School of Information Technology and Data Science

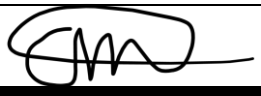
Occupational Certificate: Software Developer (Full Stack Web and Software Developer) HTML5 Software Development (HTML)	
SAQA Qualification ID	118707
Intake / year	2-2024
Assessment Name	Formative Assessment
Module NQF Level	5
Credits	16
Open date and time	24 March 2025 • ***Open time: 08:00***
Close date and time	13 April 2025 • ***Close time: 23:45***
Assessment Total Marks	365 (50% of total mark percentage)
Pass percentage	65%

Lecturer Information	
Lecturer	Thaveshan Govender
Lecturer E-mail	ThaveshanG@aie.ac

IMPORTANT NOTICE

Before submitting your assessment, please ensure that it is signed and that all your personal details are accurately filled in the designated space below (Section in red). Work will not be accepted unless the required sections below are fully completed prior to submission, resulting in a 0% grade for this assessment.

The signature must be your valid handwritten signature and NOT your name and surname typed in the space provided.

Student Number:	**242748**		
Student name and surname:	**CARLOS MAVEYA**	**Date:**	**24/03/205**
Student cell phone number:	**067 106 7078**		
Student Signature:			
Lecturer name and surname:	**Thaveshan Govender**	**Date:**	
Lecturer Signature:			
Assessor name and surname:		**Date:**	
Assessor Signature:			
Moderator name and surname:		**Date:**	
Moderator Signature:			

Instructions to Candidates
1. Read each question carefully. 2. You must answer Section B below the space provided named “STUDENTANSWER”. Once completed, the answer sheet, with your answers, must be converted to PDF and uploaded to the student portal. Make sure that all additional information is uploaded in a single Zipped folder. 3. QCTO students who achieve between 55-64% may have a further submission opportunity, subject to the departmental rules. 4. NO LATE submissions will be accepted. (System will automatically close when due date and time is reached). If you encounter any difficulty with the student portal, immediately take a screenshot, log a ticket (attention support), and attach your assessment to the ticket. 5. You must follow the departmental formatting guidelines. 6. You must write your name, surname, and student number on your script. A failure to do so will result in your script not being marked. 7. By submitting your answer sheet for marking, you declare the following: a. You are familiar with the AIE’s policies on academic dishonesty and plagiarism; b. Your submitted is your own original work. Where you have made use of someone else’s work, you have duly referenced such work making using of the APA style of referencing; c. You have not allowed, and will not allow any other student to copy your work; and d. You did not copy the work of another student. 8. Students that are dishonest will be subjected to disciplinary procedures.

Assessment Objective/s:

The Formative Assessment will cover the following assessment topics:

No	PM/PS number	PS Description
1.	**PM-08-PS01 to PM-08-PS14**	Engage in a series of practical exercises that cover a wide range of web development skills. From creating and styling HTML5 pages to implementing real-time communication with web sockets and creating web worker processes, each activity is designed to build upon your knowledge and skills in web development.

Scope:

This module focuses on providing you with hands-on experience in using HTML5 and open-source technologies to implement programming logic, develop user interfaces, handle user input, communicate with remote data sources, and much more. By the end of this module, you will have the ability to build well-structured applications that leverage the latest web development practices.

Technical Aspects:

Please verify that all pages are included when downloading this assessment by checking the page numbers at the bottom of the document.

Ensure the following font and size is used in this assessment

a. Font: Arial

b. Size: 12

c. Font colour: Black

Save and upload the report as a .PDF **(No backgrounds)** with the following naming convention:

a. StudentName_ StudentSurname _Student no_ModuleCode_FA1

Mark allocation for report

See Mark allocation sheet below

Question 1 (30)

PM No	*PS No*	*PS Description*
08	*01*	*Creating and Styling HTML5 Pages*

a)	**PA 0101: Creating HTML5 Pages** 1. **Setup Environment:** • Open your IDE and create a new project folder named **PortfolioPage**. • Inside the folder, create two files: **index.html** and **styles.css**. 2. **Structure Your HTML Page:** • In **index.html**, define the basic HTML5 structure using the **<!DOCTYPE html>** declaration. • Add **<header>**, **<nav>**, **<main>**, and **<footer>** sections. • Within the **<main>** section, create sections for "About Me", "Projects", and "Contact". 3. **Add Content:** • Populate the "About Me" section with a brief paragraph and a photo. • List at least three projects in the "Projects" section, each with a title, description, and image. • Include a simple contact form in the "Contact" section with fields for name, email, and message. **ANSWER:**	[15]

b)	**PA 0102: Styling HTML Pages** 1. **Link Your CSS File:** • In **index.html**, link the **styles.css** file within the **<head>** section using the **<link>** tag. 2. **Style the Page:** • Use CSS to style your page, ensuring it is responsive. Consider using Flexbox or Grid for layout. • Apply styles to the **<header>**, **<nav>**, **<main>**, and **<footer>** sections for a cohesive look. • Ensure text is readable and elements are well-spaced. 3. **Responsive Design:** • Implement media queries in **styles.css** to adjust the layout and font sizes for different screen sizes. **ANSWER:**	[15]

List of Evidence to be submitted:

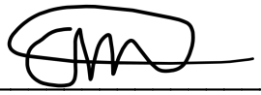
- The **index.html** and **styles.css** files.
- Screenshots of your portfolio page displayed on various devices or screen sizes.
- A brief report summarizing your design choices, including how you ensured responsiveness.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Proper HTML5 Structure	**index.html** file		
Effective Use of CSS for Styling	**styles.css** file		
Responsive Design Implementation	Screenshots/Report		

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Creativity and Aesthetics	Screenshots/Report		

Name: ___ CARLOS MAVEYA _____

Signature: 

Date: ____08/04/2025_____

Question 2 (45)

PM No	*PS No*	*PS Description*
08	*02*	*Display data and handle events by using JavaScript*

a)	**PA 0201: Displaying Data Programmatically** 1. **Create the HTML Structure:** • In your project folder, create an **index.html** file. Define the basic HTML structure including a ** ** for the To-Do List, an input field for new tasks, and an 'Add Task' button. 2. **Styling:** • Create a **style.css** file for basic styling of your To-Do List. Link this CSS file in your **index.html**. 3. **Setup JavaScript:** • Create a **script.js** file. This will contain your JavaScript code. Link this file at the end of the body in your **index.html**. **ANSWER:** A	[15]

b)	**PA 0202: Handling Events** 1. **Add Task Event:** • In **script.js**, write a function that listens for clicks on the 'Add Task' button or enter key presses in the input field. This function should retrieve the input value and add it as a new item to the To-Do List. 2. **Delete Task Event:** • Allow tasks to be removed by clicking on them. This can be done by adding an event listener to each task that removes the task from the DOM when clicked. **ANSWER:**	[15]
c)	**PA 0203: Describe Basic JavaScript Syntax** 1. **Commenting:** Throughout your **script.js**, add comments describing the basic JavaScript syntax used, including variables, functions, event listeners, and DOM manipulation. **ANSWER:**	[15]

List of Evidence to be submitted:

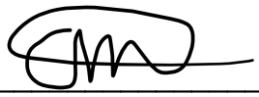
- The **index.html**, **style.css**, and **script.js** files.
- Screenshots of your To-Do List webpage showing tasks being added and removed.
- A brief explanation document highlighting how your JavaScript code manipulates the DOM and handles events.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Functional To-Do List	**index.html**, **script.js**		

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
DOM Manipulation for Displaying Data	**script.js**		
Event Handling for Adding/Removing Tasks	**script.js**		
Use of Basic JavaScript Syntax	Explanation Document, **script.js**		

Name: ____ CARLOS MAVEYA _____

Signature: 

Date: ____08/04/42025_____

Question 3 (45)

PM No	*PS No*	*PS Description*
08	*03*	*Create forms to collect and validate user input*

a)	**PA 0301: Creating Input Forms Using HTML5** 1. **Form Setup:** • In your project folder, create an **index.html** file and a **style.css** file. Add basic HTML structure to your **index.html** and link the **style.css** for styling your form. • Define a **<form>** element with **id="registrationForm"** and include the following input fields: Name, Email, Password, Confirm Password, and a Submit button. 2. **Add HTML5 Validation:**	[15]

	• Use HTML5 input types such as **type="email"** for the Email field to utilize built-in validation. • Add **required** attributes to make fields mandatory. • Use the **pattern** attribute on the Password field to enforce specific requirements (e.g., at least one number, one uppercase and lowercase letter, and at least 8 or more characters). **ANSWER:**	
b)	**PA 0302: Validating User Input Using HTML5 Attributes** 1. **Enhance HTML Validation:** • For the Confirm Password field, add a custom title attribute to display a tooltip if the pattern does not match the Password field. • Utilize the **minlength** and **maxlength** attributes where appropriate to restrict input length. **ANSWER:**	[15]
c)	**PA 0303: Validating User Input Using JavaScript** 1. **JavaScript Validation Setup:** • Create a **script.js** file and link it in your **index.html**. This script will handle the form validation that cannot be achieved with HTML5 attributes alone. • Write a JavaScript function that validates the Confirm Password field matches the Password field. Attach this validation to the form's **submit** event and prevent form submission if the validation fails. 2. **Feedback to Users:** • Provide immediate feedback to users if there are validation errors. Consider displaying error messages next to the relevant input field. **ANSWER:**	[15]

List of Evidence to be submitted:

- The **index.html**, **style.css**, and **script.js** files.

- Screenshots showing the form with various states of validation (both passing and failing).
- A brief report describing the validation logic implemented in both HTML5 and JavaScript.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Form Creation and HTML5 Validation	**index.html**, Screenshots		
Advanced HTML5 Validation Features	**index.html**, Screenshots		
JavaScript Input Validation	**script.js**, Screenshots		
User Feedback Implementation	**index.html**, **script.js**, Screenshots		

Name: ____ CARLOS MAVEYA _____

Signature: _____

Date: _____08/04/2025_____

Question 4 (30)

PM No	*PS No*	*PS Description*
08	*04*	*Communicate with a remote data source*

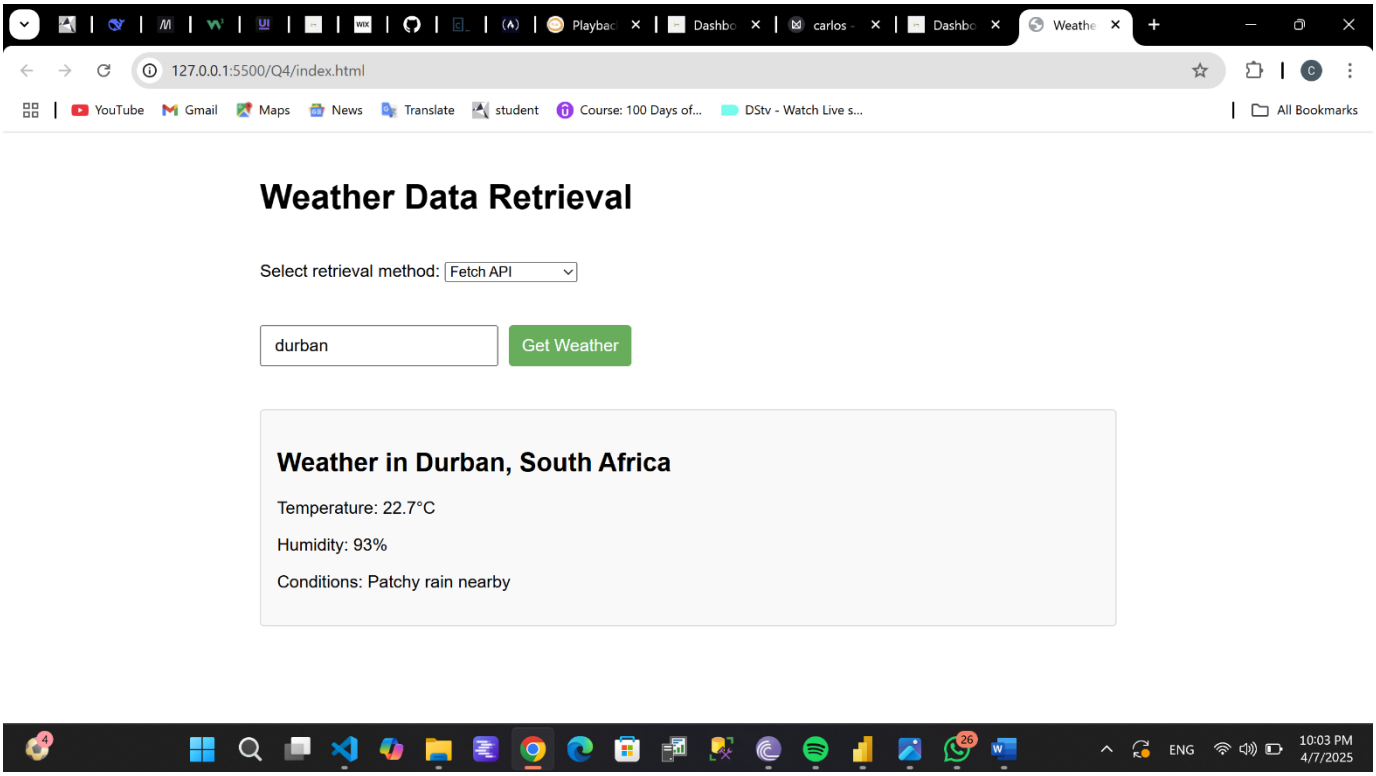
a)	**PA 0401: Retrieve Data** 1. **Setup HTML and JavaScript:** • Create an **index.html** file for your application's UI. Include input elements for the city name and a button to trigger the data retrieval.	[5]

	• Link a **script.js** file where your JavaScript code will reside. **ANSWER:**	
b)	**PA 0402: Serialize and Transmit Data** 1. **Fetch API Implementation:** • In **script.js**, write a function to fetch weather data for the specified city using the Fetch API. Serialize query parameters as needed for the API request. **ANSWER:**	[5]
c)	**PA 0403: Refactoring Using jQuery AJAX** 1. **jQuery AJAX Refactor:** • Refactor your Fetch API implementation to use jQuery's AJAX method for data retrieval. Ensure you include error handling for failed requests. **ANSWER:**	[5]
d)	**PA 0404: Handle Asynchronous Tasks** 1. **Async/Await Syntax:** • Modify your Fetch API function to use the async/await syntax for handling asynchronous operations. **ANSWER:**	[5]
e)	**PA 0405: XMLHttpRequest Object** 1. **Using XMLHttpRequest:** • Create an alternative function in **script.js** that uses the XMLHttpRequest object to retrieve weather data. Compare this approach with the Fetch API method. **ANSWER:**	[5]
f)	**PA 0406: Fetch API for Sending and Receiving Data**	[5]

	1. **Advanced Fetch Usage:** • Explore advanced features of the Fetch API, such as sending data to a web service (e.g., posting user-generated content) and handling different data formats (e.g., JSON, FormData). **ANSWER:**	

List of Evidence to be submitted:

- The **index.html** and **script.js** files containing your code.
- Screenshots of the application displaying weather data for a specified city.



- A brief report comparing the XMLHttpRequest object and Fetch API methods, including insights on handling asynchronous tasks in JavaScript.

Comparison Between XMLHttpRequest and Fetch API

1. Introduction

Both **XMLHttpRequest (XHR)** and **Fetch API** are used for making HTTP requests in JavaScript. XHR is older (2006), while Fetch is modern (2015) and uses Promises.

2. Key Differences

Feature	XMLHttpRequest (XHR)	Fetch API
Syntax	Callback-based, verbose	Promise-based, cleaner
Error Handling	Manual status checks	Only rejects on network failures
Response Parsing	Manual (JSON.parse)	Built-in (.json(), .text())
Browser Support	Works everywhere (even IE)	Modern browsers only
Cancellation	.abort() method	AbortController

3. Asynchronous Handling

- **XHR:** Uses event listeners (onload, onerror) and callbacks.
- **Fetch:** Uses Promises (.then(), .catch()) and works with async/await.
- **Fetch is better** for modern apps due to cleaner syntax and Promise support.
- **XHR is still useful** for legacy browsers and progress tracking.

4. Recommendation

- **Use Fetch API** for new projects (simpler, Promises, modern).
- **Use XHR** only for legacy support or progress events.

