EMPLOYEE TASK MANAGEMENT TOOL

A PROJECT REPORT

Submitted in partial fulfillment

of

MASTER OF COMPUTER APPLICATIONS

(MCA)

By

Harsh Gupta

23FS20MCA00002

Under the guidance of

Dr. Pradeep Kumar

Department of Computer Applications

Faculty of Science, Technology and Architecture

MANIPAL UNIVERSITY JAIPUR JAIPUR-303007, RAJASTHAN, INDIA

May 2025

DEPARTMENT OF COMPUTER APPLICATIONS

MANIPAL UNIVERSITY JAIPUR, JAIPUR – 303 007 (RAJASTHAN), INDIA

Date: 10.05.2025

CERTIFICATE

This is to certify that the project titled **Employee Task Management Tool** is a record of the Bonafide work done by **Harsh Gupta (23FS20MCA00002)** submitted in partial fulfilment of the requirements for the award of the Degree of Master of Computer Applications (MCA) in **Department of Computer Applications** of **Manipal University Jaipur,** during the academic year **2024-25.**

Dr. Pradeep Kumar
Project Guide, Department of Computer Applications
Manipal University Jaipur

Dr. Shilpa Sharma
HOD, Department of Computer Applications
Manipal University Jaipur

(*On company letterhead)*

date

CERTIFICATE

This is to certify that the project entitled **Employee Task Management Tool** was carried out by **Harsh Gupta (23FS20MCA00002)** at **GemsNY IT Solutions, Jaipur** under my guidance during **January 2025** to **May 2025**

Mr. Navneet Bhardwaj
Project Manager
GemsNY IT Solutions
Jaipur

ACKNOWLEDGMENTS

I would like to express my heartfelt thanks to the **Dean of the Faculty of Science Technology and Architecture**, for providing me with the opportunity to conduct work on this project.

My sincere thanks to **Dr. Shilpa Sharma, HOD, Department of Computer Applications** for her invaluable support and encouragement throughout the project.

I would like to express my heartfelt appreciation to my faculty guide, **Dr. Pradeep Kumar**, for his guidance, support and thoughtful suggestions throughout the project.

I would like to express my sincere gratitude to my **Project Supervisor**, **Mr. Navneet Bhardwaj**, for his valuable guidance, continuous support and patience during the entire project development process.

I would like to thank the team and mentors at **GemsNY IT Solutions**, especially **Mr. Ratnesh Gupta**, for their overview and insights and for giving me an opportunity to work on real time applications in my internship.

I appreciate the team in all its forms of work, support and guidance, with the gracious support of the **Laboratory InCharge, faculty members and technical staff of the department**, which helped with project development.

Date: 10.5.2025
Place: Manipal University Jaipur **Harsh Gupta**

ABSTRACT

In an era of more collaborative and fast-paced environments, efficiently managing tasks is critical in maintaining productivity, accountability, and timely delivery of projects within organizations. Manual task tracking has led to miscommunication, missed deadlines, and decreased transparency. The intended outcome of the project named **"Employee Task Management Tool"** was to develop a central and scalable system that allows the assigning, tracking, and management of Employee tasks for better coordination and improved operational efficiency.

The study design used a scalable and optimal back-end design of **Nest.js (**a progressive Node.js-based framework) for user authentication, assigning tasks, access by roles, and management of data flows to the front end. This back-end design is then connected to a **Next.js** frontend which has been designed to produce a responsive full stack web application. The back end and front-end connectivity using **RESTFUL API's and WebSocket** support is what has enabled the tool to support r**eal-time task updates** as well as allowing the server to respond to a client request response.

Ultimately, the tool that has been developed enables the real-time tracking of tasks, parent-child task management, and allows for tasks to be filtered dynamically across several variables, including employee, department, and task status. Overall, these outcomes significantly increase task visibility for employees and managers and promote better decision-making and resource allocation in a professional capacity.

The core tools and technologies used in this project will be Nest.js for the backend, Next.js for front-end interactions and presenting, **TypeORM** to interact against a **MySQL** primary relational database. Authentication was done using **JWT** and in order to allow for real-time data syncing we were also able to use **Web Sockets**. These technologies and frameworks were critical to the development of a scalable, secure, functioning task management application.

LIST OF FIGURES

Contents

1. Introduction

Structured systems for managing employees and the tasks they perform have become more critical than before in the fast-paced and digital work environment of today. Every organization-from a startup to an enterprise or even teams operating remotely-has to apply efficient delegation, time management, and tracking throughout productivity processes; this ensures that deadlines are met, weight is evenly distributed, and everyone in a team is accountable. But the available systems are either very inflexible or too complicated or just don't understand how complicated workflows exist inside dynamic teams.

