"Task Tracker Tool"

A Minor Project Report Submitted to
Rajiv Gandhi Proudyogiki Vishwavidyalaya

Towards Partial Fulfillment for the Award of
Bachelor of Engineering in Computer Science Engineering

Submitted by:
Adarsh Trivedi (0827CS223D02)

Guided by:
Prof. Ritika Bhatt
Computer Science and Engineering

Acropolis Institute of Technology & Research, Indore
Jan - Dec 2024

EXAMINER APPROVAL

The Minor Project entitled ***"Task Tracker Tool"*** submitted by

Adarsh Trivedi (0827CS223D02) has been examined and is hereby approved towards partial fulfillment for the award of ***Bachelor of Technology degree in Computer Science Engineering***discipline, for which it has been submitted. It understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed, or conclusion drawn therein, but approve the project only for the purpose for which it has been submitted.

(Internal Examiner)	**(External Examiner)**
Date:	**Date:**

RECOMMENDATION

This is to certify that the work embodied in this minor project entitled "***Task Tracker Tool***" submitted by ***Adarsh Trivedi (0827CS223D02)*** is a satisfactory account of the bonafide work done under the supervision of ***Dr. Kamal Kumar Sethi***, is recommended towards partial fulfillment for the award of the Bachelor of Technology (Computer Science Engineering) degree by Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal.

(Project Guide)

(Project Coordinator)

(Dean Academics)

STUDENTS UNDERTAKING

This is to certify that the minor project entitled ***"Task Tracker Tool"*** has developed by us under the supervision of ***Dr. Kamal Kumar Sethi***. The whole responsibility of the work done in this project is ours. The sole intension of this work is only for practical learning and research.

We further declare that to the best of our knowledge; this report does not contain any part of any work which has been submitted for the award of any degree either in this University or in any other University / Deemed University without proper citation and if the same work found then we are liable for explanation to this.

Adarsh Trivedi (0827CS223D02)

Acknowledgement

We thank the almighty Lord for giving me the strength and courage to sail out through the tough and reach on shore safely.

There are number of people without whom this project would not have been feasible. Their high academic standards and personal integrity provided me with continuous guidance and support.

We owe a debt of sincere gratitude, deep sense of reverence and respect to our guide and mentor **Prof. Rikita Bhatt,** Professor, AITR, Indore for his motivation, sagacious guidance, constant encouragement, vigilant supervision, and valuable critical appreciation throughout this project work, which helped us to successfully complete the project on time.

We express profound gratitude and heartfelt thanks to **Dr Kamal Kumar Sethi**, Professor & Head CSE, AITR Indore for his support, suggestion, and inspiration for carrying out this project. I am very much thankful to other faculty and staff members of the department for providing me all support, help and advice during the project. We would be failing in our duty if do not acknowledge the support and guidance received from **Dr S C Sharma**, Director, AITR, Indore whenever needed. We take opportunity to convey my regards to the management of Acropolis Institute, Indore for extending academic and administrative support and providing me all necessary facilities for project to achieve our objectives.

We are grateful to **our parent** and **family members** who have always loved and supported us unconditionally. To all of them, we want to say "Thank you", for being the best family that one could ever have and without whom none of this would have been possible.

Adarsh Trivedi (0827CS223D02)

Executive Summary

Task Tracker tool

This project is submitted to Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal (MP), India for partial fulfillment of Bachelor of Engineering in Information Technology branch under the sagacious guidance and vigilant supervision of ***Dr. Kamal Kumar Sethi***.

The Task Tracker tool project is dedicated to constructing an intuitive and This project develops a comprehensive task management system to streamline team and task management within an organization. It includes functionalities for creating teams, assigning tasks, and tracking progress through a user-friendly interface. Managers can manage teams and monitor task completion, while employees can update task statuses. The system uses Node.js and Express for the backend, React for the frontend, and MongoDB for data management. It ensures secure access via token-based authentication. Designed with user-centric principles, the system aims to enhance productivity and collaboration. Future enhancements may include real-time notifications, detailed analytics, and mobile app support.

Key words: Task Management, Team Collaboration, User-Centric Design, Task Tracking.

"Where the vision is one year,

cultivate flowers;

Where the vision is ten years,

cultivate trees;

Where the vision is eternity,

cultivate people." -

Oriental Saying

List of Figures

List of Tables

Table of Contents

Chapter 1.

Introduction

This project is a web application designed to streamline task management and team coordination for organizations. Developed using React for the front-end and leveraging a RESTful API for back-end operations, the application provides a robust platform for managers and employees to create, assign, and track tasks effectively.

The core functionality of the application revolves around task creation and management. Managers have the ability to create tasks with detailed descriptions, deadlines, and sub-tasks (referred to as "todos"). Each task is assigned to specific employees, who can then view their tasks, mark todos as complete, and track their progress. This feature ensures that all team members are on the same page regarding their responsibilities and deadlines, promoting efficiency and accountability within the organization.

Overall, this project aims to provide a comprehensive solution for task and team management within organizations. By facilitating clear communication, organized task distribution, and efficient progress tracking, the application supports teams in achieving their goals more effectively and with greater transparency.

1.1 Overview

This project involves developing a comprehensive task and team management web application using a tech stack that includes Node.js, React, and MongoDB. It features user-centric design principles to enhance usability and accessibility, ensuring an intuitive user experience. Key functionalities include task creation, tracking, and team collaboration, supported by token-based authentication for secure access. The application aims to streamline workflow management, improve productivity, and facilitate efficient team communication and coordination.

1.2 Background and Motivation

In today's fast-paced work environment, efficient task management and seamless team collaboration are critical for productivity and project success. Traditional methods of managing tasks, such as spreadsheets or paper-based systems, often fall short in providing real-time updates, clear accountability, and effective communication. This project aims to address these limitations by developing a web-based task and team management application, leveraging modern technologies to provide a robust and user-friendly platform.

The The motivation behind this project stems from the increasing need for digital solutions that cater to remote and hybrid work setups. With the rise of distributed teams, it has become essential to have tools that can facilitate remote collaboration, ensure timely task completion, and provide visibility into project progress. This application is designed to bridge the gap between team members, regardless of their physical location, by offering features like real-time updates, task assignments, and progress tracking, all accessible through a centralized platform.

1.3 Problem Statement and Objectives

Design a task tracking tool that helps teams manage and track their projects. Implement features like creating projects, assigning tasks to team members, setting deadlines, and tracking progress.

Objectives:

1.3.1 Develop a User-Friendly Interface : Create an intuitive and accessible user interface that simplifies task and team management processes. Ensure that the design is responsive and provides a seamless experience across various devices.

1.3.2 Efficient Task Management : We Implement functionalities for creating, assigning, and tracking tasks. Allow users to set deadlines, add descriptions, and update task statuses in real-time to enhance productivity and accountability.

1.3.3 Facilitate Team Collaboration: Build features that promote effective team collaboration, such as shared task lists, team member assignments, and progress tracking. Ensure that all team members can easily communicate and stay updated on project developments.

1.3.4 Integrate Secure Authentication: Ensure secure access to the application by implementing robust authentication mechanisms. Use token-based authentication to protect user data and maintain privacy.

1.3.5 Implement a Scalable and Flexible Backend: Develop a scalable backend using Node.js and MongoDB that can handle growing user demands and data volumes. Ensure the system is capable of efficient data storage, retrieval, and management to support the application's functionalities.

1.4 Scope of the Project

The scope of the Task Tracker tool project is defined to establish the boundaries and limitations of the system's functionality:
In Scope:

- **Task Management:** Develop features for creating, assigning, and tracking tasks. Implement due dates and priority settings for tasks. Provide real-time notifications for task updates and deadlines.

- **Team Management:** Enable the creation and management of teams. Allow for assigning tasks to specific teams or team members. Implement team collaboration tools such as shared task lists and progress tracking.

- **User Authentication and Authorization:** Implement secure login and registration processes. Develop role-based access control (RBAC) to differentiate permissions between managers and employees.

- **User Interface and Experience:** Design a user-friendly, intuitive interface for both web and mobile platforms. Ensure accessibility standards are met for a diverse range of users.

