TO DO LIST MANAGEMENT SYSTEM

MICRO PROJECT REPORT

Submitted by

MOHAMMED YUNUS A

(23ITR099)

KARTHICK SARAVANAN R M

(23ITR078)

MATHAN M

(23ITR095)

in partial fulfilment of the requirements

for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638 060

MAY 2025

DEPARTMENT OF INFORMATION TECHNOLOGY

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638060

MAY 2025

BONAFIED CERTIFICATE

This is to certify that the Project report entitled **TO DO LIST MANAGEMENT SYSTEM** is the Bonafiderecord of project work done by **MOHAMMED YUNUS A (23ITR099), KARTHICK SRAVANAN R M(23ITR078)** and **MATHAN M (23ITR095)** for **22ITT42 WEB TECHNOLOGY** during the year 2024–2025.

COURSE IN CHARGE **HEAD OF THE DEPARTMENT**

(Signature with seal)

Date:

Submitted for the final viva voce examination held on _____ .

DEPARTMENT OF INFORMATION TECHNOLOGY

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI ERODE – 638060

MAY 2025

DECLARATION

We affirm that the Project Report titled **TO DO LIST MANAGEMENT SYSTEM** being submitted in partial fulfilment of the requirements for the award of Bachelor of Technology is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Date:

MOHAMMED YUNUS A
(23ITR099)

KARTHICK SARAVANAN R M
(23ITR078)

MATHAN M
(23ITR095)

I certify that the declaration made by the above candidates is true to the best of my knowledge.

Date:

Name and Signature of the course in charge

ABSTRACT

The To-Do List Management System is a robust and user-friendly web application designed to help individuals and organizations efficiently manage their tasks, deadlines, and productivity workflows. In today's fast-paced digital environment, staying organized and keeping track of daily responsibilities is a common challenge for students, professionals, and teams alike. This project addresses those needs by offering a centralized platform where users can create, view, and manage their to-do items with clarity, simplicity, and accountability.Users can register and securely log in to access their personalized dashboards, where they can create new tasks, set due dates, categorize them, and mark them as complete upon completion. The system includes built-in functionalities such as email reminders for overdue tasks, highlighting pending or urgent tasks, and allowing users to filter their tasks by category, priority level, or completion status. This enhances productivity and ensures that users remain on top of their responsibilities without missing critical deadlines.Technologically, the application utilizes HTML, CSS, and JavaScript for the frontend, ensuring responsive and dynamic interfaces. The backend is built using Node.js with Express.js to handle user authentication, task management, and server-side operations. MongoDB is used for efficient and scalable data storage, enabling quick access and updates to user and task information. Security measures such as password encryption, session management, and role-based access controls are implemented to protect user data and prevent unauthorized access.+In conclusion, the To-Do List Management System is not just a productivity application—it is a comprehensive framework that encourages goal-oriented behavior and personal accountability. By digitalizing the task management process, it promotes effective time utilization and organizational efficiency. Whether used by individuals to manage personal goals or by small teams to streamline project planning, the system stands as a practical and scalable solution in the field of productivity tools.

ACKNOWLEDGEMENT

First and foremost, we acknowledge the abundant grace and presence of Almighty throughout different phases of the project and its successful completion.

We wish to express our gratefulness to our beloved Correspondent **Thiru.A.K.ILANGO B.Com., M.B.A., LLB.,** and all the trust members of Kongu Vellalar Institute of Technology Trust for providing all the necessary facilities to complete the project successfully.

We express our deep sense of gratitude to our beloved Principal **Dr.V.BALUSAMY B.E.(Hons)., M.Tech., Ph.D.,** for providing us an opportunity to complete the project.

We express our gratitude to **Dr. S. ANANDAMURUGAN M.E., Ph.D.,** Head of the Department, Department of Information Technology for his valuable suggestions.

We are highly indebted to **Mrs. R. AARTHI B.E., M.E.,** Department of Information Technology for her valuable supervision and advice for the fruitful completion of the project.

We are thankful to the faculty members of the Department of Information Technology for their valuable guidance and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

CSS	Cascading Style Sheets
HTML	HyperText Markup Language
ID	Identification
JS	JavaScript
JSX	JavaScript XML
NFC	Near-Field Communication
QR	Quick Response
URL	Uniform Resource Locator

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Effective task management is essential in today's fast-paced world, where individuals juggle multiple responsibilities across personal, academic, and professional domains. Despite its importance, many people still rely on traditional methods like notebooks, sticky notes, or basic mobile apps to manage their to-do lists. These outdated techniques are often disorganized, prone to errors, and lack the functionality needed to handle complex or time-sensitive tasks. As responsibilities increase, so does the need for a more reliable and efficient system. The To-Do List Management System is developed to address these needs—offering users a streamlined, accessible, and scalable solution to manage their tasks efficiently and stay on top of their commitments, whether personal or professional.

1.2 OBJECTIVE

The main objective of the To-Do List Management System is to offer a centralized and efficient platform for organizing, prioritizing, and tracking daily tasks and responsibilities. This system is designed to simplify personal and professional task management by allowing users to create, update, and monitor to-do items with ease and precision. It aims to help individuals and teams enhance productivity by categorizing tasks, setting due dates, and receiving timely reminders for upcoming or overdue activities. The application supports secure user registration and login, ensuring that each user has access to a personalized task dashboard. With role-based access and administrative controls, the system can also be adapted for collaborative task tracking within small groups or organizations. Features such as real-time task status updates, filtering options, and progress indicators empower users to stay on top of their schedules. Built with modern web technologies and a responsive interface, the platform provides a consistent user experience across devices. This makes it a reliable and user-friendly tool for managing time, setting goals, and boosting efficiency in a structured manner.

CHAPTER 2

SYSTEM SPECIFICATION

2.1 HARDWARE SPECIFICATION

Processor	:	12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz
Processor Speed	:	1.30 GHz
RAM	:	8.00 GB
Hard Disk	:	512GB
Keyboard	:	Standard 104 enhanced
Mouse	:	Local PS/2

2.2 SOFTWARE REQUIREMENTS

Platform	:	Visual Studio Code
Language	:	HTML, CSS, Bootstrap, Javascript, Node js
Database	:	Mongo DB
Library	:	Express.js, Mongoose,Body- Parser,Cors,Connect-Mongo,Dotenv,Bcrypt.js
Framework	:	Angular

2.3 SOFTWARE DESCRIPTION

2.3.1 Visual Studio Code

Visual Studio Code (VS Code) is a lightweight and powerful Integrated Development Environment (IDE) used for building the *To-do list management system*. It is an ideal choice for web development due to its support for multiple programming languages and rich features. VS Code provides intelligent code completion through IntelliSense, which helps developers write accurate code faster by suggesting variables, methods, and functions. Its integrated debugging tools enable developers to set breakpoints, inspect variables, and step through code, facilitating quick issue identification and resolution. Additionally, VS Code integrates seamlessly with Git, allowing for efficient version control management directly within the IDE. It also supports a variety of extensions such as ESLint, Prettier, and live-server, which enhance productivity by automating tasks like code formatting and running local servers. The built-in terminal further streamlines the workflow by allowing developers to execute commands, install dependencies, and test the application without leaving the IDE. With its clean, user-friendly interface and cross-platform compatibility (Windows, macOS, Linux), Visual Studio Code is a crucial tool that supports efficient development, debugging, and testing of the *To-do list management system*.