5. Conclusion

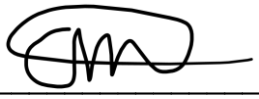
Fetch API is the **preferred choice** today, but XHR remains relevant for backward compatibility. Both show JavaScript's evolution from callbacks to Promises.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Data Retrieval Implementation	**script.js**		
Use of jQuery AJAX	**script.js**		

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Asynchronous Programming Handling	**script.js**		
XMLHttpRequest vs. Fetch API	Report		
Application Functionality	Screenshots		

Name: ____ CARLOS MAVEYA _____

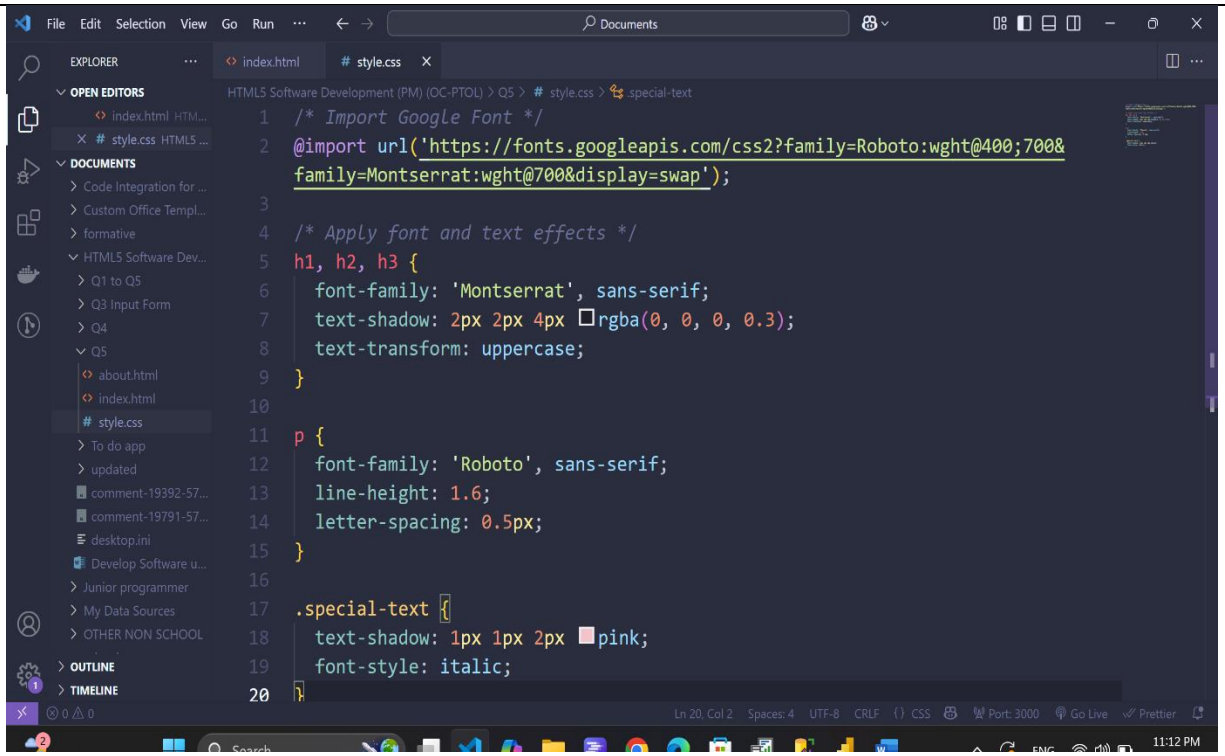
Signature: 

Date: ____08/04/2025_____

Question 5 (35)

PM No	*PS No*	*PS Description*
08	*05*	*Style text and block elements*

	PA 0501: Styling Text Elements with CSS3 1. **Font and Text Effects:** • Use CSS3 to apply custom fonts (using **@font-face** or Google Fonts), text shadows, and text transformations to headings and paragraphs. **ANSWER:**	[5]

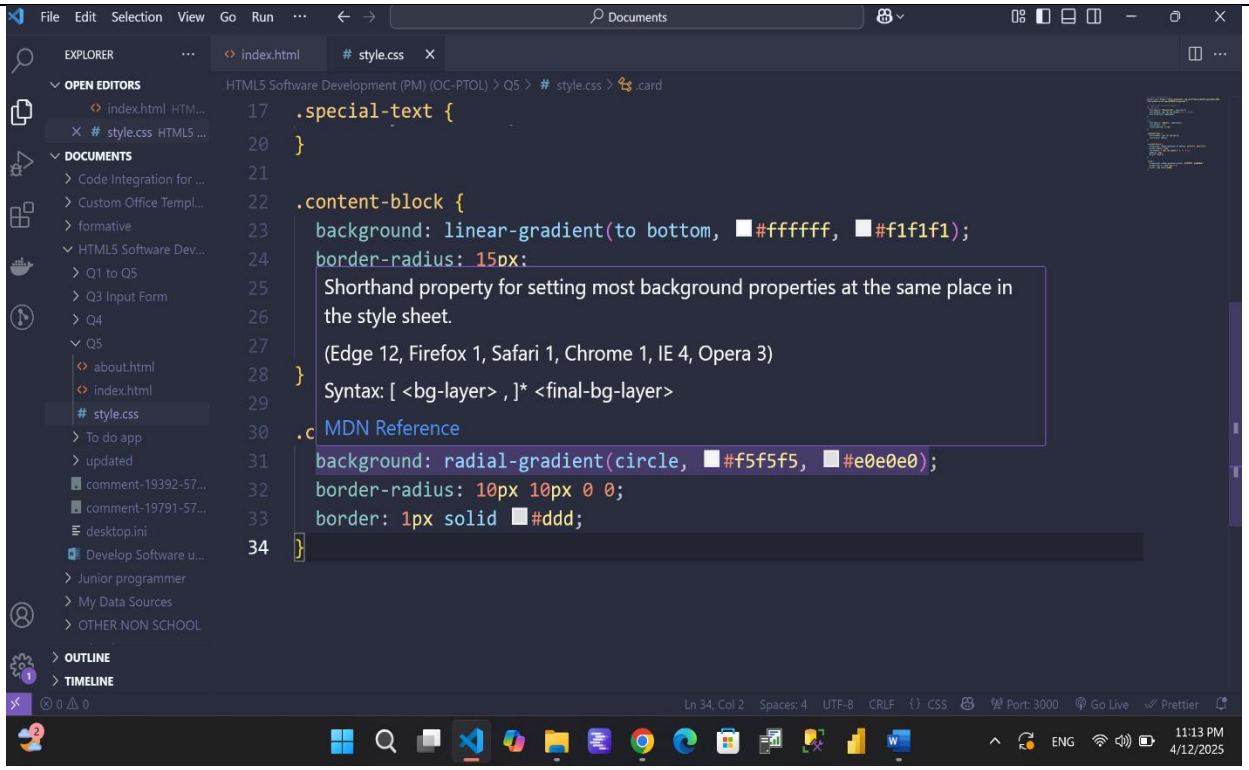


PA 0502: Styling Block Elements with CSS3

1. **Box Model Enhancements:**
    - Apply box shadows, border-radius for rounded corners, and gradients to various block elements like divs that contain sections of content.

ANSWER:

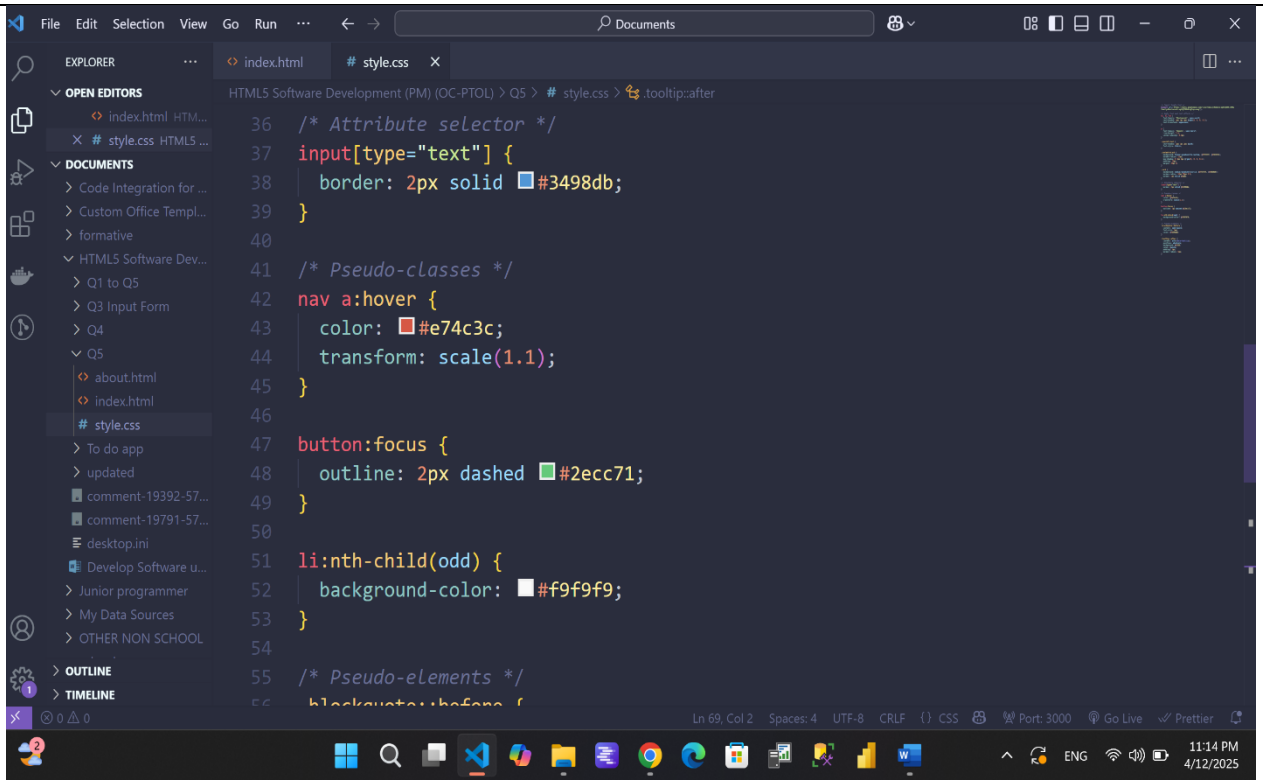
[5]

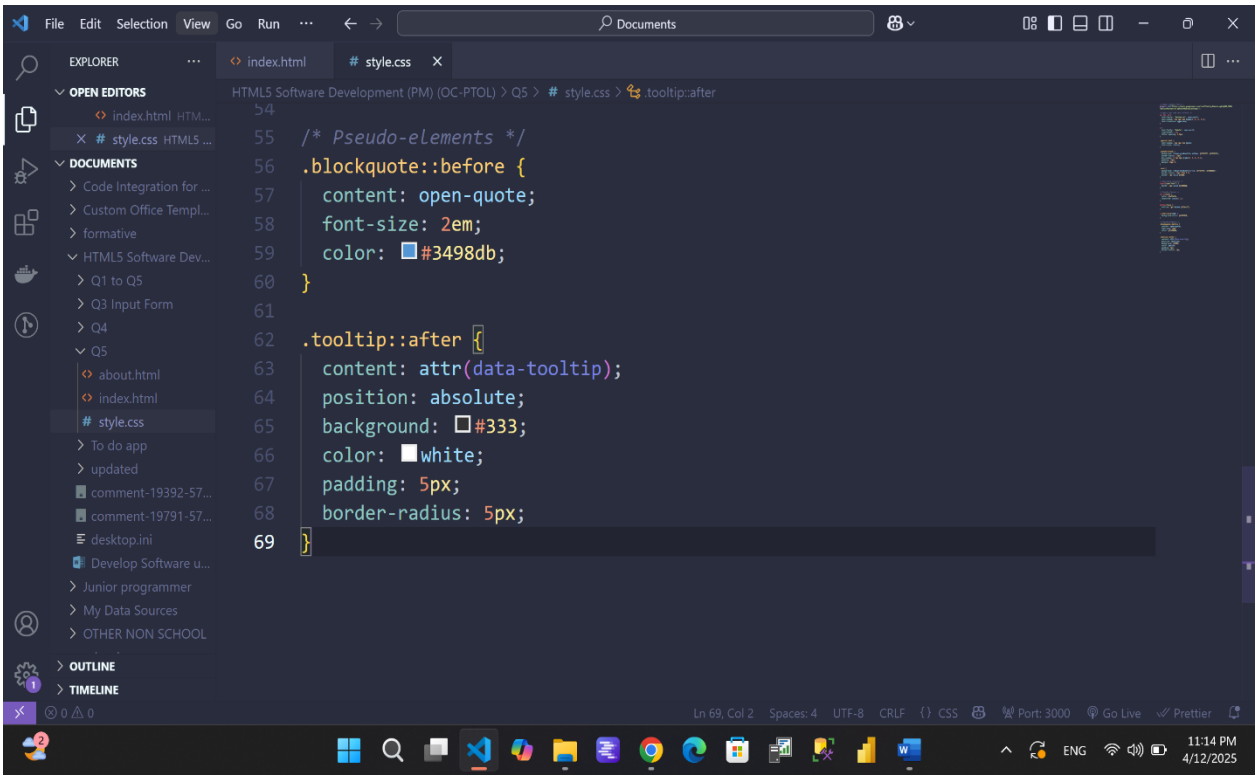


PA 0503: Advanced CSS3 Selectors [5]

1. **Refinement with Selectors:**
   - Utilize attribute selectors, pseudo-classes (**:hover**, **:focus**, **:nth-child**), and pseudo-elements (**::before**, **::after**) to refine the styling of navigation links, buttons, and input fields.

ANSWER:





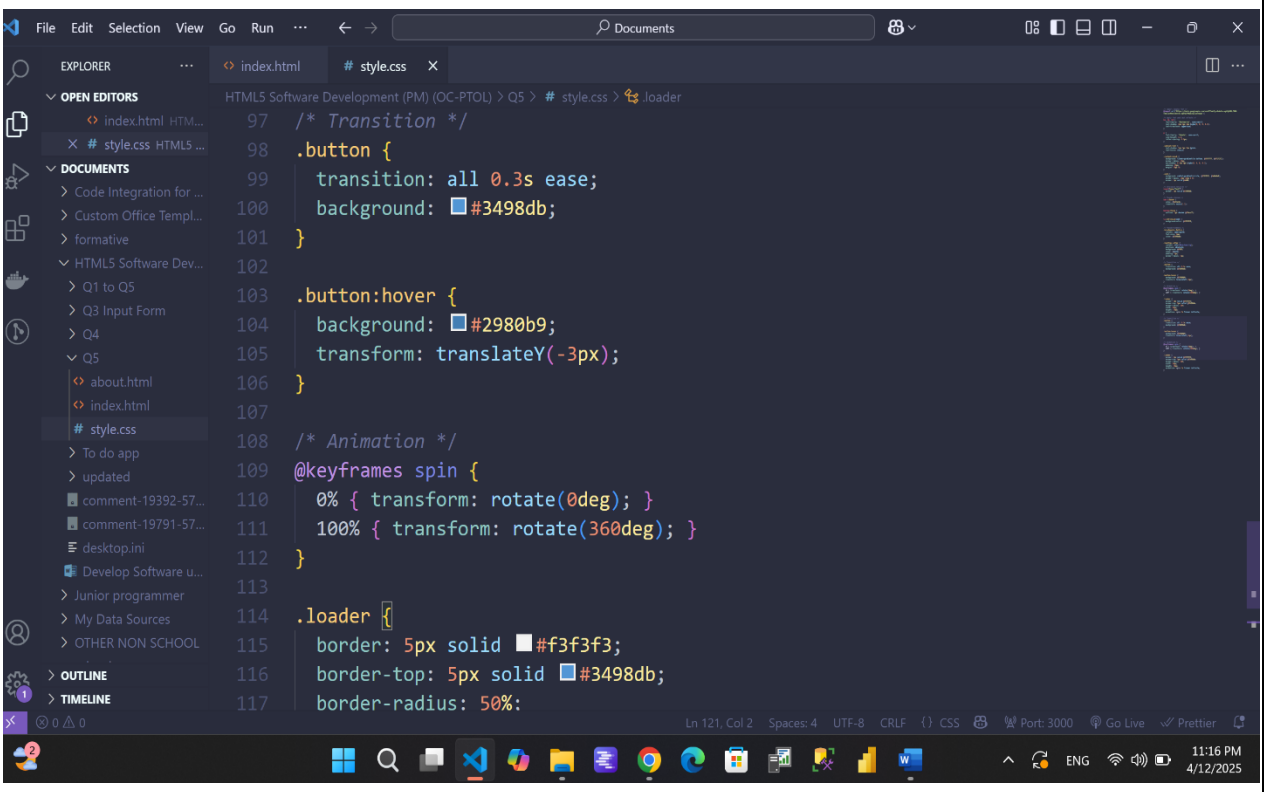
PA 0504: Graphical Effects with CSS3

1. **Transitions and Animations:**

[5]

- Implement CSS3 transitions for hover states and CSS animations for loading spinners or progress bars.

