### ITECH3108 – Dynamic Web Development

# Assessment Task – Front-end Development

## Overview

For this assessment task, you will use skills acquired through the first six weeks of material to build an interactive front-end to an API.

You will be developing a front-end for a simple forum application, using front-end JavaScript, the Document Object Model, and fetch to connect to a RESTful web API. You will also submit a written reflection on your learning.

## Timelines and Expectations

Percentage Value of Task: **30%**

Due: **Refer to Course Description**

Minimum time expectation: **34 hrs**

## Learning Outcomes Assessed

Refer to Course Description.

## Assessment Details

For this assignment, you will need to create an interactive front-end for an existing API, using HTML, CSS and JavaScript.

### Part one – Install the API server application

Using the skills you developed in the week 1 laboratory task, install the API server application. The application can be installed using the following single command:

```
deno install --allow-read --allow-net
https://cdn.jsdelivr.net/gh/ITECH3108FedUni/assignment_api@v2022.05/chat_server.js
```

Ensure that is correctly installed by running `chat_server` and pointing your browser at **https://localhost:7777**, which should give an overview of the server API and a live display of the current database.

### Data Model

The database exists in-memory only, and will be **reset** every time the server restarts, so **don't be afraid of breaking anything** in it.

There are two top-level object types: **users** and **threads**.
A **user** has a **username** and a **name** (or display name).

A **thread** has a **title**, an **id**, an **icon**, and an array of **posts**.

A **post** has a **user** (the author of the post) and a **text** field (the content of the post).

The complete database is displayed in the browser, and updates automatically.

### Server functionality

The chat_server API has (at minimum) the following resources available:

| | | |
|---|---|---|
| GET | `/api/threads` | Retrieve a list of all **threads** |
| GET | `/api/threads/{id}` | Retrieve a specific **thread** |
| GET | `/api/threads/{id}/posts` | Retrieve a specific **thread** and all the associated **posts** |
| GET | `/api/users` | Retrieve a list all **users** |
| GET | `/api/users/{user}` | Retrieve information about a specific **user** |
| GET | `/api/users/{user}/threads` | Retrieve all **threads** started by a particular **user** |
| POST | `/api/threads` | Create a new **thread**. Requires a JSON body to be submitted with a content-type of application/json<br><br>The body must be a JSON object containing the following keys:<br><br>**`user`** : The username of the user posting.<br>**`icon`**: A single character - e.g. an emoji.<br>**`title`** : The title of the thread. A string.<br>**`text`** : The content of the first post. A string. |
| POST | `/api/threads/{id}/posts` | Create a new **post** within a particular **thread**. Requires a JSON body to be submitted with a content-type of application/json<br><br>The body must be a JSON object containing the following keys:<br><br>**`user`** : The username of the user posting.<br>**`text`** : The content of the post. A string. |
| DELETE | `/api/threads/{id}` | Delete an entire **thread**, and all **posts** within it. Requires a JSON body to be submitted with a content-type of application/json<br><br>The body must be a JSON object containing the following keys:<br><br>**`user`** : The username of the current user, which must match the user who created the post otherwise an error value is returned. |

Any errors will be reported both by an appropriate status code (eg. 400, 401, 404) and by a JSON response containing an “error” key.

The API does not allow creating or deleting **users**, nor deleting individual **posts**. It does not support the PUT verb for any resources.

Creating a **thread** using the API also creates the first post within it.

## Part two – Build the application

Create a front-end HTML, CSS and JavaScript application that uses the above API.

Your application should run within the regular Deno **file_server** application. During development you will need to run both **file_server** to serve your assignment files and **chat_server** to serve up your chat forum API. In Windows, this is easiest achieved by running multiple command prompts.

The application should have the following functionality:

- Upon loading the site, the user is presented with a "**login**" screen – this does **not need to perform any authentication** and merely asks the user to type their username. (e.g. Figure 1)
- The application should check that this **username is already in the database** by requesting the appropriate API endpoint, and retain this username and use it in subsequent requests as required.


*Figure 1 Login screen*

- After login, the application should display a list of all threads, where each thread is a clickable link (e.g. Figure 2)

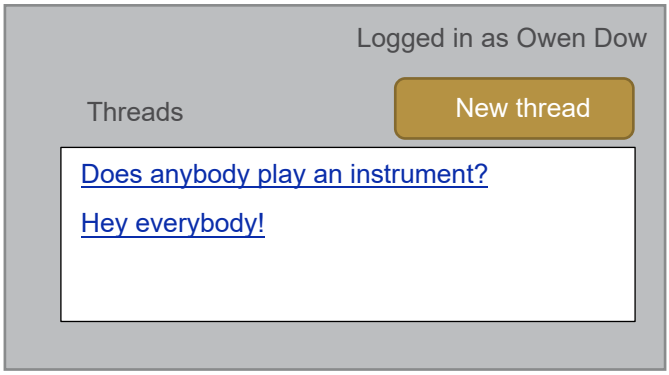


*Figure 2 Threads screen*

- The user should be able to **create a new thread**, supplying a thread **title** and the text of an initial **post**. (No mockup supplied – implement this however you like).