2.3.2 MongoDB Compus

MongoDB Compass is a graphical user interface (GUI) for interacting with MongoDB, the NoSQL database used in the *To-do list management system* project. MongoDB Compass allows developers and database administrators to visualize, manage, and optimize the database without needing to write complex queries in the command-line interface. It provides an intuitive and user-friendly interface to explore the database's collections, documents, and fields. Through Compass, users can view real-time data, filter, and query collections with ease, and gain insights into database performance through built-in analytics tools. MongoDB Compass also offers schema exploration features, enabling developers to understand the structure of stored data and make necessary adjustments to the schema when required. Additionally, it simplifies data

migration tasks and offers features like indexing, aggregation, and visual explain plans to improve query performance. This tool plays a crucial role in managing and optimizing the database for the *To-do list management system* application, ensuring efficient data storage, retrieval, and maintenance.

2.3.3 Express.js

Express.js is a minimal and flexible web application framework for Node.js, designed to simplify the process of building web applications and APIs. In the *To-do list management system* project, Express.js is used to handle HTTP requests, route incoming data, and manage server-side logic. Express makes it easy to set up middleware to process requests, handle errors, and ensure smooth communication between the client-side and server-side components. It is lightweight and allows for building RESTful APIs with ease, which is crucial for managing donation and expense records.

2.3.4 Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward way to define the structure of your data, create schemas, and interact with the MongoDB database. In the *To-do list management system* project, Mongoose is used to define models for various entities like donors, donations, and expenses. It also helps in querying the database efficiently and handling CRUD operations (Create, Read, Update, Delete) seamlessly. By using Mongoose, developers can ensure that the data is well-structured and validated before being stored in the database.

2.3.5 Body-Parser

Body-Parser is middleware for Express.js that simplifies the extraction of data from incoming HTTP requests. It allows for easy parsing of JSON and URL-encoded form data in request bodies, which is essential for processing form submissions and API requests. In the *To-do list management system*, Body-Parser is used to handle POST requests containing donor information, donation data, and other form submissions. This

library ensures that the server can properly interpret the incoming data and use it for various operations, such as storing donations or updating expense records.

2.3.6 Cors

Cors (Cross-Origin Resource Sharing) is a Node.js package used to enable cross-origin requests. When building a web application that interacts with APIs hosted on a different domain, CORS is required to allow the frontend to access resources on the backend securely. In the *To-do list management system*, **Cors** ensures that the frontend (running on one domain) can safely send HTTP requests to the backend server (which may be running on a different domain or port). This prevents security issues while enabling communication between different parts of the application.

2.3.7 Connect-Mongo

Connect-Mongo is a MongoDB session store for **Express.js** and **Connect**, which allows session data to be stored in a MongoDB database instead of in memory or in cookies. This is particularly useful for applications with many users, as it allows session data to be persistent across different requests. In the *To-do list management system*, **Connect-Mongo** is used to manage user sessions, such as keeping users logged in after they submit donations or log in. It ensures that session data is securely stored in MongoDB, allowing for easy session management and better scalability.

2.3.8 Dotenv

Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env. This allows you to store sensitive information such as database credentials, API keys, and secret keys in a secure and easily accessible way. In the *To-do list management system*, **Dotenv** is used to manage environment-specific settings, such as the MongoDB connection string, JWT secrets, and other sensitive configuration values. By keeping these values in an .env file, the project ensures better security practices and easier management of different deployment environments (development, production, etc.).

2.3.9 Bcrypt.js

Bcrypt.js is a library for hashing passwords, providing security for user authentication. In the *To-do list management system*, **Bcrypt.js** is used to securely hash user passwords before storing them in the database. This prevents plain-text passwords from being exposed in case of a data breach. Bcrypt applies a salt to the password and hashes it multiple times, ensuring that even identical passwords have unique hashes. This adds an extra layer of security and helps protect user data from unauthorized access.

2.3.10 Angular

Angular is a robust, open-source front-end framework developed by Google, used to build dynamic, single-page web applications. In the *To-do list management system* project, Angular is employed to develop the user interface, ensuring a responsive and interactive experience. With its component-based architecture, Angular allows the app to be modular and maintainable by breaking the UI into reusable components. It also features two-way data binding, which ensures that changes on the frontend, such as entering donation data, are automatically reflected in the model, creating a seamless experience. Angular's built-in routing allows for smooth navigation between pages, such as donation history, registration, and login. The framework's dependency injection (DI) system simplifies managing services, while RxJS and Observables handle asynchronous operations like making API calls to the backend, ensuring smooth communication with the Node.js and MongoDB backend. Angular CLI streamlines the development process by providing commands for generating components, services, and modules, making it easier to build and deploy the application. Through these features, Angular helps create a responsive, scalable, and maintainable front-end application that integrates smoothly with the backend to track donations, manage expenses, and provide an optimal user experience.

CHAPTER 3

SYSTEM DESIGN

3.1 USE CASE DESIGN

The **Use Case Design** defines the interactions between the users (actors) and the system, illustrating how different users will engage with the *To-do list management system* to accomplish specific tasks. These use cases provide a high-level understanding of the system's functionality and how various features support user actions.

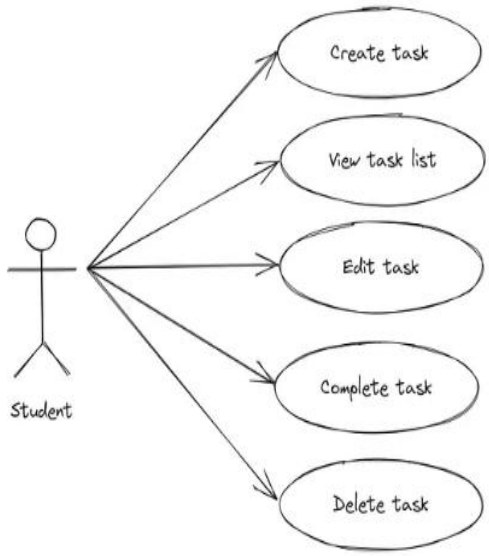


Fig 3.1 Use Case Diagram

3.2 CLASS DIAGRAM

In the Class Diagram for the *To-Do List Management System* project, several core classes represent the essential entities and functionalities that drive the system. The User class contains attributes such as **userID**, name, email, and password, along with methods for user registration, login, profile updates, and password management. The Task class is central to the system, with attributes like **taskID**, title, description, dueDate, status, and priority. It includes methods to create, update, delete, and mark tasks as completed or pending. The Reminder class handles scheduled notifications and alerts, with attributes like reminderID, taskID, scheduledTime, and isSent, and methods to create and dispatch reminders. Additionally, a Session class manages active user sessions, with attributes such as sessionID, userID, loginTime, and logoutTime, along with methods to start, validate, and terminate sessions.

