**Summative Assignment: WEBD2A**

**Web Development 2**

For Learner Use:

|  |  |
| --- | --- |
| Surname of Learner | Langa |
| Name of Learner | Thulani |
| Learner ID | 0308175686085 |
| Student Number | TLPMB044 |
| Date of Test Given | 04/06/2025 |

For Assessors Use:

|  |  |
| --- | --- |
| Name of Facilitator |  |
| Name of Assessor |  |
| Mark Allocation | 160 |
| Mark Obtained |  |
| Competency Status (C / NYC) |  |

|  |  |
| --- | --- |
| **Knowledge and Practical Outcomes Assessed** | |
| 118707 Occupational Certificate: Software Developer | * 251201-005-00-KM-06 Software Development with HTML5, Opensource Frameworks and Libraries, Level 5, 16 Credits. * 251201-005-00-PM-08, Develop Software using HTML5, Opensource Frameworks and Libraries, Level 5, 16 Credits. * 251201-005-00-PM-09 Design and Build Web Applications, Desktop Graphical User Interfaces or Mobile Apps, Level 5, 8 Credits. |

|  |  |  |  |
| --- | --- | --- | --- |
| Candidates Signature |  | Date of Submission | 09/06/2025 |
| Assessors Signature |  | Date Marked |  |

**INSTRUCTIONS**

1. Read questions carefully.
2. Answer all questions.
3. You will complete a number of practical and theoretical questions.
4. Place screenshots in the space provided.
5. Write Answers in the space provided.
6. Complete the cover page in full.
7. Plagiarism will not be tolerated
8. Ensure your work is formatted neatly and is readable by the Assessor at all times.
9. One complete assignment must be submitted in editable format.
10. Any attachments must be submitted with the original submission.

This Assignment is set up in of 4 sections:

* Task A: Design a Web Application
* Task B: Build a functional REST AP
* Task C: Build a functional Web Interface
* Task D: Testing a Web Application

All Sections are to be completed in full, according to the instructions given.

**COMPETENCY RATINGS:**

* Not Yet Competent (NYC) – Student achieves less than 60%
* Competent (C) – Student achieves 60% – 74%
* Competent with Merit (C/M) – Student achieves 75% - 84%
* Competent with Distinction (C/D) – Student achieves 85% - 100%

**SCENARIO**

You work as a freelance web developer, and a local small business has contacted you to make a request to build a web application for them.

They have asked you to make a to-do list web application so that their employees can better understand their tasks. Users should be able to manage tasks through the web application’s interface. They need to be able to see a list of existing tasks, add new tasks, and delete selected tasks.

Read through all the instructions carefully and complete the tasks in the order given.

**Task A – Design a Web Application (70)**

1. Prior to designing your website it is important that you understand exactly what is required of the web application. Firstly, you will need to interview the owner of the local business to gain the following information: (10)

• the aim and functionality of the web application

• any user requirements

• implementation of the web application

• timescales for completing the web application

|  |
| --- |
| **INTERVIEW QUESTIONS AND ANSWERS**  Interview Questions   1. What’s the biggest problem you're hoping this to-do app will solve for your team? 2. Can you walk me through what a typical day looks like for your team and how tasks are currently tracked? 3. How should tasks be added—should it be one person in charge or can anyone on the team contribute? 4. What kind of details should a task include (just a title, or also due dates, notes, etc.)? 5. Do you want tasks to be editable after they’re created? 6. Would it be helpful to group tasks into categories or departments, or keep one flat list? 7. What devices will your team be using to access the app most often? 8. Do you want any kind of undo or confirmation when deleting a task? 9. What’s your ideal outcome two weeks from now—what does success look like? 10. Is there anything you’ve seen in other apps or tools that you liked—or hated?   Interview Answers (From the Business Owner – LaughTale)   1. Honestly, we just need to stop things from falling through the cracks. A shared digital list helps us all stay aligned, especially when things get hectic. 2. Right now, we use sticky notes, whiteboards, and emails. It’s chaotic. Everyone has their own system, and nothing syncs across the team. 3. Anyone should be able to add a task. We’re a small team—no point in having gatekeepers for this. 4. Just a short title is fine for now. If we could add a note or due date later, that might be useful, but it’s not essential at this point. 5. Yes, editing would be nice. People make typos or plans change, so being able to tweak a task would be helpful. 6. If you can group tasks by department or tag them, that’d be great. But we’re okay with starting out with just one list and keeping it simple. 7. Mainly desktops and laptops. A few tablets are used occasionally in meetings. No one really uses their phone for work here. 8. An “Are you sure?” message before deleting something would prevent mistakes. An undo button might be overkill, but it’d be a nice-to-have. 9. By the end of two weeks, I want the team to be using the app daily without needing instructions. If they actually prefer it over sticky notes, we’ve won. 10. I liked Trello’s drag-and-drop feel but found it too complex. I hated apps where I had to make an account just to make a simple note. |