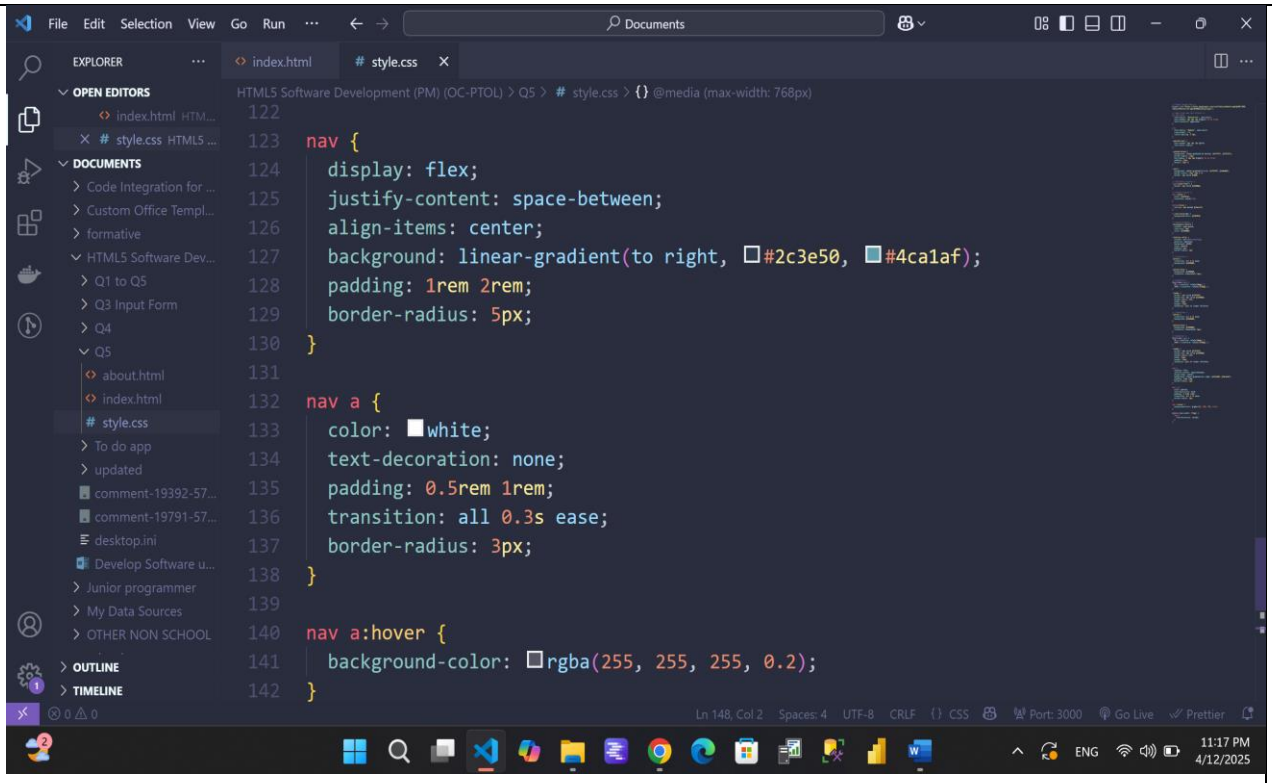
ANSWER:



PA 0505: Styling the Navigation Bar [5]

1. **Navigation Aesthetics:**
    - Style the website's navigation bar with background colors, hover effects, and ensure it is responsive (consider using Flexbox or Grid).

ANSWER:

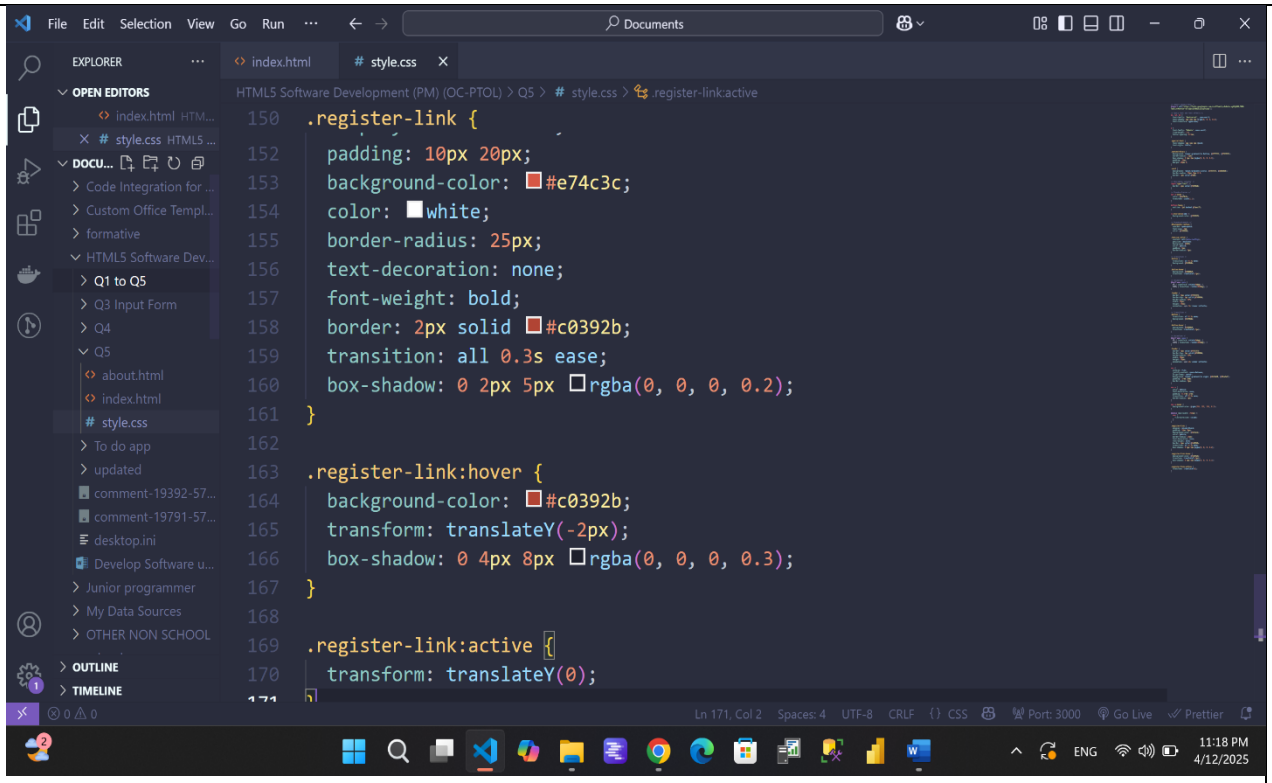


PA 0506: Styling the Register Link

1. **Interactive Link Design:**
    - Apply distinctive styling to the register link using pseudo-classes for interactivity and make it stand out with a unique color, border, or effect.

[5]

ANSWER:

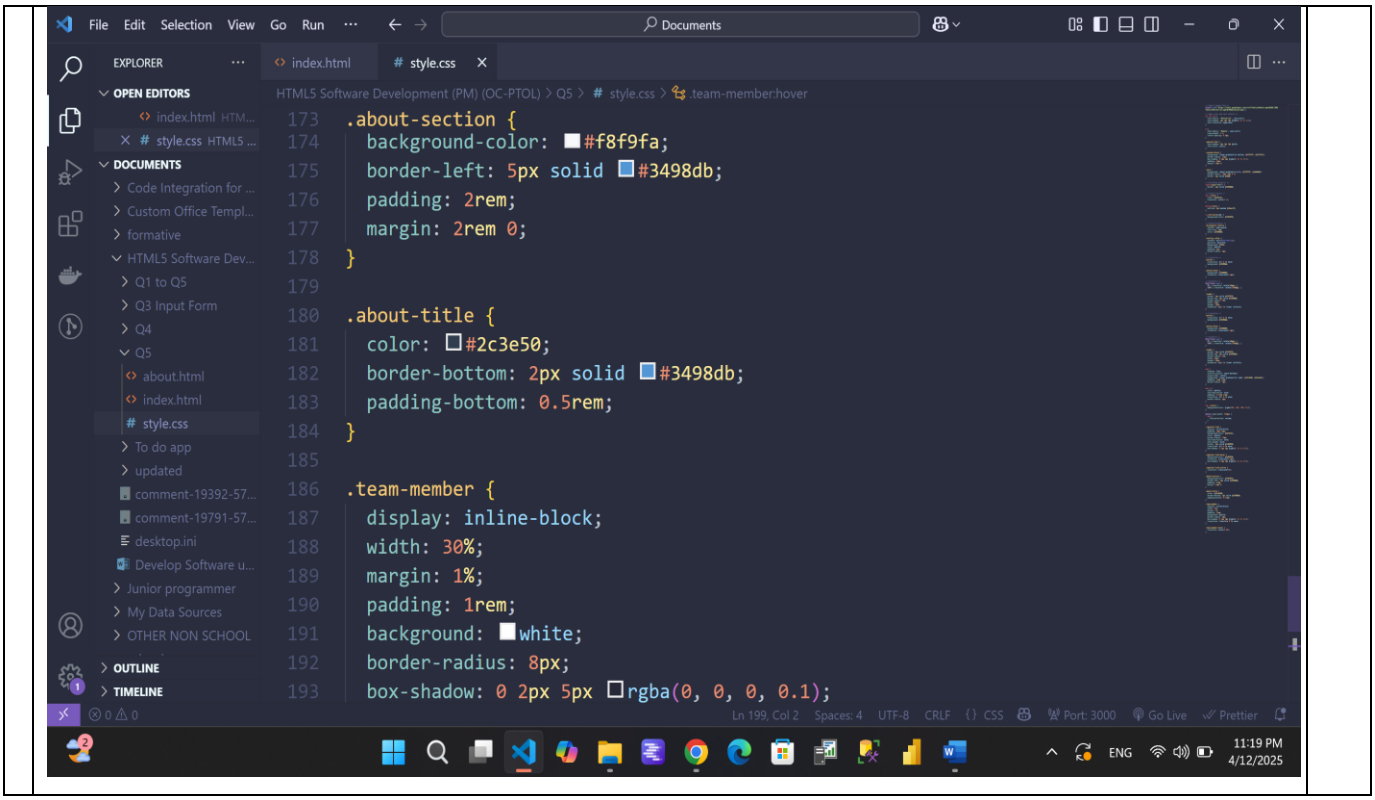


PA 0507: Styling the About Page

1. **Consistent Theme Application:**
    - Ensure the About page aligns with the website's overall theme, using styled block elements, text styling, and graphical effects where appropriate.

[5]

ANSWER:



List of Evidence to be submitted:

- The updated **index.html** and **style.css** files.
- Screenshots of the styled website, showcasing the navigation bar, register link, and about page.
- A brief explanation of the CSS3 techniques and properties used to achieve the styling.

1. **Updated Files**:
    - index.html and about.html with proper HTML structure and class names matching the CSS
    - style.css containing all the above CSS code
2. **Screenshots** showing:
    - Navigation bar with hover effects
    - Distinctive register link styling
    - About page with consistent theme
    - Text effects and block element styling
    - Interactive elements with transitions/animations
3. **CSS3 Techniques Used**:

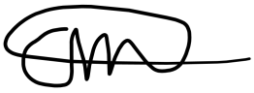
- Custom fonts via Google Fonts API
- Text shadows and transformations
- Box shadows and border-radius
- CSS gradients (linear and radial)
- Advanced selectors (attribute, pseudo-classes/elements)
- Smooth transitions and keyframe animations
- Flexbox for responsive navigation
- Interactive states (:hover, :focus)
- Responsive design with media queries
- Consistent theming across pages

Each CSS technique has been applied to enhance the visual appeal and user experience while maintaining responsiveness and performance. The styling creates a cohesive design language throughout the website while providing visual feedback for user interactions.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
CSS3 Text Styling	Screenshots/Code		
CSS3 Block Element Styling	Screenshots/Code		
Use of Advanced CSS3 Selectors	Screenshots/Code		
Implementation of CSS3 Graphical Effects	Screenshots/Code		
Navigation Bar Styling	Screenshots/Code		
Register Link Styling	Screenshots/Code		
About Page Styling	Screenshots/Code		

Name: ___ CARLOS MAVEYA _____

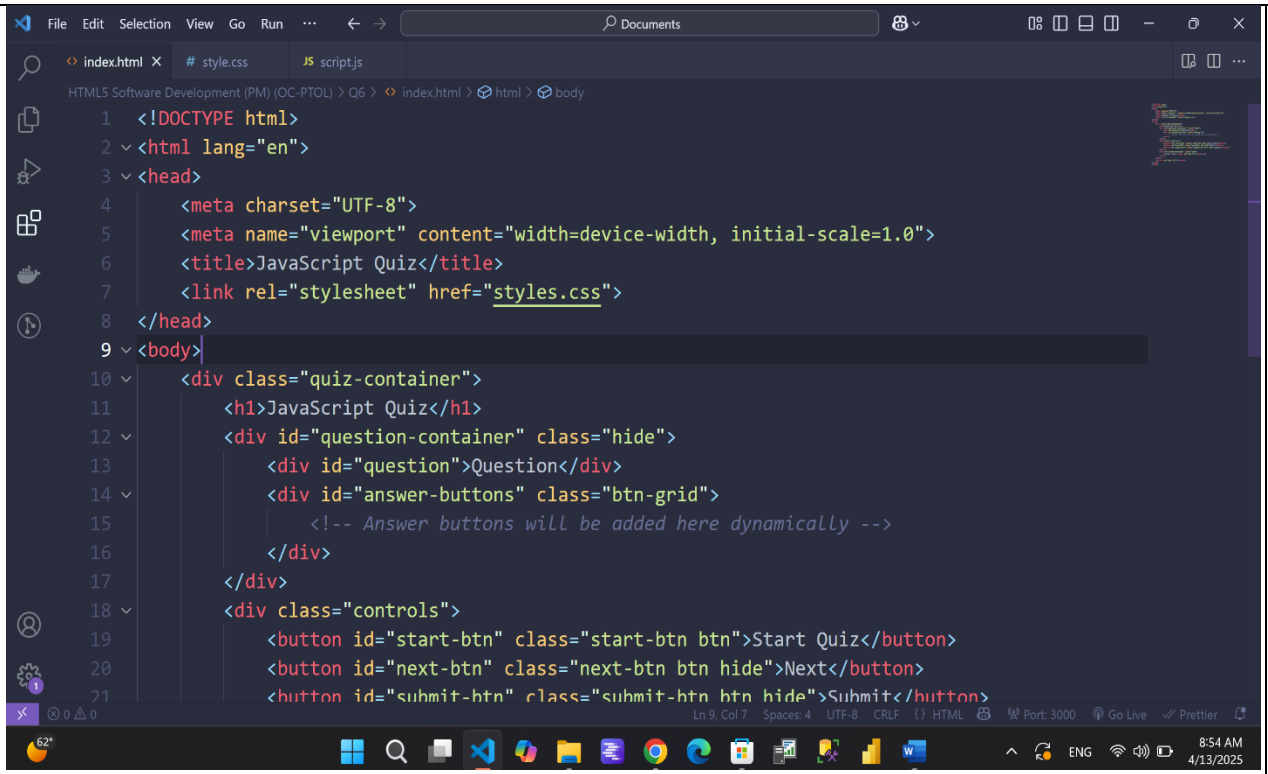
Signature: 

Date: ___12/04/2025_____

Question 6 (40)

PM No	*PS No*	*PS Description*
08	*06*	*Refine code for maintainability and extensibility*