These classes are designed to interact cohesively—for example, a single User can have multiple Task entries, each possibly linked to a Reminder. The use of session control ensures secure and personalized access to the system. This class structure supports scalability, modularity, and secure task management across both individual and collaborative environments.

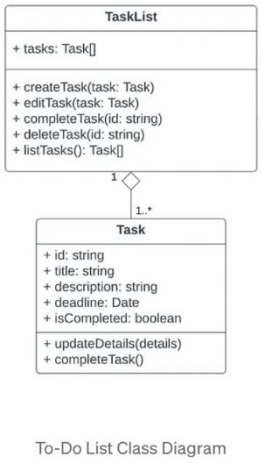


Fig 3.2 Class Diagram

3.3 SEQUENCE DIAGRAM

The Sequence Diagram of the *To-Do List Management System* illustrates the step-by-step flow of interactions between system components and users during key processes such as user login, task creation, and task completion. A common use case is the user login and task creation sequence. In this sequence, the User begins by entering login credentials through the Frontend Interface (developed using **HTML**, **CSS**, and **JavaScript**). The frontend sends an authentication request to the Backend Server (built with Node.js and Express.js), which communicates with the Database (MongoDB via Mongoose) to validate the user's credentials. Upon successful verification, a session token is generated and returned to the frontend, granting the user access to their personalized task dashboard. Once logged in, the user can create a new task by entering task details. The frontend sends the task creation request to the backend, which stores the task in the database. A confirmation message is then sent back, updating the UI with the newly added task in real-time.

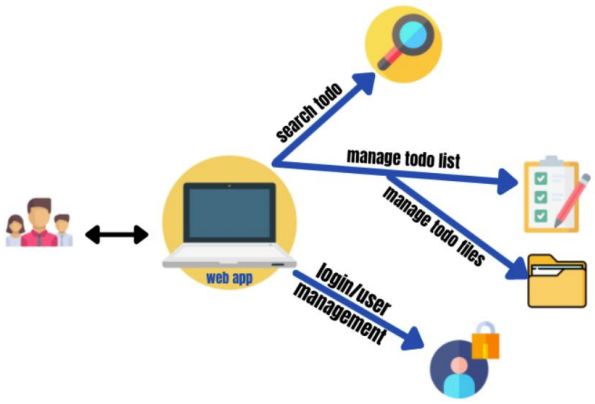


Fig 3.3 Sequence Diagram

3.4. ACTIVITY DIAGRAM

The Activity Diagram for the *To-Do List Management System* illustrates the dynamic flow of control during essential operations such as user login, task creation, updating tasks, and viewing task status. The process begins from the initial state where the user launches the application. The first activity prompts the user to either register or log in. If the user is new, the system collects registration details, performs input validation, and stores the information in the database. If the user already has an account, they input their login credentials, which are **authenticated** through the backend server. Upon successful login, the user is redirected to their personalized dashboard. From there, they can choose to add a new task, edit existing tasks, mark tasks as complete, or delete them. Each action triggers a backend update and refreshes the dashboard accordingly. The activity diagram helps visualize the user journey and how different actions transition the system from one state to another in an interactive and **logical flow**.

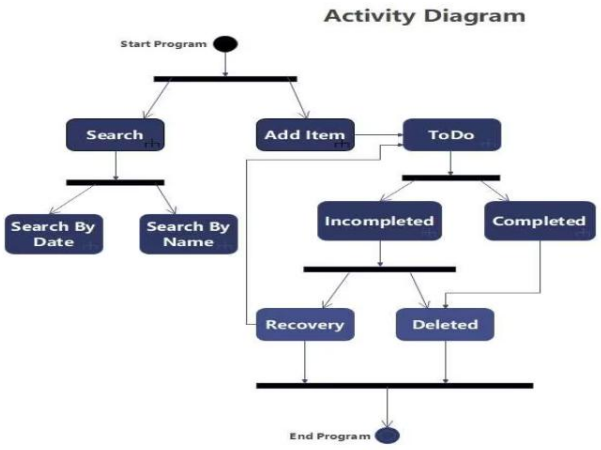


Fig3.4 Activity Diagram

3.5 DATABASE DESIGN

3.5.1 OVERVIEW

The database design of the *To-Do List Management* System is essential for efficiently managing and organizing user and task-related data. The project utilizes **MongoDB,** a NoSQL database known for its flexibility and scalability, where data is stored in JSON-like documents. The design ensures rapid access, simplified data structure, and easy scalability for growing user bases. The primary entities in the database include Users, Tasks, and Reminders, each represented as individual collections with clearly defined attributes. The Users collection holds data such as user ID, name, email, password, and role. The Tasks collection includes **task ID,** title, description, due date, status, and priority. Reminders are stored with attributes like **reminder ID**, associated task ID, time, and status. Relationships between entities are maintained through references using unique identifiers like user IDs and task IDs. This schema enables a well-structured, fast, and reliable data flow that supports the application's core features and enhances user experience.

3.5.2 SIGN UP SCHEMA

The Sign Up Schema defines the structure of user data collected and stored when a new user registers on the *To-Do List Management* System. This schema is implemented in the backend using Mongoose, allowing for a well-defined and validated data model within MongoDB. The schema ensures that all essential user information is accurately captured, securely stored, and readily available for authentication and personalized access to the task dashboard.

The key fields in the Sign Up Schema include:

Name: A string that captures the user's full name. It is a required field used for personalizing the user interface.

Email: A unique string that serves as the main identifier for login. It is validated to ensure correct email formatting.

Password: A string storing the user's password in a hashed format using bcrypt.js, ensuring strong encryption and security.
Role: A string representing the user role, such as standard user or admin, for implementing role-based functionalities.
CreatedAt: A date field that logs the registration time, useful for tracking user onboarding trends and activity history.

LOGIN SCHEMA

The Login Schema in the *To-Do List Management System* is built to authenticate existing users and provide them with secure access to their personalized task environment. While login requires fewer inputs than registration, it is a vital component for verifying user identity and maintaining a secure session. The login functionality relies primarily on the email and password fields entered by the user, which are validated against stored records in the database using Mongoose and authentication middleware.

The key fields managed in the Login Schema include:

Email: A required string that must match an existing user's email in the database. It is validated to ensure correct formatting and serves as the main identifier during the login process.

Password: A required string submitted by the user. It is securely compared to the hashed version stored in the database using bcrypt.js to confirm the user's identity and grant access to the task dashboard.

3.6 MODULES DESCRIPTION

3.6.1 AUTHENTICATION MODULE

The Authentication Module is a fundamental part of the *To-Do List Management System*. It handles user identity verification and ensures secure access control throughout the application. This module guarantees that only registered users can interact with their task lists, update information, or use administrative features if authorized. Built using **Node.js**, **Express.js**, **Mongoose**, and **bcrypt.js**, the authentication system offers robust functionalities including user sign-up, login, and session/token management.

