Todo Application with Basic CRUD Operations

A PROJECT REPORT

Submitted by

Vidhi Khurana (23BCS11503)

Neha Sharma (23BCS10907)

Hardik Shukla (23BCS11503)

Saket (23BCS10750)

in partial fulfillment for the award of the degree of

Bachelor Of Engineering

IN

Computer Science and Engineering

Chandigarh University

November-2025

BONAFIDE CERTIFICATE

Certified that this project report " **Todo Application with Basic CRUD Operations**" is the bonafide work of "VIDHI KHURANA (23BCS11503), and NEHA SHARMA (23BCS10907) , HARDIK SHUKLA (23BCS12068) and SAKET (23BCSl0750) who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Er. Gagandeep Singh	Er. Aniket Raj
HEAD OF THE DEPARTMENT	**SUPERVISOR**
BE-CSE	ByteXL Trainer
Chandigarh University	**Computer Science and Engineering**

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

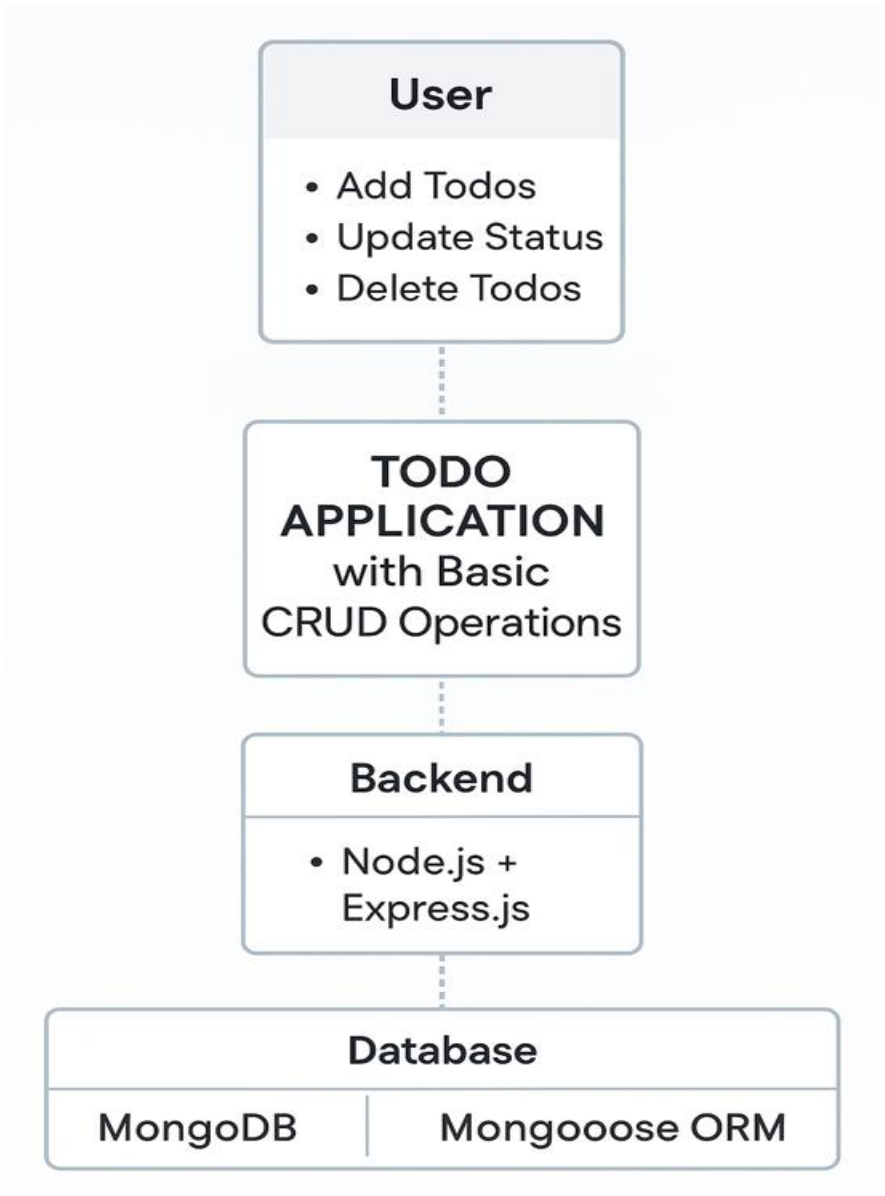
ABSTRACT

The **Todo Application with Basic CRUD Operations** is a full-stack web-based project developed using the MERN stack — **MongoDB**, **Express.js**, **React.js**, and **Node.js**. It enables users to manage their daily activities efficiently by performing four fundamental operations: Create, Read, Update, and Delete. The application aims to demonstrate how frontend, backend, and database components interact to form a functional and responsive system.

Users can add new tasks, view existing ones, mark them as completed, edit their titles, and delete them when no longer required. The frontend, built using React (Vite), provides a fast and interactive interface, while the backend, designed with Express and Node.js, handles all RESTful API requests. MongoDB acts as the persistent data store for todos, ensuring durability even after system restarts.

The project emphasizes clear separation of concerns and real-time synchronization between layers. It validates key concepts like API integration, asynchronous data flow, and dynamic rendering using React hooks. Through this implementation, the team gained hands-on experience in full-stack development, environment configuration, and cross-origin communication. The result is a lightweight yet powerful web application that exemplifies the CRUD cycle and modern software-engineering principles in action.

GRAPHICAL ABSTRACT



ABBREVIATIONS AND SYMBOLS

1. MERN - MongoDB, Express, React, Node
2. JWT - JSON Web Token
3. API - Application Programming Interface
4. User Interface
5. DB-Database

CHAPTER 1.
INTRODUCTION

1.1 Need Identification / Contemporary Issue