- **Reporting and Analytics:** Develop basic reporting features for tracking task completion rates and team performance. Provide dashboards for visualizing key metrics and progress.

Out of Scope:

- **Advanced Analytics:** Implementing complex data analytics and machine learning algorithms for predicting task outcomes or employee performance.

- **Third-Party Integrations:** Integrating with external project management tools, email systems, or other third-party software.

- **Extensive Customization:** Allowing for highly customizable workflows, templates, or user interfaces beyond the basic settings.

- **Offline Functionality:** Supporting task and team management features in an offline mode.

1.5 Team Organization

- Adarsh Trivedi : The development of the Task and Team Management System was undertaken solely by Adarsh Trivedi. Adarsh was responsible for all aspects of the project, including documentation, frontend development with React, backend development using Express and Node.js, database management with MongoDB, and thorough testing of the application. His comprehensive approach ensured a seamless integration of all components, resulting in a robust and user-friendly system designed to streamline team and task management processes effectively.

1.6 Report Structure

The project Task Tracker tool is categorized into five chapters.

Chapter 1: Introduction

This chapter provides an initial understanding of the project, outlining its relevance in the context of today's fast-paced work environment. It introduces the Task Tracker tool, discussing its background, motivation, problem statement, and scope.The chapter also offers a glimpse of the report's content.

Chapter 2: Review of Literature

This project builds upon existing literature in task and team management systems, addressing limitations such as scalability, usability, and accessibility. It integrates user-centric design principles, remote access technologies, and modern authentication methods to enhance functionality and user satisfaction, advancing beyond traditional management systems.

Chapter 3: Proposed System

Chapter 3 delves into the architecture and design of the Task Tracker tool. It describes the features and functionalities of the platform, with a focus on user interface design, including wireframes or mockups. Additionally, the technology stack used for development is explained.

Chapter 4: Implementation

This chapter offers insights into the methodologies and planning employed in the project. It outlines the approach taken for project development, including key milestones and timelines. The system architecture and how different components interact are discussed, along with the testing plan and methods used to ensure functionality and reliability.

Chapter 5: Conclusion

The final chapter serves as a conclusion to the project report. It summarizes the project's objectives, achievements, and findings, reflecting on the outcomes and benefits to the educational community. Project limitations and resource constraints are addressed, and concluding remarks underscore the significance of the Task Tracker tool in the broader context of education.

Chapter 2

Review of Literature

This project builds upon existing literature in task and team management systems, addressing limitations such as scalability, usability, and accessibility. It integrates user-centric design principles, remote access technologies, and modern authentication methods to enhance functionality and user satisfaction, advancing beyond traditional management systems.

2.1 Preliminary Investigation

2.1.1 Survey of Existing System

Existing System/Application 1: Trello

Advantages:

- Visual Workflow: Intuitive board interface.
- Easy Collaboration: Simple team management.
- Flexible: Customizable task organization.
- Integration: Connects with various tools.

Disadvantages:

- Limited Features: Lacks advanced tools.
- Fragmented Workflow: Multiple integration dependency.
- Basic Reporting: Limited analytical insights.
- Scalability Issues: May not suit large projects.

Existing System/Application 2: Asana

Advantages:

- Comprehensive Management: Extensive task features.
- Collaborative Workspace: Efficient team communication.
- User-Friendly: Intuitive interface design.
- Customizable Workflows: Flexible project setups.

Disadvantages:

- Steep Learning Curve: Challenging for beginners.
- Overwhelming for Small Teams: Excessive features.
- Limited Timeline View: Basic project visualization.
- Integration Reliance: Dependent on third-party tools.

Existing System/Application 3: Jira

Advantages:

- Comprehensive Management: Robust project features.
- Scalable: Suitable for all sizes.
- Customizable: Flexible workflows.
- Agile Support: Strong agile tools.

Disadvantages:

- Steep Learning Curve: Complex for new users.
- Overhead: High administrative effort.
- Interface Complexity: Can overwhelm users.

Existing System/Application 4: Microsoft Planner

Advantages:

- Integration**:** Seamless with Office 365.
- User-Friendly: Intuitive interface.
- Collaboration: Easy team management.
- Task Visualization: Simple board views.

Disadvantages:

- Limited Features: Basic functionality.
- No Time Tracking**:** Lacks detailed timelines.
- Scalability Issues: Not for large projects.

Existing System/Application 5: Todolist

Advantages:

- Simple Interface: Easy to use.
- Organized Tasks: Clear task management.
- Accessibility: Available on multiple devices.
- Productivity Boost: Enhances task focus.

Disadvantages:

- Limited Features: Basic functionality.
- No Collaboration: Lacks team support.
- No Advanced Reporting: Minimal insights.
- Scalability Issues: Not for complex projects.

2.2 Common Identified Gaps

Common Identified Gaps :

- Limited Advanced Features: Most tools lack comprehensive project management functionalities like Gantt charts, time tracking, and advanced reporting.

- Scalability Issues: These tools often struggle to scale effectively for large teams or complex projects.

- Basic Collaboration Tools: While collaboration features exist, they are often basic and insufficient for detailed project coordination.

- Integration Dependence: Heavy reliance on third-party integrations to achieve full functionality can lead to a fragmented workflow.

- These identified gaps can guide the final project objectives for your Task Tracker tool development to address these limitations and provide a more comprehensive solution.

2.3 Requirement Identification and Analysis for Project

In Requirement identification and analysis is a crucial phase in the development of the Task Tracker tool, as it lays the foundation for understanding the needs and expectations of end-users. This process begins with gathering detailed information from various stakeholders, including project managers, team members, and potential users, to capture a comprehensive list of functional and non-functional requirements. Methods such as interviews, surveys, and focus group discussions are employed to ensure that the collected data reflects the real-world challenges and scenarios that the tool aims to address.

Functional requirements are centered on the core capabilities of the Task Tracker, such as task creation, assignment, prioritization, and tracking. Additional features like real-time collaboration, customizable workflows, and automated notifications are also identified to enhance user experience and productivity. Non-functional requirements focus on the system’s performance, security, scalability, and usability. These include ensuring that the application can handle a growing number of users and tasks without compromising performance, providing robust data security measures, and maintaining an intuitive and user-friendly interface.

The analysis phase involves organizing and prioritizing these requirements to determine their feasibility and relevance to the project's goals. Techniques such as use case analysis, requirement modeling, and feasibility studies are utilized to create a clear and structured requirement specification document. This document serves as a reference point throughout the development process, guiding the design, implementation, and testing phases to ensure that the final product aligns with user needs and expectations.

By thoroughly identifying and analyzing requirements, the project team can mitigate potential risks and challenges early in the development cycle. This proactive approach not only enhances the quality and functionality of the Task Tracker tool but also ensures that it delivers tangible benefits in terms of improved task management, streamlined workflows, and enhanced team collaboration. Ultimately, this phase is instrumental in setting the stage for a successful project outcome that meets both user demands and organizational objectives.

2.3.1 Conclusion

In conclusion, the requirement identification and analysis phase is essential to the successful development of the Task Tracker tool, providing a clear roadmap for the project. Through comprehensive stakeholder engagement, including interviews, surveys, and focus groups, the project team ensures that the captured requirements are accurate and reflective of real user needs. Functional requirements such as task creation, prioritization, and real-time collaboration are meticulously documented, alongside non-functional requirements like system performance, security, and scalability.

This phase employs various analytical techniques, including use case analysis, requirement modeling, and feasibility studies, to prioritize and validate requirements. The resulting requirement specification document serves as a critical reference throughout the project lifecycle, ensuring that design, development, and testing efforts remain aligned with user expectations and project goals.

By thoroughly identifying and analyzing requirements, the project team can anticipate and address potential challenges early, reducing risks and ensuring a smoother development process. This proactive approach enhances the likelihood of delivering a high-quality, user-friendly Task Tracker tool that meets the diverse needs of individuals and teams, ultimately optimizing productivity, streamlining workflows, and improving project outcomes. The rigorous focus on requirement identification and analysis underscores its importance in achieving a successful, user-centered product that stands up to the demands of modern task and project management.