During sign-up, the module validates the input data—such as name, email, and password—ensures email uniqueness, and securely hashes the password using bcrypt before storing it in the MongoDB database. At login, the system fetches the user's record by email, compares the provided password against the stored hash, and if successful, generates a JWT (JSON Web Token) or session token. This token is returned to the frontend (e.g., React or HTML/CSS/JS), where it is stored for authenticating future user requests to protected routes and actions.

3.6.2 USER MANAGEMENT MODULE

The **User Management Module** is a fundamental component of the *To-Do List Management System*, responsible for handling all user-related operations including registration, authentication, and role-based access. These credentials are validated and stored securely in the MongoDB database using **Mongoose**. The backend, built with Node.js and Express.js, employs bcrypt.js to hash passwords, enhancing user security.Once registered, users can log in using their email and password. After successful authentication, a session token or JWT is generated and used to authorize further actions. The module supports profile updates, such as changing names or passwords, and ensures that only the rightful user can make these changes. Admin users, if implemented, can view and manage all user accounts, promoting centralized oversight.

This module also integrates role-based access control (RBAC), restricting access to certain features depending on whether the user is a regular task user or an administrator.To maintain accuracy and accountability, the system includes validation mechanisms that check for unique email addresses, proper formatting, and secure password practices. The User Management Module plays a critical role in ensuring smooth, secure, and personalized interaction with the application. It contributes to the overall usability, scalability, and security of the system by keeping user data organized and protected.

3.6.3 TASK MANAGEMENT MODULE

The The **Task Management Module** serves as the core functionality of the *To-Do List Management System*, enabling users to create, organize, and track their daily tasks effectively. This module is designed to streamline personal productivity by allowing users to manage to-do items based on priorities, due dates, and completion status.The task data is stored in the MongoDB database using Mongoose models, allowing each task to be linked with its respective user. Users can perform full **CRUD** operations—Create, Read, Update, and Delete—on their tasks. They can also mark tasks as complete or overdue, and use filters such as date, status, or keyword search to manage large task lists more efficiently. Real-time updates and dynamic UI changes built using Angular provide a responsive user experience.

Advanced features like color-coded priority levels, overdue notifications, and deadline reminders enhance user engagement and task tracking. Input validation ensures that empty or incorrect data is not accepted, maintaining data integrity. For better analytics and usability, tasks can also be sorted and grouped, and the module can generate summaries of pending and completed tasks.Overall, the Task Management Module empowers users with an intuitive interface and robust backend logic, ensuring efficient personal task organization and time management.

CHAPTER 4

RESULTS

The *To-Do List Management System* was successfully developed and tested, offering a reliable and intuitive platform for users to create, manage, and track their tasks efficiently. The application facilitated seamless user registration, login, and task handling, ensuring that users could focus on productivity without technical barriers.

Built with a clean and responsive Angular frontend and a robust Express.js backend, the system ensured real-time interaction with the MongoDB database. This enabled users to add, update, complete, or delete tasks with immediate feedback and persistent data storage. The platform's design prioritized user experience, offering features like task categorization, deadline setting, and priority tagging.

Security and role management were handled via JWT-based authentication, allowing only authenticated users to access and manipulate their personal task data. The authentication system enforced secure password handling and prevented unauthorized access, ensuring data privacy and integrity.

The backend architecture followed a modular approach, improving maintainability and scalability. Effective data validation and comprehensive error handling mechanisms were implemented, resulting in a dependable and user-friendly application experience.

Overall, the To-Do List project successfully met its objectives by helping users stay organized and focused. It demonstrated efficient task lifecycle management, ensured data consistency, and laid the groundwork for future improvements such as notification systems, collaborative task sharing, calendar integrations, and multi-device synchronization.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Conclusion

The *To-Do List Management* System effectively addresses the everyday challenge of task organization and personal productivity. By providing a user-friendly interface, secure authentication, and real-time task management capabilities, the platform enables users to efficiently create, update, categorize, and track their tasks. The system offers a seamless experience through its integration of modern web technologies such as **Angular, Node.js, Express.js, and MongoDB**, ensuring smooth data handling, security, and scalability. The application meets its primary goals of enhancing productivity, minimizing task management complexity, and supporting users in achieving their personal and professional objectives. With secure access control and efficient task lifecycle management, the system helps users maintain focus and control over their daily activities. Its modular and scalable architecture also makes it suitable for both individual and collaborative use cases. Overall, this project serves as a practical and reliable solution for task tracking, empowering users to stay organized, prioritize effectively, and improve time management**.**

Future Scope

The *To-Do List Management System* presents several avenues for future enhancement and development, aiming to create a more interactive and intelligent task management experience. One key area for improvement is the **integration of reminders and notifications**—both via in-app alerts and external services such as email or SMS—to help users stay on track with due dates and high-priority items. A **mobile application** version would significantly increase accessibility, allowing users to manage their tasks anytime, anywhere. Additionally, incorporating **calendar synchronization** with services like Google Calendar or Outlook could streamline scheduling and improve visibility into upcoming tasks and deadlines.

Future versions of the platform could also support **collaborative task sharing**, allowing teams or groups to manage shared to-do lists with role-based access and activity logs. Enhancing the dashboard with **data visualization tools** like charts and graphs would provide users with insights into their productivity trends, task completion rates, and time management patterns. **Multi-language support** could be introduced to expand usability across diverse user bases globally.

For a more intelligent experience, **AI-powered suggestions** could recommend task prioritization, detect task duplication, or estimate task completion times based on historical data. Integrating **voice input support** and **natural language processing (NLP)** could further simplify task creation and management. As the platform scales, the inclusion of **dark mode**, **offline access**, and **customizable themes** could enhance user satisfaction and personalization. These enhancements would position the To-Do List System as a comprehensive, scalable, and adaptive productivity tool for both personal and professional users.

APPENDIX 1

CODING

Index.html :

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>To-Do List</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">

</head>

<body>

<div class="container">

<div class="header">

<h1>To-Do List</h1>

<div class="user-info"></div>

<button type="button" class="theme-toggle" onclick="toggleTheme()" aria-label="Toggle dark mode">

<i class="fas fa-moon"></i>

</button>

</div>
```

```
<!-- ----------------------------------------STATISTICS---------------------------------------->
<div class="statistics">
<div class="stat-item"
<span class="stat-label">Total Tasks</span>
</div>
<div class="stat-item">
<span class="stat-value" id="activeTasks">0</span>
<span class="stat-label">Active</span>
</div>
<div class="stat-item">
<span class="stat-label">Completed</span>
</div>
</div>
<!-- Search and Sort -->
<div class="search-sort">
<div class="search-box">
<input type="text" id="searchInput" placeholder="Search tasks...">
<i class="fas fa-search"></i>
</div>
<select id="sortBy" onchange="sortTasks()">
<option value="priority">Sort by Priority</option>
<option value="category">Sort by Category</option>
</div>
```