In today's fast-paced digital world, efficient task management has become an essential aspect of personal and professional productivity. Individuals often juggle multiple responsibilities—academic assignments, workplace goals, household tasks, and personal reminders—that demand an organized system to manage daily activities. Traditionally, people have relied on handwritten notes, diaries, or offline desktop applications to maintain to-do lists. However, these conventional methods fail to provide synchronization, accessibility across devices, and real-time updates. As lifestyles become increasingly digital, there is a growing need for applications that offer centralized access, automated updates, and user-friendly interfaces for task tracking.

1.2 Identification of Problem

In the present digital ecosystem, several task management tools and productivity applications are available; however, most of these solutions suffer from critical limitations. A large number of them are **proprietary or subscription-based**, making them inaccessible to users who require a free, open-source, and flexible alternative. Furthermore, existing systems are often **platform-dependent**—some are designed only for mobile use, while others are optimized for desktop environments, creating a fragmented user experience. These applications may also store user data on third-party servers, raising **privacy and data security concerns**.

1.3 Identification of Tasks

- Design user interface for adding, viewing, and managing tasks
- Develop backend APIs to handle CRUD operations
- Integrate MongoDB for persistent storage
- Implement dynamic update using React state hooks
- Provide clear UI feedback for user actions

1.4 Timeline

Phase	Duration	Task
Week 1–2	Requirement analysis and UI wireframe design	
Week 3–5	Backend API and database integration	
Week 6–8	Frontend development in React	
Week 9–10	Integration and testing	
Week 11	Deployment and documentation	

1.5 Organization of Report

The project report on **Todo Application with Basic CRUD Operations** is structured into four main chapters for clarity and logical flow. **Chapter 1** introduces the project, outlining its objectives, problem identification, and development plan. **Chapter 2** explains the system design, chosen technologies, and implementation methodology used to build the application. **Chapter 3** presents the results, analysis, and validation through testing of all CRUD functionalities. **Chapter 4** concludes the report, summarizing outcomes and suggesting future enhancements. The report also includes appendices containing the user manual, configuration steps, and a list of technologies used in the project.