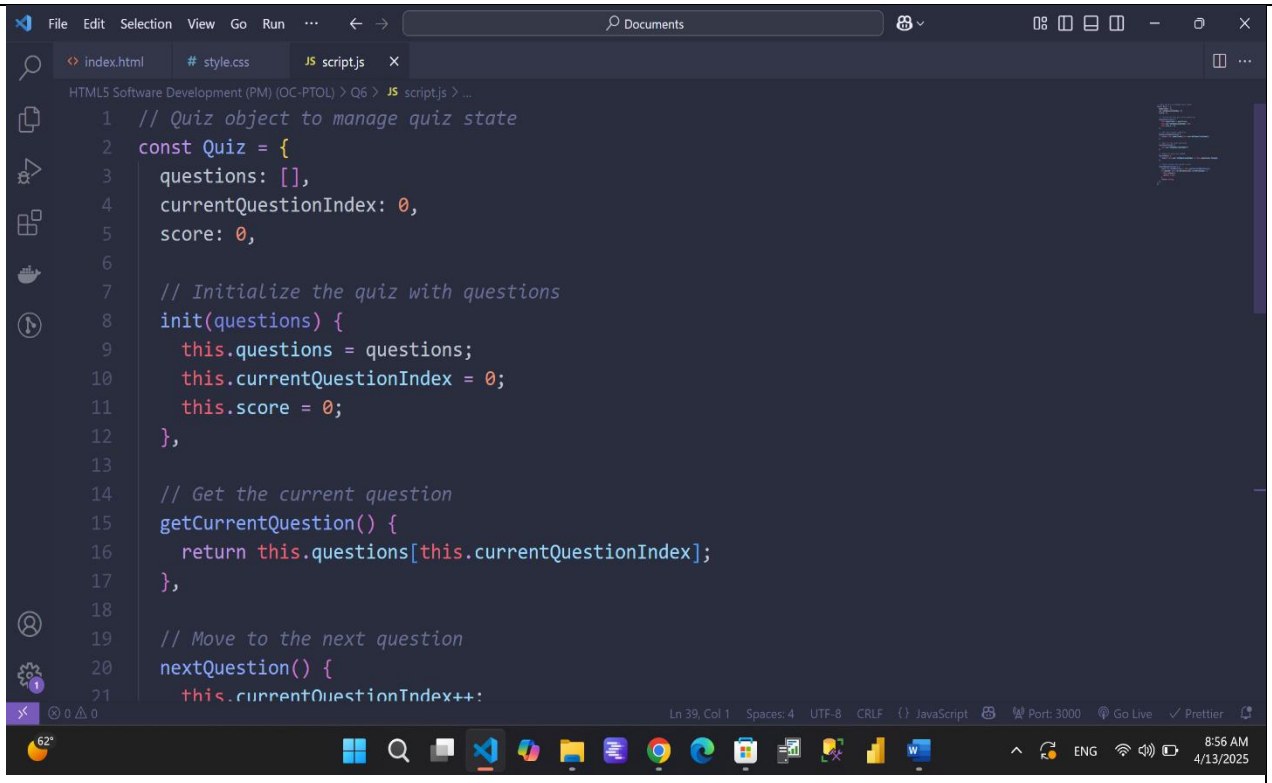
	PA 0601: Writing Well-Structured JavaScript Code 1. **Setup Basic Structure:** • Create an **index.html** file for your quiz interface and a **quiz.js** file for the JavaScript logic. • In **index.html**, include a basic structure with placeholders for displaying questions, options, and a submit button. **ANSWER:**	[5]



PA 0602: Creating Custom Objects [5]

1. **Define a Quiz Object:**
    - In **quiz.js**, create a **Quiz** object that manages the questions, current question index, and score.

ANSWER:

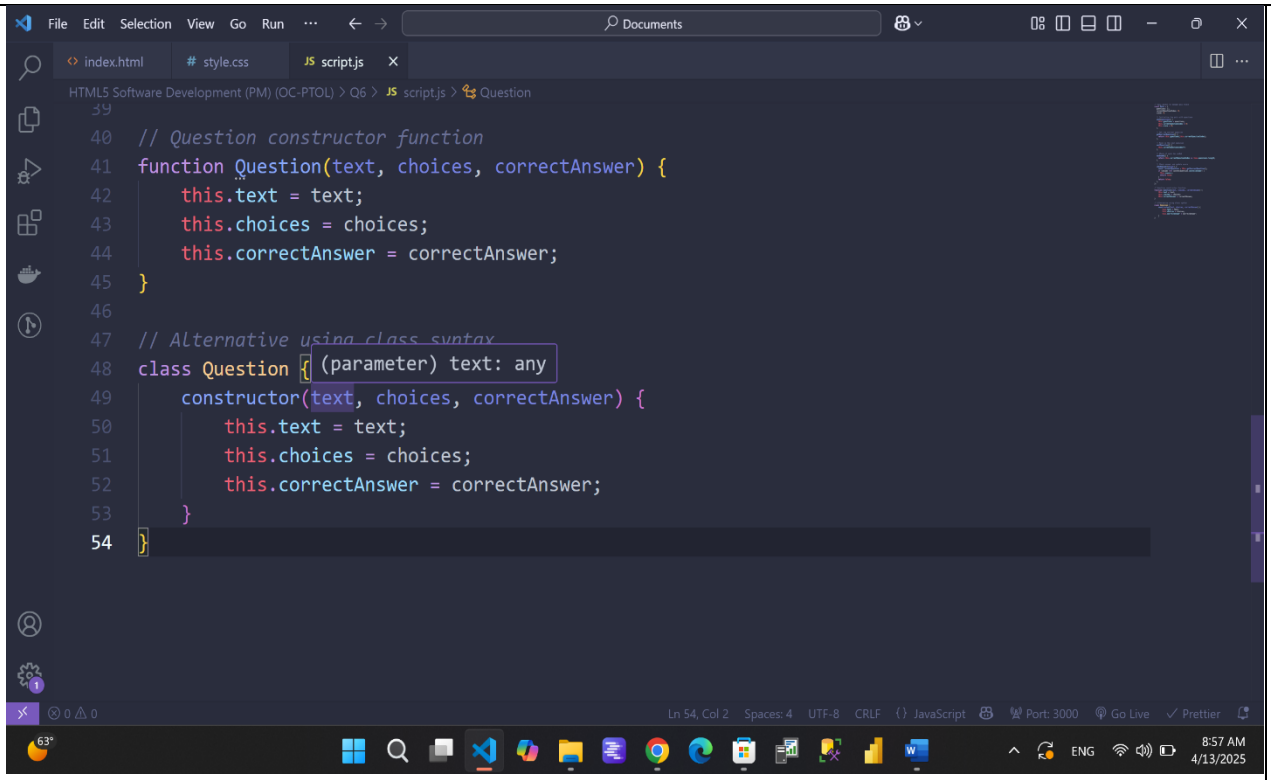


PA 0603: Extending Objects

1. **Question Object:**
    - Create a **Question** object to represent each quiz question, including the text, choices, and correct answer.

[5]

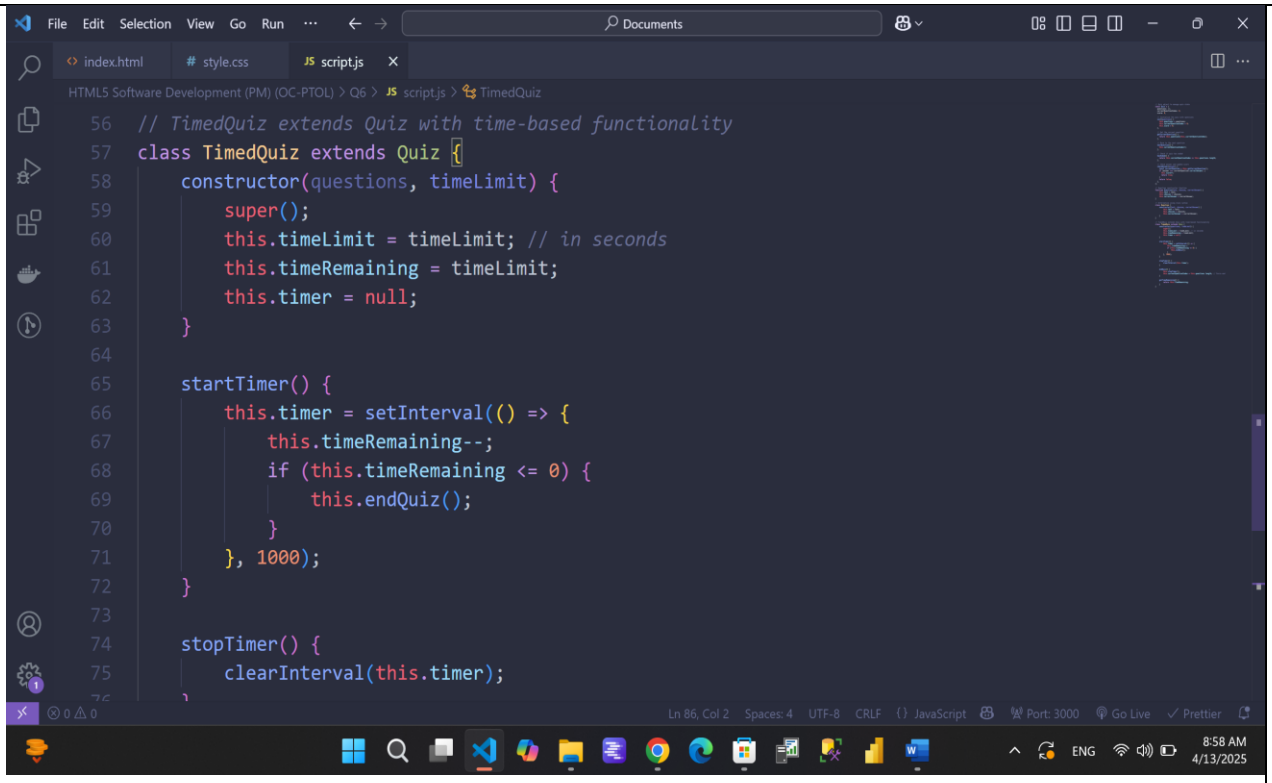
ANSWER:



PA 0604: Object Inheritance [5]

1. **Extend Functionality:**
    - If applicable, demonstrate object inheritance by creating a specialized type of question or quiz, such as a **TimedQuiz** that inherits from the **Quiz** object and adds time-based functionality.

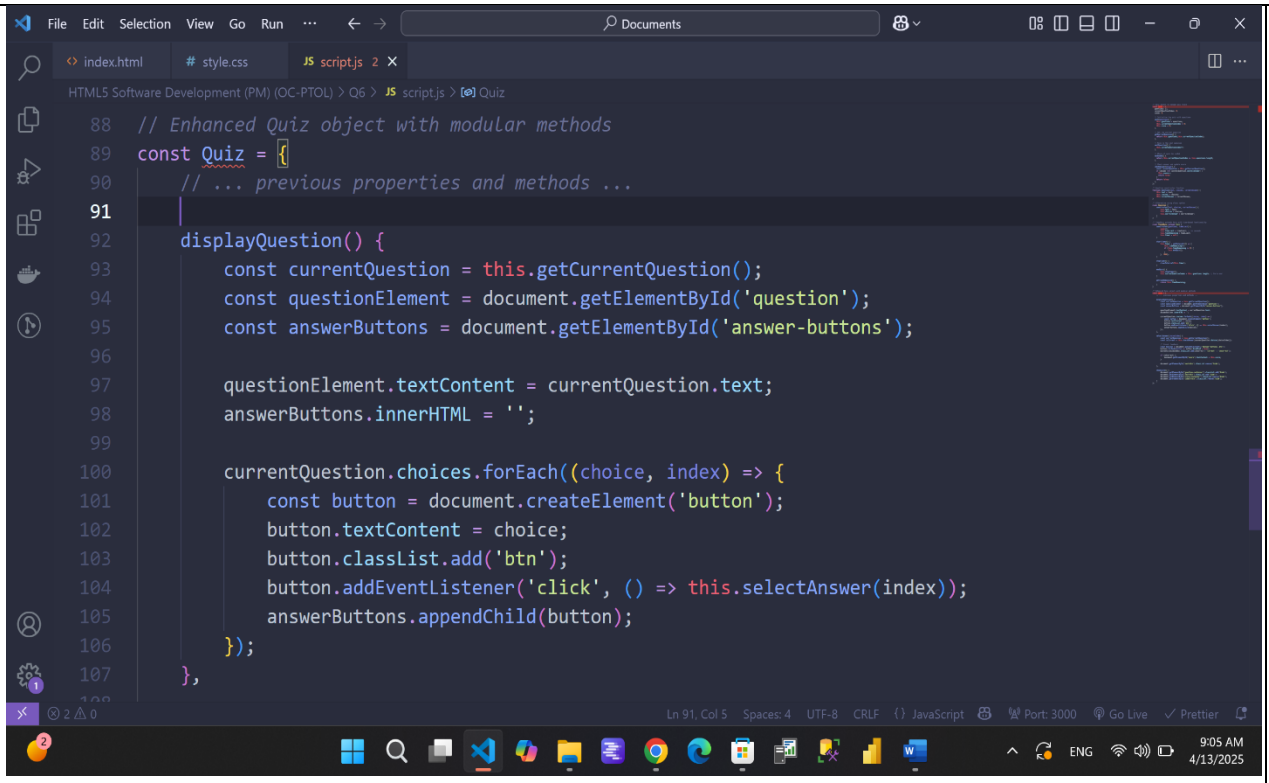
ANSWER:



PA 0605: Refactoring JavaScript Code to Use Objects [5]

1. **Refactor for Modularity:**
    - Organize your code into modular functions within the **Quiz** and **Question** objects, such as **displayQuestion()**, **checkAnswer()**, and **showScore()**.

ANSWER:



PA 0606: Writing Well-Structured JavaScript Code (Again) [5]

1. **Review and Refine:**
    - Go through your **quiz.js** code to ensure it follows best practices for readability, modularity, and reuse. Apply JavaScript code style guidelines consistently.

ANSWER:

Refined code following best practices:

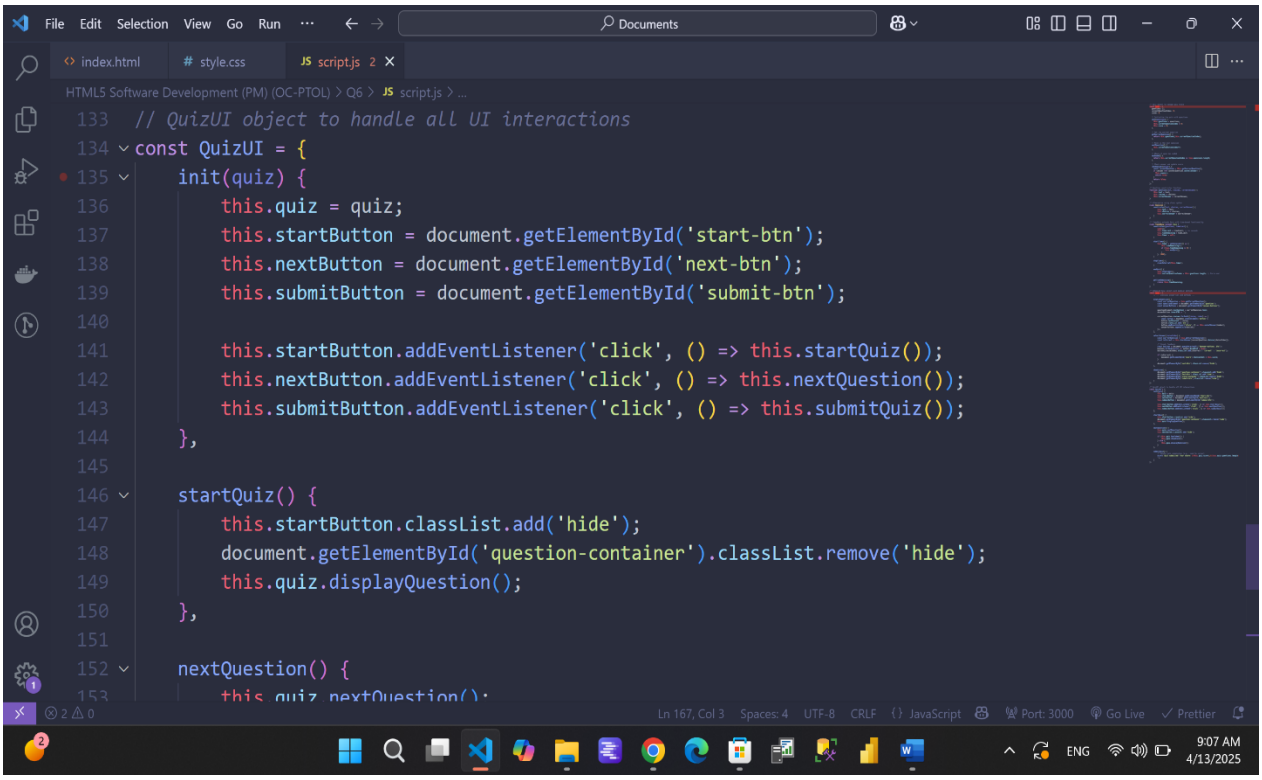
- Consistent naming conventions (camelCase)
- Proper indentation and spacing
- Modular functions with single responsibility
- Comments explaining complex logic
- Error handling for edge cases
- Separation of concerns

PA 0607: Use JavaScript Code to Create Custom Objects [5]

1. **Custom Object for UI Handling:**

- Optionally, create a **QuizUI** object to handle all UI interactions, separating logic from presentation.