```
<!-- ----------------------------------------TASK INPUT----------------------------------------
----------->

<div class="task-input">

<input type="text" id="taskInput" placeholder="Add your task">

<input type="date" id="dueDate">

<select id="priority">

<option value="low">Low Priority</option>

<option value="medium">Medium Priority</option>

<option value="high">High Priority</option>

</select>

<select id="category">

<option value="work">Work</option>

<option value="personal">Personal</option>

<option value="shopping">Shopping</option>

<option value="other">Other</option>

</select>

<button type="button" onclick="addTask()">Add</button>

</div>
```

```
<!-- ----------------------------------------FILTERS---------------------------------------- ---------->

<div class="filters">

<button type="button" onclick="filterTasks('all')">All</button>

<button type="button" onclick="filterTasks('active')">Active</button>

<button type="button" onclick="filterTasks('completed')">Completed</button>

<button type="button" onclick="filterTasks('high')">High Priority</button>

</div>


<!-- ----------------------------------------EDIT TASK MODEL---------------------------------- ------------------------>

<div id="editModal" class="modal">

<div class="modal-content">

<span class="close">&times;</span>

<h2>Edit Task</h2>

<form onsubmit="saveEdit(); return false;">

<div class="form-group">

<label for="editTaskInput">Task Title</label>

<input type="text" id="editTaskInput" required>

</div>

<div class="form-group">

<label for="editDescription">Description</label>

<textarea id="editDescription" rows="3"></textarea>

</div>
```

```
<div class="form-group">
<label for="editDueDate">Due Date</label>
<input type="date" id="editDueDate">
</div>
<div class="form-group">
<label for="editPriority">Priority</label>
<select id="editPriority">
<option value="low">Low Priority</option>
<option value="medium">Medium Priority</option>
<option value="high">High Priority</option>
</select>
</div>
<div class="form-group">
<label for="editCategory">Category</label>
<select id="editCategory">
<option value="work">Work</option>
<option value="personal">Personal</option>
<option value="shopping">Shopping</option>
<option value="other">Other</option>
</select>
</div>
<button type="submit">Save Changes</button>
</form>
```

```
</div>

</div>


<script src="script.js"></script>

</body>

</html>
```

Server.js:

```
const express = require('express');

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const path = require('path');

const cors = require('cors');

require('dotenv').config();


const app = express();

const PORT = process.env.PORT || 3002;


// Middleware

app.use(cors());

app.use(express.json());

app.use(express.static('public'));
```

```
// ------------------ MONGO_DB CONNECTION ------------------

mongoose.connect(process.env.MONGODB_URI                                    ||
'mongodb://localhost:27017/todoapp', {

useNewUrlParser: true,

useUnifiedTopology: true

})

.then(() => console.log('Connected to MongoDB'))

.catch(err => console.error('MongoDB connection error:', err));


// User Schema

const userSchema = new mongoose.Schema({

name: { type: String, required: true },

email: { type: String, required: true, unique: true },

password: { type: String, required: true },

createdAt: { type: Date, default: Date.now }

});


const User = mongoose.model('User', userSchema);


// Task Schema

const taskSchema = new mongoose.Schema({

title: { type: String, required: true },

description: String,

dueDate: Date,
```

```
priority: { type: String, enum: ['low', 'medium', 'high'], default: 'medium' },
completed: { type: Boolean, default: false },
userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
createdAt: { type: Date, default: Date.now }
});

const Task = mongoose.model('Task', taskSchema);

// Authentication Middleware
const auth = async (req, res, next) => {
try {
const token = req.header('Authorization').replace('Bearer ', '');
const decoded = jwt.verify(token, process.env.JWT_SECRET);
const user = await User.findOne({ _id: decoded.userId });

if (!user) {
throw new Error();
}

req.user = user;
next();
} catch (error) {
res.status(401).json({ error: 'Please authenticate.' });
```

```
}
};
// ------------------ AUTH ROUTES ------------------
app.post('/api/auth/register', async (req, res) => {
try {
const { name, email, password } = req.body;

// Check if user already exists
const existingUser = await User.findOne({ email });
if (existingUser) {
return res.status(400).json({ error: 'Email already registered' });
}

// Hash password
const hashedPassword = await bcrypt.hash(password, 10);

// Create new user
const user = new User({
name,
email,
password: hashedPassword
});
```

```
await user.save();

// Generate token
const token = jwt.sign(
{ userId: user._id },
process.env.JWT_SECRET,
{ expiresIn: '24h' }
);

res.status(201).json({ token, user: { name: user.name, email: user.email } });
} catch (error) {
res.status(500).json({ error: 'Error creating user' });
}
});

app.post('/api/auth/login', async (req, res) => {
try {
const { email, password } = req.body;

// Find user
const user = await User.findOne({ email });
if (!user) {
return res.status(401).json({ error: 'Invalid credentials' });
```

```
}

// Check password
const isMatch = await bcrypt.compare(password, user.password);
if (!isMatch) {
return res.status(401).json({ error: 'Invalid credentials' });
}

// Generate token
const token = jwt.sign(
{ userId: user._id },
process.env.JWT_SECRET,
{ expiresIn: '24h' }
);

res.json({ token, user: { name: user.name, email: user.email } });
} catch (error) {
res.status(500).json({ error: 'Error logging in' });
}
});

// ------------------ TASK ROUTES ------------------
app.get('/api/tasks', auth, async (req, res) => {
```

```
try {
const tasks = await Task.find({ userId: req.user._id });
res.json(tasks);
} catch (error) {
res.status(500).json({ error: 'Error fetching tasks' });
}
});

app.post('/api/tasks', auth, async (req, res) => {
try {
const task = new Task({
...req.body,
userId: req.user._id
});
await task.save();
res.status(201).json(task);
} catch (error) {
res.status(500).json({ error: 'Error creating task' });
}
});

app.put('/api/tasks/:id', auth, async (req, res) => {
try {
```

```
const task = await Task.findOneAndUpdate(
{ _id: req.params.id, userId: req.user._id },
req.body,
{ new: true }
);
if (!task) {
return res.status(404).json({ error: 'Task not found' });
}
res.json(task);
} catch (error) {
res.status(500).json({ error: 'Error updating task' });
}
});

app.delete('/api/tasks/:id', auth, async (req, res) => {
try {
const task = await Task.findOneAndDelete({
_id: req.params.id,
userId: req.user._id
});
if (!task) {
return res.status(404).json({ error: 'Task not found' });
}
```

```
res.json({ message: 'Task deleted successfully' });
} catch (error) {
res.status(500).json({ error: 'Error deleting task' });
}
});

// Port endpoint
app.get('/api/port', (req, res) => {
res.json({ port: global.PORT || PORT });
});

// Serve static files
app.get('/', (req, res) => {
res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.get('/auth', (req, res) => {
res.sendFile(path.join(__dirname, 'public', 'auth.html'));
});

// Start server with error handling
const startServer = async (port) => {
const server = app.listen(port)
```