Chapter 3

Track Mven

3.1 The Proposal

The proposal for the Task Tracker project outlines the development of a robust and user-friendly software solution designed to enhance task and project management for individuals and teams. Leveraging the modern MERN (MongoDB, Express.js, React, Node.js) stack, this project aims to address common challenges such as disorganization, poor communication, and inefficient workflow management. The tool will feature a visually intuitive interface, enabling users to create, assign, prioritize, and track tasks effortlessly. It will support real-time collaboration and communication, customizable workflows, and advanced automation capabilities to streamline processes. Additionally, the software will offer enhanced data visualization to provide insights into project progress and performance metrics. By ensuring scalability and performance, the Task Tracker will be adaptable for both small teams and large organizations. The ultimate goal of this project is to optimize productivity, facilitate better team coordination, and improve overall project outcomes, making task management more efficient and effective.

3.2 Benefits of the Proposed System

The proposed Task Tracker system offers a multitude of benefits aimed at improving task and project management for individuals and teams. By leveraging the MERN stack, the system ensures a seamless and efficient user experience with a highly intuitive interface. Key benefits include:

- **Enhanced Productivity:** The system's ability to streamline task creation, assignment, and tracking helps users focus on their priorities, reducing time wasted on administrative tasks and enhancing overall productivity.
- **Improved Collaboration:** Real-time collaboration features enable team

members to communicate effectively, share updates, and work together on tasks without delays, fostering a more collaborative and cohesive work environment.

- **Customizable Workflows:** With customizable workflows, teams can tailor the system to fit their specific project needs and processes, ensuring that the tool adapts to their way of working rather than forcing them into a rigid structure.
- **Advanced Automation:** Automation capabilities help to eliminate repetitive tasks, such as sending reminders or updating task statuses, allowing users to focus on more critical aspects of their projects.
- **Scalability:** The system is designed to scale efficiently, making it suitable for both small teams and large organizations. This scalability ensures that as the user base grows, the system can handle increased demand without compromising performance.
- **Enhanced Data Visualization:** Advanced data visualization tools provide clear insights into project progress and performance metrics. Users can generate detailed reports and dashboards, enabling better decision-making and strategic planning.
- **Centralized Task Management:** The Task Tracker provides a centralized hub for organizing and overseeing all tasks and projects, reducing the risk of tasks falling through the cracks and ensuring that deadlines are met consistently.
- **Security and Compliance:** By incorporating robust security measures, the system ensures that user data is protected, and compliance with relevant data protection regulations is maintained.

3.3 Block Diagram

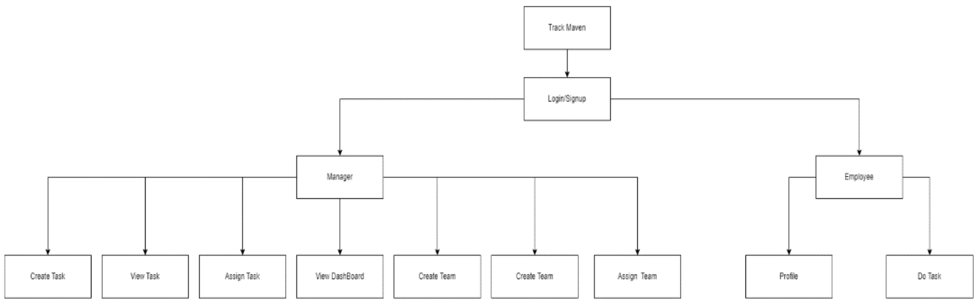


Figure 3.1 : Block Diagram

3.4 Feasibility Study

A Before embarking on the development of the Task Tracker tool, it's imperative to conduct a feasibility study to assess the project's viability.

3.4.1 Technical Feasibility:

The technical feasibility of the Task Tracker project is promising due to the mature and widely-used technologies in the MERN stack (MongoDB, Express.js, React, Node.js). Each component of the stack has extensive documentation, robust community support, and a history of successful deployment in various applications. MongoDB offers scalable and flexible data storage, while Express.js and Node.js provide a powerful backend framework for handling server-side logic and requests. React.js ensures a dynamic and responsive user interface. The integration of these technologies can be achieved effectively, considering their compatibility and interoperability.

3.4.2 Economic Feasibility:

Economic feasibility is assessed by evaluating the costs versus the benefits of the Task Tracker project. Initial costs include development expenses, infrastructure setup, and ongoing maintenance. However, the potential return on investment is substantial. By improving task management, streamlining workflows, and enhancing team collaboration, organizations can achieve significant productivity gains. This leads to better project outcomes, timely delivery, and reduced operational inefficiencies. The tool can be marketed to a broad audience, from small teams to large enterprises, ensuring diverse revenue streams.

3.4.3 Operational Feasibility:

User Acceptance The operational feasibility of the project is high, as the Task Tracker tool directly addresses common issues faced by teams and individuals in managing tasks and projects. The system's features, such as real-time collaboration, customizable workflows, and advanced automation, align with user needs for improving productivity and efficiency. The intuitive interface design minimizes the learning curve, promoting user adoption and engagement. Additionally, the project’s scalability ensures it can accommodate the growth of user bases and adapt to varying organizational sizes.

3.4.4 Legal Feasibility:

The legal feasibility considers compliance with data protection regulations such as GDPR, CCPA, and other relevant laws. The Task Tracker tool will incorporate robust security measures, including data encryption, access controls, and regular audits to ensure user data privacy and protection. Adhering to these regulations will mitigate legal risks and build user trust.

3.4.5 Schedule Feasibility :

The schedule feasibility is determined by creating a realistic timeline for project development, including planning, design, implementation, testing, and deployment phases. Given the team's expertise with the MERN stack and the availability of well-defined project management methodologies, the project can be completed within an estimated 6-9 months. Regular milestone reviews and agile development practices will ensure the project stays on track.

3.5 Deployment Requirements

The successful deployment of the Task Tracker tool necessitates a set of specific requirements encompassing hardware, software, and services. These deployment prerequisites ensure the system's seamless operation and availability for users.

3.6.1 Hardware

To support the deployment and functioning of the Task Tracker tool, the following hardware requirements are imperative:

- **Server Infrastructure:** A Robust servers capable of hosting the backend application components. Sufficient processing power and memory to handle concurrent user requests. Adequate storage space for storing application data and user files.

- **Network Infrastructure:** Desktop computers, laptops, tablets, and smartphones for accessing the Task Tracker application. Compatible hardware configurations to run modern web browsers and ensure optimal user experience.

- **Client Devices:** Desktop computers, laptops and tablets for accessing the Task Tracker application. Compatible hardware configurations to run modern web browsers and ensure optimal user experience.

3.6.2 software

The Task Tracker tool relies on a suite of software components to operateefficiently. The following software requirements are integral to the system's deployment:

- **Operating System:** Server operating system such as Linux (e.g., Ubuntu, CentOS) or Windows Server for hosting backend services. Compatibility with various client operating systems including Windows, macOS, iOS, and Android.

- **Backend Technologies:** Node.js runtime environment for running server-side JavaScript code. Express.js framework for building RESTful APIs and handling HTTP requests. MongoDB database management system for storing and managing application data.

- **Frontend Technologies:** A React.js library for building dynamic and responsive user interfaces. Modern web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge for accessing the Task Tracker application.

- **Development Tools:** Integrated development environments (IDEs) such as Visual Studio Code, Atom, or WebStorm for coding and debugging. Version control systems like Git for collaborative development and source code management.

- **Security Software:** Security software, including firewalls and intrusion detection systems, is necessary to protect the system from external threats and vulnerabilities.

- **Development Tools:** Development tools, like integrated development environments (IDEs) and version control systems, are crucial for the ongoing maintenance and enhancement of the platform.

Chapter 4

Implementation

we transition from conceptualization to the tangible realization of our Task Tracker tool. We delve into the practical aspects of the project's development and deployment. The chapter begins by introducing our chosen technology stack, encompassing programming languages, frameworks, and databases that underpin the platform's architecture. We then provide a detailed overview of the system architecture, elucidating how various components interact, offering a clear view of the platform's structure.

The user interface (UI) design, a critical element, is presented, and where applicable, wireframes or mockups are included, offering a visual representation of the platform's user experience. A thorough examination of the database structure follows, revealing the intricacies of data organization within the platform. Our implementation methodology is outlined, detailing the steps, strategies, and a Gantt chart with milestones, ensuring that the project progresses smoothly within the college minor project timeline.