ANSWER:

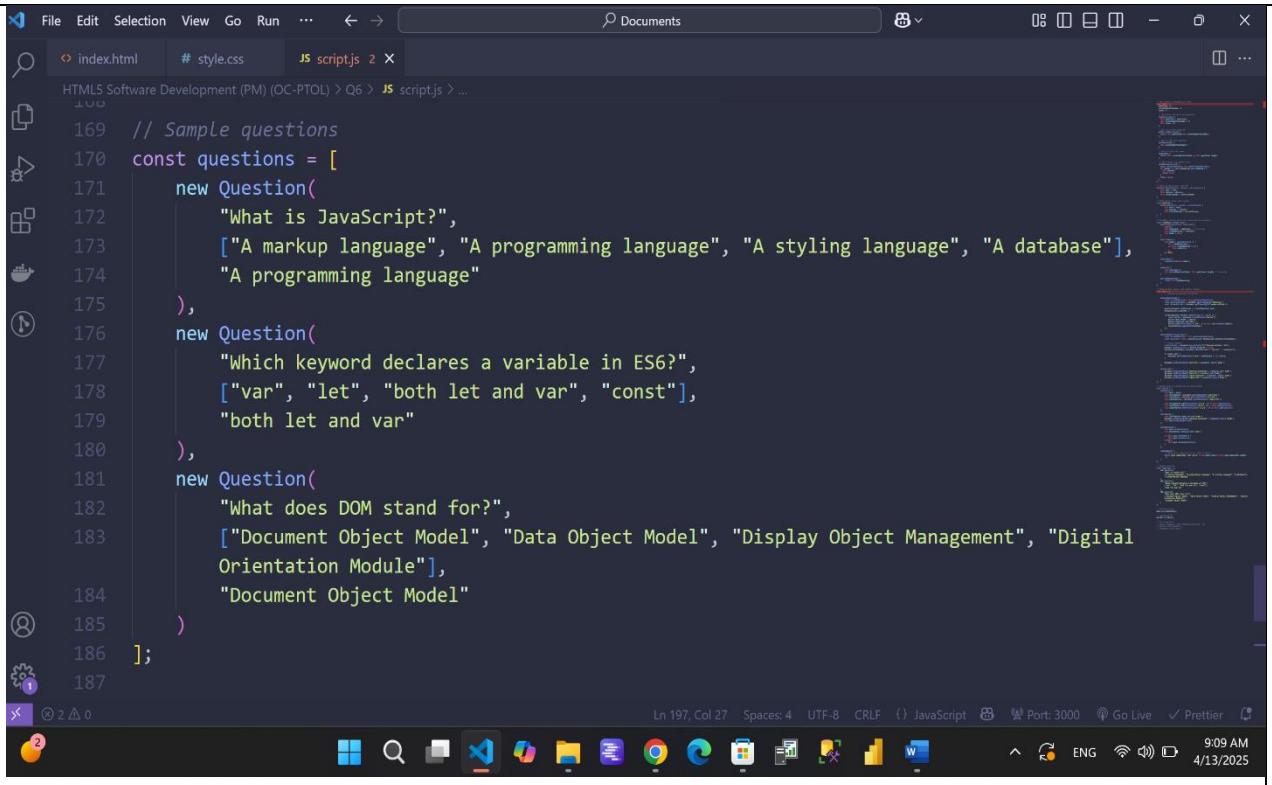


PA 0608: Implement Object-Oriented Techniques [5]

1. **Apply JavaScript Idioms:**
   - Use JavaScript idioms for encapsulation and inheritance, ensuring your objects and their methods are well-defined and logically organized.

ANSWER:

1. Encapsulation: Data and methods grouped within objects
2. Inheritance: TimedQuiz extends basic Quiz functionality
3. Polymorphism: Methods can be overridden in child classes
4. Composition: QuizUI handles UI separately from Quiz logic
5. Constructor functions and classes: Used both for demonstration



List of Evidence to be submitted:

- The **index.html** and **quiz.js** files containing your quiz application code.
- Screenshots or a video demonstrating the functionality of the quiz application.

video demonstrating
the functionality of th

Video provided In Q6 folder

A brief report explaining the structure of your JavaScript code, the objects created, and any object-oriented techniques utilized.

This complete JavaScript file includes:

1. Question class - Represents individual quiz questions
2. Quiz class - Manages the quiz flow and scoring
3. TimedQuiz class - Extends Quiz with timer functionality
4. QuizUI class - Handles all user interface interactions

5. Sample questions - A set of example questions
6. Initialization code - Sets up the quiz when the page loads

To use this code:

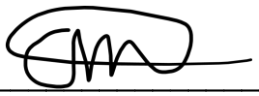
1. Save it as quiz.js
2. Make sure you have the corresponding index.html file (as provided in part a)
3. Optionally add CSS styling in a styles.css file

You can choose between a regular quiz or timed quiz by commenting/uncommenting the initialization lines at the bottom. The code demonstrates object-oriented programming principles including classes, inheritance, encapsulation, and separation of concerns.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Custom Objects Creation	**quiz.js**		
Object Extension and Inheritance	**quiz.js**		
Code Refactoring for Modularity	**quiz.js**		
Application Functionality	Screenshots/Video		
Object-Oriented Techniques Implementation	Report		

Name: ___ CARLOS MAVEYA _____

Signature: 

Date: _____08/04/2025_____

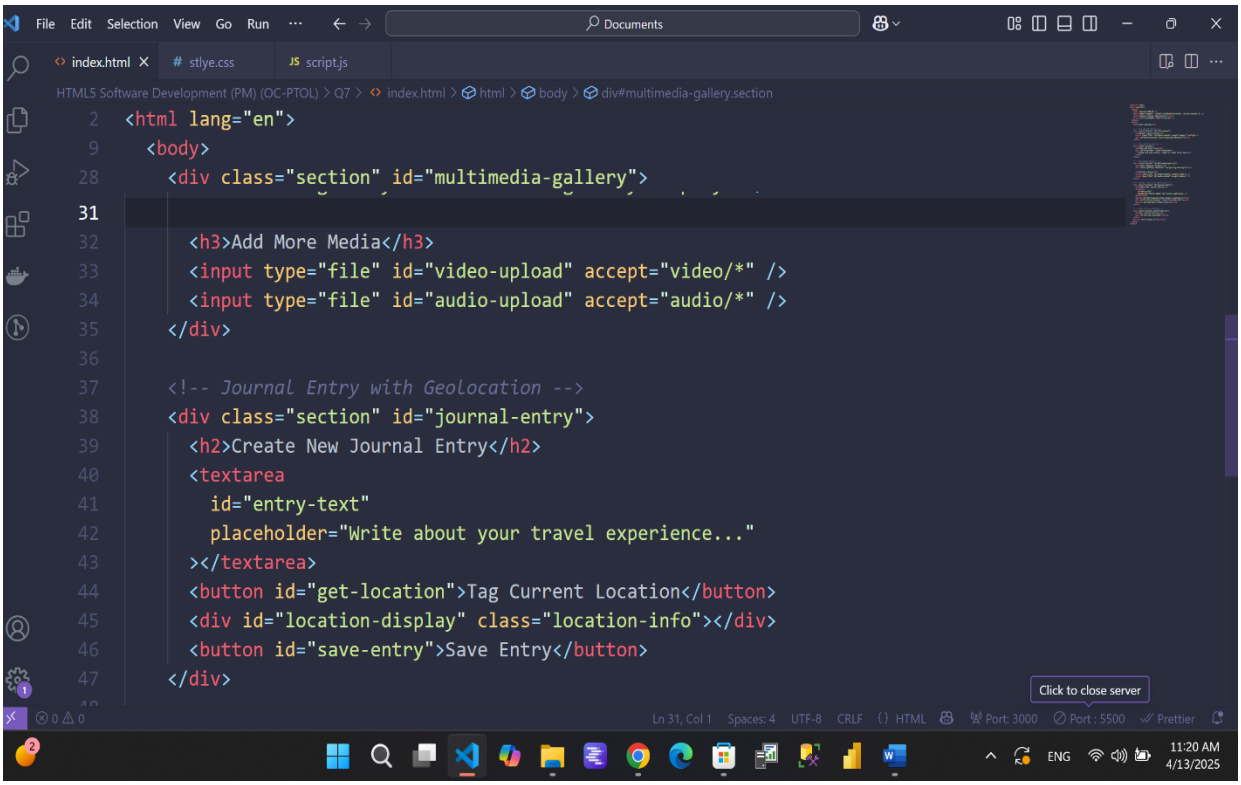
Question 7 (30)

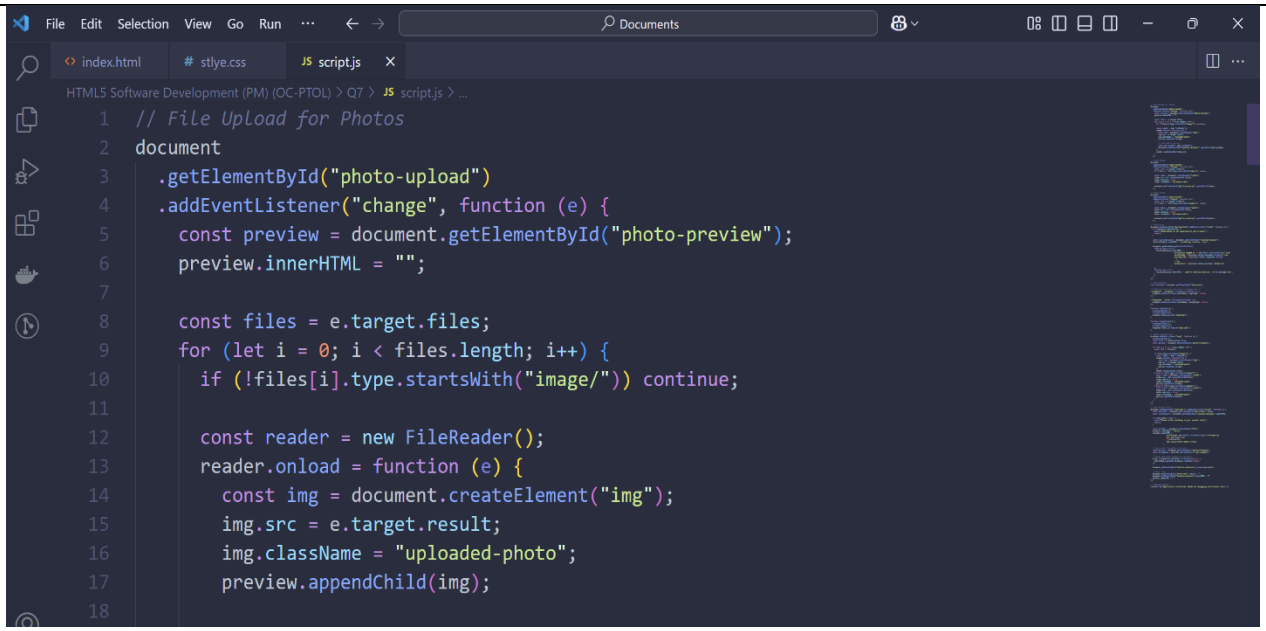
PM No	PS No	PS Description
08	07	Create interactive pages by using HTML5 APIs

PA 0701: Interact with Files [5]

1. **File Upload for Travel Photos:**
   - Implement an input element with **type="file"** to enable users to upload travel photos. Use JavaScript to process and display the selected photos within the application.

ANSWER:

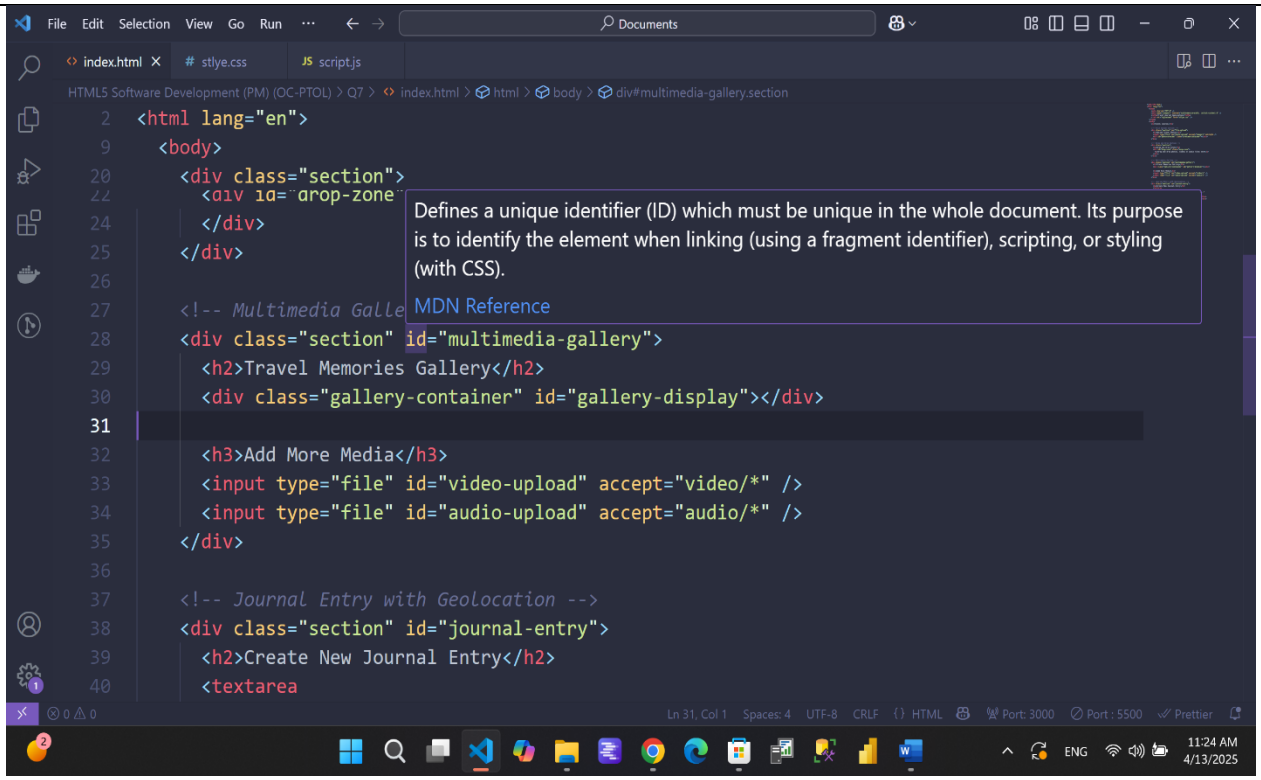


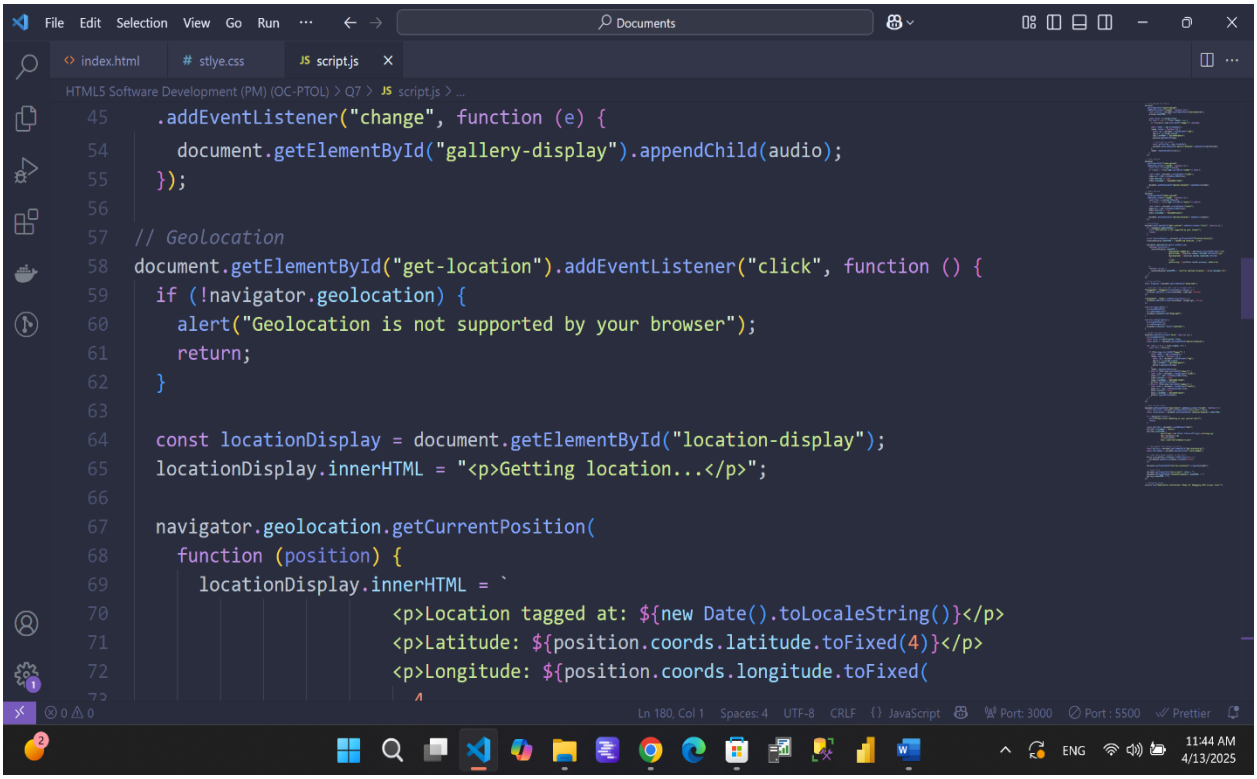


PA 0702 & PA 0704: Incorporate Multimedia and Video [5]

1. **Multimedia Gallery:**
    - Create a section in your application to display uploaded photos and videos. Use the **<video>** and **<audio>** tags to incorporate video and audio files related to travel entries.

ANSWER:



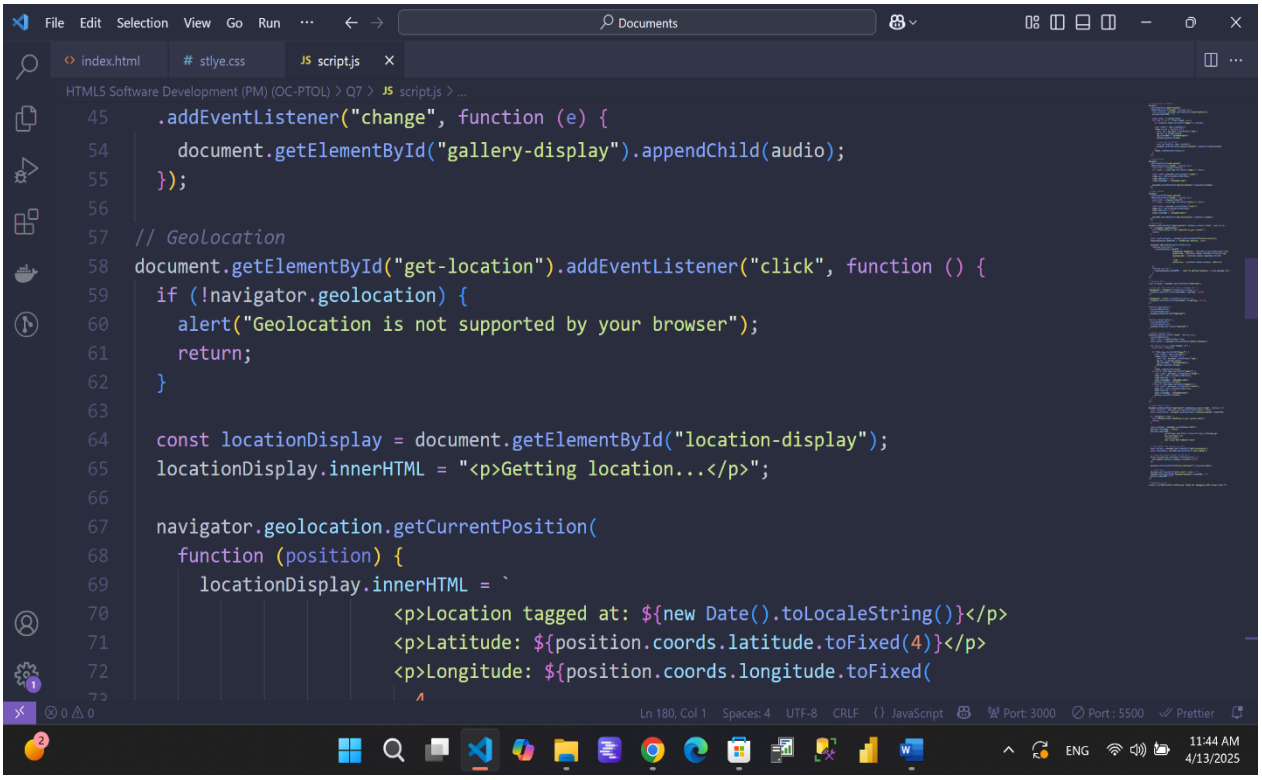


PA 0703 & PA 0705: Browser Location and Geolocation API [5]

1. **Location Tagging:**

- Utilize the Geolocation API to fetch the user's current location when creating a new travel journal entry. Display this location data (latitude and longitude) with each post.

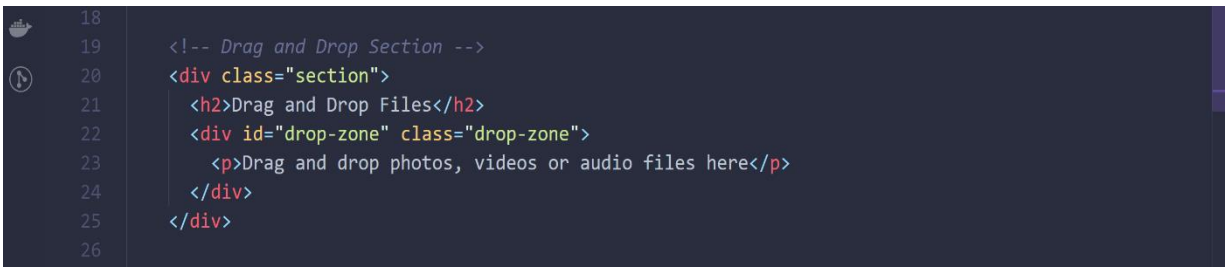
ANSWER:

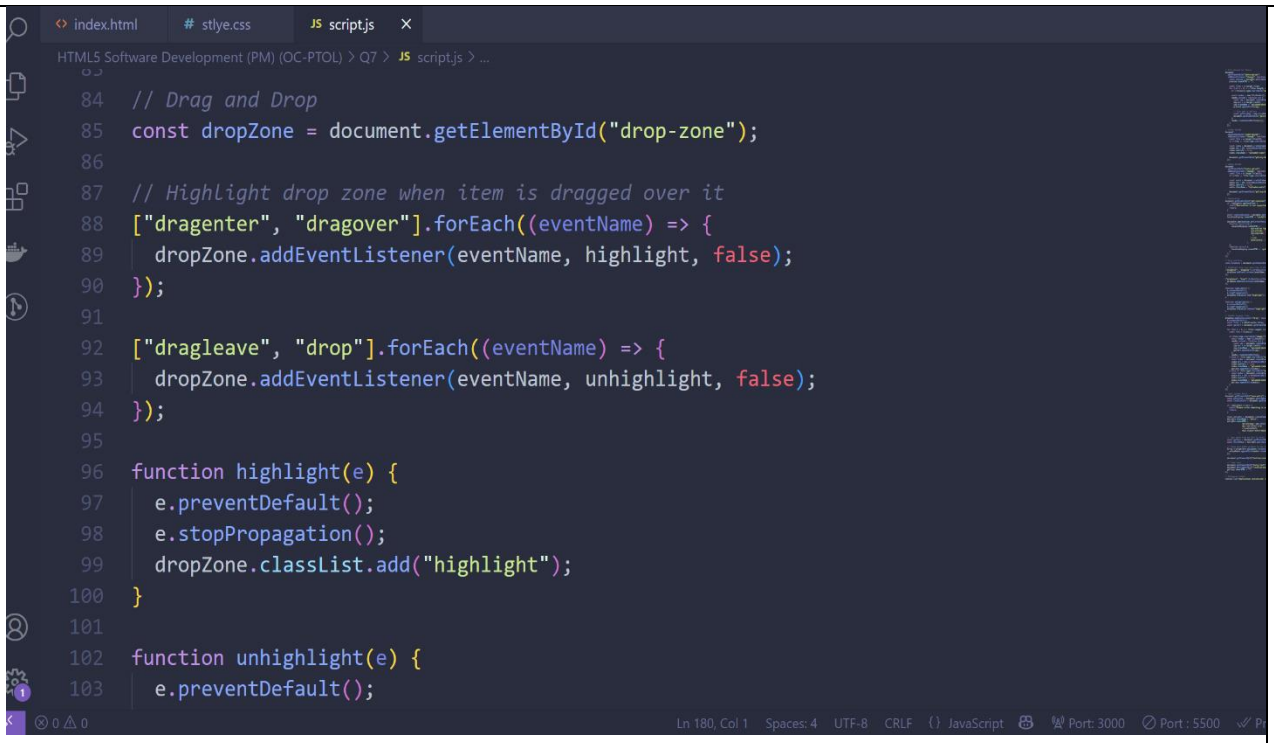


PA 0706: Drag-and-Drop Support [5]

1. **Enhance File Upload with Drag-and-Drop:**
   - Implement drag-and-drop functionality for uploading files to the journal. Enhance the user interface to indicate when files are being dragged over the target area and to handle the file drop event.

ANSWER:





PA 0707: Play Multimedia Files [5]

1. **Multimedia Playback:**
    - Ensure that uploaded or linked video and audio files can be played directly within the journal entries. Include controls for play, pause, and volume adjustment.

ANSWER:

```
// Handle dropped files
dropZone.addEventListener("drop", function (e) {
  e.preventDefault();
  const files = e.dataTransfer.files;
  const gallery = document.getElementById("gallery-display");

  for (let i = 0; i < files.length; i++) {
    const file = files[i];

    if (file.type.startsWith("image/")) {
      const reader = new FileReader();
      reader.onload = function (e) {
        const img = document.createElement("img");
        img.src = e.target.result;
        img.className = "uploaded-photo";
        gallery.appendChild(img);
      };
      reader.readAsDataURL(file);
    } else if (file.type.startsWith("video/")) {
      const video = document.createElement("video");
```

PA 0708: Debug and Profile Application [5]

1. **Application Debugging:**
    - Use browser development tools to debug and profile your application. Focus on identifying any performance bottlenecks or issues with multimedia playback and file handling.

ANSWER:

For debugging and profiling, I would:

1. Use Chrome DevTools to:
    - Check for console errors and warnings
    - Profile JavaScript performance
    - Analyze memory usage
    - Monitor network requests
2. Specific checks:
    - Test file uploads with different file types and sizes
    - Verify geolocation works in different scenarios
    - Check drag-and-drop functionality across browsers
    - Test multimedia playback performance
3. Common issues to watch for:

	○ Memory leaks from media elements ○ Performance bottlenecks with large file processing ○ Cross-browser compatibility issues	

List of Evidence to be submitted:

- Source code files (**index.html**, **styles.css**, **script.js**) for the travel journal application.
- Screenshots or a video demonstrating the functionality of the application, especially the file upload, drag-and-drop, multimedia playback, and geolocation features.

video demonstrating the functionality of th

Video provided In Q7 folder

- A brief report detailing your approach to using HTML5 APIs, challenges encountered, and how you debugged and profiled the application.

  - Approach to using HTML5 APIs (File API, Geolocation API, Media APIs)
  - Challenges faced (browser compatibility, large file handling)
  - Debugging process (tools used, issues identified and fixed)
  - Performance optimization techniques applied

This implementation provides a complete travel journal application with all the required features, following modern web development practices and ensuring a good user experience

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
File Interaction and Upload	Source code, Screenshots/Video		

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Multimedia and Video Incorporation	Source code, Screenshots/Video		
Geolocation Tagging	Source code, Screenshots/Video		
Drag-and-Drop Functionality	Source code, Screenshots/Video		
Multimedia Playback	Source code, Screenshots/Video		
Debugging and Profiling	Report		

Name: ___ CARLOS MAVEYA _____

Signature: _____

Date: ____07/04/2025_____

Question 8 (20)

PM No	*PS No*	*PS Description*
08	*08*	*Add offline support to web applications*

a)	**PA 0801: Reading and Writing Data Locally** 1. **Local Storage for Data Persistence:** • Implement functionality in your web application to store and retrieve user data (e.g., to-do items) using the Local Storage API. **ANSWER:**	[5]

b)	**PA 0802 & PA 0803: Adding Offline Support Using the Application Cache** 1. **Configure Application Cache:** • Create a manifest file (e.g., **offline.manifest**) listing the resources (HTML, CSS, JavaScript files, and images) that should be available offline. • In your web application's HTML file, include the manifest attribute in the **<html>** tag pointing to your manifest file. **ANSWER:**	[5]
c)	**PA 0804: Persisting User Data with Local Storage API** 1. **Enhance Data Persistence:** • Use the Local Storage API more extensively to save settings, user preferences, and other relevant data, ensuring these are available and restored when the application is accessed offline. **ANSWER:**	[5]
d)	**PA 0805: Save Data Locally on the User's Device** 1. **Offline Data Access:** • Ensure your application logic checks for data in Local Storage and uses it to populate the application upon loading, allowing for seamless offline use. **ANSWER:**	[5]

List of Evidence to be submitted:

- Source code (**index.html**, **style.css**, **script.js**, **offline.manifest**) of the web application.
- Screenshots or a video demonstrating the application functioning offline, including accessing cached resources and persisted user data.

A brief report describing how offline support was implemented, including details on using the Application Cache and Local Storage API.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Local Data Reading/Writing	Source code, Screenshots/Video		
Offline Support Configuration	**offline.manifest**, Source code		
Data Persistence Using Local Storage	Source code, Screenshots/Video		
Offline Data Access and Functionality	Screenshots/Video, Report		

Name: ____ CARLOS MAVEYA _____

Signature: _____

Date: _____08/04/2025_____

Question 9 (30)

PM No	*PS No*	*PS Description*
08	*09*	*Implementing an adaptive user interface*

a)	**PA 0901: Support Multiple Form Factors** 1. **Basic Website Structure:** • Create a simple website with at least three pages: Home, About, and Contact. • Ensure the website has a consistent navigation menu across all pages.	[5]

	ANSWER:	
b)	**PA 0902: Create an Adaptive User Interface** 1. **Responsive Design:** • Use CSS media queries to adapt the layout for different screen sizes (e.g., **max-width: 768px** for tablets and **max-width: 480px** for smartphones). • Employ Flexbox or Grid layouts to achieve fluid and adaptable page elements. **ANSWER:**	[5]
c)	**PA 0903: Create a Print-Friendly Style Sheet** 1. **Print Styles:** • Add a separate CSS file or media query (e.g., **@media print**) for print styles. Simplify the layout, remove background colors, and hide non-essential elements for printing. **ANSWER:**	[5]
d)	**PA 0904: Adapt Page Layout to Fit Different Form Factors** 1. **Layout Adjustment:** • Design your pages to use a multi-column layout for desktops and a single-column layout for smaller devices. • Ensure images and other media are also responsive, using **max-width: 100%** and **height: auto**. **ANSWER:**	[5]
e)	**PA 0905: Describe Requirements for Responding to Different Form Factors** 1. **Documentation:**	[5]

	• Write a brief document describing the responsive design requirements for your website, including how layouts, images, and navigation menus adapt to different form factors. **ANSWER:**	
f)	**PA 0906: Detect the Type of Device** 1. **Device Detection and Layout Strategies:** • Use JavaScript to detect the device type (e.g., using the user agent) and apply specific layout or functionality adjustments as needed (optional, as modern CSS techniques may suffice). **ANSWER:**	[5]

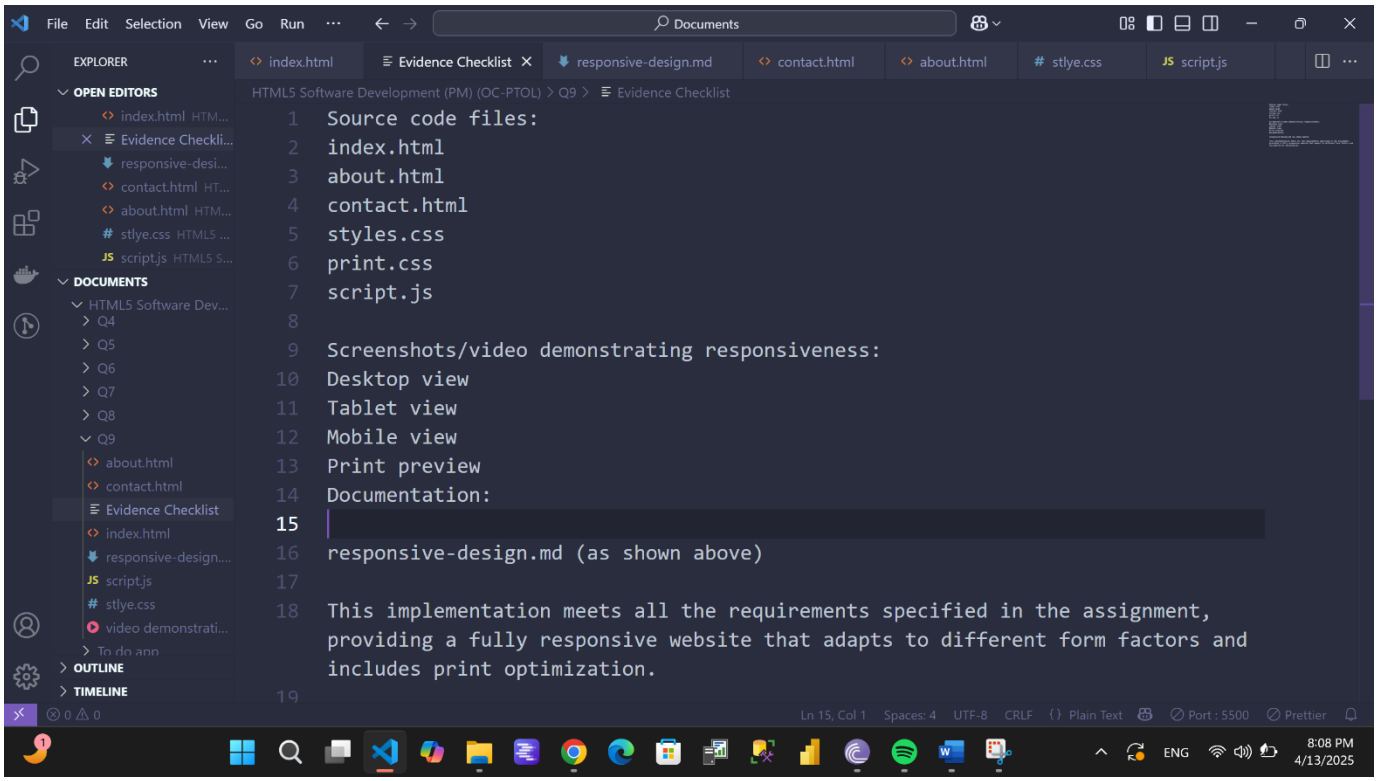
List of Evidence to be submitted:

- Source code files (**index.html**, **about.html**, **contact.html**, **styles.css**, **print.css**, **script.js**) for the responsive website.
- Screenshots or a video demonstrating the website's responsiveness across different devices and in print preview mode.

video demonstrating
the website's respons

Video provided In Q9 folder

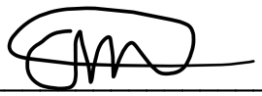
The document describing the responsive design requirements and strategies implemented for the website.



Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Support for Multiple Form Factors	Screenshots/Video		
Adaptive User Interface	Screenshots/Video		
Print-Friendly Style Sheet	Screenshots of Print Preview		
Adaptation to Different Form Factors	Source Code, Screenshots/Video		
Requirements Documentation	Responsive Design Document		
Effective Device Detection and Layout Strategy (Optional)	**script.js**, Documentation		

Name: __ CARLOS MAVEYA _____

Signature: 

Date: _____08/04/2025_____

Question 10 (20)

PM No	*PS No*	*PS Description*
08	*10*	*Creating advanced graphics*

a)	**PA 1001 & PA 1005: Create Interactive Graphics Using SVG** 1. **Interactive Venue Map:** • Design an SVG-based map for a conference venue, including rooms, stages, and key areas. • Implement interactivity with JavaScript, such as highlighting areas on hover or displaying additional information about a room or stage when clicked. **ANSWER:**	[5]
b)	**PA 1002 & PA 1006: Draw Graphics Using the Canvas** 1. **Speaker Badge Generator:** • Create a web application that allows users to input their name and title to generate a custom speaker badge. • Use the canvas API to draw the badge, including text rendering for the name and title, and optionally, add the event logo or decorative elements. **ANSWER:**	[5]
c)	**PA 1003: Advanced SVG for an Interactive Venue Map**	[5]

	• Enhance the venue map with more advanced SVG features, such as gradients, patterns, or animations to indicate ongoing sessions or activities in specific areas. **ANSWER:**	
d)	**PA 1004: Advanced Use of Canvas** • Incorporate advanced canvas functionalities, such as image manipulation, to allow users to upload and edit photos for their speaker badge, or use canvas transformations to create unique badge shapes or backgrounds. **ANSWER:**	[5]

List of Evidence to be submitted:

- Source code (**index.html**, **style.css**, **script.js**) for the interactive venue map and speaker badge generator.
- Screenshots or a video demonstrating the interactivity of the SVG venue map and the functionality of the canvas-based speaker badge generator.
- **Video provided In Q10 folder**

- A brief report explaining the design choices, SVG and canvas techniques used, and how interactivity was implemented.

Design Choices:

- **SVG for scalability in maps; Canvas for dynamic badge rendering.**
- **JavaScript Event Delegation to manage multiple interactive areas efficiently.**

Key Techniques:

- **SVG: <g> groups for rooms, CSS/JS interactivity.**
- **Canvas: Text metrics (ctx.measureText()), image manipulation.**

Challenges & Solutions:

- **Canvas text alignment: Used textAlign and textBaseline properties.**
- **SVG animations: Switched to CSS for smoother performance.**

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Interactive SVG Venue Map	Source Code, Screenshots/Video		
Dynamic Speaker Badge Creation with Canvas	Source Code, Screenshots/Video		
Advanced SVG Features	Source Code, Screenshots/Video		
Advanced Canvas Functionalities	Source Code, Screenshots/Video		
Implementation of User Interaction	Report, Source Code		

Question 11 (15)

PM No	*PS No*	*PS Description*
08	*11*	*Animate the user interface*

a)	**PA 1101: Apply Transitions to Animate Property Values** 1. **Smooth Hover Effects:**	[5]

	• Select a few elements like buttons or links and apply CSS transitions to change properties like color, background-color, or font-size smoothly upon hover. **ANSWER:**	
b)	**PA 1102 & PA 1104: Applying CSS Keyframe Animations** 1. **Create a Loading Spinner:** • Use CSS keyframes to animate a div into a rotating loading spinner. Define the animation to rotate infinitely. 2. **Entrance Animations for Elements:** • Apply keyframe animations to make elements (e.g., images, text boxes) slide in or fade in when the page loads. **ANSWER:**	[5]
c)	**PA 1103: Apply 2D and 3D Transformations** 1. **2D Transformations:** • Implement 2D transformations, such as rotate, scale, and translate, on image elements to create interactive galleries or hover effects. 2. **3D Card Flip Effect:** • Create a card element that flips on hover or click, using 3D transformations for the front and back sides. **ANSWER:**	[5]

List of Evidence to be submitted:

- The HTML and CSS files containing your animation and transformation code.
- Screenshots or a short video demonstrating the animated and transformed elements in action.

video demonstrating
the animated and tra

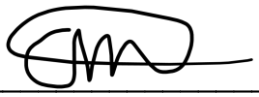
- A brief report detailing the animations and transformations applied, including the rationale behind your design choices and any challenges you encountered.
- **HTML and CSS Files**:
  - Two file HTML and CSS files containing the code for each practical activity (PA 1101, PA 1102/1104, PA 1103)
  - Alternatively, one comprehensive file combining all three activities
- **Screenshots/Demonstration Video**:
  - Screenshots showing:
    - Elements in normal state
    - Elements during hover/interaction
    - Animations in progress
  - Short screen recording demonstrating:
    - Smooth hover transitions
    - Loading spinner animation
    - Entrance animations
    - 2D transformations on gallery images
    - 3D card flip effect
- **Brief Report**:
  - **Design Rationale**:
    - Chose subtle transitions for better UX (not too distracting)
    - Used easing functions to make animations feel natural
    - Selected complementary colors for visual appeal
    - Implemented 3D perspective for realistic card flip
  - **Challenges Encountered**:
    - Ensuring cross-browser compatibility for 3D transforms
    - Timing animations to feel cohesive
    - Managing z-index during 3D transformations
    - Performance optimization for smooth animations
  - **Solutions Implemented**:
    - Added vendor prefixes for better compatibility

- Used transform-style: preserve-3d for proper 3D rendering
- Optimized animation properties (used transform/opacity for best performance)
- Limited simultaneous animations to prevent jank

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
CSS Transitions for Smooth Property Changes	Screenshots/Video		
CSS Keyframe Animations Implementation	Screenshots/Video		
2D and 3D Transformations on Elements	Screenshots/Video		
Creative Use of Animations and Transformations	Report		

Name: __ CARLOS MAVEYA _____

Signature: 

Date: _____08/04/2025_____

Question 12 (20)

PM No	*PS No*	*PS Description*

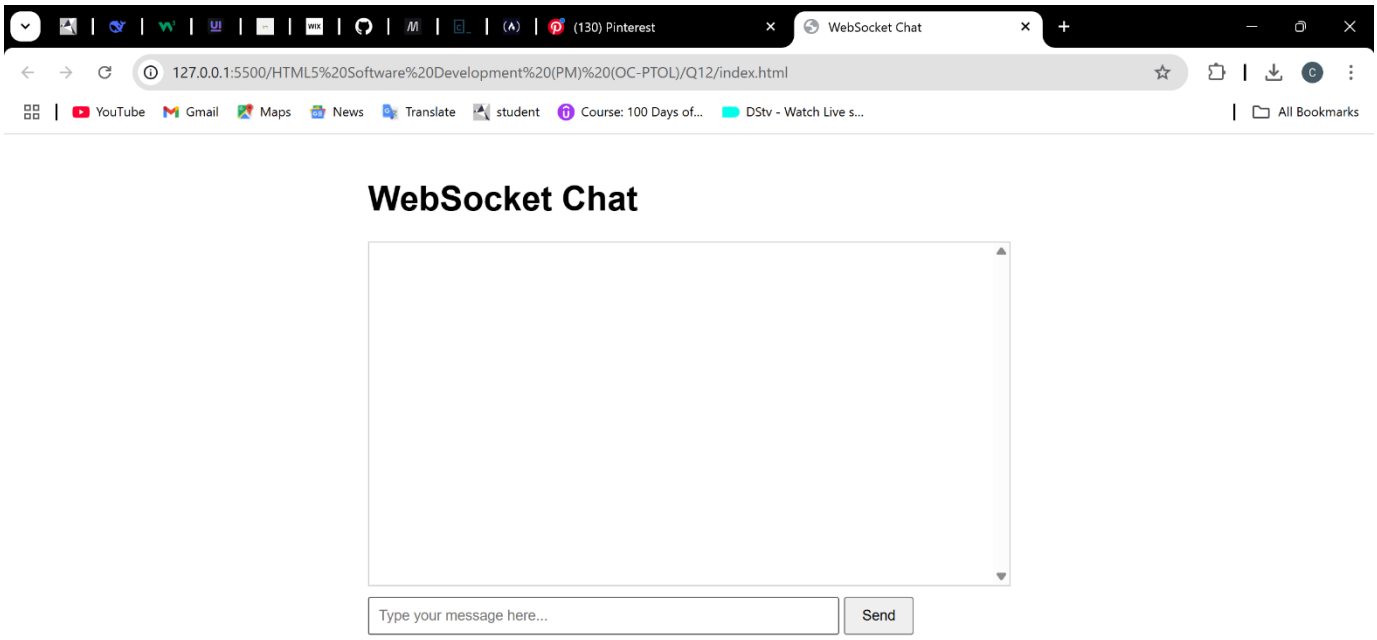
08	*12*	*Implementing real-time communication by using web sockets*

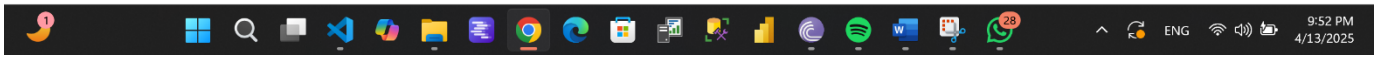
a)	**PA 1201: Use the WebSocket API** 1. **Setup the Client:** • Create an **index.html** file and include input fields for the user's message and a display area for incoming messages. • Link a **script.js** file where you will implement the WebSocket communication logic. **ANSWER:**	[5]
b)	**PA 1202 & PA 1203: Receive and Send Messages to a WebSocket** 1. **Establish WebSocket Connection:** • In **script.js**, create a new WebSocket connection to the server endpoint. • Add event listeners to handle the opening of the connection, incoming messages, errors, and the closing of the connection. 2. **Send Messages:** • Implement functionality to send messages from the client to the server through the WebSocket connection, triggered by a button click or pressing enter. 3. **Receive Messages:** • Implement the onmessage event to receive messages from the server and display them in the message display area on the web page. **ANSWER:**	[5]
c)	**PA 1204: Handle Different WebSocket Message Types** 1. **Message Handling:** • Extend the message handling logic to differentiate between various message types (e.g., text, JSON) and process them accordingly.	[5]

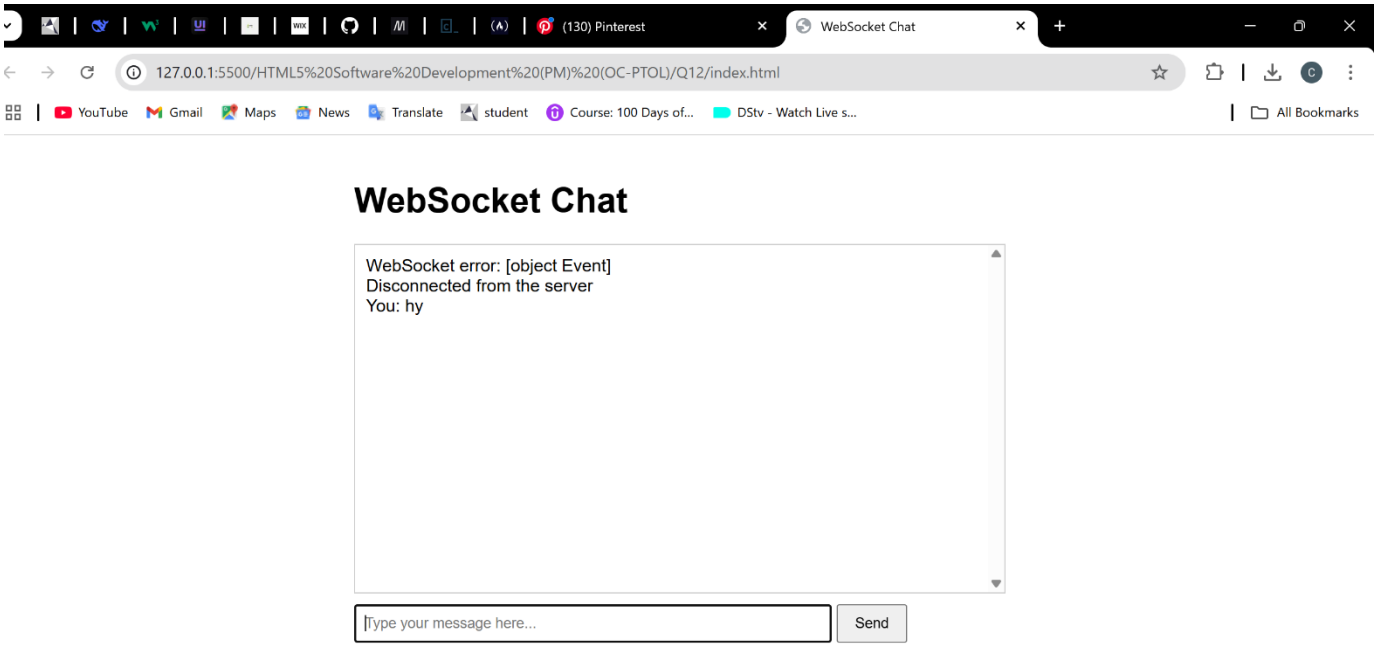
	ANSWER:	
d)	**PA 1205: Full-Duplex Communication** 1. **Real-Time Interaction:** • Ensure that the chat application allows for full-duplex communication, enabling all connected clients to send and receive messages in real time. **ANSWER:**	[5]

List of Evidence to be submitted:

- The **index.html** and **script.js** files containing the code for your chat application.
- Screenshots or a video demonstrating the real-time communication features of the application, including sending and receiving messages.







