# **Assessment 02: Django REST Framework, React & OpenTDB API Assessment Rubric**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **10-9** | **8-7** | **6-5** | **4-0** |
| **Functionality & Robustness** | Application thoroughly demonstrates functionality & robustness.  Unit tests thoroughly demonstrate coverage of components, models, views & OpenTDB API.  End-to-end tests thoroughly demonstrate coverage of creating, updating & deleting a quiz tournament & viewing a quiz tournament’s question, correct answer & incorrect answers. | Application mostly demonstrates functionality & robustness.  Unit tests mostly demonstrate coverage of components, models, views & OpenTDB API.  End-to-end tests mostly demonstrate coverage of creating, updating & deleting a quiz tournament & viewing a quiz tournament’s question, correct answer & incorrect answers. | Application demonstrates some functionality & robustness.  Unit tests demonstrate some coverage of components, models, views & OpenTDB API.  End-to-end tests demonstrate some coverage of creating, updating & deleting a quiz tournament & viewing a quiz tournament’s question, correct answer & incorrect answers. | Application does not or does not fully demonstrate functionality & robustness.  Unit tests thoroughly demonstrate coverage of components, models, views & OpenTDB API.  End-to-end tests does not or does not fully demonstrate coverage of creating, updating & deleting a quiz tournament & viewing a quiz tournament’s question, correct answer & incorrect answers. |
| **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, UI split into independent, reusable pieces. * Adhere to client-server architecture. * Components written as function components, not class components. * 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. | 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, UI split into independent, reusable pieces. * Adhere to client-server architecture. * Components written as function components, not class components. * 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. | 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, UI split into independent, reusable pieces. * Adhere to client-server architecture. * Components written as function components, not class components. * 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. | 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, UI split into independent, reusable pieces. * Adhere to client-server architecture. * Components written as function components, not class components. * 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. |
| **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. |

# **Assessment 02: Django REST Framework, React & OpenTDB API Marking Cover Sheet**

Name:

Date:

Learner ID:

Assessor’s Name:

Assessor’s Signature:

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Out Of** | **Weighting** | **Final Result** |
| Functionality & Robustness | 10 | 40 |  |
| Code Elegance | 10 | 50 |  |
| Documentation & Git Usage | 10 | 10 |  |
| **Final Result** | | | /100 |
| **This assessment is worth 25% of the final mark for the Intermediate Application Development course.** | | | |

Feedback: