**CSE 310 – Applied Programming**

**Module Plan**

Name:

Date:

Module # (1-3):

1. Identify which module you have selected to work on. Place an “X” in front of your selected module.

Cloud Databases

Data Analysis

Game Framework

GIS Mapping

Mobile App

Networking

SQL Relational Databases

Web Apps

Language – C++

Language – Java

Language – Kotlin

Language – R

Language – Erlang

x Language – JavaScript

Language – C#

Language - TypeScript

Language – Rust

1. At a high level, describe the software you plan to create that will fulfill the requirements of this module. Describe how each requirement will be met. This may change as you learn more about the technology or language you are learning.

At a high level, the software I plan to create is a **Task Management Web Application** using **JavaScript**. This application will allow users to efficiently manage their tasks by adding, editing, deleting, and marking them as completed. The application will have a user-friendly interface, task filtering and sorting features, and persist data using the browser’s localStorage. Here's how each requirement will be met:

**Requirements and How They Will Be Met:**

1. **Display Output to the Screen**:
   * **Requirement**: Display output using HTML (for web) or Node.js (for the terminal).
   * **Implementation**: The tasks will be displayed in the browser using dynamic HTML elements created and manipulated by JavaScript. JavaScript's document object will be used to add tasks as list items to an unordered list.
2. **Native Array ES6 Functions**:
   * **Requirement**: Use native ES6 functions such as .map(), .filter(), or .reduce().
   * **Implementation**:
     + The .map() function will be used to render tasks in the task list.
     + The .filter() function will filter tasks by their completion status (completed or pending).
     + The .reduce() function will be used to count completed and pending tasks, helping display task statistics to users.
3. **Recursion**:
   * **Requirement**: Demonstrate recursion in the code.
   * **Implementation**: Recursion will be used to handle nested subtasks. For example, the application will recursively check subtasks to ensure all are completed before marking a parent task as done.
4. **Use a JavaScript Library Written by Someone Else**:
   * **Requirement**: Integrate an external JavaScript library.
   * **Implementation**: The project will use **Moment.js** to manage and display due dates for tasks, including formatting dates and calculating time differences. Alternatively, **Lodash** will be used for utility functions like deep cloning objects and array manipulation.
5. **Exception Handling**:

* **Requirement**: Demonstrate throwing and handling exceptions.
* **Implementation**: Exceptions will be thrown if a user attempts to create a task with invalid input (e.g., missing a title or setting an invalid due date). The application will catch these exceptions and display error messages to the user, ensuring they understand the problem and can correct it.



1. Create a detailed schedule using the table below to complete your selected module during this Sprint. Include the task and duration for each day. You are expected to spend 24 hours every Sprint working on this individual module and other activities in the course. Time spent on this individual module should be **at least** 12 hours.

|  |  |  |
| --- | --- | --- |
|  | **First Week of Sprint** | **Second Week of Sprint** |
| **Monday** | - Initial setup: Create project structure (HTML, CSS, JS files).  - Set up libraries like Moment.js or Lodash (via CDN). 3 hours | Integrate Moment.js for date handling and display. - Add due date sorting and time remaining calculation features. 3 hours |
| **Tuesday** | Design the basic UI layout using HTML and CSS (task list, add task form, buttons). - Ensure responsive design for different devices. 3 hours | Style the completed tasks and implement task filtering (e.g., show only completed or pending tasks) using JavaScript and CSS. 3 hours |
| **Wednesday** | Implement task creation functionality using JavaScript. - Add tasks to an array and dynamically render them to the screen. 3 hours | Add animations or visual effects (e.g., fade out for deleted tasks, hover effects on buttons). 3 hours |
| **Thursday** | Add ES6 array functions to manage tasks (e.g., .map(), .filter() for rendering and filtering tasks). 3 hours | Perform testing on all features, including task creation, editing, completion, and deletion. Fix bugs if needed. 3 hours |
| **Friday** | Implement task completion and deletion features. - Use recursion to check completion of subtasks. 3 hours | Final UI and UX improvements based on testing feedback. - Optimize for mobile and desktop responsiveness. 3 hours |
| **Saturday** | Implement input validation and exception handling for task creation (e.g., invalid title, missing due date). 3 hours | Final review: Documentation (README.md creation), code cleanup, and project submission. 3 hours |

1. Identify at least two risks that you feel will make it difficult to succeed in this module. Identify an action plan to overcome each of these risks.

Risk 1: Difficulty with New JavaScript Concepts (e.g., ES6 functions, recursion)

Description: As the project requires the use of ES6 array functions (e.g., .map(), .filter(), .reduce()) and recursion, these concepts can be complex if not fully understood, potentially leading to inefficient or incorrect implementations.

Action Plan:

Study and Practice: Before diving into implementation, spend additional time reviewing ES6 functions and recursion through tutorials, examples, and small practice exercises.

Break Down Problems: Tackle complex features incrementally by breaking down tasks like recursion into smaller, manageable steps.

Leverage Resources: Use online resources such as MDN Web Docs and Stack Overflow for quick help and clarification on JavaScript features.

Test Early and Frequently: Write unit tests to ensure that individual components (such as task manipulation functions) work correctly and prevent errors from compounding later.

Risk 2: Integration Issues with External Libraries (Moment.js or Lodash)

Description: Integrating external JavaScript libraries (such as Moment.js for date handling or Lodash for utility functions) can lead to compatibility issues or unexpected bugs, especially if not properly configured or used incorrectly.

Action Plan:

Read Documentation: Thoroughly read the official documentation for any external libraries (Moment.js, Lodash) before integrating them into the project to understand their usage and limitations.

Test in Isolation: Implement and test the library functionality in a separate environment before adding it to the main project to avoid integration conflicts.

Fallback Plan: Have a fallback strategy in case of library issues (e.g., using native JavaScript Date objects if Moment.js fails to integrate smoothly).

Version Control: Use Git or other version control tools to manage changes, so that library integration attempts can be rolled back if needed without disrupting progress.