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 Three
Portfolio

Best Programming Practices (Web & Mobile Development) BCDE211

Semester One 2024

Due date: Friday 14 June 2024

Time: 5:00pm

TOTAL MARKS: **64**

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 50% 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 nine (9) pages including the cover sheet.

Learning Outcome Assessed in this Assessment

Learning Outcome 3: Apply knowledge of standards and tools to build complex systems.

Tasks

1 **Developing a React-based User Interface (UI) for your solution for Assessment 2**

In Assessment 2, you created a <<MODEL>> of a program beneficial to triathlon enthusiasts. Building on this, the focus of this task is to develop a React-based user interface (UI), i.e., a <<VIEW>>, and a <<CONTROLLER>> compatible with your previously developed <<MODEL>>. The solution of this task must meet the following generic requirements, which may necessitate updates to your solution for Assessment 2.

Generic Requirements

(1) Display Information: The solution should be able to display information about the whole that acts as a container and a gateway to the parts it contains.

(2) Create New Part: The solution should enable users to enter values required to create a new part via appropriate input controls.

(3) Sort Parts: The solution should be able to sort parts in at least two different orders.

(4) Find a Part: The solution should allow users to search for a specific part using a user-specified criterion and be able to show successful and unsuccessful searches.

Or Filter Parts: The solution should be able to toggle the filter "on/off" and provide 2+ filter options to refine results.

(5) Delete Parts: The solution should be able to delete user-specified parts and display successful and unsuccessful deletion attempts.

(6) Save Parts to localStorage: The solution should be able to save parts to the browser's localStorage.

(7) Load Parts from localStorage: The solution should be able to demonstrate that the previously saved parts can be loaded from browser's localStorage when the browser is started again.

(8) Update/Edit a Part: The solution should be able to show that any fields of a part can be edited/updated by users as needed.

(9) Discard/Revert Edits to a Part: The solution should be able to show that the user can roll back the fields of a part to the values before the last edit.

(10) Validate Inputs: The solution should validate inputs to ensure that only valid inputs are accepted and provide feedback on incorrect or missing inputs.

(11) Calculation within a Part: The solution should be able to display the result of a calculation within a part.

(12) Calculation across Many Parts: The solution should be able to display the result of a calculation across many parts.

(13) Provide Default Values: The solution should be able to show default values appearing in the input controls before a user starts entering values for adding a new part.

(14) Display All Parts: The solution should be able to display all parts in a list or table.

(15) Database Connectivity: The solution should allow users to enter configuration information to connect to a target database, e.g., IndexedDB, SQLite or MySQL[1], and demonstrate usage of the target database.

(16) Handle Unexpected and Abnormal Conditions: The solution should be able to demonstrate how to handle exceptions that can occur when connecting to and using a target database.

Note that

1) The submitted PowerPoint presentation slides need to include code arranged by feature and screen recording for each feature to demonstrate that the feature is working properly.

2) Marks will be awarded based on the successful fulfilment of each requirement, with 2 marks allocated per completed requirement.

(32 marks)

2 Completing your solution for Assessment 2

For learners who obtained less than half of the total marks in Assessment 2, this task offers an opportunity to upgrade and resubmit their solution. The marking rubrics of Assessment 2 will be applied, with marks assigned solely for any new additions or improvements made in the resubmission.

(25 marks)

3 Rewriting your solution code and test code for your Assessment 2 in TypeScript

You will need to use a HTML test coverage report generated by Jest to prove the correctness of the TypeScript code created. 100% statement and branch coverages of the Typescript code are still required. 0.5 marks for demonstrating that one of the generic requirements listed in Assessment 2 has been successfully implemented, unit tested using TypeScript with 100% statement and branch coverages.

(8 marks)

[1] If you intend to use back-end database, e.g., MySQL, you may use the Restful API server provided as a proxy. See https://moodle.ara.ac.nz/mod/equella/view.php?id=1366894

4 Creating a progressive web app by implementing Progressive Web Application (PWA) related features in your solution for Task #1

3 marks for demonstrating that one of the PWA related features listed below has been successfully and appropriately implemented in your solution for Task #1.