Thus, this project offers solutions in the form of a custom-built Employee and Task Management Tool that would facilitate structured collaboration through role-based access control, departmental structuring, and task lifecycle management. The proposed system reflects real-world organizational hierarchies and workflows where different roles-such as Admins, Assigners, and Assignees-will interact with tasks based on specific permissions and responsibilities. Moreover, it will include detailed features like daily time tracking, task status progression, commenting, and automated timeline calculations, resulting in an extensive yet scalable and secure collaborative environment.

The front end developed using Next.js will be just the element to show a prototype for testing interaction and flows. However, the focus of most development will be toward creating robust backend architecture using Nest.js. The backend will concern managing integrity, security, and authentication and logic regarding user roles, departments, and task processes. It will be modularly extensible and scalable for future developments toward things like analytics dashboards, real-time notifications, and mobile support.

1.1 Introduction into Work Done / Motivation

Overview

During the early phases of planning this project, a gap was identified in the existing tools used by mid-sized teams, educational institutions, and small enterprises. Many such teams are dealing with fragmented systems—one tool for time tracking, a second one for task assignment, and a third one for communication. The disjointed nature of these systems reduces productivity at best, with an obvious loss of accountability and higher overhead in managing even simple workflows.

The project was done to address these concerns by bringing together employee management with task handling into a unified platform. The idea was to build a system where every user can be defined with a clear role, belongs to one or more departments, and interacts with tasks based on fine-grained permissions.

The solution incorporates several real-life functions:

- Cross-department task creation and assignment
- Logging hours against tasks for effort tracking
- Updating statuses and monitoring progress
- Inline comments to facilitate team interactions
- Role-restricted access for data integrity

Motivation

The leading motivators behind the project stem from organizational needs:

- **Lack of Customization in Existing Tools:** Customization is often needed in tasks that do not afford such luxuries. These platforms follow a "one-size-fits-all" model that does not scale well with customization along role and department divisions.
- **Role Wise Need for Flexibility:** Role-wise differentiations in access levels among users are often needed within organizations. Admins need global access, while assignees only need access to tasks they are involved in.
- **Accountability and Time Tracking:** Supervisors wanted to see how much time is spent on tasks, while employees wanted an examination of deadlines and deliverables.
- **Simplification Through Technology:** The team is seeking to use Nest.js and Next.js and implement such technology to facilitate a simple yet scalable development.

Applications

The Employee and Task Management Tool may be used applicable in any number of sectors and teams including:

- **IT Teams:** To manage software development tasks across sprints and departments.
- **Educational Institutions:** For managing tasks allocated to faculty and administrative staff.
- **Startups:** An internal tool to manage a small team of people whose roles have clearly defined boundaries.
- **Remote Freelance Groups:** Who have a need for structured delegation and progress tracking but can hardly afford complex enterprise tools.

Advantages

Among many others, some key advantages of this system are:

- **Scalable Role-Based Access Control (RBAC):** Permissions that are tightly bound to user roles and automatically enforced through backend guards.
- **Translucent Task Lifecycles:** All active state tasks embody start dates, estimating timelines, status, and completion logs.
- **Effort Logging:** Employees can log daily hours with visibility to be availed for performance reviews and project estimates.
- **Staying Contextually Collaborative:** Comments remain attached to tasks, assuring that all conversation remains contextual and traceable.
- **Backend-First Design:** The system is designed to allow for subsequent integrations, e.g., mobile apps, analytics, or automation, while requiring negligible change to the core logic.
- **Departmental Organization:** All user and task data are organized department-wise, to aid large teams with multiple functional groups.

1.2 Project Statement/Objectives of the Project

Project Statement

The project aims to design and implement a robust, scalable, and modular Employee and Task Management System, which emulates real-world organizational functioning, structured hierarchically controlled role-based and department permissions and dynamic task management processes. This application will enable actual-performing users, in real-time as per designed roles-usually Admins, Assigners, and Assigns-early, deliberate enforcement with separation of permissions but at the same time help the one-stop task tracking and collaboration to check.

It is meant to:

- Organizes in a controlled, safe, and structured manner organizations regarding task assignment and management.
- Accountability through detailed timelines regarding tasks and time logging on a day-to-day basis.
- Commenting function allows collaboration on tasks.
- Increased transparency and clarity regarding performance of tasks involving several users.
- While being backend-heavy, the system makes use of Nest.js for core logic, security, and feature handling, while a quite simple Next.js prototype serves as the testing and interactive layer for the frontend.