The critical testing plan for the platform is described, ensuring that it meets the functionality and reliability expectations suitable for a college minor project. Finally, we outline the hardware, software, and service requirements for the platform's deployment, ensuring its smooth operation. This chapter acts as the bridge between vision and reality, bringing to life the Task Tracker tool.

4.1 Language Used

JavaScript: The project predominantly employs JavaScript for both front-end and back-end development. JavaScript is chosen for its versatility and extensive ecosystem, allowing for the creation of dynamic and interactive web applications.

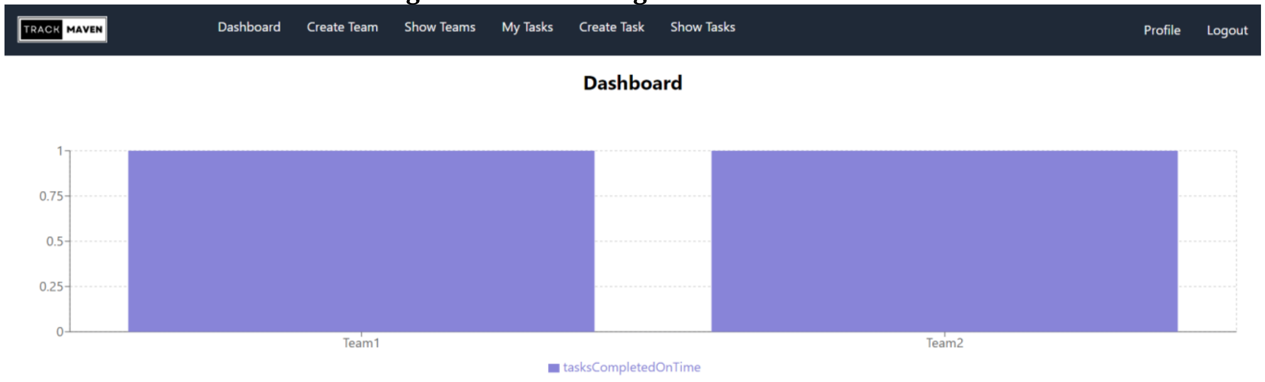
React.js: On the frontend, React.js takes center stage, presenting a JavaScript library celebrated for its capacity to build captivating user interfaces. A defining feature is React's component-based architecture, which simplifies development by allowing the creation of reusable UI components. This approach not only enhances the efficiency of development but also ensures consistency throughout the user experience. Additionally, React.js leverages its virtual DOM (Document Object Model), a key feature that optimizes performance by minimizing rendering time. This translates into a smooth and responsive user interface, a crucial aspect of our platform's appeal.

Node.js: On the back-end, Node.js is used with the Express.js framework to build scalable and performant server-side applications. Node.js provides a non-blocking, event-driven architecture, making it suitable for handling concurrent requests and real-time data processing.

4.2 Screenshots

The Following are the screenshots of the result of the project :

Figure No. 4.1- Manager Dashboard



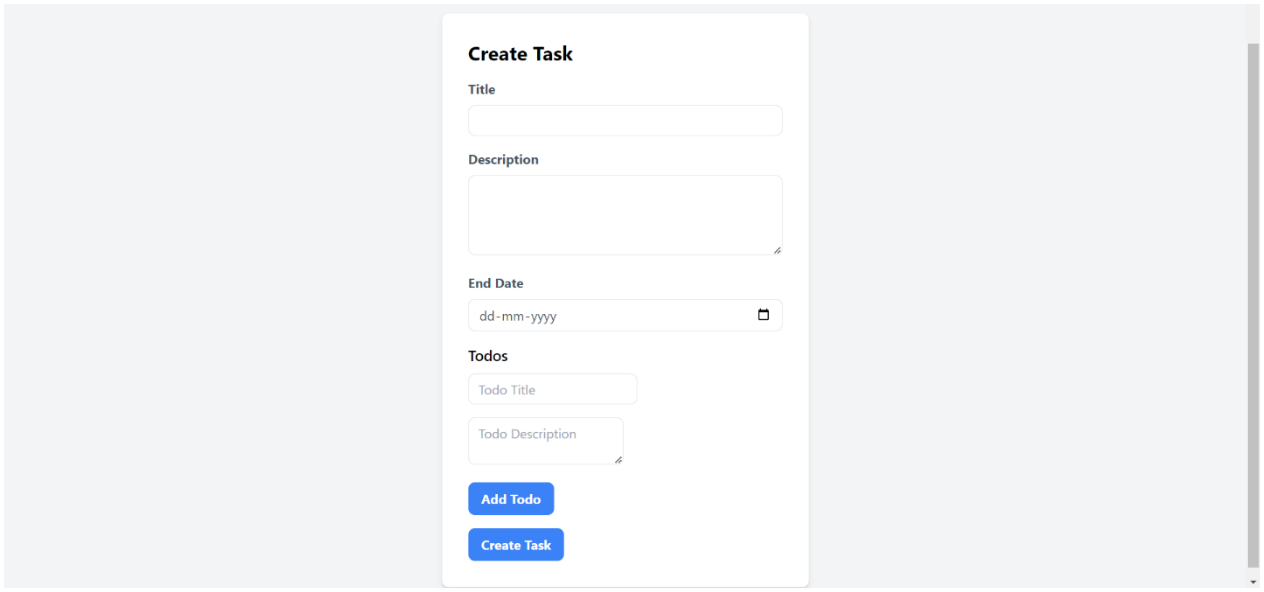
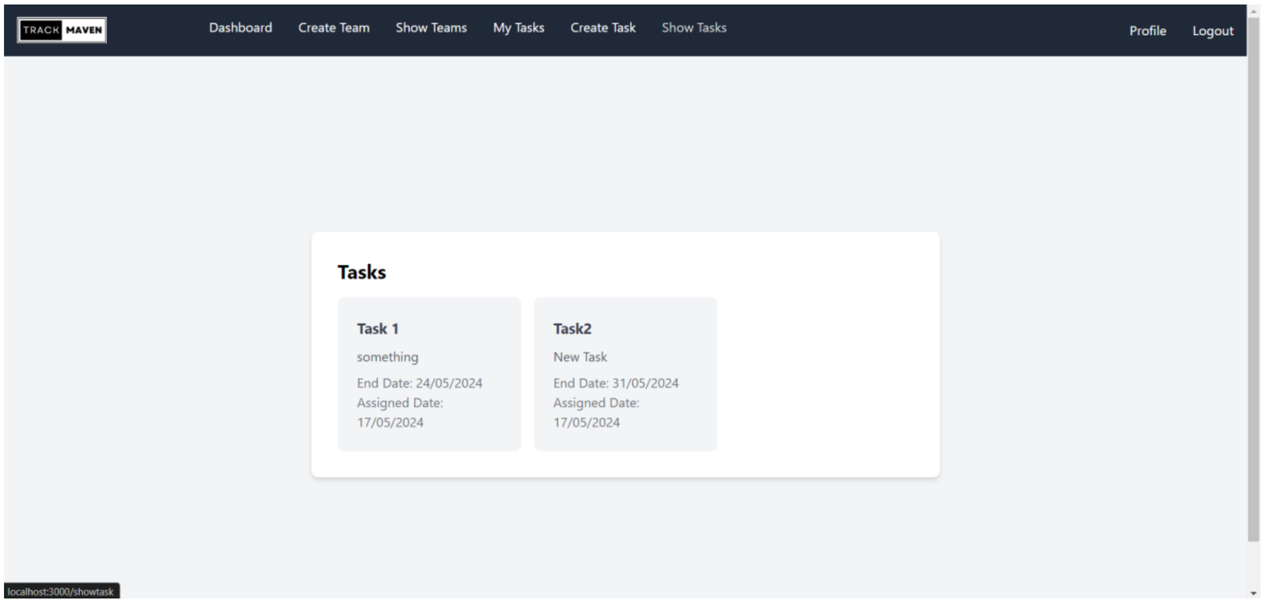


Figure 4.2 – Create Task



Tasks

Task 1
something
End Date: 24/05/2024
Assigned Date: 17/05/2024

Task2
New Task
End Date: 31/05/2024
Assigned Date: 17/05/2024

Figure 4.3 – Show Tasks

4.3 Testing

Testing is the process of evaluation of a system to detect differences between given input and expected output and also to assess the feature of the system.

