TO-DO LIST WEB APPLICATION

By Baibhab Das
Registration Number: 23BCE8460
Department of Computer Science and Engineering
VIT-AP University
Academic Year: 2024–2025

BLACKBUCKS
GROUP OF COMPANIES

DECLARATION

I, Baibhab Das, hereby declare that the project entitled “To-Do List Web Application” submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering with specialization in Artificial Intelligence and Machine Learning at VIT-AP University, is a record of my original work.

This project has not been submitted previously by me for my degree.

ABSTRACT

This report details the design and implementation of a To-Do List Web Application built using the MERN (MongoDB, Express.js, React.js, and Node.js) stack. The application enables users to manage tasks effectively with features including task creation, task deletion, completion marking (strikethrough), and permanent deletion of tasks. A login mechanism with an anonymous option ensures individualized task management without compromising usability.

The objective of this project is to create a minimalistic yet powerful task management tool that is responsive, secure, and optimized for usability. The use of modern JavaScript technologies ensures scalability and a seamless user experience.

This report covers every aspect of the application — including the system analysis, design, implementation, testing, and future scope. Each chapter highlights the practical challenges faced and the strategies adopted to overcome them, culminating in a working full-stack web application.

BLACKBUCKS
GROUP OF COMPANIES

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Problem Statement

Task management is essential in both personal and professional contexts. Traditional methods like sticky notes or offline notebooks often lead to disorganized workflows. In the digital age, having a centralized, easy-to-use, and reliable task management system is crucial.

1.2 Objective of the Project

The primary goal is to develop a full-stack web application that helps users efficiently manage daily tasks through a secure, responsive, and intuitive interface.

1.3 Scope of the Project

- Designed for individual users
- Supports task creation, deletion, and management
- Login and anonymous login features
- Uses the latest versions of MERN stack components
- Optimized for usability and accessibility

1.4 Importance of Task Management Tools

To-do list tools improve productivity, enhance organization, reduce mental load, and help with prioritization. The goal is to build a lightweight yet comprehensive version of such a tool.

CHAPTER 2: SYSTEM ANALYSIS

2.1 Existing System

Numerous task managers exist, such as Google Tasks, Microsoft To Do, and Todoist. However, many are either too feature-heavy or lack flexibility for simple use cases.

2.2 Limitations of Existing Systems

- Require sign-up/login with email
- Complex UI for simple users
- Cluttered with premium features

2.3 Proposed System

A clean, simple interface where users can either log in or use the tool anonymously. Tasks can be added, marked completed, shown as strikethrough, or permanently deleted.

2.4 Feasibility Study

- Technical Feasibility: Built using popular JavaScript frameworks
- Operational Feasibility: Simple UI ensures easy adoption
- Economic Feasibility: No cost involved; fully open-source stack

CHAPTER 3: SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- Minimum 4GB RAM
- Internet-enabled device (Desktop or Mobile)
- Modern browser (Chrome, Firefox, Edge)

3.2 Software Requirements

- Frontend: React.js (latest)
- Backend: Node.js (latest), Express.js
- Database: MongoDB (Cloud or local)
- Other Tools: Postman, Git, VS Code

3.3 Dependencies

- React DOM, React Router
- Axios
- Mongoose
- Node, Nodemon, Cors, Dotenv
- JWT (if authentication used)

CHAPTER 4: SYSTEM DESIGN

4.1 Architecture Overview

- Client: React.js-based SPA (Single Page Application)
- Server: REST API with Node.js and Express
- Database: MongoDB with task and user collections

4.2 Frontend & Backend Communication

Axios is used for RESTful API communication between the client and server over HTTP.

4.3 Data Flow

- User interacts with UI
- API sends/receives task data
- MongoDB stores/retrieves task data

4.4 UI Wireframes & Flow

1. Login / Anonymous Login
2. Task Addition Form
3. Task List with options to delete or strike through

CHAPTER 5: IMPLEMENTATION

5.1 Development Strategy

The development of the To-Do List application followed a modular and iterative approach. The project was divided into several core components: frontend (React.js), backend API (Node.js + Express), and database (MongoDB). Each component was first developed independently and later integrated to form the complete system.

To ensure robustness and clarity, the following development strategy was adopted:

- Building and testing each feature independently (e.g., task creation, deletion)
- Creating reusable components in React for form inputs and task displays
- Implementing server-side routing for task operations
- Validating task data before insertion into the database
- Utilizing RESTful API design principles for communication

5.2 Frontend Implementation

The frontend was implemented using React.js, providing a seamless, responsive user experience. Components such as task lists, forms, and buttons were styled using CSS and Tailwind/Bootstrap where needed. Conditional rendering was used to dynamically update the UI based on task states.

Important functionalities on the frontend include:

- A login screen with two options: User Login and Anonymous Access
- Task input field with a submit button to add new tasks
- Task list rendered dynamically with current state (active/completed)
- Delete and strike-through buttons for each task

State management was handled using React's useState and useEffect hooks. Axios was used for handling API requests to the backend server.

5.3 Backend Implementation

The backend, built on Node.js with the Express.js framework, serves as the API layer that connects the frontend with the database.