1) Establish a local web server utilizing HTTPS protocol for the purpose of distributing your PWA to end-users. To validate the successful completion of this task, kindly provide a brief screen recording that clearly demonstrates the process and the resulting functionality.
2) Create a web app manifest and set up its related resources.
3) Create and register a service worker that can handle install and fetch events in order to provide offline support via Fetch and Cache APIs.

You are only allowed to implement the following PWA related features if you have successfully implemented all the PWA related features listed above.

4) Use File System Access API to store and access to files on disk.
5) Display a badge on the app icon.

(15 marks)

5 Creating a React application using Vite or Next.js in JavaScript/TypeScript in a Node.js environment

1) This task requires you to recreate the solution for Task #1 using Vite[2] or Next.js[3], rather than implementing React within a local HTML page[4], which you might do in Task #1. A correct solution is worth 2 marks.
2) You can further upgrade your solution for this task to an installable[5] progressive web app by merging your solution for Task #4. A correct solution is worth another 2 marks. You should reuse the code from your solutions for Tasks #1 & #4 whenever possible.

(4 marks)

[2] https://react.dev/learn/add-react-to-an-existing-project#step-1-set-up-a-modular-javascript-environment
[3] https://react.dev/learn/add-react-to-an-existing-project#using-react-for-an-entire-subroute-of-your-existing-website
[4] https://react.dev/learn/installation#try-react-locally
[5] See the minimum requirements for an installable PWA on https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Installable_PWAs#requirements

MARKING RUBRIC

The marks of each task will be awarded based on the correctness of the corresponding solution. The final marks for this assessment will also depend on how well coding style and standards have been applied in the solution of this assessment. The final marks of this assessment will be the total marks awarded for all tasks plus the marks awarded for “coding style and standards”.

Requirements of Task 1	0 Mark	1 Mark	Weight
1. Display Information	Not attempted	Successfully fulfilling the requirement	
2. Create New Part	Not attempted	Successfully fulfilling the requirement	
3. Sort Parts	Not attempted	Successfully fulfilling the requirement	
4. Find a Part or Filter Parts	Not attempted	Successfully fulfilling the requirement	
5. Delete Parts	Not attempted	Successfully fulfilling the requirement	
6. Save Parts to localStorage	Not attempted	Successfully fulfilling the requirement	
7. Load Parts from localStorage	Not attempted	Successfully fulfilling the requirement	
8. Update/Edit a Part	Not attempted	Successfully fulfilling the requirement	* 2
9. Discard/Revert Edits to a Part	Not attempted	Successfully fulfilling the requirement	
10. Validate Inputs	Not attempted	Successfully fulfilling the requirement	
11. Calculation within a Part	Not attempted	Successfully fulfilling the requirement	
12. Calculation across Many Parts	Not attempted	Successfully fulfilling the requirement	
13. Provide Default Values	Not attempted	Successfully fulfilling the requirement	
14. Display All Parts	Not attempted	Successfully fulfilling the requirement	
15. Database Connectivity	Not attempted	Successfully fulfilling the requirement	
16. Handle Unexpected and Abnormal Conditions	Not attempted	Successfully fulfilling the requirement	
			Total marks out of 32 marks:

Requirements of Task 2	0 Mark	1 Mark	2 Marks	3 Marks	4 Marks	5 Marks	Weight
MUST-HAVE app feature list proposed (Feature coverage)	Not attempted	Relevant and reasonable MUST-HAVE app features, which will fulfil 1-3 generic requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 4-6 generic requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 7-10 generic requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 11-14 generic requirements provided, have been proposed.	Relevant and reasonable MUST-HAVE app features, which will fulfil 15+ generic 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 generic requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 4-6 generic requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 7-10 generic requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 11-14 generic requirements provided, have been implemented correctly.	The proposed MUST-HAVE app features, which fulfil 15+ generic requirements provided, have been implemented correctly.	* 4
Unit tests with test coverage	Not attempted	The MUST-HAVE app features implemented, which fulfil 1-3 generic 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 generic 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 generic 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 generic 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+ generic 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
					Total marks out of 25 marks:		