- When the user clicks on one of the listed threads, the list of posts should be **dynamically loaded** and displayed underneath (e.g. Figure 3).

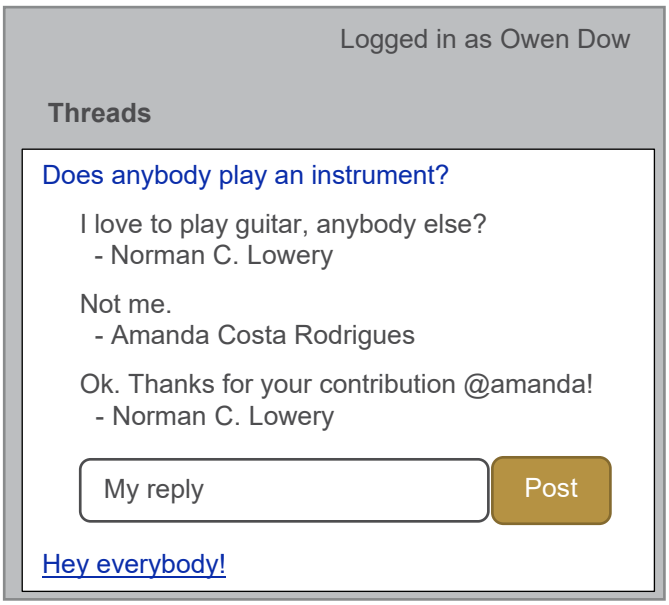


*Figure 3 Posts screen*

- The user should be able to **add a post** to a **thread**.
- The user should be able to **delete** a thread that they created themselves.
- Every 10 seconds, the data currently being displayed should be refreshed from the server, *without losing user input* (for example in the Reply box).

> Hint: look at using window.setInterval or window.setTimeout

### Challenge task (optional!)

- Use the History API to make it possible to use the Back and Forward browser buttons to navigate (eg, clicking back after clicking a thread).

### Part three – Written report

Include in your submission a **written report** which includes:

- A **personal reflection** describing your approach to the assignment, any difficulties encountered, and what you learned in completing the task (approx. 300-400 words, about 1 page). Note that *any* plagiarism in this reflection is absolutely unacceptable, and will be reported if discovered.
- A **statement of completion** indicating which parts of the assignment you did or did not attempt.
- A **statement of assistance** showing what sort of help you obtained from external resources or peer groups.
- Any instructions necessary to run your application (if not obvious)
- Anything cool or extra you've done.

**This page intentionally left blank**

## Submission

Zip your assignment files, preserving the directory structure, and submit via Moodle.

## Marking Criteria / Rubric

Refer to the attached marking guide.

## Feedback

Feedback will be supplied through Moodle.

Authoritative results will be published on fdlMarks.

## Academic Misconduct

To submit your assessment task, you must indicate that you have read and understood, and comply with, the Federation University Australia Academic Integrity and Student Plagiarism policies and procedures (http://policy.federation.edu.au/learning_and_teaching/compliance/academic_integrity/ch02.php).

You must also agree that your work has not been outsourced, and is entirely your own except where work quoted is duly acknowledged. Additionally, you must agree that your work has not been submitted for assessment in any other course or program.

**ITECH3108 – Dynamic Web Development**

# Marking Guide – Front-end Development

| Criteria | Maximum | Obtained |
|---|---|---|
| **Implementation** | | |
| • Login screen allows user to type username | 1 | |
| • Login screen validates username | 2 | |
| • Threads list displays list of threads | 2 | |
| • New threads can be created successfully | 3 | |
| • Thread items can be selected to show posts | 2 | |
| • Post listing includes both text and display name | 1 | |
| • New posts can be added to a thread | 3 | |
| • Users can delete their own threads | 3 | |
| • The page automatically loads new data | 3 | |
| • Challenge task (optional). Navigation using the History API | (+3) | |
| **Implementation penalties** | | |
| • Implementation does not use REST API | (-5) | |
| • Complete page reload (e.g. window.location.reload or clickable links) | (-2) | |
| • Users can delete the posts of others | (-1) | |
| • Errors in console | Up to (-3) | |
| **Written report** | | |
| • Appropriate reflective task | 5 | |
| • Penalty – missing or incorrect statement of completion or statement of assistance | (-1 each) | |
| **Total** | **25** | |