Testing assesses the quality of the product. It is a process that is done during the development process. .

4.3.1 Test Case and Analysis

TEST CASE: 1

Test Case ID	TC001
Test Case Summary	Verify whether the system successfully adds a new Task
Test Procedure	Navigation: Navigate to the Crate Task. Action: Click on the "Create Task" button. Input: Enter the Title, Description, End Date, Todos and Todo Description. Submit: Click the "Create Tasks" button to add the Task to Tasks.
Expected Result	The system should add the Tasks to theTasks without any errors.
Actual Result	The Task is added to theTask without issues.
Status	Pass

Table 1 : Test Case 1

TEST CASE 1 OUTPUT

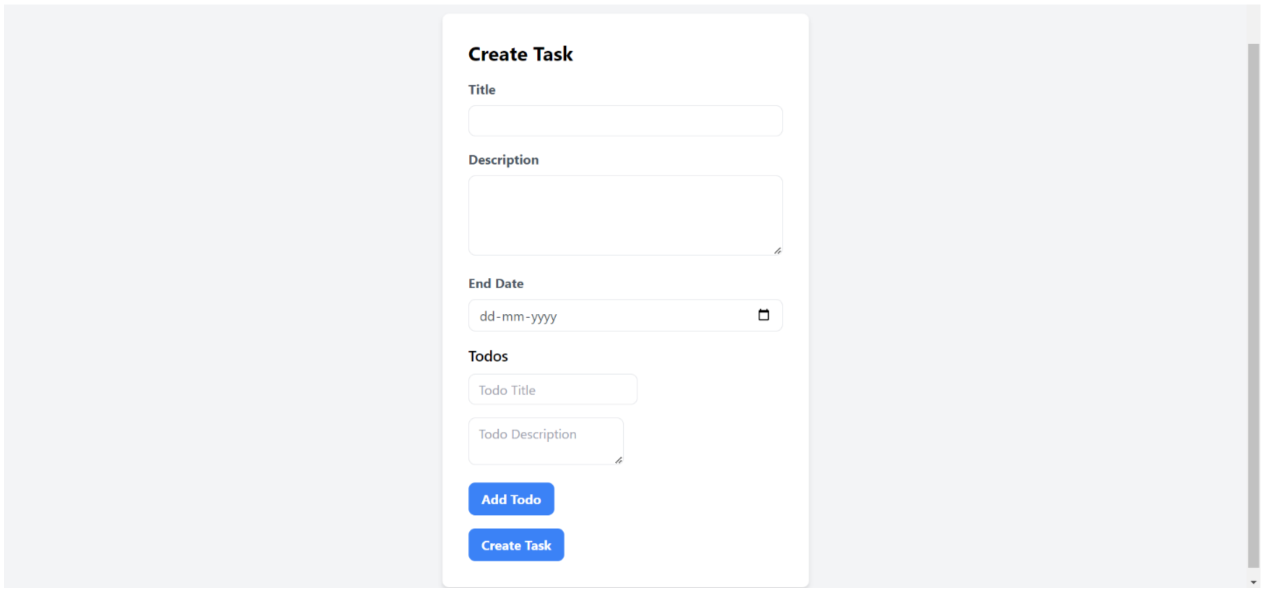


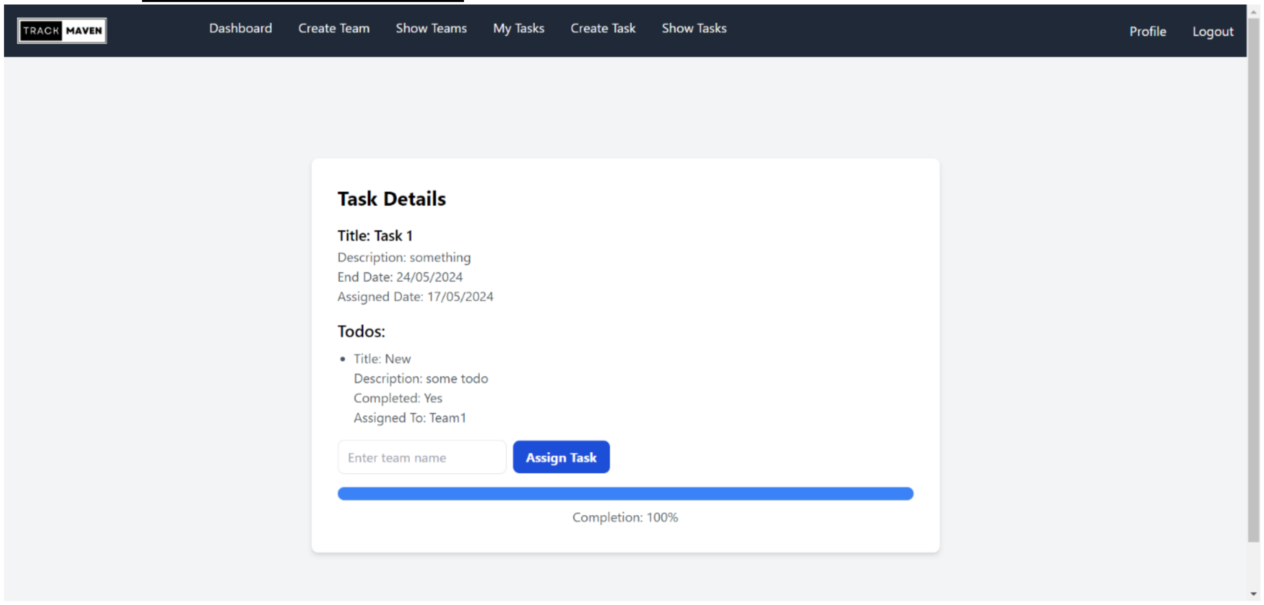
Figure4.4 : Test Case 1 output

<u>TEST CASE: 2</u>

Test Case ID	TC002
Test Case Summary	Verify whether Created task Assign to the team successfully.
Test Procedure	Login: Log in to the Task Tracker tool as a registered Manager. Navigation: Access the list of available Tasks. Selection: Choose a Task to Assign for teams. Assign to team: Enter the Team Name and click to Assign.
Expected Result	The assign task to show to the Employees in my Tasks Section.
Actual Result	The Tasks Show in the Employees My Task Section
Status	Pass

Table 2: Test Case 2

<u>TEST CASE 2 OUTPUT</u>



Task Details

Title: Task 1
Description: something
End Date: 24/05/2024
Assigned Date: 17/05/2024

Todos:

- Title: New
  Description: some todo
  Completed: Yes
  Assigned To: Team1

Completion: 100%

Figure 4.5 : Test Case 2 Output 1

Chapter 5.

Conclusion

5.1 Conclusion

The Task Tracker project is designed to transform task and project management for individuals and teams by offering a comprehensive and user-friendly solution. Utilizing the MERN stack (MongoDB, Express.js, React, Node.js), the project ensures a robust, scalable, and efficient application that addresses issues such as disorganization, poor communication, and inefficient workflows. The application features an intuitive interface, real-time collaboration, customizable workflows, and advanced automation capabilities, all aimed at enhancing productivity, streamlining processes, and improving project outcomes.

The project's feasibility has been carefully considered in terms of technical, operational, economic, legal, and schedule aspects, ensuring a high likelihood of success. Deployment requirements, encompassing both hardware and software, have been meticulously planned to guarantee reliability and performance. Ultimately, the Task Tracker tool aims to significantly enhance task and project management, providing a valuable resource for individuals and organizations seeking greater efficiency and effectiveness in their work.

5.2 Limitations of the Work

- **Scalability Constraints**

  While the MERN stack provides a scalable architecture, extreme scaling requirements, such as handling thousands of concurrent users in real-time, might necessitate more complex infrastructure and optimization techniques, which are beyond the scope of this initial project.

- **Customization Flexibility**

  Although the system offers customizable workflows, there might be limitations in accommodating highly specific or complex business processes without significant custom development work.

