9530

St. MOTHER THERESA ENGINEERING COLLEGE

COMPUTER SCIENCE ENGINEERING

NM-ID: D98DDCEBB8E8BB997484B143AE89341D

REG NO: 953023104079

DATE: 15-09-2025

Completed the project named as

Phase II

FRONTEND TECHNOLOGY

TO DO LIST APPLICATION

SUBMITTED BY:

M. Navanitha

6385015298

Tech Stack Selection

The success of any software project depends largely on the right choice of technology stack. For a To-Do List Application, the stack must be lightweight, scalable, secure, and developer-friendly. The chosen stack here is the MERN stack (MongoDB, Express.js, React.js, and Node.js), which is widely used for full-stack web development.

On the frontend, React.js is selected because of its component-based structure, which enables the creation of reusable and interactive UI components. It also provides efficient state management and fast rendering with the virtual DOM. Styling can be managed using Tailwind CSS or Bootstrap for a clean, responsive interface. API requests from the frontend to the backend can be made using Axios or the native Fetch API.

On the backend, Node.js is the runtime environment and Express.js is the web framework. This combination is ideal because it is lightweight, asynchronous, and efficient in handling REST API calls. Express provides built-in routing and middleware features, making it easier to build secure and maintainable APIs. Authentication can be implemented using JSON Web Tokens (JWT), ensuring that only registered users can access their tasks.

The database chosen is MongoDB, a NoSQL database that stores data in a flexible JSON-like structure. This is particularly suitable for tasks because they may include varying details such as title, description, due date, and priority. With MongoDB Atlas (cloud-hosted), data storage and scaling become simpler.

For hosting, the frontend can be deployed on Netlify or Vercel, while the backend can run on Render or Heroku. MongoDB Atlas provides a cloud database solution, eliminating the need for manual server setup.

In conclusion, this stack provides speed, flexibility, and scalability. It is also widely supported, meaning resources, tutorials, and libraries are abundant. This makes it a strong choice for developing a simple yet production-ready To-Do List Application.

UI Structure / API Schema Design

A well-designed User Interface (UI) structure and API schema form the backbone of the To-Do List Application. The UI must be simple, intuitive, and user-friendly so that users can add, edit, delete, and manage tasks without difficulty. Similarly, the API schema should clearly define how the frontend communicates with the backend to perform these operations securely and efficiently.

UI Structure

The application consists of the following main screens:

1. **Authentication Pages** – Login and Signup pages allow users to register and securely log in. After successful authentication, the system issues a JSON Web Token (JWT) stored on the client side.
2. **Dashboard** – Once logged in, the user is redirected to the dashboard, which displays a list of tasks.
3. **Task Form** – This form is used to create or edit tasks. Fields include title, description, due date, and priority level.
4. **Task List View** – Tasks can be displayed under categories such as Pending, Completed, or All. Each task has buttons for editing, marking as complete, or deleting.
5. **Navigation Bar** – Provides quick access to dashboard, profile, and logout functions.

This structure ensures a smooth user experience, where every essential action is a click away.

API Schema Design

The REST API provides endpoints that allow the frontend to interact with the backend. The design follows standard CRUD operations:

Authentication Routes:

- POST /api/auth/signup → Registers a new user.
- POST /api/auth/login → Authenticates user and returns a token.

Task Routes:

- GET /api/tasks → Fetches all tasks for the logged-in user.
- POST /api/tasks → Creates a new task.
- PUT /api/tasks/:id → Updates an existing task.
- DELETE /api/tasks/:id → Deletes a task.

Data Handling Approach

The data handling approach of a To-Do List Application is crucial for ensuring that user information and tasks are managed securely, efficiently, and accurately across the system. It involves the coordination of the frontend, backend, and database layers to maintain smooth functionality.

Frontend Handling

On the frontend, built with React.js, the primary responsibility is to collect user input, display tasks, and maintain state. When a user logs in, the system issues a JWT (JSON Web Token), which is stored securely in localStorage or sessionStorage. This token is included in every subsequent request to the backend for authentication. React's state management (via Context API or Redux) is used to store and update the task list in real time. Whenever a task is added, edited, or deleted, the UI immediately reflects the change, ensuring responsiveness.