Objectives of the Project The following are the detailed and specific objectives that this project hopes to achieve:

1. ***Role-Based User System A system will be developed such that every user can hold one of the following roles:***

   Admin:

   - Can manage all users, assign roles, create departments, and view all system data.
   - It has global access across all departments.

   Assigner:

   - Can create and assign tasks to Assignees within the same department.
   - He has permission to view all tasks in his/her department, leave comments, and review time logs.

   Assignee:

   - Sees and modifies assigned tasks only.
   - Can change task status (e.g., "In Process" to "Finished"), log time against the task, and comment on it or leave a note besides that.

   It will include stringent enforcement of these permissions by role guards and middleware at the backend level to prevent unauthorized access.

2. ***Departmental Structuring Each user has at least one department.***
   - Departments are the boundaries within which user interaction takes place; no task is assigned and visible outside of that boundary unless a user has Admin rights.
   - User A is associated with Department A and Department B but can do an action only under Department A and not Department B.
3. ***Task Metadata***

   All tasks have the following metadata:

- Title and Description
- Department and Assignee
- Start Date
- Estimated Time (in days/hours)
- Computed End Date (Start Date + Estimated Time)
- Actual End Date (gets noted when task status is marked as Completed)
- Status:
  - Pending (Created but has not been started)
  - In Process (Work has started)
  - Completed (Work has finished and verified)
- Task transitions are logical and validated at the server side to prevent wrong status jumps.
- Daily Task Time Tracking Assignees will log the hours spent on a task on any given day.
- Time logs are linked to a specific date and saved within a database for future analysis.
- Admins and Assigners have access to time logs for performance evaluation.
- Commenting System Each task features a comment thread visible to relevant users (typically assigner and assignee within the same department).
- Timestamped and stored for audit and collaboration purposes.

4. Secure Authentication and Authorization The JWT-based token system ensures secure user authentication.
5. Backend APIs are protected with authentication guards.
   - Role and department checks are applied on endpoints to ensure that users access only their permissible data.
   - Modular Backend Architecture The backend is developed using Nest.js and adopts a modular pattern in implementation.
   - Thus, every feature (users, departments, tasks, comments, logs) is implemented as a separate module with isolated services, controllers, and guards among them.
   - It's easier for testing, debugging, and future enhancements.
6. Frontend Prototype using Next.js A basic frontend is developed to:
   - Allow users to log in
   - View their dashboards and assigned tasks
   - Update task status
   - Enter time logs and comments
   - The frontend communicates with the backend through RESTful APIs.
   - Scalability and Extensibility The system was made and designed to be extensible in the part of future additions, such as:
   - Email or notification services
   - Analytics dashboards
   - Role customization
   - Integration with third-party platforms (e.g., Slack, Google Calendar)

1.3 Organization of the Report

The Employee Task Management System has traveled a long way from inception to final development, and to provide an orderly approach to providing the development process in this final report, each chapter offers a phase of the project throughout its development. In this report, there are separate chapters to portray each stage in the life of the project from the initial idea of the system through to implementation and evaluation. The intent of the approach taken was to take the reader step-by-step through the rationale, planning, designing and developing to completion of this system.

The first chapter, Introduction, introduces the reader to the impetus behind the project, and specifies the problem that the system attempted to address, and the origins of the objectives to determine the attributes and functionality within the system. The full chapter also incorporates a description of the report components that made up the entire report, so the reader understands how the project report summarizes each of the chapters that proceed in the report.

Next, the Survey of Technologies are descriptions of the tools, frameworks and technologies that we were considering when developing the ETMS, and the tools that were selected as part of the ETMS development process. It also explains the reasoning behind selecting the tools, including React for the front-end, Node.js for the back-end, and certain databases like MongoDB or MySQL for data storage, and how the tools are to build components identified by the plan to provide performance and scale.

Difference between requirement analysis and design the next chapter, Requirement Analysis, looks at the system's functional and non-functional requirements, as well as the needs of the users (roles) across three of the user roles: Admin, Assigner, Assignee, and describes their expected inputs and outputs and boundary constraints. The requirement analysis set a foundation that led to design decisions throughout the other chapters.

System Design outlines the architectural structure used for the ETMS. This chapter provides data flow diagrams (DFDs), entity-relationship (ER) diagrams, and diagrams that break down components of the ETMS and illustrate in detail how the pieces fit together. This section also concludes that development work was conducted with a plan that used foresight.

Next, the Coding chapter specifically explains the technical implementation of the system. Key elements of the codebase will be highlighted with examples to show how the logic works, and breakdowns will help show how the theoretical design was later translated into a usable application. Challenges that were met during development implementation will be noted.

Following the coding chapter, the Testing section describes how the system was tested against the requirements previously defined. There are multiple levels of tests that occurred including a variety of levels: unit tests, integration tests, and system level tests and this noted the intended reliability, security, and usability when the platform is in use.

The report then proceeds with Results and Discussion, in which the outcomes of the project will be discussed. There are performance indicators, screenshots of the working application, and user feedback, if applicable. In this section of the report a thorough evaluation of the ETMS will be critically reviewed to identify if the aims of the project were achieved.

Then in the last sections, Conclusion and Future Scope, we can reflect on what has been achieved with the project and consider future improvements, upgrades and features that could be integrated into future

versions. These sections demonstrate the limitations of the current system but allow us to remain positive about future development.

The report fulfils this organization structure because it outlines a journey to build the Employee Task Management System and allows for reading clarity of readers from a technical review perspective, as well as an academic review perspective, while ensuring the relevant context and insights comply to support the development.

2. Background Material

2.1 Conceptual Overview (Concepts / Theory Used)

The Employee and Task Management Tool draws its basis upon certain fundamental theoretical and architectural concepts that guide the behavior of modern digital systems dealing with people, processes, and information. These concepts shape not only the behavior of the tool but also make it secure, scalable, and satisfactory to the needs of the real organizational world.

Central to this system is a trusted security theory called Role-Based Access Control (RBAC), which has been and still remains a hot security topic pertinent to enterprise systems. Simply put, RBAC guides the assigning of typical user roles: an Admin role (with full controls over users and departments), an Assigner role (to assign tasks), and an Assignee role (to view and update their assigned tasks); where each such role has a set of generic and set permissions. Simply put, the user does an abstraction, separating duties, and minimizes the risk of unauthorized actions. These rules are tied in the system/programmatically, such as an Appliant being able to work with user data only while the admin is fast asleep. This way, the very interface used to apply these rules closely resembles the real organizational landscape that separates the data from the eyes of unauthorized people.

Another very important theory realized through the application is called the Task Lifecycle Model. This deals with how tasks are said to progress over time, from being initiated (Pending), to say being worked on (In Progress), and finally to being completed (Completed). Each one of these transitions has significance as tangible effort and decisions by the user are reflected in these transitions and are often coupled with initiating automatic processes—for example, the actual end date may be updated when the task is marked as “Completed”. By attaching further metadata of start date, estimated duration, calculated end date, and actual completion date to the task, the system allows moving beyond just observing project timelines or delays and puts itself in the position of appraising individual performance.

The concept of tracking time as it applies here is also very important. This development encourages users to record, every single day, the hours they actually spent completing the assignment in question, instead of merely allocating a task. This task model is based on project management and billing systems for consultancy and IT services, wherein such transparency in work logs directly imposes the basis for invoicing and progress measurement. In our case, these daily logs are connected to tasks, departments, and users, empowering managers to thus analyze workloads, calibrate expectations, and distribute work accordingly and fairly.

The principle of modularity in software design is another concept followed in this tool development. Basically, following clean architecture, each functionality (user management, task management, commenting, time logging, etc.) is developed as an independent module on the backend. This way maintainability, parallel development, and the ease of extending the system in further releases are achieved.

The last model guiding the whole communication at the backend is RESTful API design. REST (Representational State Transfer) provides an application programming interface through which the frontend and backend are meant to cooperate using HTTP methods (GET, POST, PUT, DELETE). Therefore, every user action, be it logging in, creating a task, or posting a comment, is mapped to a corresponding API route with clear expected inputs and outputs. Hence, it guarantees that the system is predictable, testable, and easily integrated with any front-end or third-party service.

By integrating these concepts-RBAC-task lifecycle modeling-time logging-modular design-RESTful API communication-the project creates a system that is technically sound, yet capable of being used practically across a host of organizational settings.

2.2 Technologies involved

In building a fully-fledged task and employee management tool, it is necessary to be really careful while choosing the stack of the technology to pepper with scalability and performance as well as ease of development. In this project, the core technologies include Nest.js for the back end and Next.js for the front-end prototype. Supplementary tools such as MySQL, Prisma, JWT, and TypeScript together create a robust-maintainable system.

Nest.js forms the backbone of the application. Progressive Node.js framework built with TypeScript and directed heavily towards Angular. Nest.js comes with a high degree of support for modular architecture, dependency injection, and decorators, which makes it greatest for this project. Each feature implemented in the system, for example, authentication, task creation, comment posting, becomes its own module with dedicated services, controllers, and guards. This allows having cleaner code which is easier to scale and test. One of the other sources for keeping the project clean is the facility by which Nest.js allows one to easily put up custom middleware and guards, which will be extensively used in this particular project to enforce authentication and role-based access.