Requirements of Task 3	0 Mark	1 Mark	Weight
1. Display Information	Not attempted	Successfully rewriting solution code and test code in TypeScript	* (statement coverage + branch coverage) / 4
2. Create New Part	Not attempted	Successfully rewriting solution code and test code in TypeScript	
3. Sort Parts	Not attempted	Successfully rewriting solution code and test code in TypeScript	
4. Find a Part or Filter Parts	Not attempted	Successfully rewriting solution code and test code in TypeScript	
5. Delete Parts	Not attempted	Successfully rewriting solution code and test code in TypeScript	
6. Save Parts to localStorage	Not attempted	Successfully rewriting solution code and test code in TypeScript	
7. Load Parts from localStorage	Not attempted	Successfully rewriting solution code and test code in TypeScript	
8. Update/Edit a Part	Not attempted	Successfully rewriting solution code and test code in TypeScript	
9. Discard/Revert Edits to a Part	Not attempted	Successfully rewriting solution code and test code in TypeScript	
10. Validate Inputs	Not attempted	Successfully rewriting solution code and test code in TypeScript	
11. Calculation within a Part	Not attempted	Successfully rewriting solution code and test code in TypeScript	
12. Calculation across Many Parts	Not attempted	Successfully rewriting solution code and test code in TypeScript	
13. Provide Default Values	Not attempted	Successfully rewriting solution code and test code in TypeScript	
14. Display All Parts	Not attempted	Successfully rewriting solution code and test code in TypeScript	
15. Database Connectivity	Not attempted	Successfully rewriting solution code and test code in TypeScript	
16. Handle Unexpected and Abnormal Conditions	Not attempted	Successfully rewriting solution code and test code in TypeScript	
		Total marks out of 8 marks:	

Requirements of Task 4	0 Mark	1 Mark	Weight
1. Establish a local web server utilizing HTTPS protocol	Not attempted	Successfully fulfilling the requirement	* 3
2. Create a web app manifest and set up its related resources	Not attempted	Successfully fulfilling the requirement	
3. Create and register a service worker	Not attempted	Successfully fulfilling the requirement	
4. Use File System Access API	Not attempted	Successfully fulfilling the requirement	
5. Display a badge on the app icon	Not attempted	Successfully fulfilling the requirement	
		Total marks out of 15 marks:	

Requirements of Task 5	0 Mark	1 Mark	Weight
1. Recreate the solution for Task 1 using Vite/Next.js	Not attempted	Successfully fulfilling the requirement	* 2
2. Upgrade the solution for Task 1 using Vite/Next.js to an installable progressive web app	Not attempted	Successfully fulfilling the requirement	
		Total marks out of 4 marks:	

Marks	0	1	2	3	4	5
Coding style and standards	Not attempted	Code hard to follow in one reading, poorly 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 well-structured and presented (e.g., modularized, commented).	Well-formatted, understandable code and well-organized file structure, non-redundant, well-structured, properly presented (e.g., modularized, commented).

Total marks (out of 64 marks) for this assessment are:

SUBMISSION

Learners must submit their work as a single .zip file to the drop box on BCDE211 course Moodle site by the deadline indicated. The .zip file should contain the following.

(a) A digital copy of all your project code (including source code, test code and configuration files) for this assessment, so that your project can be run and tested without any intervention.

(b) A simple presentation explaining what you have coded for this assessment. Your presentation MUST be in PowerPoint format.

In general, your presentation needs to show:

(a) The task you would like to claim marks for.

(b) The expected (i.e., self-estimated) marks for the task based on the marking rubric

(c) The evidence, i.e., a short screen recording, showing that the task has been successfully completed.

(d) The source code behind the completed task, i.e., snapshots of the code.

NOTE

Learners are encouraged to seek clarification on any doubts or ambiguities regarding the assessment instructions. This is an individual assessment, and each learner must submit their own individual work. Learners are also expected to show their progress on their assessment work to their course tutor on a weekly basis and receive formative feedback.