```
.on('error', (err) => {
if (err.code === 'EADDRINUSE') {
console.log(`Port ${port} is busy, trying ${port + 1}`);
startServer(port + 1);
} else {
console.error('Error starting server:', err);
}
})
.on('listening', () => {
console.log(`Server is running on port ${port}`);
// Update the PORT variable to match the actual port being used
global.PORT = port;
});
};

startServer(PORT);
```

APPENDIX 2

SNAPSHOTS

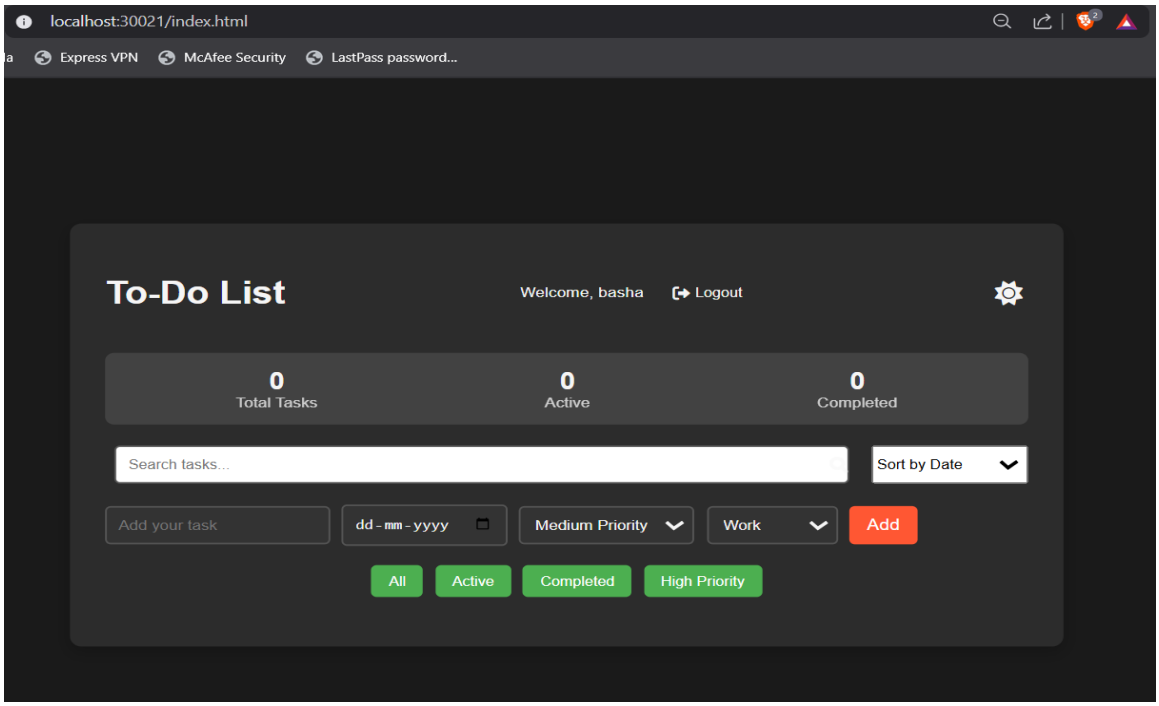


Figure A2.1 HOME PAGE

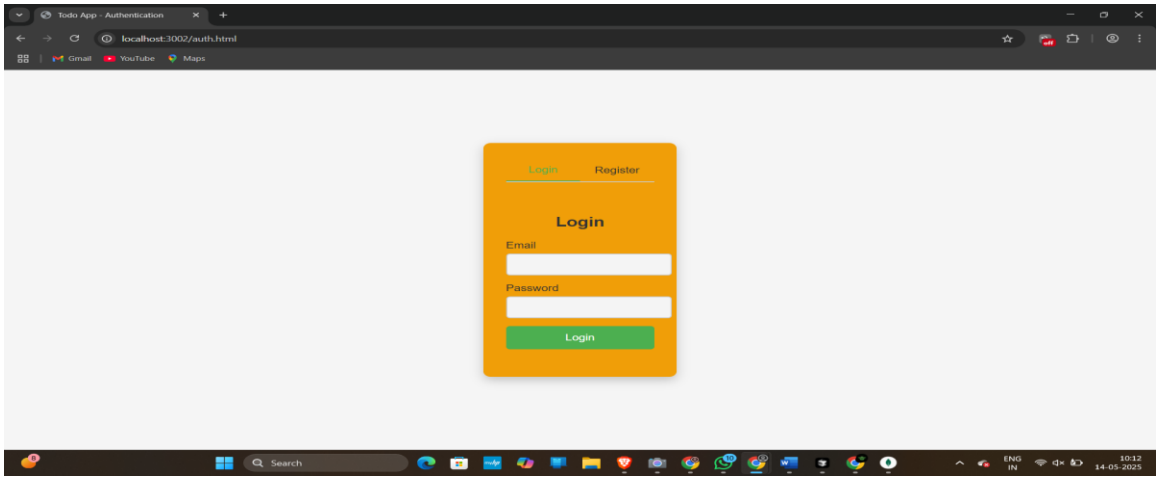


Figure A2.2 LOGIN PAGE

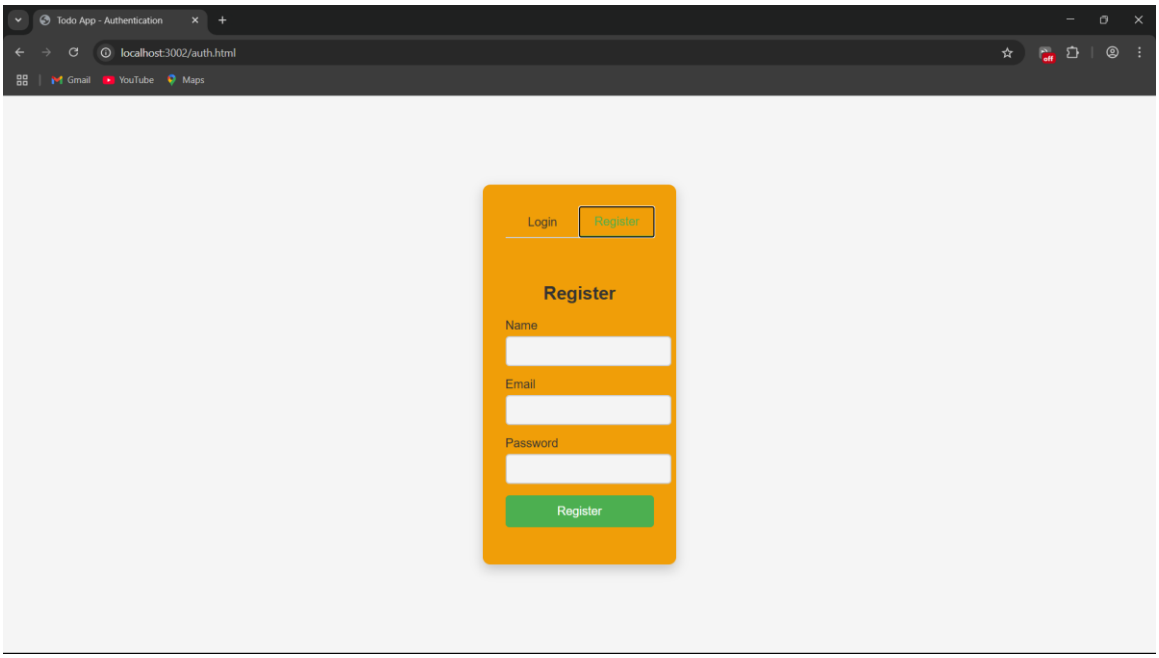


Figure A2.3 SIGNUP PAGE