CHAPTER 2.
DESIGN FLOW/PROCESS

2.1 Evaluation & Selection of Specifications / Features

The design of the **Todo Application with Basic CRUD Operations** began with an evaluation of several existing task management systems such as Microsoft To-Do, Google Tasks, and Todoist. These platforms were analyzed for their functionalities, including task creation, categorization, synchronization, and reminders. However, most of these solutions were either proprietary, required paid subscriptions, or lacked flexibility for customization and local deployment. Based on this evaluation, the team decided to develop an **open-source, lightweight, and responsive web-based Todo application** that could perform all essential CRUD operations efficiently.

To ensure high performance, **Vite** was used as a fast build tool, providing instant hot reload and optimized rendering. Features such as task filtering (All, Active, Completed), live status updates, and data synchronization with the backend were finalized. These specifications were selected keeping usability, scalability, and maintainability in mind, ensuring the system remained simple yet powerful.

2.2 Design Constraints

During the design phase of the Todo Application, several constraints were considered to ensure feasibility, reliability, and efficient performance. From a **technical standpoint**, one major constraint was achieving synchronization between the client and server without performance lag. Since the application relied on RESTful APIs for communication, network latency and server response time were carefully monitored to maintain an optimal user experience. Another important factor was ensuring **data consistency** within MongoDB to avoid duplication or task corruption during CRUD operations.

From a **design and usability perspective**, the interface was kept minimalistic to ensure ease of use even for non-technical users. It needed to function effectively on both desktop and mobile browsers. **Environmental constraints** were minimal, as the project was developed entirely using digital tools. Collectively, these design constraints ensured the system achieved the desired balance between performance, simplicity, and user satisfaction.

2.3 Analysis and Feature Finalization Subject to Constraints

After analyzing the identified design constraints, the team refined the Todo Application's features to maintain a balance between functionality and efficiency. The primary focus was to include essential CRUD operations while keeping the system lightweight and easy to maintain. Features such as **task creation**, **status toggling**, **real-time updates**, and **task deletion** were finalized as core functionalities.

Additionally, the system's architecture was modularized, allowing future upgrades without affecting existing modules. React's reusable components and hooks simplified UI updates, while Express routes ensured maintainable backend structure. This selective inclusion of features resulted in a stable and responsive Todo Application that effectively demonstrated CRUD principles while operating within the defined technical and economic constraints.

2.4 Design Flow

Two design approaches were initially considered for developing the Todo Application. The first was a **monolithic structure**, where both frontend and backend would run under a single environment. Although this design simplified initial deployment, it lacked flexibility and scalability for future updates. The second approach was a **modular MERN architecture**, separating the frontend, backend, and database into distinct layers, allowing each part to evolve independently.

This workflow ensures a **bidirectional data flow** between client and server, where every CRUD action performed by the user (Add, Update, Delete, or Read) is instantly reflected on the interface. The frontend dynamically re-renders components using React state management hooks, ensuring seamless user experience without page reloads. This design flow not only improves system performance but also enhances modularity, allowing developers to test, debug, and extend individual layers independently.

2.5 Design Selection

After comparing both design alternatives, the **MERN architecture** (MongoDB, Express.js, React.js, Node.js) was selected as the most suitable design for the Todo Application. The choice was influenced by several factors, including scalability, ease of maintenance, and the use of a single programming language—**JavaScript**—across all layers. This unified approach simplified development and improved communication between the frontend and backend.

The MERN architecture provides a **non-blocking, event-driven backend** with Node.js and a flexible NoSQL database through MongoDB, allowing dynamic storage of todo items without predefined schema constraints. Express.js efficiently manages routing and middleware, ensuring that CRUD APIs function smoothly. On the frontend, React.js supports reusable components and virtual DOM rendering, resulting in a highly responsive user interface.