- **Integration Challenges**

Integrating with other third-party tools and systems can be challenging and might require additional middleware or APIs. Not all external systems may be easily compatible, leading to potential integration issues.

- **Security Concerns**

Ensuring robust security measures is a continuous challenge. While the project includes basic security protocols, advanced security features like real-time threat detection, comprehensive encryption, and detailed audit trails may need further enhancement.

- **Feature Completeness**

The initial version of the Task Tracker might not include all the desired features and functionalities. Continuous development and iterations will be needed to fully meet user needs and preferences.

5.3 Suggestion and Recommendations for Future Work

To further enhance the Task Tracker project and address its limitations, several suggestions and recommendations are proposed. First, enhancing scalability through advanced solutions such as microservices architecture, load balancing, and distributed databases will better handle high user loads and large data volumes. Improving customization options with a more flexible workflow editor or a plugin architecture can accommodate complex business processes. Strengthening integrations by developing robust APIs and middleware solutions will facilitate seamless integration with a wider range of third-party tools.

Investing in advanced security measures, including real-time threat detection and multi-factor authentication, will ensure robust protection against emerging threats. Continuous performance optimization, through techniques like database indexing and query optimization, will maintain system responsiveness under heavy loads. Providing comprehensive training materials and a responsive support team will assist users in maximizing the tool's capabilities. Expanding the feature set regularly based on user feedback and market trends will keep the tool competitive, while developing data migration tools will aid organizations in transitioning from existing systems.

Implementing offline accessibility through progressive web applications or local data caching will enhance usability for users with unreliable internet connections. Lastly, adding multi-language support will make the Task Tracker accessible to a global audience. By addressing these areas, the Task Tracker project can evolve into a more robust, versatile, and user-friendly tool, capable of meeting the diverse needs of its user base and staying ahead in the competitive task management solutionslandscape.

Bibliography

[1] Atlassian, "Jira Software," in Atlassian, [Online]. Available: https://www.atlassian.com/software/jira.

[2] B. Trello, "Trello," in Trello, [Online]. Available: https://trello.com/.

[3] S. Asana, "Asana," in Asana, [Online]. Available: https://asana.com/.

[4] Doist, "Todoist," in Todoist, [Online]. Available: https://todoist.com/.