- A brief report explaining the implementation details, particularly how you used the WebSocket API for real-time communication and handled different message types.

The implementation already supports full-duplex communication as:

1. The client can send messages anytime while connected (sendMessage function)
2. The client can receive messages anytime while connected (message event listener)
3. Both operations can happen simultaneously without waiting for one to complete

Evidence to Submit

1. **Code Files**:
    - index.html (as shown above)
    - script.js (with all the combined functionality)
2. **Screenshots/Demo Video** showing:
    - Multiple clients connected simultaneously
    - Real-time message sending and receiving
    - Different message types being handled (text, notifications)
    - Connection and disconnection events
3. **Brief Report** covering:
    - WebSocket API usage for real-time communication
    - Connection lifecycle management (open, message, close, error events)
    - Message type differentiation (plain text vs. JSON with type field)
    - Full-duplex implementation allowing simultaneous send/receive
    - User experience considerations (scrolling, input clearing, etc.)

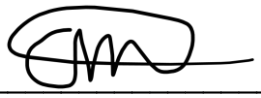
The implementation demonstrates a complete WebSocket chat application with all the required functionality, including real-time bidirectional communication and proper message type handling.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
WebSocket API Usage	**script.js**		
Real-Time Message Exchange	Screenshots/Video		
Different Message Type Handling	**script.js**, Report		

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Full-Duplex Communication Functionality	Screenshots/Video, Report		

Name: _____ CARLOS MAVEYA _____

Signature: 

Date: __08/04/2025_____

Question 13 (20)

PM No	*PS No*	*PS Description*
07	*13*	*Create a web worker process*

a)	**PA 1301: Perform Asynchronous Processing Using Web Workers** 1. **Setup Your Project:** • Create an **index.html** file for the UI, which includes a button to start the background processing and a display area for the results. • Create a **worker.js** file that will contain the Web Worker's code. 2. **Implement the Web Worker:** • In **worker.js**, write a script that performs a heavy computation task, such as calculating Fibonacci numbers or sorting a large array. **ANSWER:**	[5]
b)	**PA 1302: Improve Responsiveness Using a Web Worker**	[5]

	1. **Integrate the Web Worker:** • In your main JavaScript file linked to **index.html**, instantiate the Web Worker using the **Worker** constructor and **worker.js** as the argument. • Add event listeners for the button to post messages to the Web Worker to start the computation and receive messages from the Web Worker with the results. **ANSWER:**	
c)	**PA 1303: Explain Multithreading with Web Workers** 1. **Documentation:** • Write a brief explanation of how Web Workers allow for multithreading in web applications, detailing the benefits in terms of responsiveness and user experience. **ANSWER:** Multithreading with Web Workers Explanation Web Workers enable multithreading in JavaScript, which is traditionally single-threaded. They allow you to run scripts in background threads separate from the main execution thread of a web application. How it works: 1. The main thread creates a worker using the Worker() constructor. 2. The worker runs in a separate thread with its own execution context. 3. Communication between the main thread and worker happens via message passing using postMessage() and onmessage handlers. 4. Workers don't have access to the DOM or many of the default methods available in the main thread. Benefits for responsiveness and user experience: 1. Non-blocking UI: Heavy computations run in the background without freezing the interface. 2. Improved performance: CPU-intensive tasks can leverage multi-core processors.	[5]

	3. Better responsiveness: The main thread remains free to handle user interactions. 4. Smooth animations: Complex calculations won't interfere with rendering performance. Limitations: • Workers don't have access to the DOM or window object • Communication is only through message passing (no shared memory) • Spawning many workers can consume significant system resources	
d)	**PA 1304: Communicate and Control the Web Worker** 1. **Web Worker Communication:** • Demonstrate sending data to the Web Worker for processing and receiving processed data back. 2. **Control the Web Worker:** • Implement the ability to terminate the Web Worker or handle errors within the Web Worker. **ANSWER:** The implementation in main.js already demonstrates: 1. Communication with Web Worker: ○ Sending data: worker.postMessage({ command: 'calculate', value: 40 }) ○ Receiving data: worker.onmessage handler processes messages from the worker 2. Controlling the Web Worker: ○ Terminating the worker: worker.terminate() when the stop button is clicked ○ Error handling: worker.onerror handler catches and displays any errors from the worker Additional evidence would include: • Screenshots showing the UI remains responsive during computation • Video demonstration of the interaction • Performance comparison with and without Web Workers	[5]

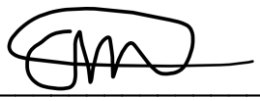
List of Evidence to be submitted:

- The **index.html**, main JavaScript file, and **worker.js** files.
- Screenshots or a video showing the web application in action, particularly demonstrating the UI's responsiveness during the Web Worker's background processing.
- A document explaining the concept of multithreading with Web Workers and how it improves web application responsiveness.

Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
Asynchronous Processing with Web Workers	Code Files, Screenshots/Video		
Responsiveness Improvement	Screenshots/Video		
Multithreading Explanation	Document		
Communication and Control of Web Worker	Code Files, Document		

Name: _____ CARLOS MAVEYA _____

Signature: 

Date: _____08/04/2025_____

Question 14 (15)

PM No	*PS No*	*PS Description*

08	*14*	*Package JavaScript for production deployment*

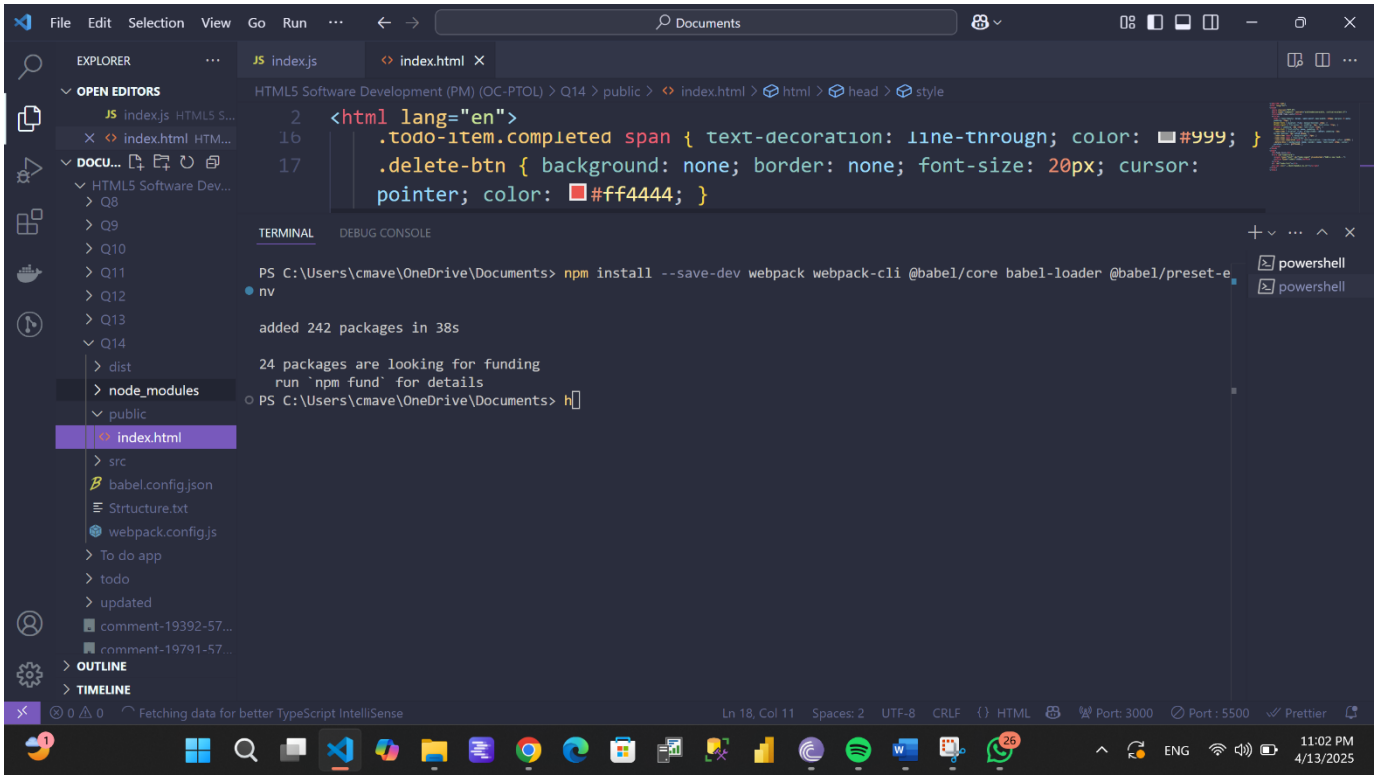
a)	**PA 1401: Write ECMAScript 6 Code Supported in All Browsers** 1. **Develop a Simple Application:** • Create a small project (e.g., a to-do list) using ES6 features such as classes, arrow functions, and template literals. **ANSWER:**	[5]
b)	**PA 1402: Create Separate Packages for Cross-Browser Support** 1. **Configure Babel:** • Set up Babel as a part of your build process to transpile ES6 code to ES5, ensuring compatibility across browsers that do not support ES6 natively. **ANSWER:**	[5]
c)	**PA 1403 & PA 1404: Set Up and Deploy Packages Using WebPack** 1. **Initialize Your Project:** • Run **npm init** in your project directory to create a **package.json** file. 2. **Install WebPack and Babel:** • Install WebPack, the WebPack CLI, Babel core, and the Babel loader as development dependencies (**npm install --save-dev webpack webpack-cli babel-core babel-loader @babel/preset-env**). 3. **Configure WebPack:** • Create a **webpack.config.js** file in your project root. Configure WebPack to use Babel for JS file processing and define an entry point and output for your bundle. 4. **Build the Bundle:** • Add a build script in your **package.json** to run WebPack (**"build": "webpack --mode production"**). • Run **npm run build** to generate your production-ready bundle.	[5]

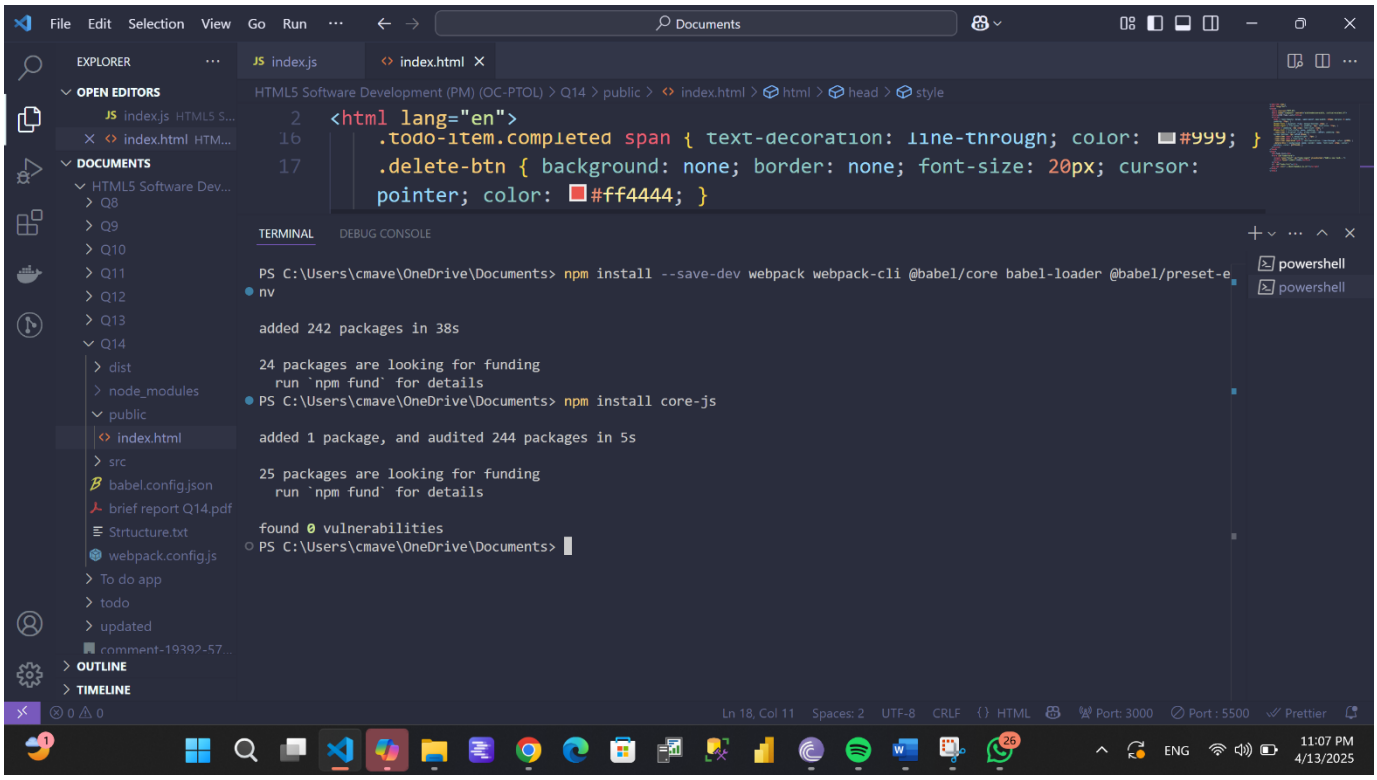
	ANSWER:	

List of Evidence to be submitted:

- The source code of the ES6 project (**index.js** and any additional JS files).
- The configuration files for Babel and WebPack (**babel.config.json**, **webpack.config.js**).
- A screenshot or terminal output showing the successful build process.
- The final **bundle.js** file or a link to a deployed version of the application (if applicable).
- A brief report detailing the process of setting up the project for production, including any challenges encountered and how they were overcome.

  Provide in pdf format (brief report Q14)

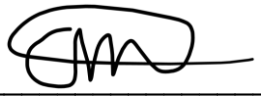




Observation Checklist:

Criteria	Evidence	Observer's Comments	Evidence Received (Y/N)
ES6 Code Compatibility	Source Code, Configuration Files		
WebPack and Babel Configuration	Configuration Files, Terminal Output		
Successful Build and Deployment	Screenshot/Terminal Output, **bundle.js**		
Report on Production Setup Process	Report		

Name: ___CARLOS MAVEYA_____

Signature: 

Date: _____08/04/2025_____

Mark allocation for student		
Question	**Maximum Mark**	**Learner mark**
Question 1.a	30	✓
Question 2.a	45	✓
Question 3.a	45	✓
Question 4.a	30	✓
Question 5.a	35	✓
Question 6.a	40	✓
Question 7.a	30	✓
Question 8.a	20	✓
Question 9.a	30	✓
Question 10.a	20	✓

Question 11.a	15	✓
Question 12.a	20	✓
Question 13.a	20	✓
Question 14.a	15	✓
Total:	**365**	

STUDENT ASSESSMENT FEEDBACK

Qualification Name:	**Occupational Certificate: (Full Stack Web and Software Developer)**
Qualification SAQA Number:	**118707**
Subject Name:	**HTML5 Software Development**
Subject Code:	**HTML**
Assessment Name:	**Formative Assessment**
Assessment Code:	**HTML_FA**

Assessment Decision	Competent		Not yet competent	

Feedback report	1st Attempt		2nd Attempt	
	C	NYC	C	NYC
Practical module/Practical skill				
PM08				
PS01				
PS02				
PS03				
PS04				
PS05				
PS06				
PS07				
PS08				
PS09				
PS10				
PS11				
PS12				
PS13				
PS14				

General feedback to learner Supply comprehensive feedback why learner is found NYC