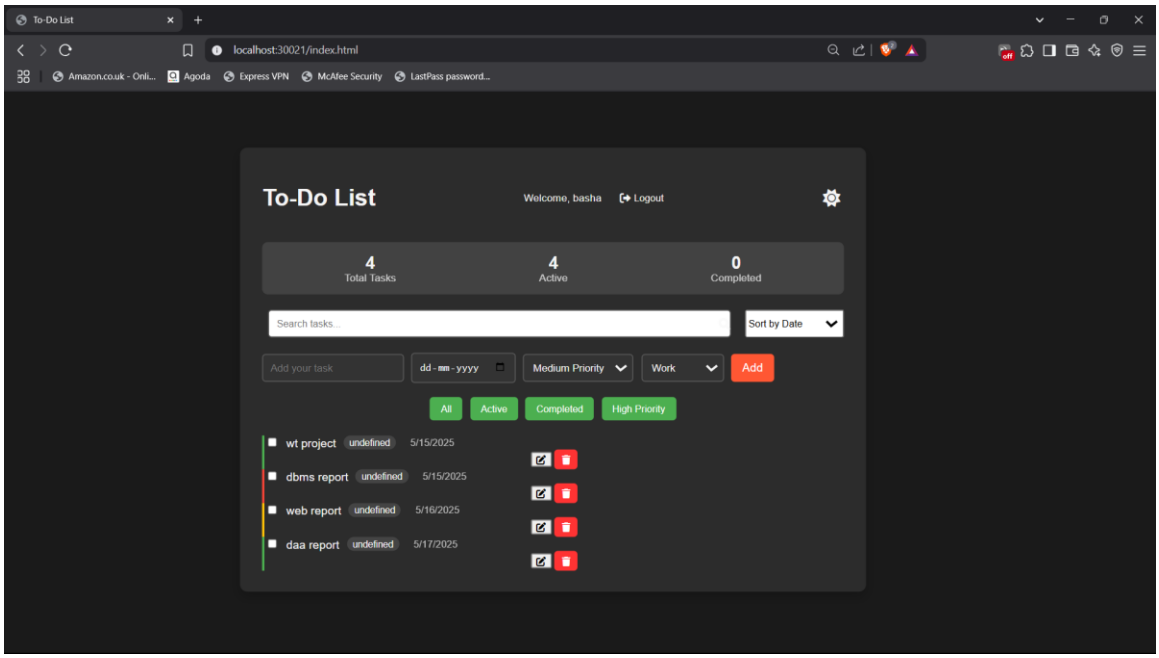
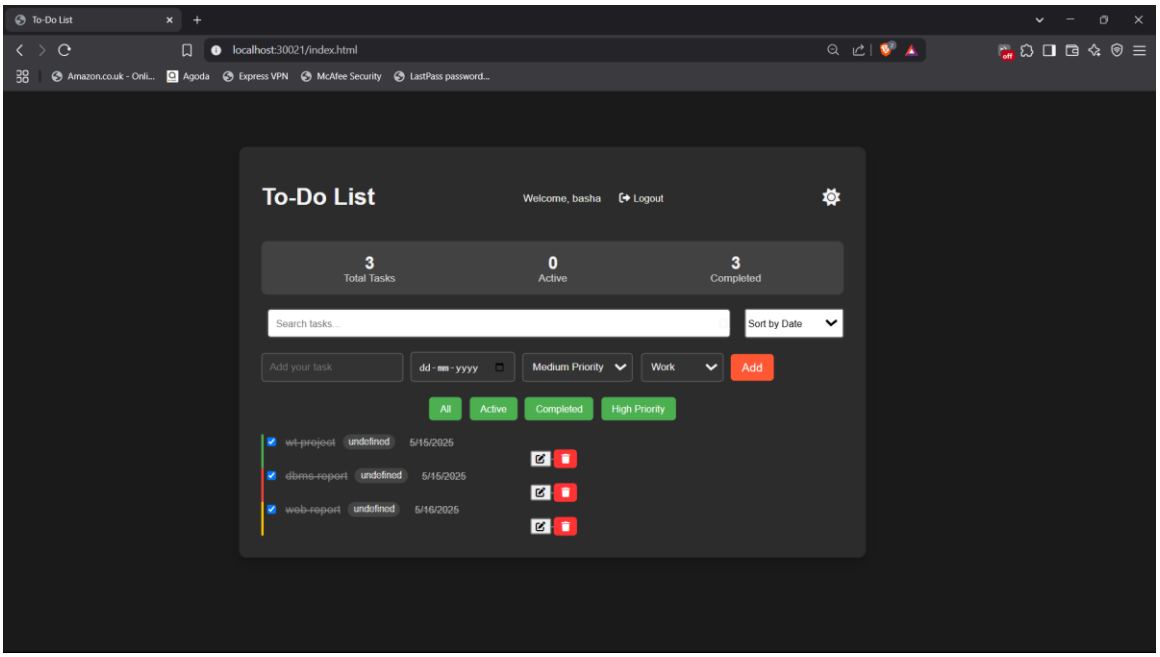


Figure A2. 4 ACTIVE TASKS PAGE



FigureA2.5 COMPLETED TASKS

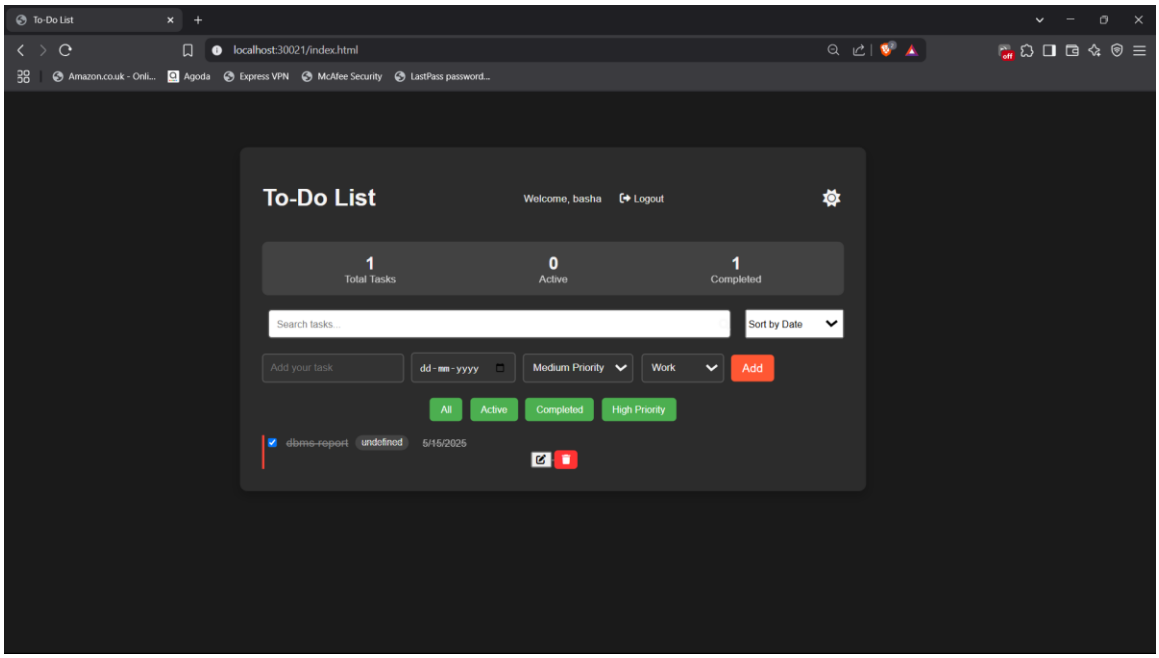


Figure A2.6 PRIORITY TASKS PAGE

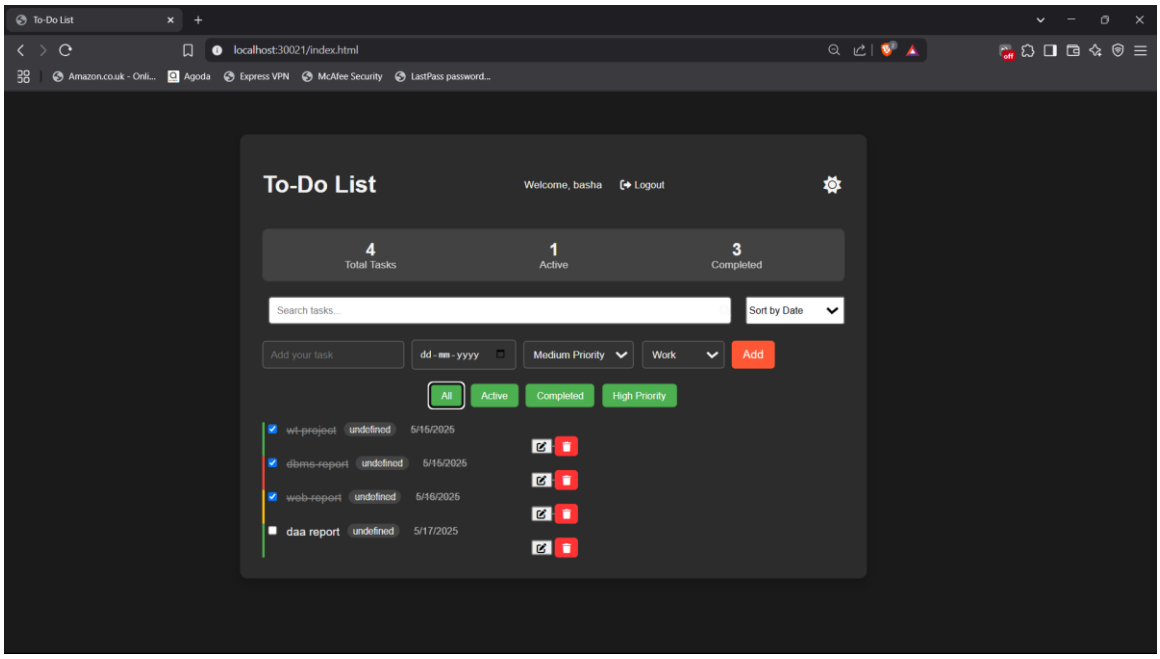


Figure A2.7 TASKS INFO PAGE

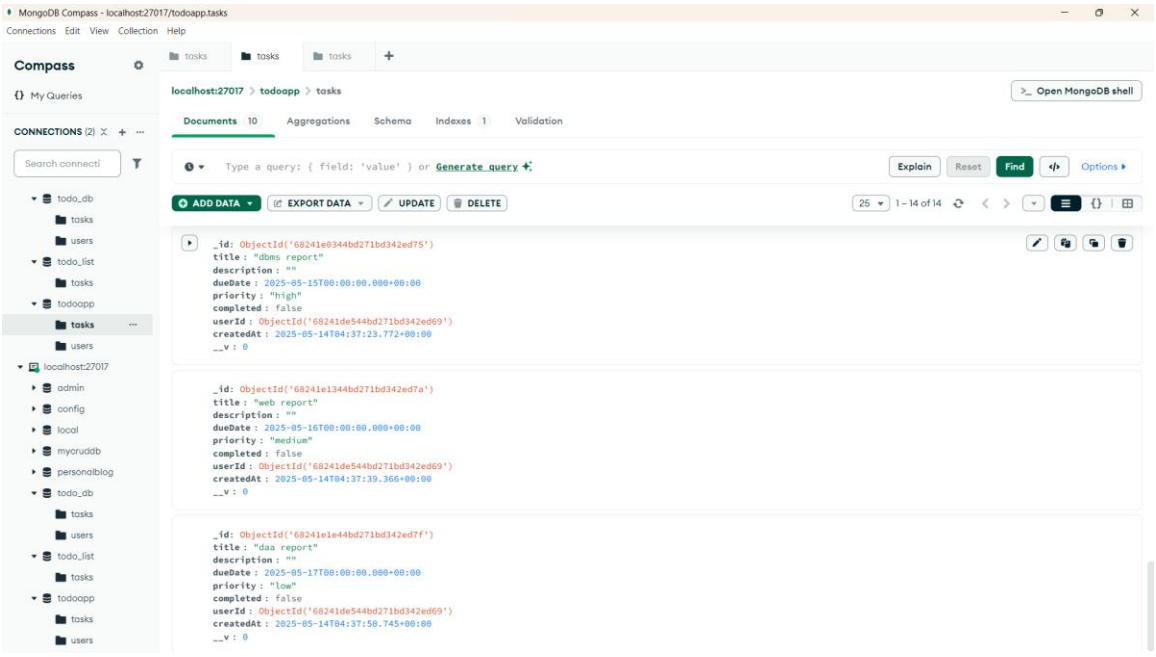


Figure A2.8 DATABASE PAGE

REFERENCES

1. MongoDB Official Documentation – https://www.mongodb.com/docs
2. Node.js Documentation – https://nodejs.org/en/docs
3. Express.js Guide – https://expressjs.com/en/starter/installing.html
4. Angular Framework Documentation – https://angular.io/docs
5. Mongoose ODM Documentation – https://mongoosejs.com/docs
6. Bcrypt.js GitHub Repository – https://github.com/kelektiv/node.bcrypt.js/
7. Visual Studio Code User Guide – https://code.visualstudio.com/docs
8. CORS Middleware for Node.js – https://www.npmjs.com/package/cors
9. Body-Parser Middleware – https://www.npmjs.com/package/body-parser
10. Dotenv Package Documentation – https://www.npmjs.com/package/dotenv
11. W3Schools – https://www.w3schools.com
12. GeeksforGeeks – https://www.geeksforgeeks.org
13. TutorialsPoint – https://www.tutorialspoint.com
14. Stack Overflow – https://stackoverflow.com