Te Hoe Hōkai Pakihi
Department of Business and Digital Technologies
Bachelor of Information and Communication Technologies
Graduate Diploma in Information and Communication Technologies

Best Programming Practices (Web and Mobile Development)

BCDE211

Assessment 2 Practical Project

Semester 1, 2022

Due date: Friday, 08 April 2022

Time: 5pm

TOTAL MARKS: **60**

Student Name/ID .....

Ara Institute of Canterbury and its division members reserve the right to use electronic means to detect and help prevent plagiarism. Students agree that when submitting this assignment, it may be subject to submission for textual similarity review to Turnitin.com.

Submissions received late will be subject to a penalty of 10% of the student's mark per working day.

This assessment is worth 25% of the total marks for this course. To pass this course, students must gain an average of at least 50% across all assessments, and gain at least 50% in Assessment 3.

This paper has four (4) pages including the cover sheet.

Ara Institute of Canterbury
Ara rau, taumata rau

This assessment relates to the following Learning Outcome(s):

- Demonstrate ability to implement a prototype system.

Overview

You are to create the first part of a program that will be of use to a cycling enthusiast. This first iteration of the program must be written in JavaScript / ECMAScript 2021 (not required to use TypeScript). Naturally you will be expected to use good design, programming and testing practices.

There are many existing cycling apps[1]. You are encouraged to explore them to form your ideas about the possible features of your app, and then design, code and test the <<MODEL>> of it.

Outside of scope

- Creating a user interface <<VIEW>>
- Creating a <<CONTROLLER>>
- Connecting to hardware specific APIs

Tasks

You need to analyse, design, write, and test a <<MODEL>> that will be able to be used later in association with a <<VIEW>> in Assessment 3.

It means you need to:

1. Decide on the features and GUI that your program will have based on the assessment instructions
2. Develop appropriate UML diagrams (including at least a class diagram) for your <<MODEL>> and a wireframe for your GUI proposed
3. Develop appropriate unit tests to test the correctness of your <<MODEL>> code
4. Code and test your <<MODEL>>

You **MAY** work in groups for creating the code for this assessment but must make **INDIVIDUAL** presentations. Clearly comment the source of all code that is not your own in your source code files and presentation slides. In your presentation you may use code provided by others in the class but **will only be marked on code you have written**. See the marking rubric for details of how the amount of code you use from others will impact on your mark.

Students should show the progress of their assessment work to the course tutor every week and get formative feedback.

[1] For example, https://www.bikeradar.com/advice/buyers-guides/best-cycling-apps/ and https://www.cyclingweekly.com/group-tests/best-cycling-apps-143222

Submission

Students must zip and submit their work including the items listed below into the corresponding drop box on the course Moodle site by the deadline indicated.

1. A 10-minute MAXIMUM (optionally narrated) PowerPoint slide presentation explaining how much of the <<MODEL>> you have implemented and tested based on your analysis and design
2. A digital copy of the project code including unit testing code
3. A digital copy of all your analysis and design documentation used

The presentation should cover the followings.

1. The expected (i.e., self-marking) marks for each task you expect to get marks for. Please refer to the marking rubric.

2. Analysis and design of the GUI you intend to implement in association with your <<MODEL>>. This includes at least:
   - Wireframe

3. List of MUST-have application features which are required to achieve requirements listed below.
   1. Create a whole that acts as a container and gateway of accessing to parts
   2. Add a part
   3. Sort parts
   4. Filter parts
   5. Delete a selected part
   6. Save all parts to LocalStorage
   7. Load all parts from LocalStorage
   8. Update/edit a part
   9. Discard /revert edits to a part
   10. Validate inputs
   11. A calculation within a part
   12. A calculation across many parts
   13. Provide default values
   14. Find a part given a search criterion
   15. Get all parts

4. Analysis and design of the <<MODEL>> classes you have implemented. This includes at least:
   - UML 2 Class diagram(s)

5. How many of the MUST-have features of the <<MODEL>> have been implemented?
   - Use snapshots to clearly indicate which block of code is behind which MUST-have feature of the <<Model>> you have implemented

6. How many of the unit tests which prove the correctness of the <<MODEL>> can your code pass?
   - Use snapshots to clearly indicate which unit test is to test which MUST-have feature of your <<MODEL>>
   - Use snapshots to clearly indicate which unit test can pass against your <<MODEL>> implemented

Marking rubric

Element	0	1	2	3	4	5	Weight
Wireframe	Not attempted	Layout	Layout with elements	Layout with elements and containers	Layout with elements and containers and ids	Layout with elements and containers and ids and CSS classes	* 1
MUST-have feature list (Feature coverage)	< 3 relevant and reasonable features	3-5 relevant and reasonable features	6-8 relevant and reasonable features	9-11 relevant and reasonable features	12-14 relevant and reasonable features	15+ relevant and reasonable features	* 1
Class diagram	Not attempted	Analysis level diagram	Design level diagram of data structures	Design level diagram of structures and attributes	Design level diagram of structures, attributes and methods	Design level diagram of structures, attributes, methods, and interfaces	* 2
% Implemented requirements	< 3 relevant and reasonable features from the listed MUST-have feature list have been implemented correctly.	3-5 relevant and reasonable features from the listed MUST-have feature list have been implemented correctly.	6-8 relevant and reasonable features from the listed MUST-have feature list have been implemented correctly.	9-11 relevant and reasonable features from the listed MUST-have feature list have been implemented correctly.	12-14 relevant and reasonable features from the listed MUST-have feature list have been implemented correctly.	15+ relevant and reasonable features from the listed MUST-have feature list have been implemented correctly.	* 4
Unit tests and % tests passed	Not attempted	1-4 relevant and reasonable tests to partially test coverage of functionality; All tests pass.	5-9 relevant and reasonable tests to partially test coverage of functionality; All tests pass.	10-14 relevant and reasonable tests to partially test coverage of functionality; all tests pass.	15-20 relevant and reasonable tests to partially test coverage of functionality; all tests pass.	> 20 relevant and reasonable tests to fully test coverage of functionality; all tests pass.	* 3
Coding style and standards	Not attempted	Code hard to follow in one reading, poor formatted and structured	Relatively well-formatted, understandable code and relatively well-organized file structure	Well-formatted, understandable code and well-organized file structure	Relatively well-structured and presented (e.g., modularized, commented)	Non-redundant, Well-structured, properly presented (e.g., modularized, commented)	* 1