1. Based on the information that you have gathered from the owner of the local business, research **THREE** web applications that relate to what the owner requires. Include the following: (30)
2. Overview of the site
3. Screenshots
4. Pros and Cons (focus on User-Experience and the site objective)
5. Compare three web applications [[1]](#footnote-1)
6. What you would adopt (why and how would it benefit your site and how can you make it better)
7. Draw a conclusion

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RESEARCH** Site Overviews  1. **Monday.com** – a cloud-based work operating system designed to help teams manage tasks, projects, and workflows. It offers customizable boards, automation, and collaboration tools to streamline team operations. 2. **Trello** – a visual collaboration tool that uses boards, lists, and cards to organize tasks. It's known for its simplicity and ease of use. 3. **Microsoft To Do** – a task management application that integrates seamlessly with Microsoft 365 services. It allows users to create tasks, set reminders, and organize their day.  ScreenShots Please Refer to Folder Pros & Cons  |  |  |  | | --- | --- | --- | | Application | Pros | Cons | | Monday.com | Highly customizable, advanced features, good UI, automation | Can be overwhelming, paid tiers can be expensive | | Trello | Easy to use, visual, free tier, flexible Power-Ups | Lacks depth in features without add-ons | | Microsoft To Do | Simple interface, free, integrates with Microsoft tools | Limited features, not ideal for team collaboration |  Comparitve Analysis  |  |  |  |  | | --- | --- | --- | --- | | Feature | Monday.com | Trello | Microsoft To Do | | User Interface | Modern and customizable | Kanban-style | Simple and basic | | Task Management | Advanced boards & automation | Cards, checklists, Power-Ups | Lists and subtasks | | Collaboration | Team comments, sharing | Basic sharing, comments | Limited (mostly personal use) | | Integration Options | Slack, Google Drive, etc. | Limited without Power-Ups | Strong for Microsoft users | | Customization | High | Medium | Low | | Cost | Paid plans | Free with paid upgrades | Free |  Features To ~~Steal~~ Adopt **Monday.com:** Customizable task views and optional automation  **Trello:** Kanban-style board layout and drag-and-drop simplicity  **Microsoft To Do:** Simplicity and task-focused design Conclusion These features will provide a clean, visual, and easy-to-use task manager tailored to the client’s needs. We can make it even better by:   * Removing login complexity * Streamlining UI for office display use * Hosting it locally for privacy and speed   Although the client wants a to-do list, I imagine the market research will involve the creation of said project which would take weeks if not months to do, especially since all three applications offer strong features, but Monday.com may be too complex, and Microsoft To Do too simple. Trello strikes a good balance for visual task management. instead I will deliver the absolute bear minimum required of the client(project) although these questions and research will prove to be a great resource if the client wants to invest further into the project in which funding/down-payment is required. Doing what the client requests and delivering that will result in an ideal tool tailored specifically for the client’s workflow. |

1. Once you have this information you will need to produce a short plan that outlines the details you have learned. Include use case(s) that defines how users will interact with the system