2.6 Implementation Plan / Methodology

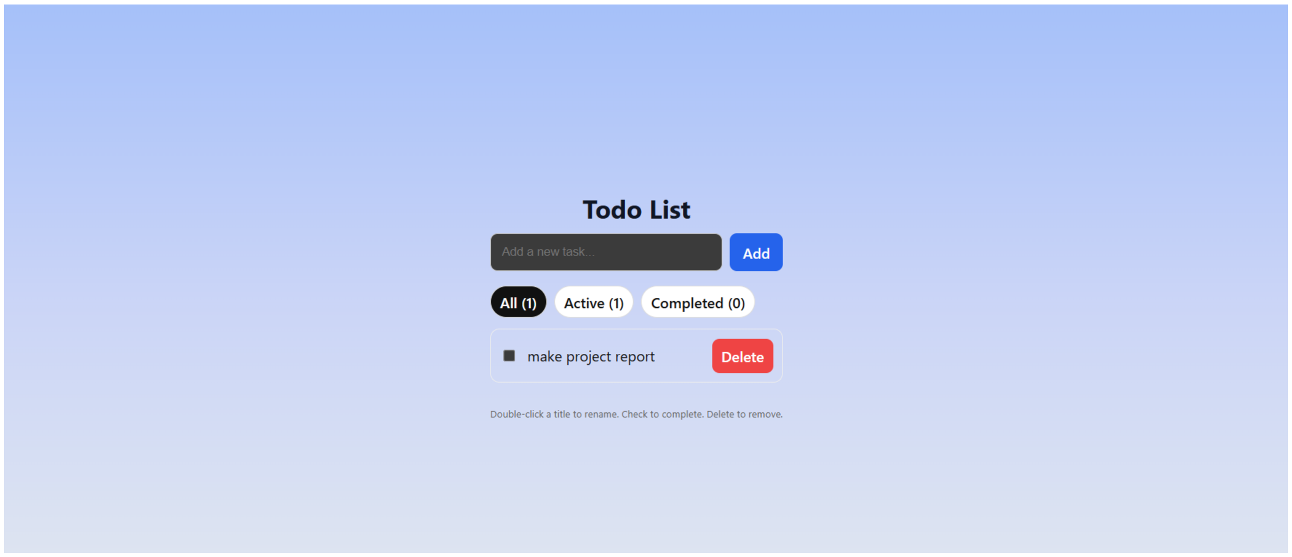
The implementation of the Todo Application followed a structured, iterative methodology to ensure smooth progress and timely completion. The process began with the setup of the **GitHub repository** for version control and collaboration. The backend was developed first using **Node.js** and **Express.js**, defining all CRUD endpoints for creating, reading, updating, and deleting tasks. These APIs were tested individually using **Postman** to ensure accuracy and stability before integration with the frontend.

Next, the **MongoDB database** was connected using Mongoose. A simple schema was designed containing fields such as title, completed, and timestamps, ensuring efficient data management. Once the backend was stable, the frontend was developed using **React.js** integrated with **Vite** for optimized build speed and hot module replacement.

CHAPTER 3.
RESULTS ANALYSIS AND VALIDATION

3.1 Implementation of Solution

The implementation of the **Todo Application with Basic CRUD Operations** was carried out using modern full-stack web technologies to ensure functionality, scalability, and efficiency. The system architecture followed the MERN model, integrating **React.js** for the frontend, **Node.js** and **Express.js** for the backend, and **MongoDB** for database management. The project began with designing a clear **data flow** between components, ensuring smooth communication between the client and server through RESTful APIs.



Double-click a title to rename. Check to complete. Delete to remove.

Backend APIs were developed and tested using **Postman**, validating CRUD endpoints for creating, retrieving, updating, and deleting tasks. Each API call was checked for proper response codes, data accuracy, and error handling. The **frontend interface** was implemented using React components and hooks such as useState and useEffect, ensuring real-time updates without requiring page reloads. The **Vite** build tool was used for faster rendering and hot module reloading, improving developer productivity and user experience.

For collaboration, **GitHub** was used for version control and progress tracking. Testing and debugging were performed iteratively at each stage to identify issues early.