Next.js is the chosen frontend technology, a very widely used React framework that supports server-side rendering and the generation of a static site. In this project, the frontend is not heavily developed because it just serves as a test and demonstration layer. Nevertheless, Next.js was chosen because of its performance and flexibility. Thus, with Next.js, it becomes easier to create pages that load quickly, implement routing, and cookie-based authentication with the backend APIs.

For data storage, there is MySQL-an open-source relational database known for its stability and known for not rejecting complex queries. MySQL was preferred on NoSQL alternatives because the relations between entities, i.e. users, roles, tasks, departments, comments, and time logs, are really structured and relational by nature. For instance, many tasks belong to departments to which they are assigned to users and include time logs and comments-all of which map well into SQL tables with foreign key constraints.

Prisma ORM (Object Relational Mapping) is used to interact with the database. At the same time, Prisma is a very elegant and type-safe way of querying and manipulating data in MySQL. Since Prisma auto-generates TypeScript type definitions and synchronizes schema, it reduces runtime errors and enhances developer productivity. In addition, it greatly simplifies the most common features like pagination, filtering, and relational queries, which both are commonly required in this project.

Authentication and session management are handled through JSON Web Tokens (JWTs). The flow is that once a user logs in, a JWT is created and sent back to the front end for future API calls. On the back end, the token is validated, and payload decoding takes place by Nest.js guards to assure that the user is authorized to act on the requested operation. JWTs provide a secure as well as a stateless way of managing sessions without server-side storage.

The entire project is written in TypeScript, the strictly typed superset of JavaScript. With TypeScript, the quality of code improves with the through compile-time error checking and forces consistent interfaces across modules, enabling the developers to work more confidently with complex data models.

All these technologies form such an integrated modern stack as Nest, Next, Pg, Prisma, JWT and TypeScript. They keep the application secure and manageable, waiting for future expansions like changing into real-time notifications, a mobile app, or advanced analytics.

2.3 Present Systems / Literature Survey

In the recent past, the arena of software solution related to management of employees and managing tasks has innumerably proliferated in different business commensurate to their requirements and need. Popularly used tools like Jira, Trello, Asana and Monday.com offer variations in task tracking and project management capabilities along with employee management in terms of human resource management systems (HRMS) like Zoho People and BambooHR, maintaining the employee records along with leave tracking. Yet though quite a few people have adopted these systems, limitations persist on their use around applicability regarding custom, role specific, and department segmented environments-especially for small and growing organizations that are looking for greater flexibility and control over workflow logic.

For instance, Jira is an excellent incorporation into teams bringing power to software development, addressing issue tracking, sprint planning, and Kanban boards. The catch here is that, for non-technical teams, the structure of Jira is way too strict and also its cost mounts up very fast when teams swell. Also, customizing permissions and departmental structure involves a steep learning curve within Jira.

According to their overall nature, usability, and simplicity of interfaces, Trello and Asana provide methods to generate a simple task board and to plan time. However, they lack defined role-based access control in a more refined manner out of the box. For example, tasks can be assigned to users, but the structuring of complex role hierarchies (an assigner can assign but not complete tasks) or organization-specific rules like automatic end date calculation based on estimated time is often lacking.

The academic literature on task and workflow management systems is primarily on process automation, doing productivity analysis, and access control. Role-Based Access Control (RBAC) can be defined, as by Sandhu et al. in their historic paper (1996), for the establishment of an approach towards access systems that are both secure and scalable. In this model, permissions are acquired by users through roles instead of a direct assignment, which reduces administrative overhead and increases security. This is a funda to produce the intended design of the tool.

Another area of study has been extensive modeling of the task lifecycle in the BPM literature. The normal life of a task-initiation, planning, execution, monitoring, and closure-is reflected in our system through the status flow of "Pending" to "In Process" to "Completed."

Time tracking, an index of productivity and accountability, has been a rich theme for management science. Many studies have set up that recording time frequently improves ownership of tasks, reduces delays, and enhances resource planning. Implementing this concept in the task system will enable the manager to analyze the effort versus output and indicate how an individual has performed on a user-specific basis.

In summary, even though existing systems have matured and rich features, they still lack customizability, require a paid tier for a lot of their baseline features, and do not fully support finer degrees of control over tasks, users, and departments. The project conceptualizes the theories identified within existing academic research and commercial systems yet transforms them into a lightweight, extensible, and role-aware solution appropriate to a wide range of organizational contexts.

