systems. • Application deployed on Heroku or PythonAnywhere. • Custom Django admin command that populates at least five players. • URL to the deployed application in the repository	• Create a new player using the Django's authentication system. • Display ongoing, upcoming, past & taken tournaments. • Participate in ongoing tournaments. • Player should not be able to participate in upcoming, past & taken tournaments. • Questions presented separately. • Provided feedback after each answer is submitted. • Allow the player to decide when to proceed to the next question. • When the player's tournament attempt is finished, display their score out of 10. • Visually attractive user-interface with a coherent graphical theme and style. • Clear & well-structured navigation. • Unknown URLs handled correctly. • Data persistently stored in MariaDB or other database management systems. • Application deployed on Heroku or PythonAnywhere. • Custom Django admin command that populates at least five players. • URL to the deployed application in the repository	• Create a new player using the Django's authentication system. • Display ongoing, upcoming, past & taken tournaments. • Participate in ongoing tournaments. • Player should not be able to participate in upcoming, past & taken tournaments. • Questions presented separately. • Provided feedback after each answer is submitted. • Allow the player to decide when to proceed to the next question. • When the player's tournament attempt is finished, display their score out of 10. • Visually attractive user-interface with a coherent graphical theme and style. • Clear & well-structured navigation. • Unknown URLs handled correctly. • Data persistently stored in MariaDB or other database management systems. • Application deployed on Heroku or PythonAnywhere. • Custom Django admin command that populates at least five players. • URL to the deployed application in the repository

	README.md • Efficient performance using optimization techniques.	README.md • Efficient performance using optimization techniques.	README.md • Efficient performance using optimization techniques.	README.md • Efficient performance using optimization techniques.
Automation Testing	Unit & integration tests thoroughly demonstrate coverage of models, views & APIs. End-to-end tests thoroughly demonstrate coverage of user-interface.	Unit & integration tests mostly demonstrate coverage of models, views & APIs. End-to-end tests mostly demonstrate coverage of the application's user-interface.	Unit & integration tests demonstrate some coverage of models, views & APIs. End-to-end tests demonstrate some coverage of the application's user-interface.	Unit & integration tests do or not fully demonstrate coverage of models, views & APIs. End-to-end tests do or do not fully demonstrate coverage of the application's user-interface.
Code Elegance	Application thoroughly demonstrates code elegance on the following: • Classes adhere to a general OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Code adhere to pycodestyle style guide. • Correct use of setup & teardown in test case classes. • Appropriate use of inheritance. • Header comments appropriately explain the input, output & computational logic of each class & method.	Application mostly demonstrates code elegance on the following: • Classes adhere to a general OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Code adhere to pycodestyle style guide. • Correct use of setup & teardown in test case classes. • Appropriate use of inheritance. • Header comments appropriately explain the input, output & computational logic of each class & method.	Application demonstrates some code elegance on the following: • Classes adhere to a general OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Code adhere to pycodestyle style guide. • Correct use of setup & teardown in test case classes. • Appropriate use of inheritance. • Header comments appropriately explain the input, output & computational logic of each class & method.	Application does not or does not fully demonstrate code elegance on the following: • Classes adhere to a general OO architecture, e.g., classes, methods, concise naming & methods assigned to the correct classes. • Correct use of intermediate variables, e.g., no method calls as arguments. • Idiomatic use of control flow, data structures & other in-built functions. • Sufficient modularity, e.g., code adheres to the KISS, SOLID & YAGNI principles. • Efficient algorithmic approach. • Code adhere to pycodestyle style guide. • Correct use of setup & teardown in test case classes. • Appropriate use of inheritance. • Header comments appropriately explain the input, output & computational

	• Inline comments appropriately explain the logic of construct of each computational statement. • Well-designed models containing essential fields & behaviour. • Flexible URL design. Not coupled to the underlying code.	• Inline comments appropriately explain the logic of construct of each computational statement. • Well-designed models containing essential fields & behaviour. • Flexible URL design. Not coupled to the underlying code.	• Inline comments appropriately explain the logic of construct of each computational statement. • Well-designed models containing essential fields & behaviour. • Flexible URL design. Not coupled to the underlying code.	logic of each class & method. • Inline comments appropriately explain the logic of construct of each computational statement. • Well-designed models containing essential fields & behaviour. • Flexible URL design. Not coupled to the underlying code.
Git Usage	Git commit messages thoroughly reflect the functional requirement changes. Git branches thoroughly named & describe the context of the functional requirements.	Git commit messages mostly reflect the functional requirement changes. Git branches mostly named & describe the context of the functional requirements.	Git commit messages reflect some of the functional requirement changes. Git branches named & describe some of the context of the functional requirements.	Git commit messages do not or do not fully reflect the context of each solution. Git branches incorrectly named & do not or do not fully describe the context of the functional requirements.

Marking Cover Sheet

OTAGO POLYTECHNIC
Te Kura Matatini ki Otago

Assessment 02: MVT (Model, View, Template)

IN710: Object-Oriented Systems Development
Level 7, Credits 15
Bachelor of Information Technology

Name: _____ Date: _____

Learner ID: _____

Assessor's Name: _____

Assessor's Signature: _____

Criteria	Out Of	Weighting	Final Result
Functionality & Robustness	**10**	**40**	
Automation Testing	**10**	**30**	
Code Elegance	**10**	**20**	
Git Usage	**10**	**10**	
		Final Result	**/100**
This assessment is worth 25% of the final mark for the Object-Oriented Systems Development course.			