CHAPTER 4.
Conclusion and Future Work

4.1 Conclusion

The **Todo Application with Basic CRUD Operations** successfully accomplished its objectives of developing a lightweight, responsive, and efficient task management system using the **MERN stack**—MongoDB, Express.js, React.js, and Node.js. The project demonstrates how full-stack web technologies can be integrated to perform essential CRUD operations while ensuring real-time synchronization between the frontend and backend.

Through systematic development and testing, the application achieved seamless functionality for creating, viewing, updating, and deleting tasks. The system's modular structure, combined with RESTful APIs, provided high scalability and ease of maintenance. React's component-based architecture ensured fast rendering and smooth user interactions, while MongoDB guaranteed reliable data storage and retrieval.

4.2 Future Work

While the current version of the Todo Application meets its primary goals, there remains significant potential for future expansion and enhancement. One major improvement could be the **integration of user authentication** using JWT (JSON Web Token), allowing multiple users to maintain separate task lists securely. This would make the application suitable for broader, multi-user environments.

Additionally, **task categorization and prioritization** can be added to help users organize their work more effectively. Features like **due dates, reminders, and notifications** would increase the app's utility and align it with modern productivity tools. The inclusion of **search and sorting filters** would further enhance the usability of the interface.

From a deployment perspective, hosting the application on cloud platforms such as **Render, Vercel, or Firebase** would enable online accessibility and real-world scalability. A **Progressive Web App (PWA)** version could also be developed to make the application installable and functional offline.

For performance enhancement, **state management libraries** like Redux can be incorporated for large-scale data handling. Integration with external APIs, such as Google Calendar or task-tracking services, could transform the Todo App into a comprehensive productivity suite. These future developments would make the application more interactive, intelligent, and suitable for both personal and professional task management.

REFERENCES

1. React Documentation. (2024). *React: A JavaScript library for building user interfaces.* Meta Open Source. Retrieved from https://react.dev/
2. Node.js Documentation. (2024). *Node.js: JavaScript runtime built on Chrome's V8 engine.* Node.js Foundation. Retrieved from https://nodejs.org/en/docs
3. Express.js Official Guide. (2024). *Fast, unopinionated, minimalist web framework for Node.js.* Retrieved from https://expressjs.com/
4. MongoDB Documentation. (2024). *Introduction to MongoDB – NoSQL Database for Modern Applications.* MongoDB Inc. Retrieved from https://www.mongodb.com/docs/
5. Mozilla Developer Network (MDN). (2024). *Using Fetch API for HTTP Requests in JavaScript.* Retrieved from https://developer.mozilla.org/
6. Vite Documentation. (2024). *Next Generation Frontend Tooling – Fast Build Tool for Modern Web Apps.* Retrieved from https://vitejs.dev/
7. W3C Web Application Standards. (2023). *Web Application Security and Best Practices.* World Wide Web Consortium. Retrieved from https://www.w3.org/
8. GeeksforGeeks. (2024). *CRUD Operations in MERN Stack using React, Node.js, Express.js, and MongoDB.* Retrieved from https://www.geeksforgeeks.org/

APPENDIX: USER MANUAL

Steps to Run the Project Locally

1. **Install Dependencies:**
   Ensure Node.js, npm, and MongoDB are installed on your system.

2. **Run Backend:**
   Open terminal →

   - cd todo-fullstack/server
   - npm install
   - npm run dev

Backend runs on http://localhost:4000.

3. **Run Frontend:**
   Open new terminal →

   - cd todo-fullstack/client
   - npm install
   - npm run dev

Frontend runs on http://localhost:5173.