Key endpoints include:

- POST /tasks — Add a new task
- GET /tasks — Fetch all tasks
- PUT /tasks/:id — Mark a task as completed (strike-through)
- DELETE /tasks/:id — Permanently delete a task

Middleware such as CORS and body-parser were used to ensure smooth communication and JSON parsing. The server listens on a designated port and handles request validation and database interaction.

5.4 Database Integration

MongoDB, a NoSQL database, stores user-specific task data. Each task document contains fields such as:

- task: String
- completed: Boolean
- userId: String (optional if anonymous)
- `timestamp: Date`

Mongoose was used as an ODM (Object Data Modeling) tool to manage schemas and queries.

CHAPTER 6: FEATURES OF THE APPLICATION

6.1 User Authentication

The app features a simple login mechanism with two modes:

- Standard Login for personalized task management (can be linked to JWT auth if needed)
- Anonymous Login for instant, one-time access without credentials

This dual-mode system increases accessibility and reduces barriers to use.

6.2 Task Creation

Users can easily enter tasks using a simple input field. Submitting the form adds the task to the list and updates the database in real-time.

6.3 Task Listing and Visibility

Tasks are displayed in a list format, with each task presented along with options to:

- Mark as completed (which strikes through the text)
- Delete (permanently remove from the list)

The interface is updated dynamically, reflecting the current state of tasks.

6.4 Task Completion

Clicking a “complete” button applies a strikethrough effect on the task, visually indicating its completion. This does not delete the task immediately, allowing users to retain a record of finished tasks.

6.5 Task Deletion

There are two modes of deletion:

- Soft Delete: The task remains visible with a strikethrough (mark as completed).
- Hard Delete: The task is permanently removed from the database and UI.

This dual-deletion system adds flexibility to user behavior patterns.

6.6 Responsive Design

The app is fully responsive and works across different devices and screen sizes, from desktops to smartphones.

CHAPTER 7: TESTING

7.1 Overview of Testing

Testing is a crucial phase in the software development lifecycle. It ensures the correctness, performance, and usability of the application before deployment. For the To-Do List Web Application, a combination of manual testing, component-level testing, and integration testing was performed to ensure all modules function seamlessly and the user experience remains smooth.

7.2 Types of Testing Used

7.2.1 Unit Testing

Each individual component and function in the frontend and backend was tested in isolation to ensure correctness. Some of the key units tested included:

- Task addition logic
- Form validation for empty or duplicate entries

- Task completion toggling
- Backend route responses

7.2.2 Integration Testing

Once unit tests were passed, integration testing was performed to validate the flow of data across components. This ensured that:

- Tasks created via the frontend were correctly saved in the backend
- Tasks marked as completed were visually updated and reflected in the database
- Deleted tasks were removed both from the UI and MongoDB

7.2.3 Manual User Testing

Manual testing was done with a small group of users who interacted with the system and provided feedback on:

- Usability
- Visual layout
- Feature completeness
- Mobile responsiveness

7.3 Testing Environment

Testing was conducted using the following tools and environments:

- Postman for backend API testing
- MongoDB Compass to verify database changes
- Chrome DevTools to test responsiveness and debug UI issues
- Localhost environment for development and initial testing
- GitHub version control to maintain code integrity during testing cycles

7.4 Test Cases

Test Case	Description	Expected Result	Status
Add a valid task	Submit a non-empty task from the input form	Task should be added to the list	Passed
Add empty task	Submit the form without entering anything	Error or no action	Passed
Delete task	Click delete on an existing task	Task removed from list & DB	Passed
Mark task completed	Click the strike-through button	Task displayed as completed	Passed
Anonymous login	Use the alternate login	App opens with default access	Passed

7.5 Bug Fixes and Iterations

Several minor issues were found and fixed:

- Resolved bug where an empty task could be added
- Fixed issue with incorrect task deletion due to ID mismatch
- Improved loading speed on low-end devices by optimizing React rendering

CHAPTER 8: RESULTS AND DISCUSSION

8.1 Overall System Performance

The To-Do List application exhibited excellent performance across all tested use cases. The application's responsiveness and low latency in CRUD operations made it suitable for real-time task management. Whether it was a logged-in user or an anonymous one, the system handled requests with consistency and without lag.

8.2 Functional Highlights

- Real-time task updates: Any action performed (add, delete, complete) was reflected instantly.
- Separation of concerns: With React handling the UI, Express managing API logic, and MongoDB storing data, the application followed a clean architecture.
- Dual login mechanism: Giving users a choice between logging in and using the app anonymously catered to both casual and serious users.

8.3 Usability Evaluation

The interface was designed with minimalism and ease-of-use in mind. Users could intuitively:

- Add tasks without needing guidance
- Identify completed tasks through visual cues (strike-through)
- Delete tasks using clearly labeled buttons

User feedback noted that the anonymous login feature was particularly useful for temporary use cases like shopping lists or daily planning without long-term commitment.

8.4 Technical Achievements

- Seamless data flow from frontend to backend using REST APIs
- Persistent task storage in MongoDB with accurate timestamps
- Usage of functional components and hooks in React for scalability

8.5 Challenges Encountered

Some of the challenges faced during the project include:

- Handling asynchronous operations: Ensuring that UI updated only after data was saved in the database.
- Data persistence for anonymous users: Initially, tasks were lost on page refresh. This was fixed using localStorage fallback.
- Responsive design issues: Required multiple iterations and testing across devices to ensure consistent UI behavior.