Appendix A. Source Code

```
const user =require("./../Model/User");
const jwt=require("jsonwebtoken");
exports.loginHandler=async(req,res,next)=>{
    const {email,password}=req.body;
    const User=await user.findOne({Email:email}).select('+Password');
    if(!User){
        return res.status(404).json({
           status:"Failed"
        })
    }

    const authenticated=User.checkPassword(password,User.Password);
    if(!authenticated){
        return res.status(401).json({
            status:"unauthorized"
        })
    }

    const token=jwt.sign(email,process.env.JWT_Secret);
        res.status(200).json({
            status:"Success",
            token:token,
            role:User.Role
        })

    next()
}
const Task = require("./../Model/Task");
const User=require("./../Model/User")
const jwt = require("jsonwebtoken");
const { roleChecker } = require("./../utils/validate")

exports.getTasks = async (req, res, next) => {
    try {
        const Tasks = await Task.find();
        res.status(200).json({
            status: "Success",
            data: Tasks
        })
        next();
    }
    catch (err) {
        console.log(err);
    }
}

exports.getTask = async (req, res, next) => {
    try {
        const id=req.params.id;
        const Tasks = await Task.findOne({_id:id});
        res.status(200).json({
            status: "Success",
            data: Tasks
        })
        next();
    }
    catch (err) {
        console.log(err);
    }
}
exports.createTask = async (req, res, next) => {
    try {
        const token = req.headers.authorization.split(" ")[1];
        const decoded = jwt.verify(token, process.env.JWT_secret);
        const validated = await roleChecker("Manager", decoded);
        if (!validated) {
            return res.status(401).json({
                status: "Unauthorized"
            })
        }
        const task = await Task.create(req.body)
        res.status(201).json({
            status: "Success",
            data: task
        })
        next();
    }

    catch (err) {
        console.log(err);
        res.status(501).json({
            status: "Failed"
        })
    }

}

exports.assignTask = async (req, res, next) => {
    try {
        const token = req.headers.authorization.split(" ")[1];
        const decoded = jwt.verify(token, process.env.JWT_secret);
        const validated = await roleChecker("Manager", decoded);
        if (!validated) {
            return res.status(401).json({
                status: "Unauthorized"
            })
        }

        const TaskId=req.body.id;
        const teamName=req.body.teamName;
        const team=await User.updateMany({Team:teamName},{
            $push:{TaskId}
        })

        const assignTask=await Task.findByIdAndUpdate(TaskId,{Assigned_to:teamName});
        res.status(200).json({
            status: "Success",
            message: "Task assigned to all team members"
        });
        next();
    }

    catch (err) {
        console.log(err);
        res.status(501).json({
            status: "Failed"
        })

    }

}

exports.getEmployeeTasks=async(req,res,next)=>{
    try{
    const token=req.headers.authorization.split(" ")[1];
    const decoded=jwt.verify(token,process.env.JWT_secret);
    const user=await User.findOne({Email:decoded}).populate("TaskId");
    const tasks=user.TaskId;

    res.status(200).json({
        status:"Success",
        tasks
    })
    next();
}

catch(err){
    console.log(err);
}

}


exports.deleteTask=async(req,res,next)=>{
    try{
        const token = req.headers.authorization.split(" ")[1];
        const decoded = jwt.verify(token, process.env.JWT_secret);
        const validated = await roleChecker("Manager", decoded);
        if (!validated) {
            return res.status(401).json({
                status: "Unauthorized"
            })
        }
    const taskId=req.query.id;

    const task=await Task.findOneAndDelete({_id:taskId})
    res.status(200).json({
        status:"Success",
        task
    })
    next();
}

catch(err){
    console.log(err);
}

}

exports.updateTask = async (req, res, next) => {
    try {
      const token = req.headers.authorization.split(" ")[1];
      const decoded = jwt.verify(token, process.env.JWT_secret);
      const user = await User.findOne({ Email: decoded });
      if (!user) {
        return res.status(401).json({
          status: "unauthorized",
        });
      }
      const taskId = req.params.id;
      const updatedTask = req.body;
      console.log(req.body);
      const task = await Task.findOneAndUpdate(
        { _id: taskId },
        updatedTask,
        {
          new: true,
          runValidators: true,
        }
      );

      const allTodosCompleted = task.todos.every((todo) => todo.completed);
      if (allTodosCompleted && !task.completed) {
        task.completed = true;
        await task.save();
      }

      res.status(200).json({
        status: "Success",
        task,
      });
      next();
    } catch (err) {
      console.log(err);
    }
```

```
  };

  exports.getTaskCompletionRate = async (req, res, next) => {
    try {
      const aggregation = await Task.aggregate([
        {
          $group: {
            _id: "$Assigned_to",
            totalTasks: { $sum: 1 },
            completedTasks: { $sum: { $cond: [{ $eq: ["$completed", true] }, 1, 0] } },
          },
        },
        {
          $project: {
            _id: 1,
            totalTasks: 1,
            completedTasks: 1,
            completionRate: { $divide: ["$completedTasks", "$totalTasks"] },
          },
        },
      ]);

      res.status(200).json({
        status: "Success",
        data: aggregation,
      });
    } catch (err) {
      console.log(err);
      res.status(500).json({
        status: "Failed",
      });
    }
  };
  const jwt=require("jsonwebtoken");
const User=require("./../Model/User")
const {roleChecker}=require("./../utils/validate")

exports.getTeams=async(req,res,next)=>{
    try{
        const token=req.headers.authorization.split(" ")[1];
        const decoded=jwt.verify(token,process.env.JWT_secret);
        const validated= await roleChecker("Manager",decoded);
        if(!validated|| !decoded){
            return res.status(401).json({
                status:"Unauthorized"
        })}

        const teams=await User.find({Team:{$ne:null}});

        const Teams=teams.map(el=>{
           return el.Team
        })
        const uniqueTeamNames = [...new Set(Teams)];
        res.status(200).json({
            status:"Success",
            Teams:uniqueTeamNames
        })
        next();
    }
    catch(err){
        res.status(501).json({status:"Failed"})
        console.log(err);
    }
}
exports.createTeam=async(req,res,next)=>{
    try{
        const token=req.headers.authorization.split(" ")[1];
        const decoded=jwt.verify(token,process.env.JWT_secret);
        const validated= await roleChecker("Manager",decoded);
        if(!validated|| !decoded){
            return res.status(401).json({
                status:"Unauthorized"
            })
        }
        const name=req.body.name;
        const EmployeeID=req.body.EmployeeID;
        const team=await User.findOneAndUpdate({EmployeeID},{Team:name});
        res.status(201).json({
            message:"Team created successfully",
            team:team
        })
        next();
    }

    catch(err){
        res.status(501).json({
            status:"Failed"
        })
        console.log(err);

    }
}

exports.getTeam=async(req,res,next)=>{
    try{
        const Team=req.query.id;
        const TeamData=await User.find({Team});

        res.status(200).json({
            status:"Success",
            TeamData
        })

        next();
    }
    catch(err){
        res.status(501).json({
            status:"Failed"
        })
        console.log(err);
    }
}

exports.deleteTeams=async(req,res,next)=>{
    try{
        const token=req.headers.authorization.split(" ")[1];
        const decoded=jwt.verify(token,process.env.JWT_secret);
        const validated= await roleChecker("Manager",decoded);
        if(!validated|| !decoded){
            return res.status(401).json({
                status:"Unauthorized"
            })
        }
        const Team=req.query.id;
        const TeamData=await User.updateMany({Team},{Team:null});

        res.status(200).json({
            status:"Success",
            TeamData
        })

        next();
    }
    catch(err){
        res.status(501).json({
            status:"Failed"
        })
        console.log(err);
    }

}
const User=require('./../Model/User');
const jwt=require("jsonwebtoken")

exports.createUser=async(req,res,next)=>{
    try{
        const user=await User.create(req.body);
         res.status(200).json({
            success:true,
            data:user
        })
        next();
    }
    catch(err){
        console.log(err);
    }

}
```

```
exports.showProfile=async (req,res,next)=>{
    try{
        const token=req.headers.authorization.split(" ")[1];
        if(!token){
            return res.status(401).json({
                message:"Unauthorized"
            })
        }

        const email=jwt.decode(token,process.env.JWT_Secret);
        const user=await User.findOne({Email:email});
        res.status(200).json({success:true,data:user});
        next();
    }
    catch(err){
        console.log(err);
    }
}
const mongoose = require("mongoose");

const TodoSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true
    },
    description:{
        type:String,
        required:true
    },
    completed: {
        type: Boolean,
        default: false
    }
});

const TaskSchema = new mongoose.Schema({
    Title: {
        type: String,
        required: true
    },
    Description:{
        type:String,
        required:true
    },
    todos: [TodoSchema],
    completed: {
        type: Boolean,
        default: false
    },
    End_Date:{
        type:Date
    },
    Assigned_Date:{
        type:Date,
        default:Date.now
    },
    Assigned_to:String
});

const Task = mongoose.model("Task", TaskSchema);

module.exports = Task;
const mongoose=require("mongoose")

const userSchema=new mongoose.Schema({
    Name:{
        type:String,
        require:[true,"An Employee Must have a name"]
    },

    Email:{
        type:String,
        require:[true,"An Employee must have an email"],
        unique:true
    },

    Password:{
        type:String,
        require:true,
        select:false
    },

    Role:{
        type:String,
        enum:["Employee","Team Lead","Manager"]
    },

    EmployeeID:{
        type:String
    },

    Team:{
        type:String
    },

    TaskId:[{
        type:mongoose.Types.ObjectId,
        ref:"Task"
    }]

})

userSchema.methods.checkPassword=function(password,orginalPassword){
    return password===orginalPassword;
}
```

```
const userModel=mongoose.model("user",userSchema,"user");

module.exports=userModel;
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import './CreateTask.css';

function CreateTask() {
  const [task, setTask] = useState({
    Title: '',
    Description: '',
    End_Date: '',
    todos: [{
      title: '',
      description: '',
      completed: false
    }]
  });

  const navigate=useNavigate();
  const handleTaskChange = (e) => {
    const { name, value } = e.target;
    setTask({ ...task, [name]: value });
  };

  const handleTodoChange = (e, index) => {
    const { name, value } = e.target;
    const todos = [...task.todos];
    todos[index][name] = value;
    setTask({ ...task, todos });
  };

  const addTodo = () => {
    setTask({
      ...task,
      todos: [...task.todos, { title: '', description: '', completed: false }]
    });
  };

  const removeTodo = (index) => {
    const todos = [...task.todos];
    todos.splice(index, 1);
    setTask({ ...task, todos });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const token = localStorage.getItem('token');
      await axios.post(
        'http://localhost:8000/api/v1/task',
        task,
        {
          headers: {
            Authorization: `Bearer ${token}`
          }
        }
      );
      navigate("/showtask")
    } catch (error) {
      console.error(error);
      alert('Failed to create task');
    }
  };

  return (
    <div className="min-h-screen bg-gray-100 flex justify-center items-center">
      <div className="bg-white p-8 rounded shadow-md w-full max-w-md">
        <h2 className="text-2xl font-bold mb-4">Create Task</h2>
        <form onSubmit={handleSubmit}>
          <div className="mb-4">
            <label htmlFor="title" className="block text-gray-700 font-bold mb-
2">Title</label>
            <input
              type="text"
              id="title"
              name="Title"
              value={task.Title}
              onChange={handleTaskChange}
              className="border rounded w-full py-2 px-3 text-gray-700 leading-tight
focus:outline-none focus:shadow-outline"
              required
            />
          </div>
          <div className="mb-4">
            <label htmlFor="description" className="block text-gray-700 font-bold mb-
2">Description</label>
            <textarea
              id="description"
              name="Description"
              value={task.Description}
              onChange={handleTaskChange}
              className="border rounded w-full py-2 px-3 text-gray-700 leading-tight
focus:outline-none focus:shadow-outline"
              rows="4"
              required
            />
          </div>
          <div className="mb-4">
            <label htmlFor="end_date" className="block text-gray-700 font-bold mb-2">End
Date</label>
            <input
              type="date"
              id="end_date"
              name="End_Date"
              value={task.End_Date}
              onChange={handleTaskChange}
              className="border rounded w-full py-2 px-3 text-gray-700 leading-tight
focus:outline-none focus:shadow-outline"
              required
            />
          </div>
          <div className="mb-4">
            <h3 className="text-lg font-semibold mb-2">Todos</h3>
            {task.todos.map((todo, index) => (
              <div key={index} className="mb-4">
                <input
                  type="text"
                  name="title"
                  value={todo.title}
                  onChange={(e) => handleTodoChange(e, index)}
                  placeholder="Todo Title"
                  className="border rounded py-2 mb-4 px-3 text-gray-700 leading-tight
focus:outline-none focus:shadow-outline mr-2"
                  required
                />

                <textarea
                  name="description"
                  value={todo.description}
                  onChange={(e) => handleTodoChange(e, index)}
                  placeholder="Todo Description"
                  className="border rounded py-2 px-3 text-gray-700 leading-tight focus:outline-
none focus:shadow-outline mr-2"
                  rows="2"
                  required
                />
                {index > 0 && (
                  <button
                    type="button"
                    className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4
rounded focus:outline-none focus:shadow-outline"
                    onClick={() => removeTodo(index)}
                  >
                    Remove
                  </button>
                )}
              </div>
            ))}
            <button
              type="button"
              className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
              onClick={addTodo}
            >
              Add Todo
            </button>
          </div>
          <button
            type="submit"
            className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
          >
            Create Task
          </button>
        </form>
      </div>
    </div>
  );
}

export default CreateTask;
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function CreateTeam() {
  const [formData, setFormData] = useState({
    name: '',
    EmployeeID: ''
  });

  const navigate=useNavigate();

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const token = localStorage.getItem('token');
      const res = await axios.post('http://localhost:8000/api/v1/team', formData, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      });
      console.log(res.data);
      if(res.status===201){
        navigate('/show-team');
      }
    } catch (error) {
      console.error(error);
    }
  };

  return (
    <div className="min-h-screen bg-gray-100 flex justify-center items-center">
      <div className="bg-white p-8 rounded shadow-md w-96">
        <h2 className="text-2xl font-bold mb-4">Create Team</h2>
        <form onSubmit={handleSubmit}>
```

```
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="name">
              Team Name
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-
700 leading-tight focus:outline-none focus:shadow-outline"
              id="name"
              type="text"
              name="name"
              value={formData.name}
              onChange={handleChange}
              required
            />
          </div>
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2"
htmlFor="EmployeeID">
              Employee ID
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-
700 leading-tight focus:outline-none focus:shadow-outline"
              id="EmployeeID"
              type="text"
              name="EmployeeID"
              value={formData.EmployeeID}
              onChange={handleChange}
              required
            />
          </div>
          <button
            className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline transition duration-300"
            type="submit"
          >
            Create Team
          </button>
        </form>
      </div>
    </div>
  );
}

export default CreateTeam;
import React, { useState, useEffect } from "react";
import axios from "axios";
import { BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend, ResponsiveContainer
} from "recharts";

const Dashboard = () => {
  const [data, setData] = useState([]);
```

```
useEffect(() => {
  fetchData();
}, []);

const fetchData = async () => {
  try {
    const res = await axios.get("http://localhost:8000/api/v1/task");
    const tasks = res.data.data;

    // Count tasks completed on time by each team
    const completionMap = {};
    tasks.forEach(task => {
      if (task.Assigned_to in completionMap) {
        completionMap[task.Assigned_to] += task.completed ? 1 : 0;
      } else {
        completionMap[task.Assigned_to] = task.completed ? 1 : 0;
      }
    });

    const chartData = Object.keys(completionMap).map(teamName => ({
      teamName: teamName,
      tasksCompletedOnTime: completionMap[teamName]
    }));

    setData(chartData);
  } catch (error) {
    console.error(error);
  }
};

return (
  <div className="flex flex-col items-center justify-center gap-8">
  <p className="font-bold text-2xl my-4">Dashboard </p>
  <div style={{ width: "100%", height: 300 }}>
    <ResponsiveContainer>
      <BarChart
        data={data}
        margin={{
          top: 20,
          right: 30,
          left: 20,
          bottom: 5,
        }}
      >
        <CartesianGrid strokeDasharray="3 3" />
        <XAxis dataKey="teamName" />
        <YAxis />
        <Tooltip />
        <Legend />
        <Bar dataKey="tasksCompletedOnTime" fill="#8884d8" />
      </BarChart>
    </ResponsiveContainer>
  </div>
```

```
    </div>
  );
};

export default Dashboard;
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import { useParams } from 'react-router-dom';

function EmployeeTask() {
  const { id } = useParams();
  const [task, setTask] = useState(null);
  const [allChecked, setAllChecked] = useState(false);

  useEffect(() => {
    const fetchTask = async () => {
      try {
        const token = localStorage.getItem('token');
        const res = await axios.get(`http://localhost:8000/api/v1/task/${id}`, {
          headers: {
            Authorization: `Bearer ${token}`
          }
        });
        const formattedTask = {
          ...res.data.data,
          End_Date: new Date(res.data.data.End_Date),
          Assigned_Date: new Date(res.data.data.Assigned_Date)
        };
        setTask(formattedTask);
      } catch (error) {
        console.error(error);
      }
    };
    fetchTask();
  }, [id]);

  useEffect(() => {
    if (task) {
      const allTodosChecked = task.todos.every(todo => todo.completed);
      setAllChecked(allTodosChecked);
    }
  }, [task]);

  const handleTodoCompletion = async (todoId, completed) => {
    try {
      const updatedTask = { ...task };
      updatedTask.todos = updatedTask.todos.map(todo => {
        if (todo._id === todoId) {
          todo.completed = completed;
        }
        return todo;
      });
      setTask(updatedTask);
      const allTodosChecked = updatedTask.todos.every(todo => todo.completed);
      setAllChecked(allTodosChecked);
    } catch (error) {
      console.error(error);
    }
  };

  const handleSubmit = async () => {
    try {
      const token = localStorage.getItem('token');
      await axios.post(
        `http://localhost:8000/api/v1/task/${id}`,
        task,
        {
          headers: {
            Authorization: `Bearer ${token}`
          }
        }
      );
      setAllChecked(true);
      setTask(prevTask => ({ ...prevTask, completed: true }));
    } catch (error) {
      console.error(error);
    }
  };

  return (
    <div className="min-h-screen bg-gray-100 flex justify-center items-center">
      <div className="bg-white p-8 rounded shadow-md w-full max-w-3xl">
        <h2 className="text-2xl font-bold mb-4">Task Details</h2>
        {task && (
          <div>
            <div className="mb-4">
              <p className="text-lg font-semibold">Title: {task.Title}</p>
              <p className="text-gray-700">Description: {task.Description}</p>
              <p className="text-gray-700">End Date: {task.End_Date.toLocaleDateString('en-
GB')}</p>
              <p className="text-gray-700">Assigned Date:
{task.Assigned_Date.toLocaleDateString('en-GB')}</p>
            </div>
            <div>
              <h3 className="text-xl font-semibold mb-2">Todos:</h3>
              <ul className="list-disc pl-5">
                {task.todos.map((todo, index) => (
                  <li key={index} className="text-gray-700">
                    <div className="flex items-center">
                      <input
                        type="checkbox"
                        checked={todo.completed}
                        onChange={(e) => handleTodoCompletion(todo._id, e.target.checked)}
                        className="mr-2"
                      />
                      <div>
```

```
                        <p>Title: {todo.title}</p>
                        <p>Description: {todo.description}</p>
                      </div>
                    </div>
                  </li>
                ))}
              </ul>
            </div>
            <div className="mt-4">
              <button
                className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4
rounded focus:outline-none focus:shadow-outline"
                onClick={handleSubmit}
              >
                {allChecked ? 'Task Completed' : 'Complete Task'}
              </button>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
}

export default EmployeeTask;
```