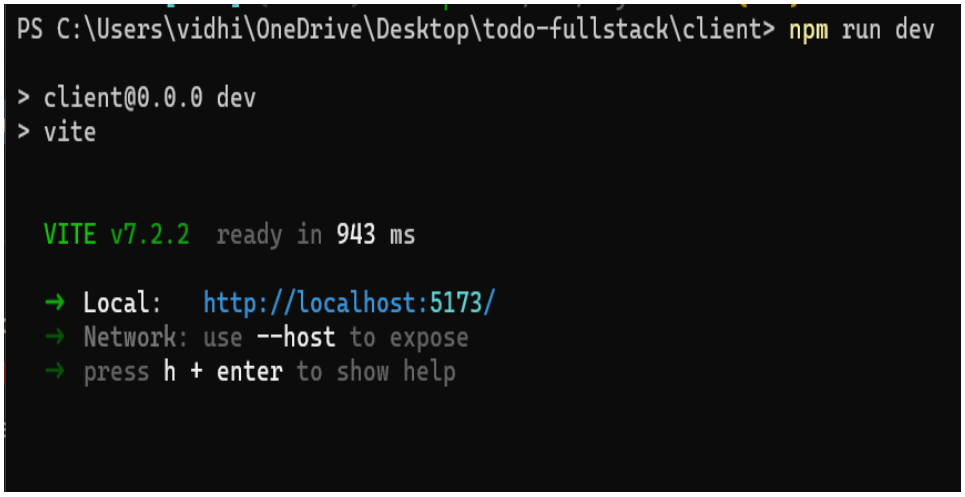
4. **Configure .env Files:**
   - In **server**: MONGO_URI=mongodb://localhost:27017/todoApp
   - In **client**: VITE_API_URL=http://localhost:4000

5. **Access Application:**
   Open http://localhost:5173 in your browser to use the Todo app.

USER MANUAL

(Complete step by step instructions along with pictures necessary to run the project)

For Frontend:



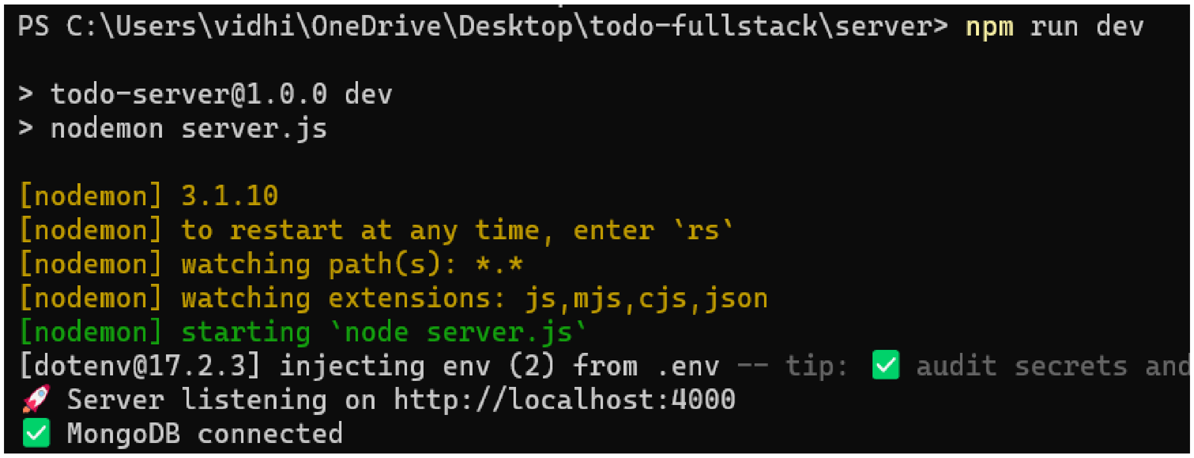
```
PS C:\Users\vidhi\OneDrive\Desktop\todo-fullstack\client> npm run dev

> client@0.0.0 dev
> vite


  VITE v7.2.2  ready in 943 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

For Backend:



```
PS C:\Users\vidhi\OneDrive\Desktop\todo-fullstack\server> npm run dev

> todo-server@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (2) from .env -- tip: ✅ audit secrets and
🚀 Server listening on http://localhost:4000
✅ MongoDB connected
```