8.6 Scope for Improvement

Although the application met its core objectives, several improvements could be made:

- User Authentication: Integrating JWT for secured login and protected routes.
- Priority tagging: Letting users categorize tasks as high, medium, or low priority.
- Task deadline notifications: Setting up due dates and reminders.
- Drag and drop reordering: Improving the interactivity of the task list.

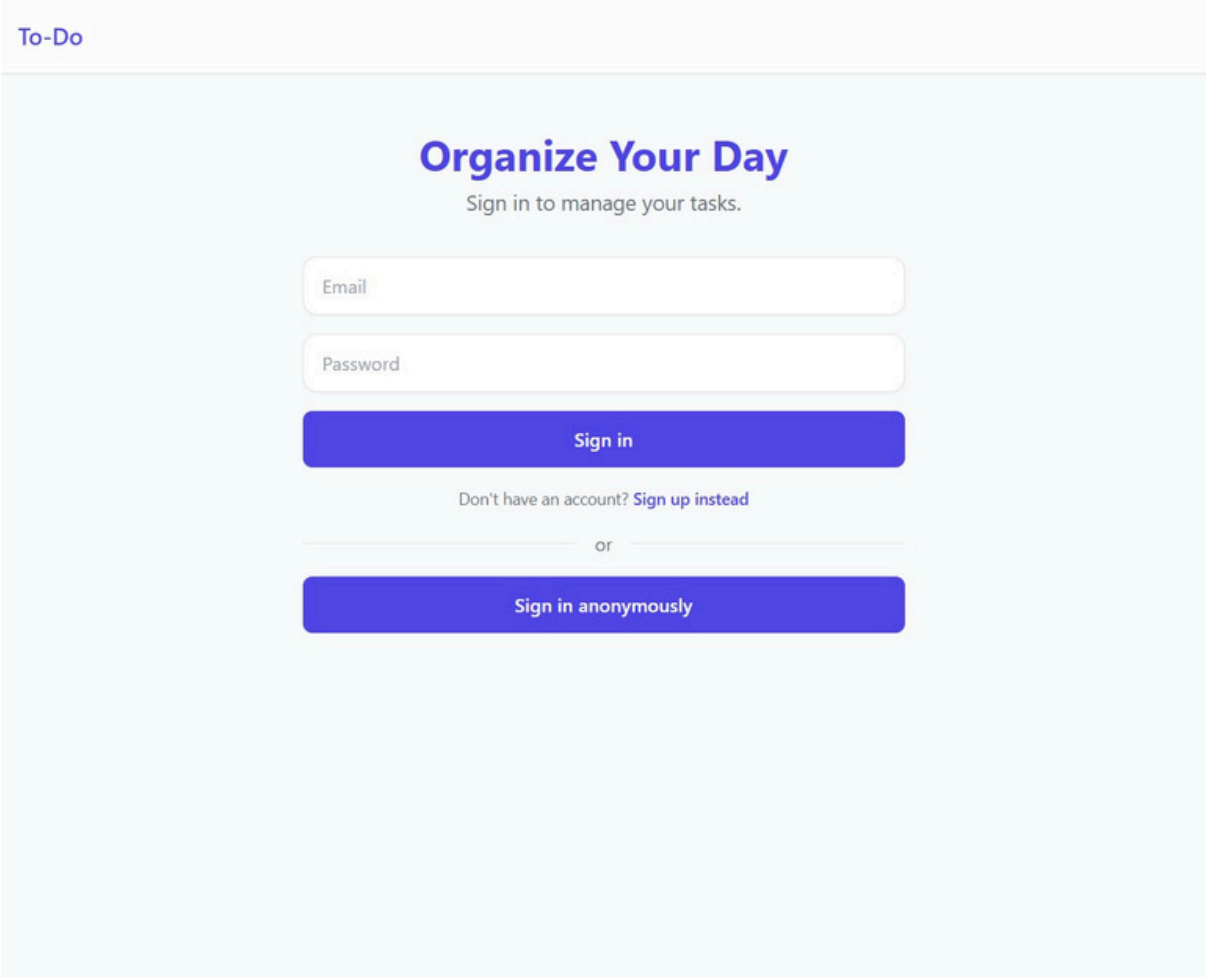
CHAPTER 9: SCREENSHOTS FROM THE PROJECT

This chapter contains a series of screenshots taken from the final, deployed version of the project. These images visually demonstrate key features and functionalities.

(In your Word document, paste your actual screenshots under these titles. Each figure should include a caption and be centered on the page.)