Appendix B. User Manual

Introduction

The Task Management System is designed to streamline the process of creating, assigning, and tracking tasks within an organization. This system allows managers to assign tasks to team members, monitor progress, and ensure timely completion. The system includes features for task creation, updating task status, and team management, providing a comprehensive solution for effective task management.

System Requirements

Hardware Requirements

- A computer with at least 4GB of RAM.
- Minimum 500MB of free disk space.

Software Requirements

- Operating System: Windows 10, macOS, or Linux.
- Node.js installed (version 14 or higher).
- Modern web browser (e.g., Chrome, Firefox, Edge).

Installation

Step 1: Clone the repository:

```
git clone https://github.com/yoddha/task-management-system.git
```

Step 2: Navigate to the project directory:

```
cd task-management-system
```

Step 3: Install dependencies:

```
npm install
```

Step 4: Start the server:

```
npm start
```

Step 5: Open your web browser and go to `http://localhost:8000`.

Getting Started

After installation, follow these steps to start using the Task Management System:

Login/Register:

- Navigate to the login page.
- If you don't have an account, click on the "Register" link to create a new account.
- Log in with your credentials.

Dashboard:

- Upon successful login, you will be directed to the dashboard where you can view your tasks and teams.

Features

Creating a Task

- Navigate to the "Create Task" page from the dashboard.
- Fill in the required details:
    - Title: Name of the task.
    - Description: Brief description of the task.
    - End Date: Deadline for the task.
    - Todos: Sub-tasks associated with the main task.
- Click on the "Create Task" button to save the task.

Viewing Tasks

- Navigate to the "Show Tasks" page from the dashboard.
- View the list of tasks assigned to you or created by you.
- Click on a task to view its details.

Updating Task Status

- On the task details page, you can mark the sub-tasks (todos) as completed.
- Once all sub-tasks are marked as completed, the main task will be marked as completed.

Managing Teams

- Navigate to the "Create Team" page from the dashboard.
- Fill in the required details:

    - Team Name: Name of the team.
    - Employee ID: ID of the team members.

- Click on the "Create Team" button to save the team.