2.4 Feasibility Study

The more critical component of developing the Employee and Task Management Tool was the conduct of a feasibility study within which the rounded view of the idea's technical, operational, or economic feasibility was developed within the scope of a college undertaking or an internal organization project. All aspects examined during this feasibility study helped in ensuring that the system could be developed and deployed without exceeding time, budget, or resource limitations.

A. Technical Feasibility

From a technical point of view, it seems entirely possible that the proposed system would be feasible. The technology stack chosen-Nest.js, MySQL, Prisma, Next.js, and TypeScript-is widely used in modern web development and has a strong support community behind it, along with rich documentation and proven instrumental performance. Most of the major key concepts can be implemented with standard backend patterns and RESTful APIs, including user roles and task metadata, a comments system, and time logs. Authentication using JWT tokens is also adopted as a very secure method for session management.

Cloud-based hosting such as Render, Vercel, or Heroku streamlines deployment. The above means that the technical environment required to develop and run the application is well within the infrastructure available.

B. Operational Feasibility

An easy-to-use structure tailored to role-specific interfaces and intuitive task workflows established a minimum confusion among users with clear demarcation of responsibilities among Admins, Assigners, and Assignees. Familiar patterns from well-known productivity tools, for example, task status updates, comment threads, and time logging, serve to ease the training or onboarding of users.

Operational processes like email notifications and app support can be integrated in the future without interrupting the current system.

C. Economic Feasibility

Since the project builds with open source tools and manages its infrastructure privately, it generates no direct charges for licensing or usage. Usually, the only effort needed for development is time, without a need for costly hardware or commercial APIs. Some extra bells and whistles are not added on the frontend, so a backend-first approach is good practice. All in all, this project is considered economically feasible for a student or small-team environment.

D. Schedule feasibility

The project was done in phases-getting requirements, designing, developing the backend, prototyping the frontend, and testing. The timeline suited the academic timeline well, and the modular aspect of development enabled separate features to be completed and tested independently. The result was good progress without seeming already entrenched delays in the project.

The above considerations indicate that the feasibility study completed has opened great potential for real-world uses and future enhancement in the project.

2.5 System Requirements

The successful implementation and use of the Employee and Task Management Tool depends fundamentally on understanding functional and non-functional requirements. These are key parameters guiding the design, development, and testing stages of the system so that it adequately caters to all user requirements.

A. Functional Requirements

- User Authentication and Role Assignment
  - The users should be able to register, log in, and log out securely.
  - Admins can assign roles (Admin, Assigner, Assignee) on registration.
  - For securing any session, JWT-based authentication will be used.
- Department Management
  - Admins can create and manage departments.
  - Users can be assigned to many departments.
- Task Creation and Assignment
  - Assigners can create tasks and assign them to users belonging to the same department.
  - Tasks must have fields for title, description, start date, estimated time, and status.
  - End date calculation is done automatically with actual end date capture upon completion.
- Task Workflow Management
  - Tasks must allow status transitions: Pending → In Process → Completed.
  - Only authorized users can change their status.
- Time Logging
  - Assignees shall log time for a task on specific dates.
  - Logs will be stored so that Admins and Assigners can see them.
- Commenting System
  - Users must add comments to tasks they are involved in.
  - Comments would be visible to relevant users in the same department.
- Role-Based Access Control
  - Access to actions and data must depend on user role and department.

B. Non-Functional Requirements

- Performance
  - In normal load conditions, the system should respond to API requests in a few hundred milliseconds.
- Security
  - It must hash passwords securely (e.g., using bcrypt).
  - JWT token end ought to be appropriately implemented.
  - APIs should impose access control and sanitize input.
- Usability
  - UI must be simple and role specific.

    - Task workflows and logging will require minimum training for users.
- Scalability
    - Backend must adopt feature-wise growth in the future and for users.
    - With a modular structure, straightforward addition of new modules will be possible.
- Maintainability
    - The system must adhere to clean code and documentation practices.
    - Code needs commenting, and API contracts need documentation using Swagger or similar tools.

With these requirements, the system is poised to eventually offer all users a meaningful, secure, and extensible experience.

3. Methodology

3.1 A Detailed Methodology Adopted

The approach adopted was stepwise based on live methodologies of real software engineering methodologies to ensure clearness, modularity, and completeness of Employee and Task Management Tool in this report. The flow essentially followed the layered and iterative approach, with feedback and validation at every end of a phase. A really elaborate walkthrough of the adopted methodology is listed hereunder:

a). Requirement Gathering and Planning for the System

The initiation of the continuing process of development was an in-depth requirement study and visualization of what an enterprise organization typically would need the system to support. Included would be imagining what would be the behavior of the Admins, Assigners, and Assignees for all these tasks and how they would interact within such a structure concerning departments and what kinds of permissions they should hold. Very careful distinction of functional and nonfunctional requirements was made. Functional requirements are assigning task management status comments, and time logs. Nonfunctional requirements characterized performance, scalability, and security.

b). Design of Application Architecture

The next step after the requirements were finalized is system design. The application was split into modules for separation of concerns, enabling maintenance. The core functionality of the system was divided into five modules: User Management, Department Management, Task Management, Time Tracking, and Commenting. A Nest.js module would be implemented for each of these modules, holding its controller, service, data model, and routes.