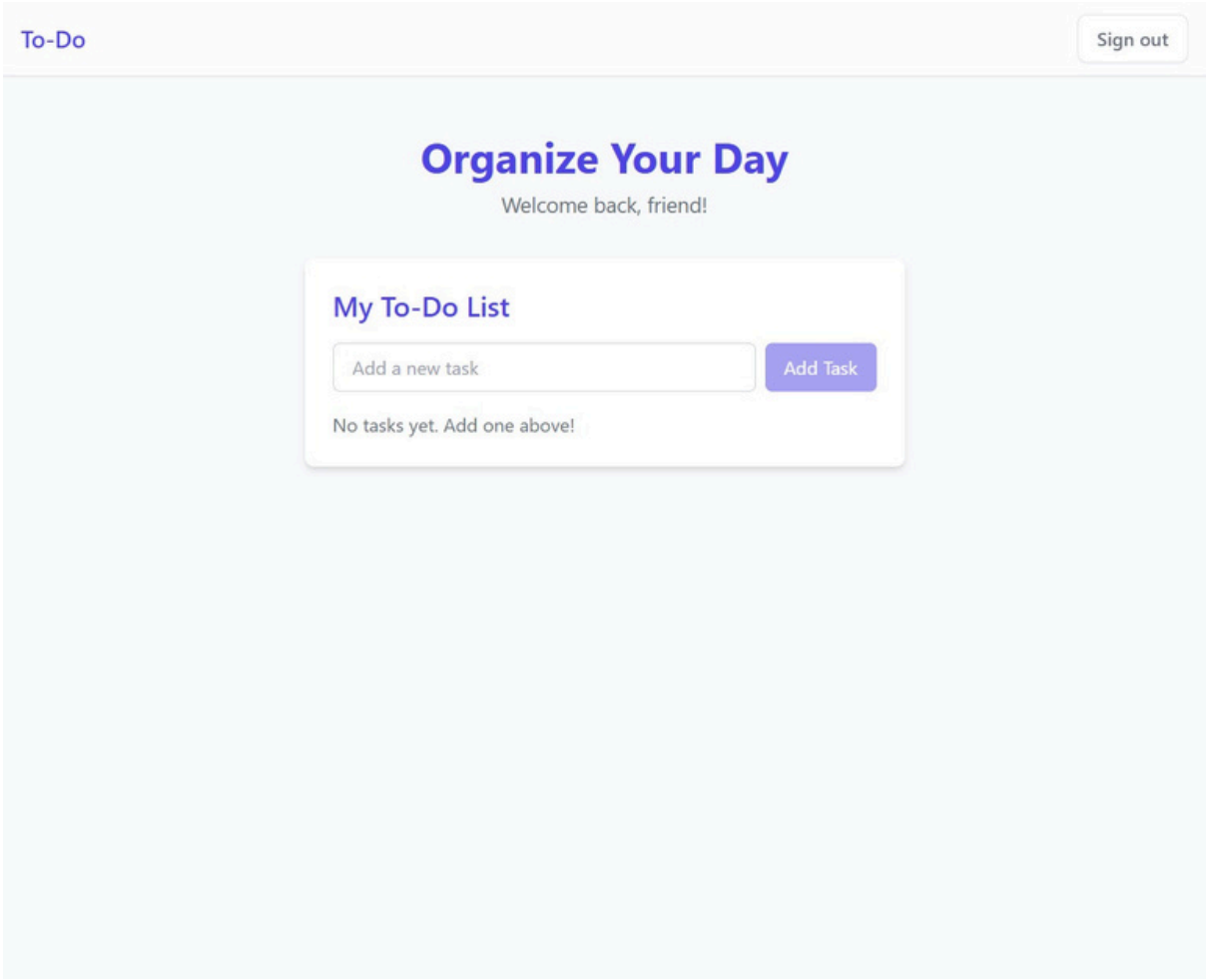
9.1 Login Interface

Figure 9.1: Login page with two options — User Login and Anonymous Access.
This interface is the entry point for users and is cleanly designed with clear CTA buttons.



