School of Games and Creative Technology
BSc (Hons) Computer Science

University for the Creative Arts

FGCT6021 MOBILE APPLICATION DEVELOPMENT
LAB 7 – TOPIC 7 PWA

Submission

1. Submit your Learning Journal link (GitHub), your PythonAnywhere link and weekly journal entry on myUCA.
2. During the workshop, you will show your progress and share what you have learned.

Workshop Overview

You will extend your Learning Journal Progressive Web App (PWA) by applying PWA technologies to make it more reliable, installable, and interactive. This week's focus is on using a **manifest**, creating a **service worker**, implementing caching with the **Cache Storage API**, and dynamically retrieving data with the **Fetch API**.

Your goal this week is to:

- Configure a **manifest** to make your app installable.
- Create a **service worker** to manage caching and offline access.
- Use the **Cache Storage API** and **Fetch API** to handle dynamic data.
- Explore and implement at least one **extra feature** that enhances the PWA experience.
- Modify your `HTML` and `flask_app.py` to integrate these PWA features with your existing backend.

This builds on your previous work with Flask, PythonAnywhere, HTML, CSS, and JavaScript, now extending the Learning Journal into a more **robust, offline-capable, and app-like experience**.

Goal

Enhance your Learning Journal PWA with PWA technologies to improve usability, reliability, and offline capabilities, while maintaining dynamic data integration through Flask.

All files should be committed to your **GitHub** repository so progress is visible and version-controlled.

What to Do

The following tasks are suggestions. You may adapt them as appropriate to your own design.

Folder and File Structure

Flask works best with its conventional **static/** and **templates/** folders. A suggested structure:

```
/mysite
   flask_app.py            Main Flask backend file
   /templates
      index.html           HTML pages served by Flask
      journal.html
      about.html
      projects.html
   /static                 Client-side assets
      /css                 Stylesheets
         style.css
      /js                  JavaScript files
         script.js
         sw.js             Service worker
      /images              Media assets
         icon.png
      manifest.json        HTML pages served by Flask
   /backend                Backend data files
      reflections.json     JSON file storing reflections
```

Manifest

1. Consider creating a **manifest file**, `manifest.json`, to make your app installable.
2. Include fields that support installability, such as **app name, short name, icons, start URL, display mode, and theme colours**.
3. You may also explore optional fields such as **orientation, description, or categories** to enhance the app experience.
4. Linking the manifest in your `HTML <head>` can allow your app to be installed on desktop or mobile for testing purposes.

Service Worker & Dynamic Data Handling

1. Create a **service worker file**, `sw.js`, to manage caching and offline access.
2. Decide which **static assets, pages, or backend data** should be cached.
3. Explore ways to dynamically retrieve and display backend data using **web APIs**, such as **Cache Storage API** or **Fetch API**, in combination with your service worker.
4. Modify your `HTML` and `flask_app.py` as needed to integrate dynamic data fetching, ensuring the interface updates when new reflections are added.
5. Consider strategies for updating the UI when the app returns online after being offline.
6. Test your app in both **online and offline modes** to ensure a reliable experience.

Extend with an Extra Feature

1. Implement at least **one additional PWA feature** that enhances the user experience.
2. Examples could include:
   - Notifying users when they are offline.
   - Background syncing of new reflections when connectivity is restored.
   - Optional push notifications or other creative enhancements.
   - Any other feature that improves usability, accessibility, or engagement.

Deployment

1. Test your PWA on **PythonAnywhere**.
2. Verify that **offline support** works for both static assets and dynamic content.
3. Ensure the app can be installed on multiple devices (desktop and mobile).
4. Confirm that updates or new reflections are reflected correctly when back online.

Journal Questions

For your weekly entry, **submit it on myUCA** and **post it on the Learning Journal Web**:

1. Why is it useful to enhance your Flask app with PWA features?
2. What did you use to support offline access and dynamic data?
3. What extra feature did you add, and why?
4. Did you face any challenges deploying your PWA, and how did you solve them?