

LAB ASSIGNMENT NO – 11

Aim: Create a web application that performs CRUD operations (database connectivity).

Theory:

CRUD (Create, Read, Update, Delete) operations are fundamental actions used in web applications to manage data within a system. These operations enable interaction with databases, APIs, and servers to perform various tasks:

Create (C):

In a web application, creating refers to adding new data or records to a database or backend server. This involves gathering user input, processing it, and sending it to the server to be stored.

Read (R):

Reading involves retrieving data from a database or a server. It often includes fetching and displaying information on a webpage based on user requests or system events.

Update (U):

Updating means modifying existing data in a database or server. Users can edit their information, and the changes are sent to the server to update the respective record.

Delete (D):

Deleting involves removing data or records from a database or server. Users can remove information, and this change is sent to the server for deletion.

To-Do List Web Application Overview:

The to-do list web application is a simple CRUD (Create, Read, Update, Delete) application that allows users to manage a list of tasks they need to complete. The application has two main components: the frontend, built using React, and the backend, built using Node.js and MongoDB.

Frontend

The frontend of the application is responsible for providing the user interface and interaction. It's built using React, a popular JavaScript library for building user interfaces. The frontend communicates with the backend to perform CRUD operations on the to-do list.

Functionality Overview

1. Displaying To-Do List: The application displays the existing to-do items fetched from the backend upon loading.
2. Adding To-Do Items: Users can add new to-do items by typing in the input field and clicking the "Add" button. This action sends a request to the backend to create a new to-do item.
3. Updating To-Do Items: Users can mark a to-do item as completed or incomplete by clicking on it. This action sends a request to the backend to update the completion status of the respective to-do item.
4. Deleting To-Do Items: Users can delete a to-do item by clicking the "Delete" button next to it. This action sends a request to the backend to delete the respective to-do item.

Backend

The backend of the application is built using Node.js, a JavaScript runtime, and Express, a web application framework. It handles HTTP requests from the frontend, interacts with the MongoDB database, and performs CRUD operations on the to-do items.

Functionality Overview

1. Creating To-Do Items: The backend provides an API endpoint to create a new to-do item. It receives the to-do item data from the frontend, saves it to the MongoDB database, and sends back the created to-do item.
2. Reading To-Do Items: The backend provides an API endpoint to fetch all existing to-do items from the MongoDB database. It retrieves the to-do items and sends them back to the frontend.
3. Updating To-Do Items: The backend provides an API endpoint to update the completion status of a to-do item. It receives the updated status from the frontend, updates the corresponding to-do item in the database, and sends back the updated to-do item.
4. Deleting To-Do Items: The backend provides an API endpoint to delete a to-do item. It receives the ID of the to-do item to be deleted from the frontend, removes the corresponding to-do item from the database, and sends a success response.

Conclusion: The to-do list web application seamlessly integrates a user-friendly React frontend with a robust Node.js backend, allowing for smooth interaction and management of tasks. Leveraging MongoDB, it offers reliable data storage and retrieval, enhancing the overall user experience. The intuitive UI permits users to effortlessly add, update, and delete tasks, promoting efficient task management. This application provides a solid foundation for future enhancements, catering to personalized features and improved scalability. In summary, it offers a practical solution for organizing tasks with a streamlined and expandable architecture.

Technologies Used:

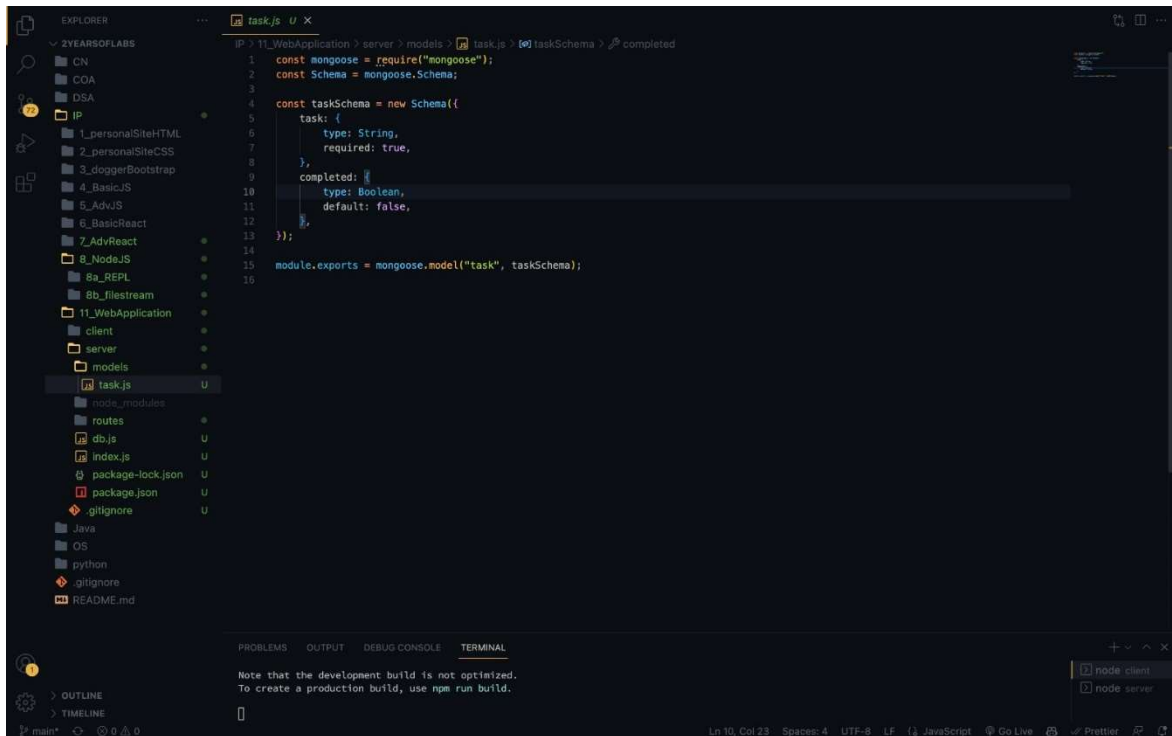
Frontend:

- React: A JavaScript library for building user interfaces.
- Axios: A popular HTTP client for making API requests.
- HTML and CSS: Standard web technologies for structuring and styling web pages.

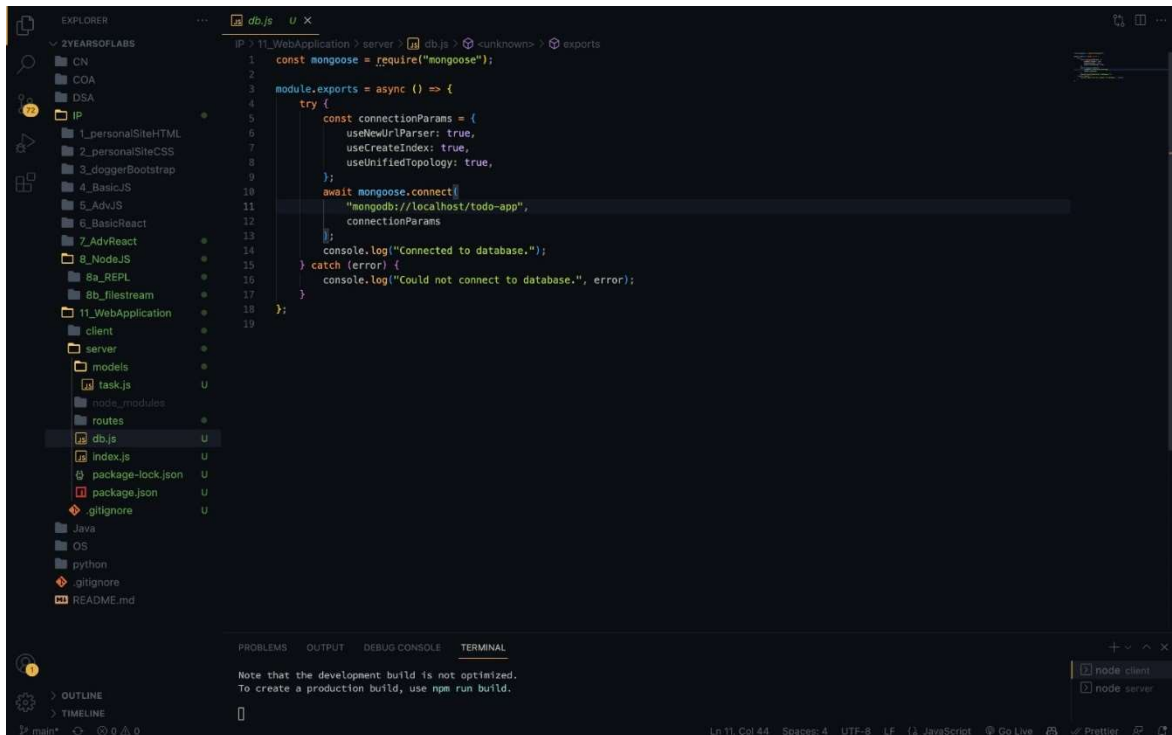
Backend:

- Node.js: A JavaScript runtime for executing server-side code.
- Express.js: A web application framework for building APIs and web applications in Node.js.
- MongoDB: A NoSQL database for storing and managing the to-do list data.
- Mongoose: An ODM (Object-Document Mapping) library for MongoDB, simplifying interactions with the database.

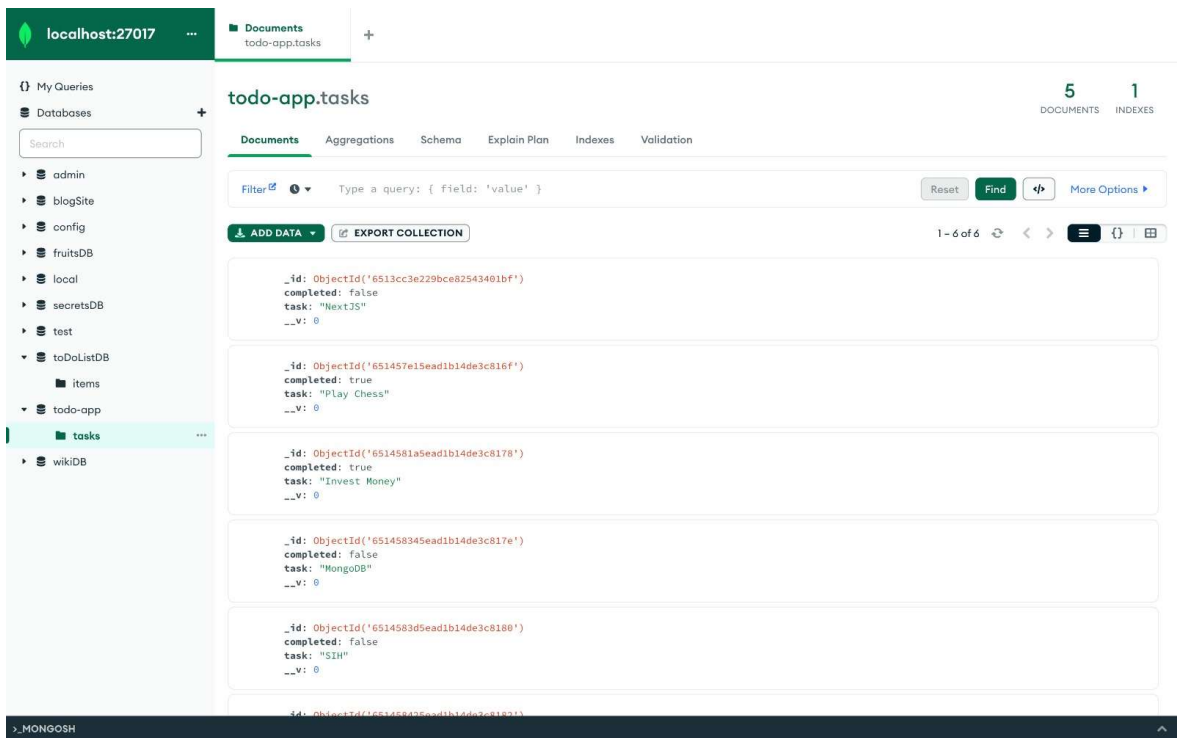
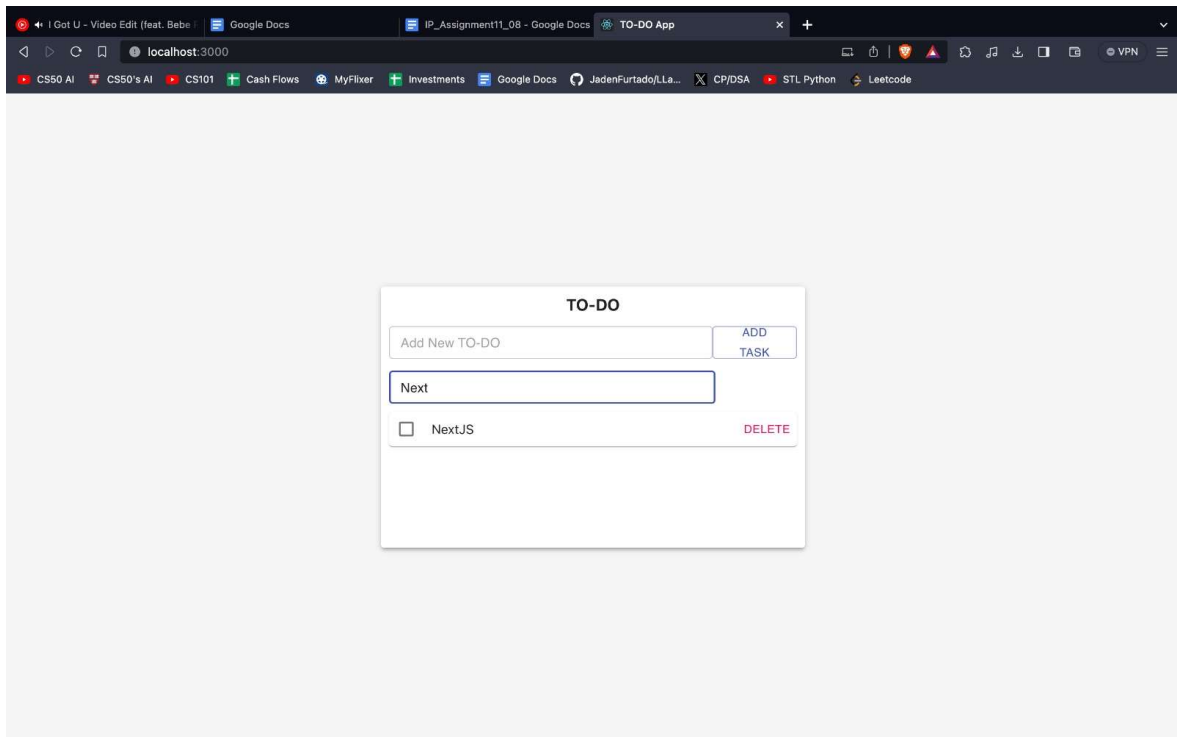
Code:

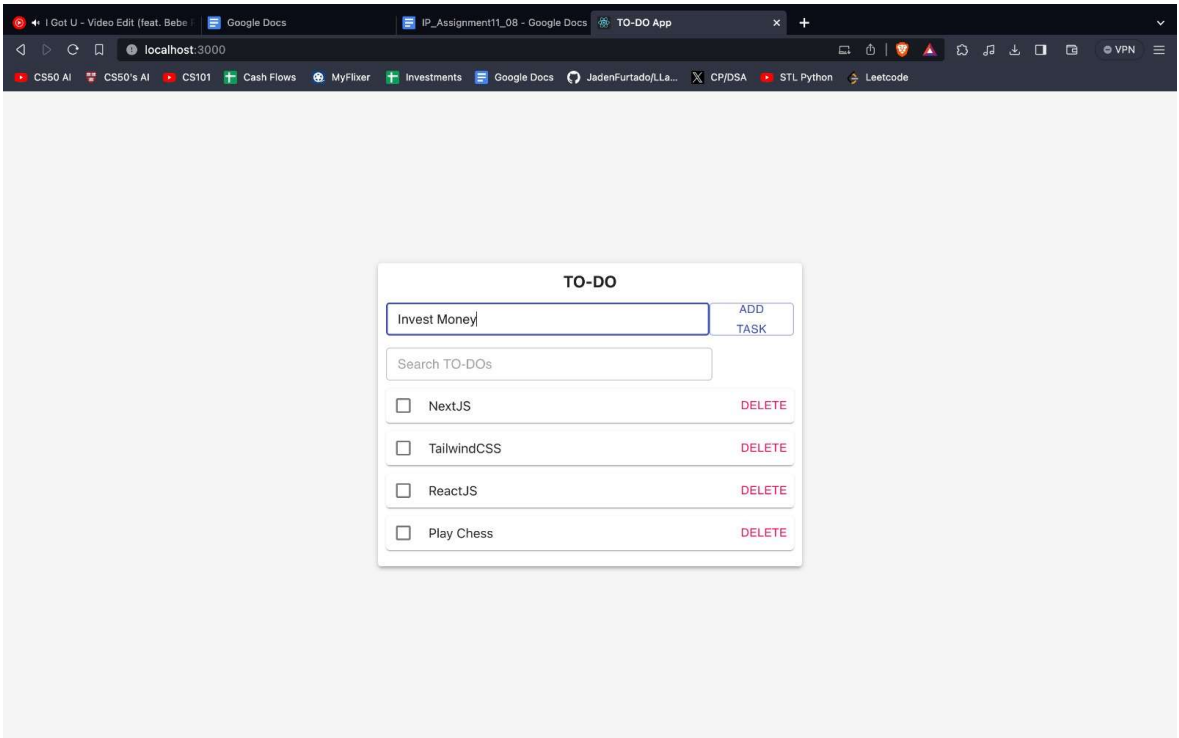
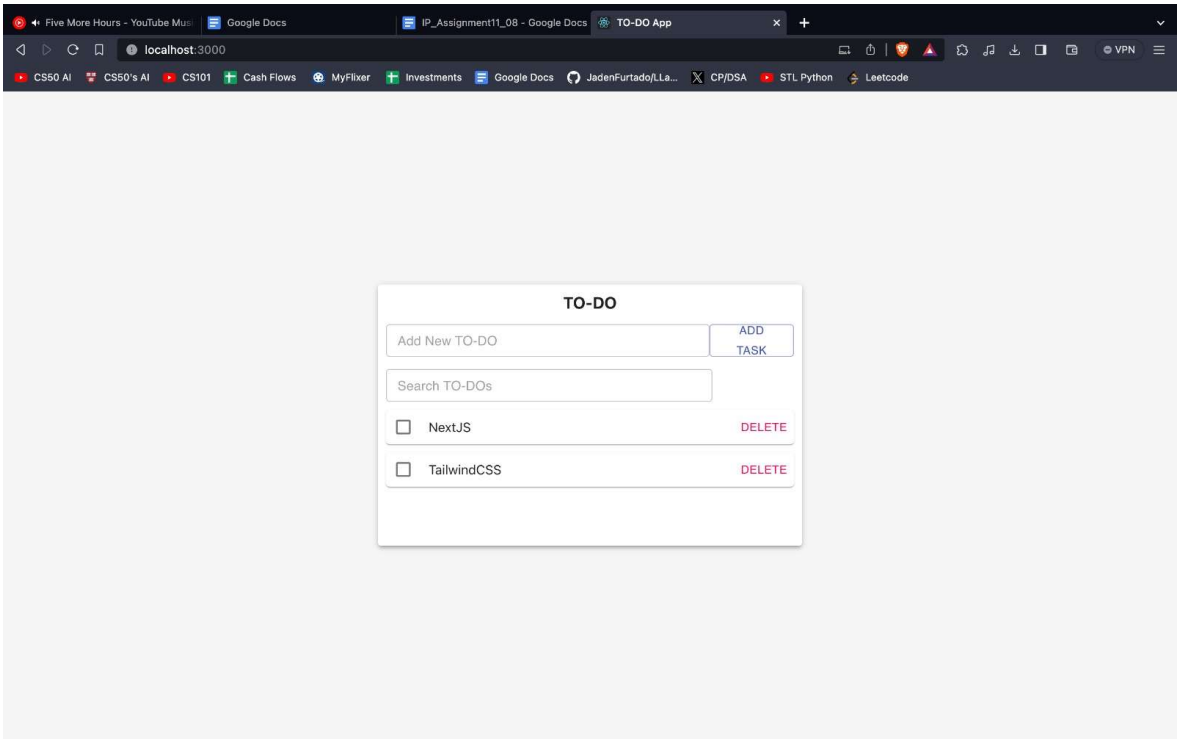