Role-based access control was also heavily considered in this phase. We created role-specific access policies enforced by backend guards. The main permits only the validation action (e.g., Assigner can assign tasks but not change task status). Clean layered architecture was adopted to achieve separation between business logic and data access/request handling.

c). Back-End Implementation using Nest.js

The majority of the development effort was focused on building a robust backend using Nest.js. This phase comprised building controllers for handling HTTP requests, building services to tie application logic, and employing Prisma ORM to map almost everything to a database behind the scenes. The features were developed incrementally in the following order:

User registration, login and role assignment in JWT-based authentication.

Next-in-line department creation and user-department mapping.

Then task creation, assignment, and status keeping.

Finally, time logging, alongside tracking tasks, and a function to comment further enables collaborative working.

Modular coding was also emphasized considering that each feature was to work isolated and be testable or updated without affecting others.

d). Frontend Prototype Development

In the backend focus, a prototype interface was developed using the Next.js framework, which is minimalistic but allows the users to perform registration, login, see their tasks, and update task statuses. This prototype was used as an engine to prove the backend APIs and give end users a peek into future experiences.

e). Integration, Testing, and Validation

After the successful implementation of the core features, extensive and thorough testing was done. Various test users were created with various roles assigned to departments and run through every task-flow creation, assignment, status update, comment, and time tracking. Edge cases such as invalid token access or status updates with the wrong role were tested to assure that guards and validations were working as expected.

f). Documentation and Reporting

DB schema preparation, writing setups, documents APIs, and preparing diagrams like ERD and DFD constitute the final phase. Code was reviewed and commented so that future developers or contributors could go through it and have a clear understanding for easy maintenance of the project. By this method, the system was made functional, secure, extensible, and easily understood, among other things.

3.2 Data Flow Diagrams (DFD)

The Data Flow Diagram (DFD) is a structured modeling tool designed to represent the flow of information within a software system. In our project, Employee and Task Management System, DFD helps really a lot to understand how data flows among users, internal modules, and storage systems in the system. Thus, we can say that it is a blueprint to the logical design of the system.
This system is designed to support three types of users (Admin, Assigner, Assignee), such as task assignment, time tracking, status update, and collaboration through comments. The DFD explains the interaction between these user roles and how data is processed internally amongst various components of the system.