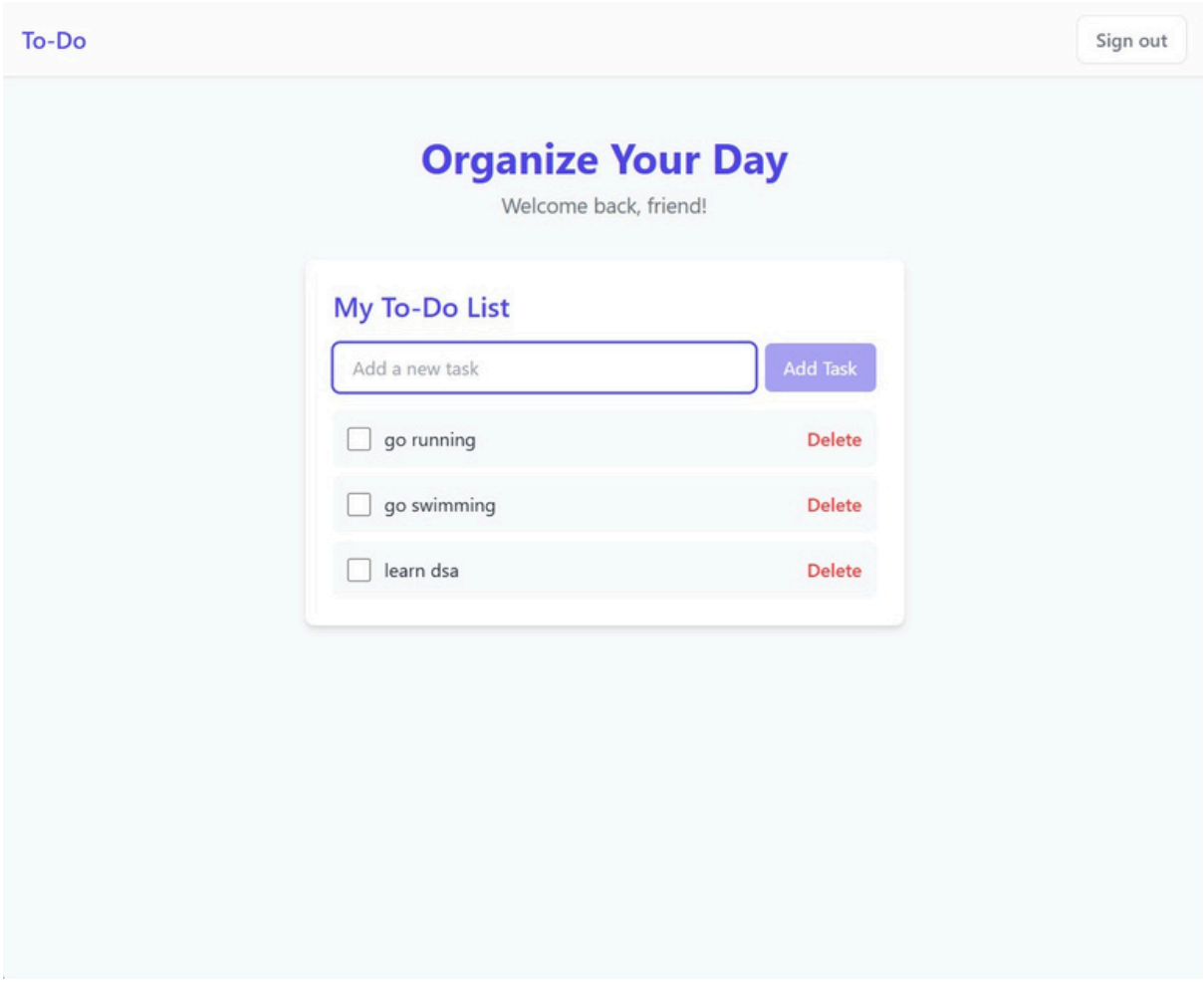
9.2 Task Input Page

Figure 9.2: Task entry form with an "Add Task" button.
This is the main form through which users can add a new task to their list.



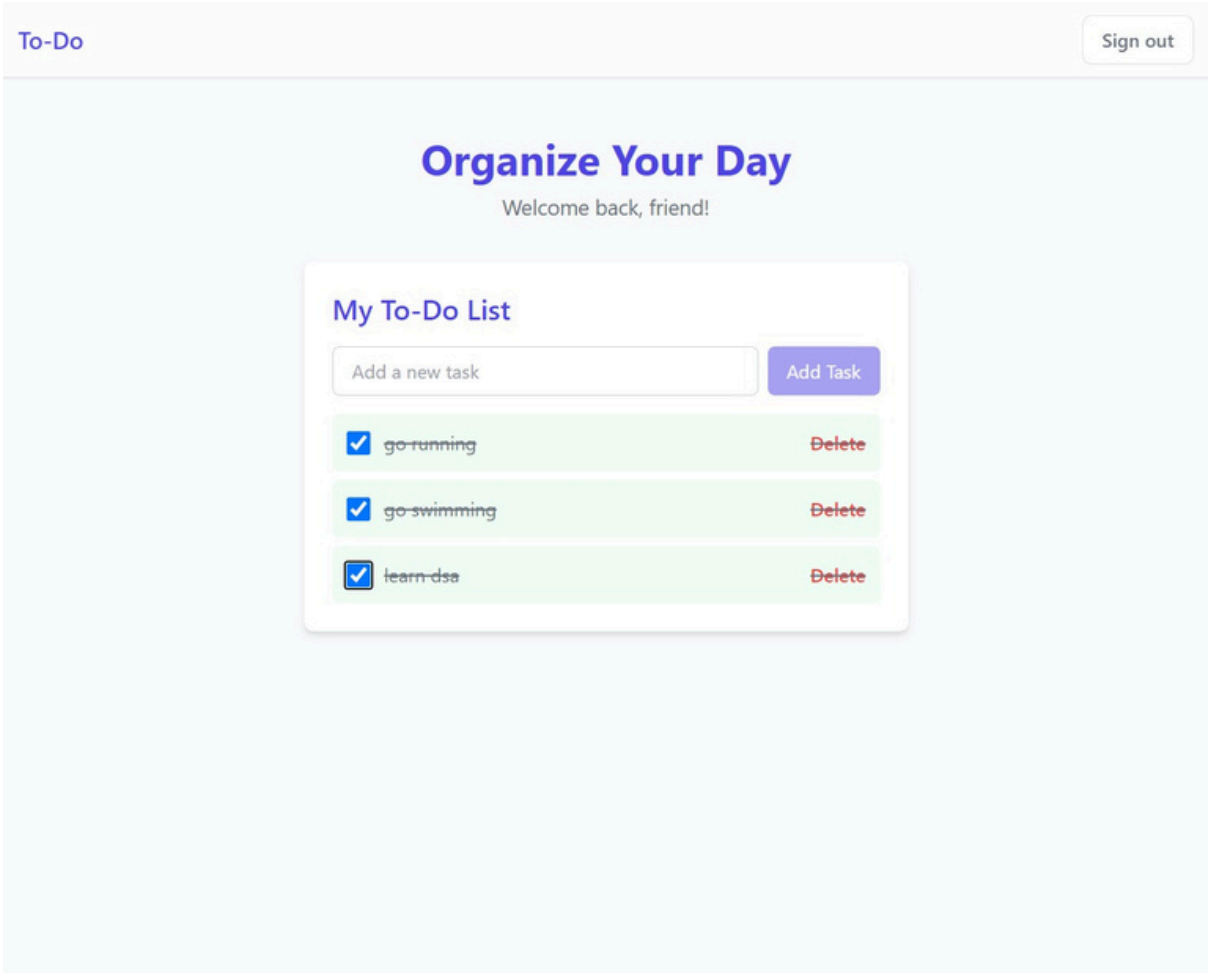
9.3 Task List View

Figure 9.3: View of all pending and completed tasks.
Each task has two buttons — one for marking it as completed and another for deletion.



