Te Hoe Whakairi Kōrero
Department of Creative Arts and Digital Information

Bachelor of Information and Communication Technologies
Graduate Diploma in Information and Communication Technologies

Assessment Two:
Practical Assessment 2

Best Programming Practices (Web and Mobile Development) BCDE211

Semester One 2025

Due date: Friday 11 April 2025

Time: 5:00 pm

TOTAL MARKS: 55

Learner Name/ID .....

If a learner needs to apply for an extension, they can do so by completing the extension request form (app505m-extension-of-time-application.pdf). Extension requests must be submitted to the lecturer prior to the assessment due date.

If an assessment is handed in late without an approved extension, a penalty of 10% per day will apply, up to a maximum of 50%. If an assessment is received more than five days after the due date without an approved extension, it will not be marked. Should a learner wish to appeal any decisions, they may do so in writing to the Head of Department within ten days of receiving the decision.

This assessment is worth 25% of the total marks for this course.

To pass this course, learners must gain an average of at least 50% across all assessments and gain at least 50% in Assessment 3.

This paper has six (6) pages including the cover sheet.

LEARNING OUTCOME

This is an individual assignment, which relates to the following Learning Outcome 2: 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 triathlon enthusiast. This first iteration of the program must be written in ES6+ (not required to use TypeScript). Naturally you will be expected to use good design, coding and testing practices.

There are many existing triathlon related apps[1]. You are encouraged to explore them to form your ideas about the possible MUST-HAVE 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>> (or <<ViewModel>>)
- Connecting to hardware specific APIs

TASKS

You need to analyse, design, implement, and test a <<Model>> that will be able to be used later in association with a <<View>> and a <<Controller>> (or <<ViewModel>>) in Assessment 3.

It means that you need to:

1 Decide on the MUST-HAVE features that your program will have based on the assessment instructions.

2 Develop appropriate UML 2 diagrams (including at least a class diagram) for your <<Model>>

3 Implement your <<Model>>.

4 Develop appropriate Jest unit tests to test the correctness of your <<Model>> code.

You **MAY** work in groups for creating the code for this assessment but must make **INDIVIDUAL** presentations. Clearly acknowledge the source of all code that is not your own in your submitted source code files and presentation slides. You may use code provided by others in the class but **will only be awarded marks on code written by you**.

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

[1] For example, https://www.220triathlon.com/gear/tri-tech/best-triathlon-training-apps-review

SUBMISSION

Learners 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 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 documents developed.

The presentation should cover the following:

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

2 A list of MUST-HAVE app features which are required to accomplish the following general requirements. Each feature needs to be described through a **User Story in Who-What-Why format**. Please also specify which of the following general requirements your feature fulfils.

- 1 Create a whole that acts as a container and gateway of accessing to parts
- 2 Add a part
- 3 Sort parts
- 4 Find a part given a search criterion or filter parts[2]
- 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 suitable default values for input parameters of the method of adding a new part
- 14 Get all parts
- 15 Appropriate database (e.g., IndexedDB) connectivity, i.e., some kind of **CRUD operations can be performed on the data in the database after successfully connecting to the database**
- 16 Handle unexpected and abnormal conditions

[2] Searching begins from nothing and adds to a list of results based on criteria that matches. Filtering begins from the full list of results and eliminates from that list based on which results do not match certain criteria.

3 Analyzing and designing the <<Model>> you intend to build. This includes at least:

- Class diagram in UML 2

4 How many of the general requirements have been covered by the implemented MUST-HAVE app features in the <<Model>>?

- Use snapshots to clearly indicate which general requirements are covered by which MUST-HAVE app feature implemented in the <<Model>> code

5 How well is the correctness of your <<Model>> proven by your created Jest unit tests?

- Use snapshots to clearly indicate which unit test is to test which MUST-HAVE app feature implemented in your <<Model>>
- Use HTML test coverage report to clearly indicate how well your unit tests cover the MUST-HAVE app features implemented in your <<Model>>

MARKING RUBRIC

Element	0	1	2	3	4	5	Weight
MUST-HAVE app feature list proposed (Feature coverage)	Not attempted	Relevant and reasonable MUST-HAVE app features, which will fulfil 1-3 general requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 4-6 general requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 7-10 general requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 11-14 general requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 15+ general requirements provided, have been proposed.	* 1
Class diagram	Not attempted	Correctly identify all required classes.	Correctly identify all required classes and relationships.	Correctly identify all required classes, attributes and relationships.	Correctly identify all required classes, attributes, methods, and relationships.	Correctly identify all required classes, attributes, methods, relationships, access modifiers, navigability, and multiplicity.	* 2
Completeness of requirements	Not attempted	The proposed MUST-HAVE app features, which fulfil 1-3 general requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 4-6 general requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 7-10 general requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 11-14 general requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 15+ general requirements provided, have been implemented correctly.	* 4

Element	0	1	2	3	4	5	Weight
Unit tests with test coverage	Not attempted	The MUST-HAVE app features implemented, which fulfil 1-3 general requirements provided, have been **successfully** unit tested and the HTML coverage report shows 100% **statement & branch** coverage of the code for these features.	The MUST-HAVE app features implemented, which fulfil 4-6 general requirements provided, have been **successfully** unit tested and the HTML coverage report shows 100% **statement & branch** coverage of the code for these features.	The MUST-HAVE app features implemented, which fulfil 7-10 general requirements provided, have been **successfully** unit tested and the HTML coverage report shows 100% **statement & branch** coverage of the code for these features.	The MUST-HAVE app features implemented, which fulfil 11-14 general requirements provided, have been **successfully** unit tested and the HTML coverage report shows 100% **statement & branch** coverage of the code for these features.	The MUST-HAVE app features implemented, which fulfil 15+ general requirements provided, have been **successfully** unit tested and the HTML coverage report shows 100% **statement & branch** coverage of the code for these features.	* 2
Coding style and standards and naming conversion	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.	Well-formatted, understandable code and well-organized file structure, relatively properly presented (e.g., modularized, commented).	Well-formatted, understandable code and well-organized file structure, non-redundant, properly presented (e.g., modularized, commented).	* 2