CITS3403/5505 Final Test.

Handing Your Project Over to Someone Else.

*This Exam is worth **50%** of your final grade and must be done individually. For equity reasons, the teaching staff will not be able to answer individual questions between the exams release and the due date. Questions may be posted to the **Final Test** teams channel and may be answered by teaching staff. Any communication, commentary, or discussion of the test during this period will be considered academic misconduct. Submit a zip containing your HTML document and any required media files to https://secure.csse.uwa.edu.au/run/cssubmit by **5pm Wednesday, May 31, 2023***

Suppose that you are going to hand your Agile Web Development Group Project over to a new group of 2-4 students who will continue to develop and refine that project. This test requires you to build a webpage that describes how to build and run the projects, provides suggestions for additional features to be incorporated into the project, and gives some advice on working in teams.

For the final unit test, you will need to produce a HTML document describing how you might address these challenges. The questions must be answered with respect to the current state of your group project. The document should satisfy the following constraints:

1. The document should consist of a single HTML file, with embedded CSS and JavaScript code. You may include some additional image or icon files in the zip, but you must ensure that these will render in the document when the zip is expanded.
2. You may not use any libraries or external code, like Bootstrap or JQuery, and the page must run without using live server, so AJAX should not be included in the document.
3. The page should render in a modern web browser (Chrome, Edge, Firefox, Safari etc) from the file explorer, and should be easy to read and use. You may assume that the web page is going to be rendered using a large desktop monitor, and the page does not have to be responsive.

Specification: The web page should have the unit title and code, your name and student number in a header at the top of the page. The web page should have four clearly separated sections with easy navigation between them, where the sections are: *Overview*; *Build and Run Instructions*; *Suggested Features*; and *Advice for Groups*. They have the following specifications.

1. **Overview:** Present a description of the application you developed in the project, describing the purpose of the application, the intended users, and the way the application works. You should include a screenshot of the application.
2. **Build and Run Instructions:** These instructions are the main part of the page and should give the steps describing how to launch the application given the git repository of the application. You may assume that the students have already cloned the repository, and the app only needs to be hosted on localhost. Each step should have a *title*, some detailed text *description* of the process, an *image* showing a screenshot of the process being described, and an *aside* containing useful information or context for the step.

   Examples of popular sites that present instructions or procedures in similar formats include iFixit or Instructables. Your instructions should also satisfy the follow properties:

   a) The text *description* should be detailed enough to allow a competent student to build and run the application.

   b) CSS styling should be used to make each step appear neat and readable, and clearly distinct from other steps.

c) An aside should be have a *label* and some *note*, and should be presented as a button with the given *label*, that when pressed reveals the *note*. Pressing the button again hides the *note*.

d) You must include two *asides* satisfying the following specification:

1. One aside should have the label *Object Relational Mapping* and the note should describe how an ORM like SQLAlchemy allows you to synchronise python classes with database tables, using examples from your project. The note should have 100-200 words.

2. One aside should have the label *Routing* and the note should explain how a flask routes handle requests using a stateless client server architecture. You should use examples from your project, and the note should have 100-200 words.

e) The steps should be described by a list of JSON objects, which are defined in a script in the page header. These objects should be rendered into the HTML when the page loads. An example JSON object describing a step is below.

```
{
title: "Clone the repository",
text: "Given Git is installed, type <code>git clone github.com/my/repo</code>",
image: "./gitCloneSuccess.jpg",
aside: {
   label: "The origin of git",
   note: "Git was created by Linus Torvalds, who also made the Linux operating system"
   }
}
```

3. **Suggested Features:** Provide two suggestions for additional features that could be included in the app. For each feature provide:

a) a short description of the feature, and a reason for why it would be useful to have (approximately 100 words);

b) a user story describing how the feature works from the user's point of view;

c) a set of test cases, described using the *given-when-then* convention to give a complete acceptance test for the proposed feature.

4. **Advice for Groups:** Give a some advice for working in groups to future student teams taking on this project. Address the following points (50-100 words each):

a) A set of software tools you would recommend using, and the best way to incorporate them into the project. Include hyperlinks to the tool, and possibly link useful tutorials.

b) Recommendations for managing communication, and task allocation within a team of student developers, citing your own experiences in the project.

c) Best practice for prioritizing deadlines and deliverables, particularly when team members have competing deadlines from other units.

Marking Criteria

Name:_____ **Student#:**_____

Criteria	Excellent	Good	Satisfactory	Inadequate	Comments	Weight
Basic Data.	Name, unit code and student number are clearly rendered at top of page. All instructions followed.	Name, unit code and student number are rendered. Instructions mostly followed.	Name, unit code and student number are rendered.	Missing data or significant instructions ignored.		/2
HTML	Valid code, well formatted, with excellent navigation.	Valid code, well formatted, and good navigation.	Mostly valid code, adequate navigation.	Invalid/incomplete code. Poor navigation.		/4
CSS	Valid code with intuitive classes, and distinctive sections with aesthetic appeal	Valid, maintainable code with intuitive classes, and distinctive sections	Mostly valid code, using novel style elements, and intuitive classes.	Invalid code, minimal style applied.		/4
JavaScript	Valid, well formatted code, with perfect function, and best practice coding	Valid, well formatted code, with perfect function.	Valid code meeting most of the requirements.	Faulty or incomplete code.		/4
Overview of Application	Clear well formatted description and complete overview.	Clear well formatted description and basic overview.	Basic description and overview of project.	Does not convey application's purpose or appeal.		/3
Instructions	Clear well formatted instructions to easily launch project, with interesting asides.	Clear instructions, sufficient to launch project, with useful images and asides.	Clear instructions with most details present and some useful asides	Incomplete instructions, or poor, ill-considered formatting.		/6
Aside: ORM	Clear description of ORM with insightful reference to project	Clear description of ORM with some reference to project	Clear description n of ORM in Flask.	Missing information or detail.		/4
Aside: Routing	Clear description of routes in stateless client server architecture, with insightful reference to project.	Clear description of routing in stateless client server architecture with reference to project	Clear description of routing with some reference to project.	General information only with no insight or reference to project.		/4
New Feature: Overview	Well thought out features, with clear rational	Reasonable features with some explanations.	Reasonable features, lacking explanation.	Inappropriate or irrelevant features.		/2
New Feature: User Story	Clear, well formatted user story showing the feature from the user's view.	Clear user story demonstrating the feature from user's point of view.	User story demonstrating essential elements of the feature.	Unclear behaviour, or incomplete description.		/4
New Features: Test Cases	Clear well formatted acceptance test cases in GWT format, giving good coverage of the feature.	Clear acceptance test cases in GWT format, with reasonable coverage.	Clear test cases giving some coverage of the feature	Incomplete or poorly described features, lacking coverage of the feature.		/4
Group Work: Tools	Good selection of tools, well motivated with well chosen links.	Good selection of tools with well chosen links	Some sensible tools described.	Generic tools, with no discussion or motivation.		/3
Group Work: Processes	Insightful discussion of processes with references to project.	Good discussion of processes with some references to project.	Sensible processes described, with some discussion.	Basic processes lacking relevance or insight.		/3
Group Work: Prioritization	Well described techniques, with references to project, and coursework	Reasonable techniques clearly described, in the University context	Reasonable techniques described, but only at a generic level	Basic techniques, with no insight or practical value.		/3
Overall						/50