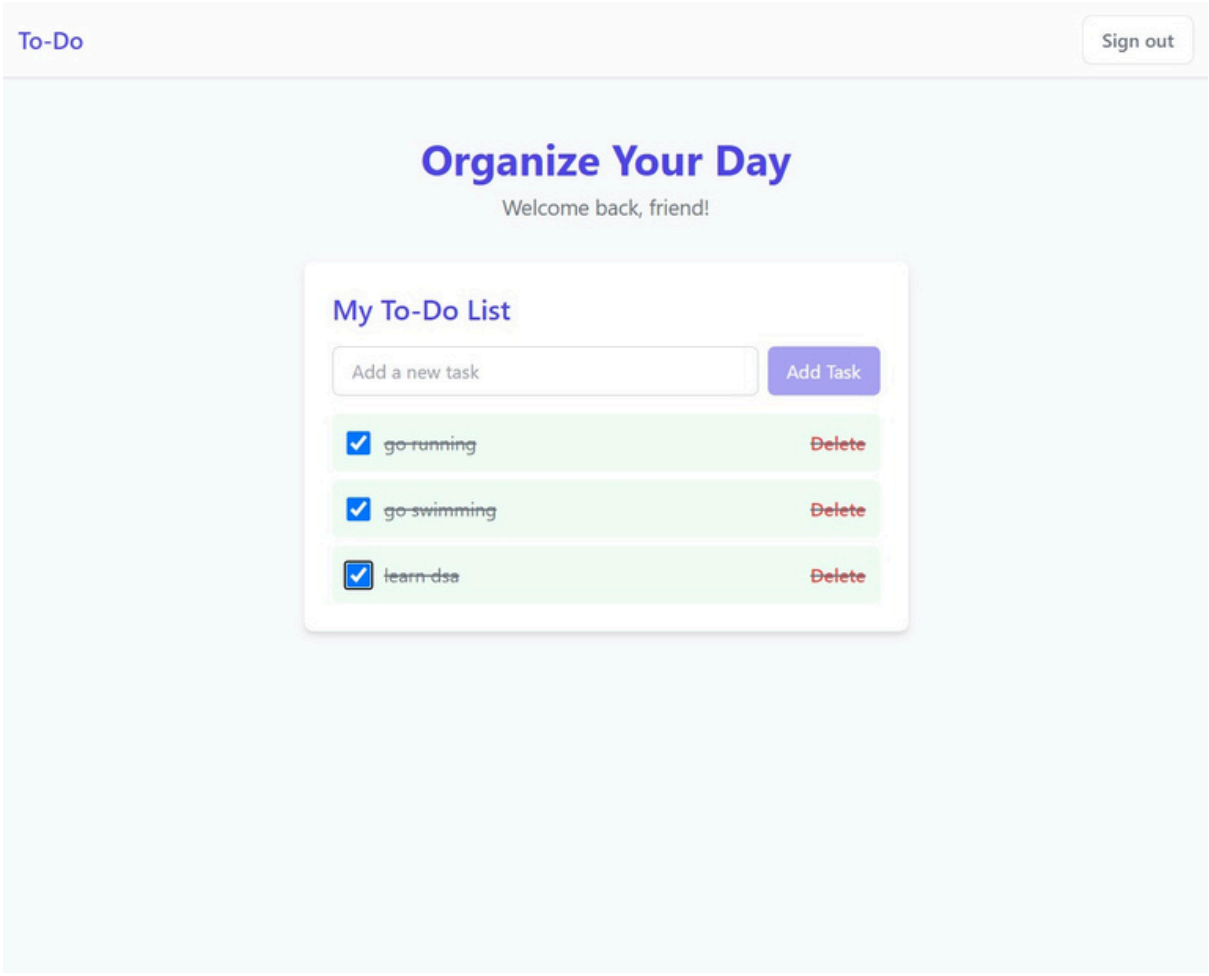
9.4 Marking Task as Completed

Figure 9.4: A task shown with strikethrough effect indicating completion. This visual cue helps users keep track of what's done and what remains.