Backend Handling

The backend, powered by Node.js and Express.js, processes incoming API requests. Each request first passes through an authentication middleware that validates the JWT token. Once verified, the corresponding controller handles the operation, such as creating a new task or updating an existing one. Input validation is enforced to ensure correct data formats (e.g., valid dates, non-empty task titles). Error handling ensures proper feedback is given in case of failures, such as unauthorized access or invalid inputs.

Database Handling

MongoDB stores data in collections. A User collection maintains login credentials (with encrypted passwords), and a Task collection stores tasks, each linked to a userId. Each task record includes fields like title, description, due date, priority, and status. This relational link ensures that tasks are isolated to individual users. Indexing is applied for faster retrieval of tasks based on status or due date.

Security&Reliability

Passwords are hashed using bcrypt, and JWT ensures secure session management. Both client-side and server-side validations protect against malicious data entry.

Component / Module Diagram

A well-structured component/module diagram is crucial for organizing the development of a To-Do List Application. It shows how different parts of the system interact with each other and ensures the application is built in a modular and scalable way. Both the frontend (React) and backend (Node.js + Express) are broken into smaller, manageable units for clarity and maintainability.

Frontend Components (React.js)

1. **App Component** – The root component that defines routes and acts as the entry point.
2. **Navbar Component** – Displays navigation links such as Home, Dashboard, Login, and Logout.
3. **Auth Component** – Handles Login and Signup forms, manages authentication states.
4. **TaskList Component** – Renders a list of tasks retrieved from the backend.
5. **TaskItem Component** – Represents a single task, showing details like title, status, and action buttons (Edit/Delete/Complete).
6. **TaskForm Component** – Provides a form for adding or editing tasks with fields for title, description, due date, and priority.

Backend Modules (Node.js + Express)

1. **server.js** – Entry point that initializes Express and connects to the database.
2. **routes/** – Defines API routes (authRoutes.js, taskRoutes.js).
3. **controllers/** – Contains logic for handling requests and sending responses (e.g., taskController.js for CRUD operations).
4. **models/** – Defines MongoDB schemas for User and Task.
5. **middleware/** – Implements authentication checks using JWT to protect private routes.

Interactions

- ➢ The frontend sends requests via Axios/Fetch → routes in Express → controllers → database → response returned to frontend.
- ➢ Each layer is independent but works together, making the app easy to extend (e.g., adding notifications or categories).

In conclusion, the component/module diagram ensures that the application is organized, scalable, and maintainable, with clear separation of concerns between frontend UI and backend logic.

5. Basic Flow Diagram

The basic flow diagram of a To-Do List Application illustrates how data and requests move between the user interface, backend, and database. It helps in understanding the overall functioning of the system by breaking it down into a step-by-step process.

Step 1: User Interaction

The flow begins when a user interacts with the frontend interface (React.js). For example, they may sign up, log in, add a task, mark it complete, or delete it. The frontend captures these actions and sends the necessary request to the backend API.

Step 2: Frontend to Backend Communication

Requests are made using Axios or Fetch API. For protected actions (like managing tasks), the frontend attaches the user's JWT (JSON Web Token) in the request header to prove authentication.

Step 3: Backend Processing

The backend (Node.js + Express.js) receives the request and passes it through middleware that checks whether the JWT token is valid. If valid, the request proceeds to the corresponding controller function. For example, when adding a task, the controller validates the input, assigns the logged-in user's ID to the task, and prepares it for database storage.

Step 4: Database Operations

The MongoDB database handles all storage. User data (credentials) is stored in the users collection, while task details are stored in the tasks collection. Each task references the userId to ensure tasks are linked to the correct user. CRUD operations (Create, Read, Update, Delete) are performed here.

Step 5: Response Back to Frontend

Once the database operation is successful, the backend sends a JSON response back to the frontend (e.g., { success: true, message: "Task added successfully" }). The frontend then updates its state and re-renders the UI to reflect the latest data.

Overall Flow

User → Frontend (React) → Backend (Node + Express) → Database (MongoDB)

← Response (JSON) ←