| University of Central Lancashire logo  **School of Psychology and Computer Science** | UCLan Coursework Assessment Brief | | 2021-2022 |
| --- | --- | --- | --- |
| Module Title: Distributed Enterprise Applications | | |
| Module Code: CO3409 | | Level 6 |
| **Development of a RESTful Web Service** | This assessment is worth **50**% of the overall module mark | |

Deadline

The Deadline is **23:59** on the **28/03/22.**

Overview

This assignment requires you to design, build and test a RESTful web service using some of the ASP.NET technologies you learned in Semester 1.

Background

Your web service should provide an API for an interactive game. The game is intended to be played by a client app developed for a mobile device, **however you are not expected to provide a client app**. Instead you will test your API by other means (e.g. Postman or equivalent).

The game API should approximate the quiz/treasure hunt game you may or may not remember from the Four Week Challenge (which most students completed in their first year). Full details of how it should work are given below. The API you may have used for that game was not RESTful, and the server-side technology is long obsolete. Your job is to build an up-to-date version of the Server.

A sample set of quiz questions and answers will be provided as an appendix. You are not obliged to use them.

Deliverables

You need to **upload the following to Blackboard** by the deadline:

* An Excel spreadsheet containing a design for your Restful API – For high marks this will be Level 3on the Richardson maturity model (i.e. HATEOAS)
  + Use the spreadsheet from the exercise in week 8 as the template for your design.
  + Your HTTP response bodies should be JSON.
* An ASP.NET server MVC implementation, written in C#
* A Word document containing:
  + A class diagram and database schema (E-R diagram) for your server
    - You may choose to implement your design following a code-first or database first approach.
    - You should use EF-Core and LINQ to connect to your database.
  + Documentary evidence that your server has been tested and works.
    - A suitable test strategy for determining your web service meets the requirements. This should include unit tests, and system tests.
    - Evidence (which may include screen-shots and messages (requests/responses)) of your actual testing, which should cover at least the following:
      * CRUD operations
      * Error handling
      * Persistence
      * Unit testing using a suitable framework
* A short video demonstration of your API being called successfully (either as a .mp4 file or a link to e.g. YouTube)

Server Requirements

The server provides a game that can be played in two modes – a simple multiple choice quiz, or a more complex treasure hunt which incorporates different types of questions and is location aware.

To get a first class mark, you will need to implement at least some of the features of the more complex treasure hunt game.

The following information is pre-loaded into the database:

* A list of question categories.
* For each category, a list of valid numbers of questions that can be requested for a game.
* For each category, a number of questions.
* Simple version (Quiz mode)
  + Each question has associated with it 4 possible answers, labelled A – D, and an indication of which answer is correct.
* Complex version (Treasure Hunt mode)
  + Each question can be one of 3 different types:
    - **multiple choice,** as described above;
    - **text,** where the answer is a word or phrase which must match exactly the given answer, rather than the single letter A - D);
    - **picture**, which requires the uploading of a compressed photograph to the database.
  + Each complex question has a latitude and longitude representing the geolocation of the correct answer.
  + Each question has a text clue – a hint as to the answer.

Basic Game

* When a player starts a game, they must first select a category and a valid number of questions from that category. They must also give a unique name to identify themselves, and an app ID.
* The game server will select a set of questions at random from those available in the chosen category, and present them in a specific order to the player, keeping track of the player’s current question.
* The player can ask for their current question at any time, and will be given the question text and possible answers (where applicable).
* The game will only move on to presenting the next question when the current question has either been answered correctly, OR been passed.
* For multiple choice questions, the answers should be the text “A”, “B”, “C”, “D” or “PASS”.
* Other “live” data may need to be stored in the database as appropriate…

For the advanced game:

* For text questions, the answer can be any text string, including “PASS”
* For picture questions, no text answer is required, although the answer “PASS” may be submitted.
* The client should be able to update the server with its latitude and longitude at any point.
* If a question has a latitude and longitude then the clients reported location must be within a given distance (20m) of the questions location AND the answer must be correct in order for the question to be deemed to have been answered correctly.
* The client may ask for the text hint for the current question at any time.
* The client may also ask for a location hint, which will give them the latitude and longitude of the answer (this is to allow the client app to have something like a magic compass which points to the answer based on the device location and orientation).
* When the client answers the final question, they will be made aware that the quiz is over.
* If the client asks for the question, a hint, or attempts to answer a question once the quiz has finished, they will be reminded that the quiz is over.
* The server should keep track of the player’s score during the quiz. A correctly answered question should attract 2 points. If a player asks for either type of hint, they only get one point when they answer the question correctly. If they ask for both hints, they get no points. If they pass on a question, they get no points.
* Final scores should be persistent, and the server should record the category and number of questions along with the actual score.
* The client should be able to ask for their current score at any time.
* After the quiz, the client should be able to see a high scores table, or start another quiz.

Marking Scheme

A fully working, comprehensively tested implementation of the “basic” game requirements, where all the required documentation has been submitted and is complete, will attract an upper second class mark.

For a first class mark, in addition to the above, some or all of the “advanced” game features must have been successfully implemented, and also tested.

RESTFul API design documentation – **25 marks**

* Appropriate mapping of CRUD operations onto HTTP verbs (**5**)
* Appropriate naming convention (see Kenneth Lange’s blog – week 7) (**5**)
* HATEOAS responses (**5**)
* Sensible operations covering the basic game requirements (**5**)
* Operations covering the advanced game features (**5**)

Database design / Class diagram (will be the same if a code-first strategy is followed) – **10 marks**

For a basic version of the game, 5 marks are available. Additional marks are available if one or more advanced features have been implemented.

Testing Documentation – **25 marks**

* Appropriate overall strategy (**4**)
* Comprehensive unit tests with evidence of them being carried out (**8**)\*
* Comprehensive system tests with evidence of them being carried out (e.g. with Postman) (**8**)\*
* Evidence that tata is being stored in the database appropriately (**5**)

\*For more than 5 marks in these sections, the advanced game features must also be tested.

**Note** – comprehensive testing should include testing of error conditions. Basic tests only will attract no more than half marks.

Implementation – **40 marks**

* Implementation conforms to MVC architecture (**5**)
* Code commented appropriately (**5**)
* Working basic game functionality (as reported in testing section) (**20**)
  + Listing question categories and available numbers of questions
  + Starting a quiz and obtaining questions
  + Answering/passing questions
  + Finishing a quiz appropriately
  + Keeping score and high scores
* Additional game features (**10**)
  + Text questions
  + Picture questions
  + Location awareness
  + hints