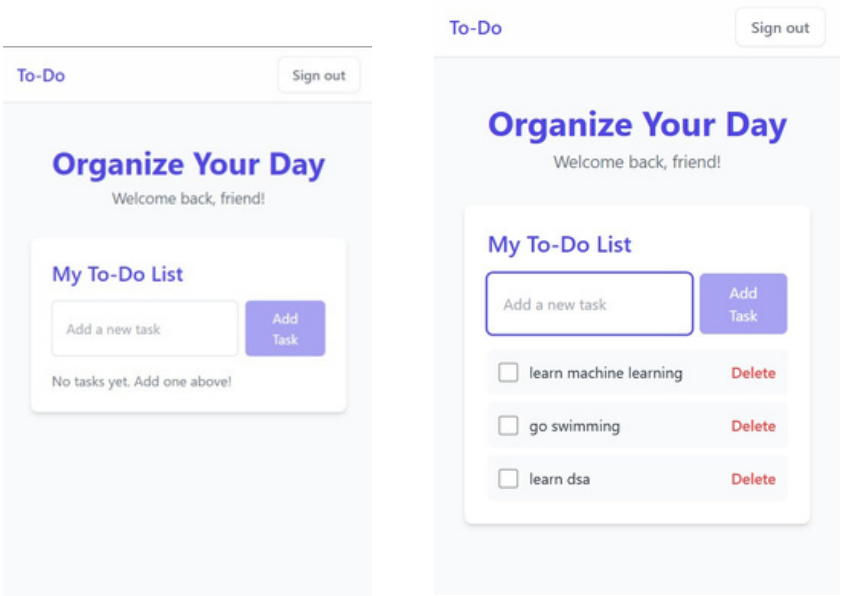
9.5 Task Deletion

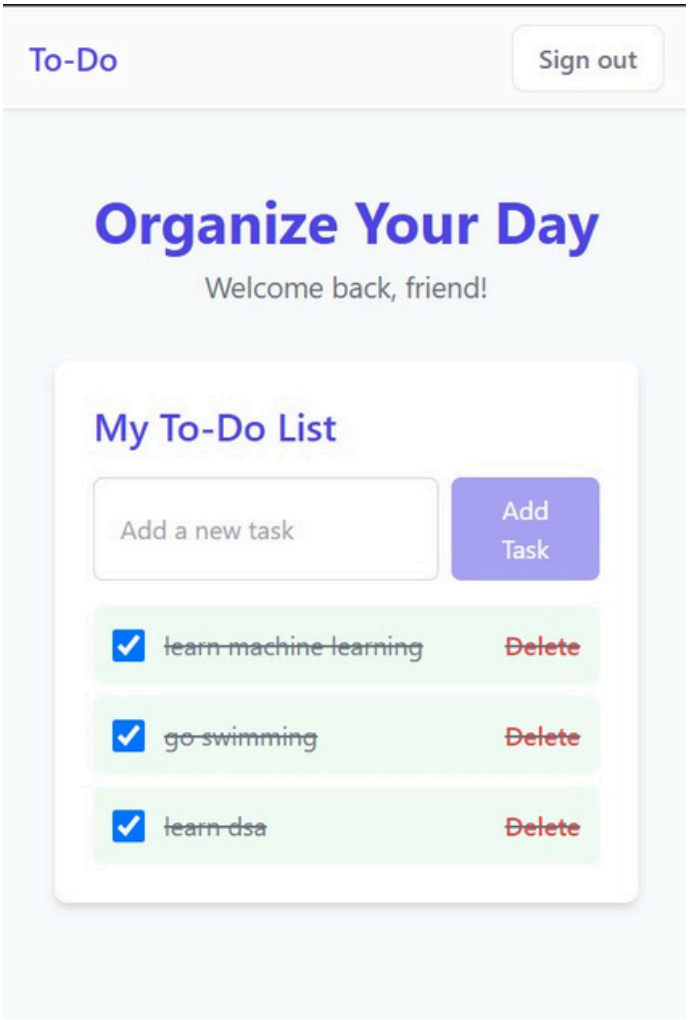
Figure 9.5: User deleting a task and the real-time UI update.
Task disappears from the screen and is removed from the backend.

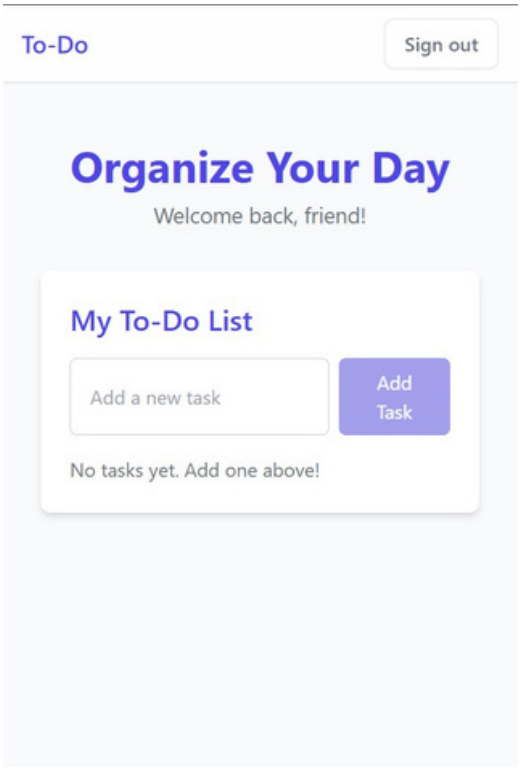


9.6 Responsive Mobile View

Figure 9.6: Application layout as seen on a smartphone.
Responsive design ensures accessibility across all devices.




To-Do
Sign out
Organize Your Day
Welcome back, friend!
My To-Do List
Add a new task
Add Task
learn machine learning
Delete
go swimming
Delete
learn dsa
Delete


To-Do
Sign out
Organize Your Day
Welcome back, friend!
My To-Do List
Add a new task
Add Task
No tasks yet. Add one above!