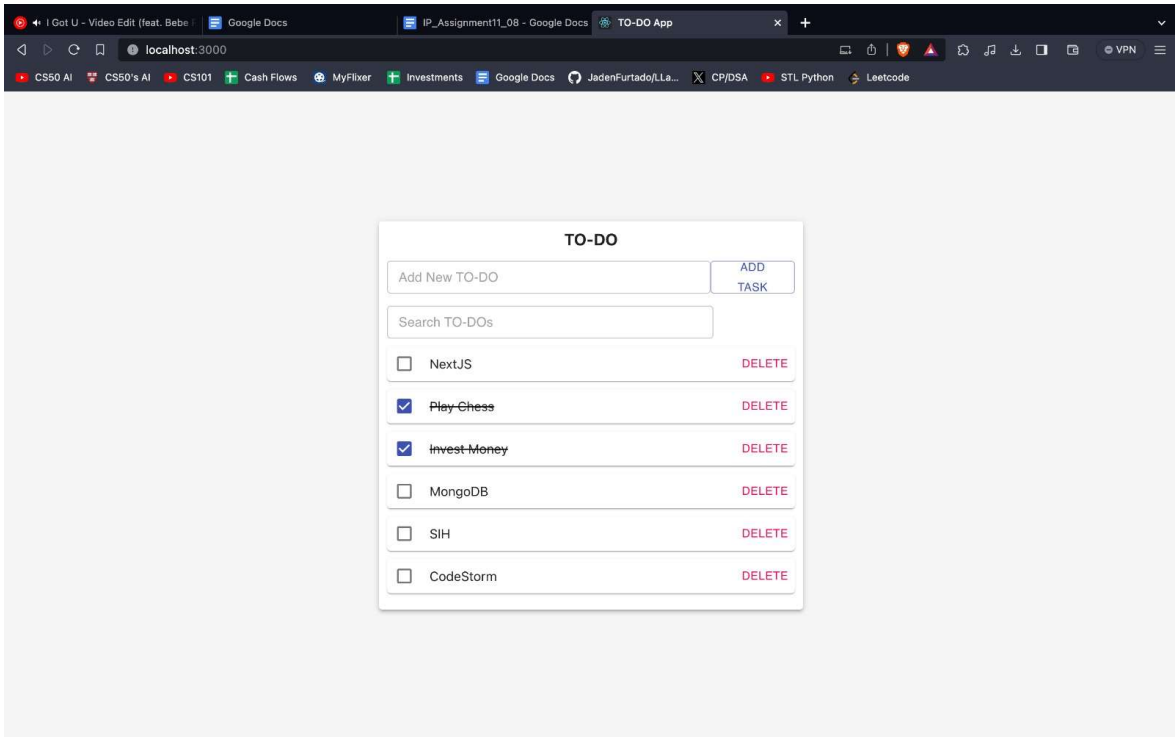
```
1 const mongoose = require("mongoose");
2 const Schema = mongoose.Schema;
3
4 const taskSchema = new Schema({
5   task: {
6     type: String,
7     required: true,
8   },
9   completed: {
10    type: Boolean,
11    default: false,
12  },
13});
14
15 module.exports = mongoose.model("task", taskSchema);
```



```
1 const mongoose = require("mongoose");
2
3 module.exports = async () => {
4   try {
5     const connectionParams = {
6       useNewUrlParser: true,
7       useCreateIndex: true,
8       useUnifiedTopology: true,
9     };
10    await mongoose.connect(
11      "mongodb://localhost/todo-app",
12      connectionParams
13    );
14    console.log("Connected to database.");
15  } catch (error) {
16    console.log("Could not connect to database.", error);
17  }
18};
```







Lab Outcome : LO-6: Construct back end applications using Node.js/Express.