WhileSmart

ENMEBE VERINE ICHUMBEI
Assessment task
Code Documentation

Overall Architecture

The Todo app follows a simple architecture, where the front-end (HTML, CSS, and JavaScript) handles the user interface, user interactions, and manages the tasks data. The front-end communicates directly with the user's browser.

The app uses a client-side approach, where tasks are stored in the JavaScript `tasks` array, and the UI is dynamically updated based on the changes to this array. The tasks data is not persisted, and there is no separate back-end or database implemented in this example.

The app follows a modular structure with functions that handle specific tasks, such as adding tasks, updating completion status, removing tasks, and updating the productivity score. These functions are called in response to user actions and ensure the UI is updated accordingly.

Key Functions:

- `createTaskObject(name, activity, state, dateline)`: Creates a new task object with the provided details and returns it.

- `addTask(event)`: Handles the form submission event to add a new task. Retrieves input values, creates a task object, adds it to the tasks array, and updates the UI.

- `renderTasks()`: Renders the tasks in the task lists based on the tasks array. Creates list items for each task, including checkboxes, labels, and buttons, and appends them to the appropriate list (incomplete or completed).

- `toggleTaskCompletion(event)`: Handles the change event of checkboxes to toggle the completion status of a task. Updates the tasks array, re-renders the task lists, and updates the productivity score.

- `removeTask(event)`: Handles the click event of the remove buttons to remove a task from the tasks array. Updates the task lists and productivity score.

- `updateProductivityScore()`: Calculates the productivity score based on the number of completed tasks and updates the productivity score element in the UI.

- `resetForm()`: Resets the form inputs to their initial state.

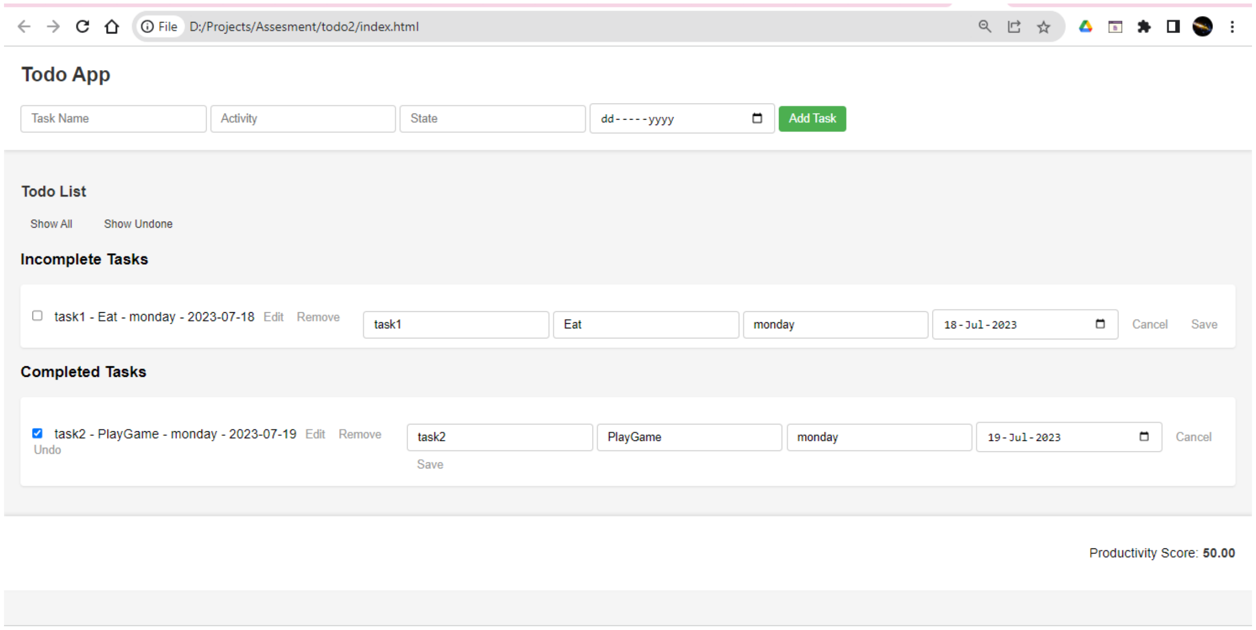
- Event Listeners: Sets up event listeners for form submission, task completion toggling, and task removal. Calls the appropriate functions based on the user actions.

HTML (index.html)

The `index.html` file serves as the entry point for the Todo app. It defines the structure of the web page and contains the necessary elements to interact with the application.

Deployment:

The application was deployed on my local host



Key Elements:

- `<header>`: Represents the header section of the web page, containing the app title and the form to add new tasks.

- `<main>`: Represents the main content area of the app, including the filter buttons, incomplete tasks list, and completed tasks list.

- `<footer>`: Represents the footer section of the app, displaying the productivity score.

CSS (styles.css)

The `styles.css` file contains the styles and layout rules that define the visual appearance of the Todo app.

Key Styles:

- Header Styles: Styling rules for the app header, including background color, padding, and box shadow.
- Form Styles: Styling rules for the form inputs and button, such as padding, border, and background color.
- Task List Styles: Styling rules for the task lists, including list item styles, background color, padding, and box shadow.
- Button Styles: Styling rules for the filter buttons, including margin, background color, border, and transitions.
- Footer Styles: Styling rules for the footer section, including background color, padding, and box shadow.

JavaScript (script.js)

The `script.js` file contains the client-side logic of the Todo app. It handles user interactions, manages the tasks data, and updates the UI dynamically.

Future Improvements

While the current implementation fulfills the requirements of the Todo app, there are several potential areas for future improvement and expansion:

- Backend Implementation: Add a server-side back-end using a suitable technology stack, such as Node.js, along with a database for persistent storage.
- API Integration: Implement API endpoints for the front-end to communicate with the back-end, allowing tasks to be saved and retrieved.

- User Authentication: Add user authentication and authorization mechanisms to secure the app and enable individual user task management.

- Error Handling: Implement robust error handling to provide feedback to users in case of failures or incorrect inputs.

- Task Sorting and Filtering: Add functionality to sort and filter tasks based on various criteria, such as task name, state, or completion status.

- Task Prioritization: Implement features to set task priorities or due dates and provide visual cues for time-sensitive tasks.

- User Interface Enhancements: Improve the UI by adding animations, responsive design, and additional visual feedback to enhance the user experience.

These potential improvements can further enhance the functionality, usability, and scalability of the Todo app.

This documentation provides an overview of the code structure, key functions, and concepts used in the implementation of the Todo app. It serves as a reference for understanding the implementation details and can be used by developers and stakeholders involved in the development and maintenance of the application.