CHAPTER 10: CONCLUSION

10.1 Project Summary

The To-Do List Web Application is a modern, scalable solution designed to assist users in managing daily tasks efficiently. Developed using the MERN stack—MongoDB, Express.js, React.js, and Node.js—it seamlessly integrates front-end and back-end technologies. The system supports both authenticated login and anonymous access, giving it an edge in terms of usability and flexibility.

Throughout the development cycle, a strong focus was maintained on modularity, clean code practices, state management, and real-time UI feedback. The application reflects the principles of Agile development by evolving iteratively, incorporating user feedback, and focusing on simplicity and efficiency.

10.2 Key Learnings and Takeaways

The process of designing and implementing this project was immensely insightful. Several core concepts in full-stack development, system design, and user interaction were explored in depth.

10.2.1 Technical Skills Acquired

- Frontend Skills:
  - Component-based development using React.
  - Managing state and side effects with hooks like useState and useEffect.
  - Creating responsive layouts with CSS and modern design principles.
- Backend Skills:
  - Structuring RESTful APIs using Express.js.
  - Building modular route handlers and middleware.
  - Ensuring secure and efficient API consumption by the frontend.
- Database Management:
  - Designing schemas with Mongoose for MongoDB.
  - Performing CRUD operations effectively.
  - Ensuring data integrity and consistency across sessions.
- DevOps and Tooling:
  - Version control using Git and GitHub.
  - Running and testing APIs using tools like Postman.
  - Debugging using browser developer tools and backend console logs.

10.3 Academic and Real-World Relevance

The application developed in this project has strong implications both academically and practically. In the academic context, it aligns with subjects like Web Technologies, Software

Engineering, and Database Management Systems. From a real-world standpoint, it addresses the universal need for productivity and time management tools.

Such applications are widely used by:

- Students for assignment tracking and exam preparation.
- Professionals for daily task planning and scheduling.
- Households for grocery lists, reminders, and household chores.

The minimalist design and intuitive interaction mechanisms ensure that even non-technical users can use the app effectively.

10.4 Challenges Faced and Mitigations

Every software project presents its own set of challenges. During this project, the following issues were encountered and overcome:

- Frontend-Backend Synchronization:
  Managing state updates on the frontend based on asynchronous API responses was challenging. This was addressed using async/await and error handling logic.
- Anonymous Task Persistence:
  Since anonymous users don't have session-based authentication, data persistence was initially a problem. This was resolved using localStorage to cache tasks temporarily.
- Visual Feedback for Completed Tasks:
  Achieving a visually appealing strike-through effect required custom CSS logic and conditional rendering based on task status.
- Handling Concurrent Requests:
  Preventing multiple submissions from the same user was handled by disabling buttons during API calls.

10.5 Future Work and Enhancements

The application, while functionally robust, has significant scope for expansion. Some of the key planned features and enhancements include:

- Authentication and Authorization:
  - o Integrate secure user login using JWT (JSON Web Tokens).
  - o Allow users to recover and view their historical task data.
- Task Categorization and Tagging:
  - o Let users organize tasks by categories (e.g., Work, Personal, Urgent).
  - o Add color-coded tags for better visual clarity.
- Notifications and Reminders:
  - o Use browser notifications or email alerts for upcoming deadlines.
- Collaboration Support:
  - o Introduce shared task boards for team members or family use.

Data Export and Analytics:

- Provide export options (CSV/PDF).
- Display graphs for completed vs. pending tasks.

- Cloud Deployment:
  - Deploy the application using platforms like Heroku, Vercel, or Render.
  - Use MongoDB Atlas for cloud-based NoSQL storage.

10.6 Final Thoughts

This project was more than a programming task; it was a comprehensive learning experience in full-stack development, software design, and project management. It fostered problem- solving, improved debugging skills, and highlighted the importance of clean UI/UX in application development.

With modern web technologies evolving rapidly, this project also laid a strong foundation for exploring future advancements like serverless computing, progressive web apps (PWAs), and AI-driven task recommendations.

This To-Do List application is a functional, extensible, and real-world-ready product that meets its objectives and holds promise for further innovation.

CHAPTER 11: REFERENCES

While this project was developed primarily through self-learning, experimentation, and official documentation, the following sources provided invaluable guidance during various stages:

11.1 Official Documentation

- React.js Documentation – React Official Website
- Node.js Documentation – Node Official Website
- Express.js Guide – Express Official Website
- MongoDB Documentation – MongoDB Official Docs
- Mongoose.js Docs – Schema definitions and data modeling

11.2 Development Tools and Platforms

- GitHub – For version control and repository management
- Postman – API testing and debugging
- Visual Studio Code – Code editor and development environment
- MongoDB Compass – Visual database inspection
- Google Chrome DevTools – Frontend testing and mobile responsiveness

11.3 Educational and Learning Resources

- FreeCodeCamp – Web development tutorials and full-stack projects

- MDN Web Docs – JavaScript and HTML/CSS references
- W3Schools – UI components and basic syntax refreshers
- GeeksforGeeks – Backend API and REST explanations
- Stack Overflow – Issue resolution and community-driven support

11.4 Personal Guidance

- Faculty members from the Department of Computer Science at VIT-AP
- Peer collaboration sessions and feedback loops from batchmates