Module 2 – React JS

(Connecting Frontend to Backend using React and RESTful APIs)

- By Ajeet Kumar (Batch: 15th June)

Project Title: Task Master

Date Prepared: 10th October 2023

Deployed On: https://github.com/ajeetkumarrauniyar/Task-Master_To-Do-App

## Introduction

In this assignment, we were tasked with creating a complete To-Do list application "**Task Master"** with both frontend and backend components. The frontend is built using React, and the backend uses Node.js and Express.js with the Node-persist storage. Here are some key points and insights:

## Project Structure

The system follows a client-server architecture each having its own package.json and node modules. This separation allows for clear organization of the frontend and backend code.

- **Client Side:** The client side consists of web browsers through which users interact with the system's graphical user interface (GUI).

- **Server Side:** The server side is built using Node.js and Express.js. It handles incoming HTTP requests, processes data, and communicates with the storage system. Storage system is built using the **Node-persist.**

## Resources Used:

- Visual Studio Code as the code editor
- Google Chrome as the web browser
- POSTMAN to test the APIs

## Getting Started

To get started with the Task Master, follow these steps:

1. Clone the repository to your local machine.

https://github.com/ajeetkumarrauniyar/Task-Master_To-Do-App

2. Install the required dependencies by running the following command:

    npm install

3. Start the server:

    npm start

## Implementation

The system's implementation involves key components and libraries:

- **Express.js:** Express is used to create the web server and handle HTTP requests.
- **Express JSON** : An Express in-built middleware is used to parse request data, enabling the system to handle POST requests effectively.
- **Node-Persist :** Node-Persist is used as a storage mechanism for persistently storing the tasks. It allows data to be saved and retrieved between server sessions.
- **Tailwind CSS:** It is used to create the Client Side (UI) of the App.
- **React-hot-toast :** It is utilized to generate an alert indicating the successful addition of a task, instead of relying on the JavaScript alert function.

## Key aspects :

Frontend (App.js):

1. **State Management:** The frontend of the application is built using React, a popular JavaScript library for building user interfaces. In the **App.js** component, state management is crucial. Two pieces of state are managed:
    - **taskInput**: This state stores the user's input in the task input field. It's initialized with an empty task object containing **id** and **task** properties.
    - **taskItemList**: This state holds an array of tasks that are displayed in the task list. Initially, it's an empty array.
2. **Form Handling:** The application provides a form for users to add tasks. When the user enters a task and clicks the "Add Task" button, a form submission event is triggered

(**handleSubmit** function). The **preventDefault()** method prevents the default form submission behavior.

- A POST request is then sent to the backend server running at "http://localhost:5000/". This request contains the user's task input, which is transformed into a JSON string using **JSON.stringify(taskInput)** and sent in the request body.
- If the POST request is successful (status code 201), a success toast message is displayed using the **react-hot-toast** library. The input field is cleared, and the new task is added to the **taskItemList** state to update the list on the frontend.

3. **Fetching Data:** he application uses the **useEffect** hook to fetch the task list from the backend when the component mounts. This is done by calling the **getData** function.
   - The **getData** function sends a GET request to "http://localhost:5000/get" to retrieve the list of tasks. The retrieved data is then set in the **taskItemList** state, updating the task list displayed on the frontend.
4. **Rendering Tasks:** The **taskItemList** state is mapped through to render the list of tasks. Each task is displayed as a list item in the UI. Currently, there are options for editing and deleting tasks in the code, but they are commented out. These options can be implemented in the future to allow users to perform these actions on tasks.

Backend (server.js):

1. **Express Server:** We create an Express server to handle HTTP requests. It listens on port 5000.
2. **Node-Persist Storage:** Node-persist is used as a simple storage system to store and retrieve task data. It provides a way to persist data across server sessions.
3. **POST and GET Routes:** The backend defines two routes:
   - **POST Route:** This route is used for adding tasks. When a POST request is received at "/", the server extracts the task data from the request body and stores it in the Node-persist storage. A 201 status (Created) is sent as a response if the task is added successfully.
   - **GET Route:** This route is used for retrieving the list of tasks. When a GET request is made to "/get", the server retrieves all task items from the Node-persist storage and sends them as a response with a 200 status (OK).
4. **CORS:** To enable communication between the frontend and backend, the application uses the CORS middleware. It allows cross-origin requests from the frontend to the backend, ensuring that the two can interact seamlessly..
5. **Storage Clear :** We had added a function to clear the storage whenever the server restarts using the **storage.clear () function.**

## Improvements and Future Work

- The code for editing and deleting tasks is currently commented out and can be implemented to provide a more complete To-Do list application.
- Error handling on the frontend and backend can be enhanced to provide better user feedback in case of failures.
- Additional features such as task prioritization, due dates, and user authentication can be added to make the application more feature-rich.

## Conclusion

This assignment provides valuable insights into building a full-stack web application using modern web development technologies. It demonstrates how frontend and backend components can work together to create a functional application.