(30)

|  |
| --- |
| **IMPLEMENTATION PLAN (based on answers)** Tech stack  * **Frontend:** HTML, CSS for layout and styling, JavaScript for logic and communication * **Backend:** Node.js with Express and SQLite for local testing * **Database:** SQLite with a tasks table (id, title, description) * **Deployment:** Localhost or internal network  Overview The goal of this project is to design and build a clean, lightweight to-do list web application that the company employees can use every day in the office. Right now, the team uses sticky notes and verbal reminders, which can lead to missed tasks and confusion. This app will centralize all task tracking in one shared space. What the App Will Do The application will be simple, but effective. It will:   * Let employees add new tasks * Display a live list of all current tasks * Allow users to delete tasks when they’re completed * Store all tasks so they don’t disappear when the page is refreshed   No login system is required — any team member can access it from their device in the office. |

|  |
| --- |
| **USE CASE(S)**  **Actor:** Employee (User)  **Use Case 1: Adding a Task**   * An employee opens the app in a browser. * They type a new task like “Send report to client” in the input box and click “Add”. * The task instantly appears on the shared list for everyone to see.   **Use Case 2: Deleting a Task**   * A team member finishes a task. * They click a small delete button (or checkbox) next to the task. * The task disappears from the list and is removed from the database.   **Use Case 3: Viewing Tasks**   * The app is always open on a shared office screen. * Any employee walking past can see what’s been done and what’s left to do. * The list updates live (or with a quick refresh). |

**Task B – Build a functional REST API (20)**

For your web application to respond to user requests and store any data, you will need to build REST API to establish some form of communication between the server and client. You will also need to set up a connection to a SQL Database to store data for individual tasks. **\***

Create a SQL database that contains a table called **tasks** that is structured as follows: (5)

|  |  |
| --- | --- |
| **COLUMN NAME** | **DATA TYPE** |
| id | INTEGER (PRIMARY KEY) |
| title | TEXT |
| description | TEXT |

P.S. Use **CREATE TABLE IF NOT EXISTS** at server startup to ensure the table gets created before the server begins accepting requests.

Your server should run on a suitable port that will not be in use by any other service or protocol.

Ensure your REST API supports the following requests:

1) GET / tasks (5)

This GET request should respond with a list of all the tasks that exist in the database. In the request handler, you should be able to run a **SELECT** query on your database and respond with a list of all the tasks returned and a status code of **200**.

2) POST / tasks (5)

This POST request should accept JSON data in the body of the request containing a title and a description of the task. In the request handler, you should be able to run an **INSERT** query on your data in to add a new task with the provided title and description. If everything works, your API should send a success response with a status code of **201**.

3) DELETE /tasks/:id (5)

This DELETE request should accept a task ID as a request parameter in the URI. In the request handler, you should be able to run a **DELETE** query on your database to delete the task with the given ID. If everything works, your API should send a success response with a status code of **200**.

***\*The actual code will be assessed as well as the functional server. You need not host the server on a functional registered domain – it can be local.***

**Task C – Build a functional Web Interface (50)**

Now that your REST API is complete, it’s time to build a web interface for your users. **\***

**1)** Write HTML and CSS code to build an interface for your users. (30)

Your web interface must feature the following:

• a list with a heading to display all available tasks

• a UI element to display the title and description of a single task

• a delete button on each task

• a form with 2 text inputs for a title and a description

• suitable CSS styling for each element

**2)** Write JavaScript code to communicate with the server using the REST API you built in Task B. (20)

Your code needs to be able to do the following:

• fetch the tasks data from the server using fetch

• dynamically render the tasks list from tasks data received

• delete selected tasks when the delete button is clicked

• add new tasks using the information entered in the form

• display user-friendly error messages when errors occur

***\*The actual code will be assessed as well as the functional web application. You need not host the web application on a functional registered domain – it can be local.***

**Task D – Testing a Web Application (20)**

You’ve finished your Web Application! Now it’s time to test it to make sure it’s suitable for users.

1. Test your REST API. Make a test log comparing the expected results and actual results of running each request. Provide screenshots as evidence for all testing done. (5)

|  |
| --- |
|  |

1. Test your Web Application’s user interface. Make a test log comparing expected results to actual results of testing each piece of functionality in the interface. In addition, test your web application in at least 2 different browsers. Provide screenshots as evidence for all testing done. (15)

|  |
| --- |
|  |

Please reference the files and folders attached to this document below.

|  |
| --- |
|  |

1. [↑](#footnote-ref-1)