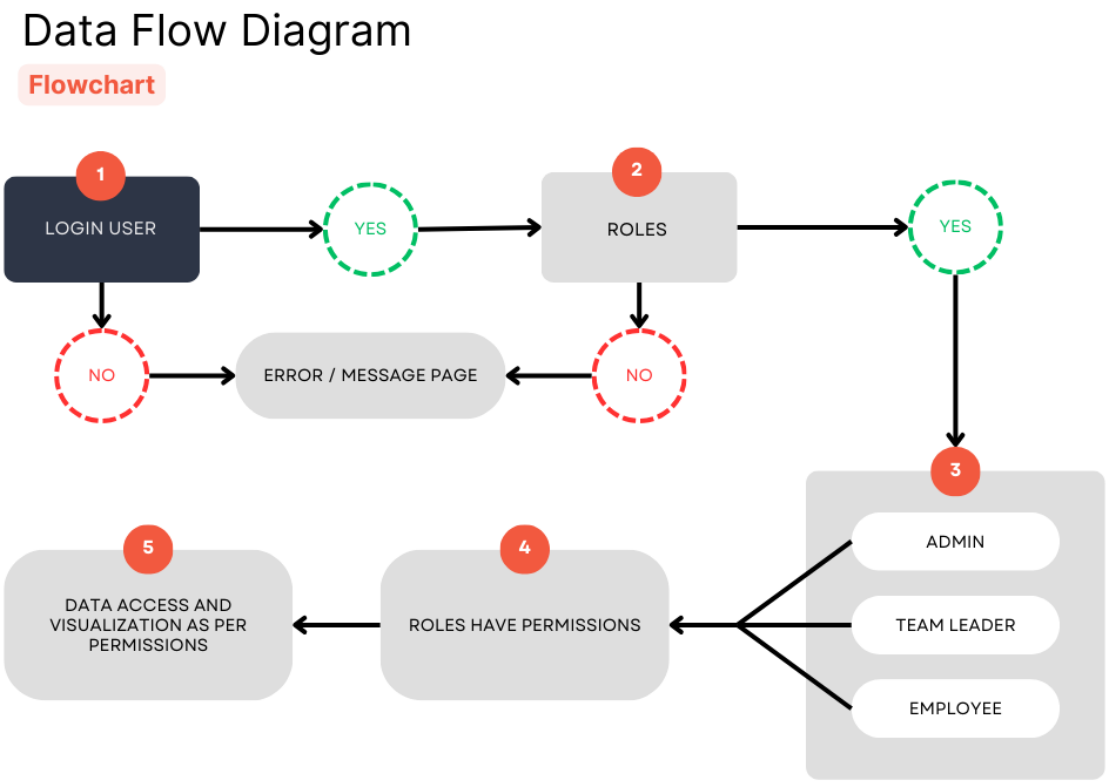

Fig.3.1 - Data Flow Diagram

External Users (Entities)
Admin:

- The admin is the highest authority in the system.
- They are responsible for creating departments and user accounts.
- Admins can assign roles to users and manage all the data in the system.
- They also have access to view analytics such as progress reports and task summaries.

Assigner (Team Leader):

- An Assigner is a user with permission to create and assign tasks.
- They can view the list of assignees in their department and assign tasks accordingly.
- They track task progress and check time logs of assignees.

Assignee (Employee):
- An Assignee is a user who receives tasks from Assigners.
- They are responsible for updating their assigned tasks status (Pending → In Process → Completed).
- They also log their daily working hours and can add comments to each task to better facilitate communication and clarity.

Core Processes in the System

User Authentication:
- All three types of users have to log in by providing their own credentials.
- The system verifies the entered credentials with those stored in the User Database.
- If authenticated, the system generates a secure session token.

Department Management:
- Admins can create new departments and assign and manage users within them.
- These records are stored in the Department Database.

Task Creation & Assignment:
- Assigners create tasks by specifying the title, description, estimated time, and assignee.
- The system records the task's start date.
- The estimated end date is calculated automatically based on the estimated time.
- Each task is associated with a defined department and user.
- The database stores all information about task-related activities in the task-database.

Execution of the tasks and status updates:
- The assignees update task statuses during execution: Pending → In Progress → Completed.
- The actual end date is automatically recorded by the system on completion.
- With such updates, users get to compare planned vs actual completion time.

Daily Time Logging:
- Each assignee logs, on a daily basis, the number of hours worked on a particular task.
- Such entries are submitted through the interface and stored in the Time Log Database.
- This time-tracking helps in evaluating individual performance and in workload monitoring.

Commenting System:
- Users (mainly Assignees and Assigners) are allowed to comment on any assigned task.

- This provides more communication and clarity.
- Comments have timestamps and are stored in the Comment Database.

Data Retrieval and Reporting:

- Admins and Assigners can obtain detailed reports such as:
- Task completion progress
- Team member's time logs
- Comments and task history

These reports are dynamically generated based on live data from all databases.

Data Stores Used in the System

User Database:

- Stores information about each user including name, email, password (hashed), role, and department associations.

Department Database:

- Contains details of each department including name and description.
- Helps in grouping users and organizing tasks department-wise.

Task Database:

- Stores all information related to tasks such as title, description, assigned user, status, start date, estimated and actual end date.

Time Log Database:

- Records daily working hours logged by users for each task.
- Includes task ID, user ID, date of entry, and number of hours.

Comment Database:

- Stores all task-specific comments made by users.
- Contains user ID, task ID, comment message, and timestamp.

Summary of Data Flows

- Initiate the creation of department and user accounts by an Admin, after which the system processes and stores the data into Department DB and User DB.
- Creators and Assigner, Create or assign task → Store in Task DB → send notification as task data to respective assignees.
- Assignee logs in, views their task, and starts working on it.
- During the process of executing tasks, Assignees update task status, record their daily working hours, and make comments against their specific tasks → and this data is stored in Task DB, Time Log DB, and Comment DB respectively.

- Admins and assigners access all databases to create reports and know the progress of tasks.

Major Advantages of using DFD in this Project:

- It gives a clear orientation in understanding the logic of the system.
- Visuals how each module and user role interact.
- Also assist in finding the bottlenecks and optimizing system flows.
- It will serve as a reference when backend development and database design are done.

3.3 Entity Relationship Diagram (ERD)

An Entity Relationship Diagram is a basic conceptual blueprint of the database structure of a software system. The ERD describes the main entities the software manages- meaning the tables, their attributes (through columns), and the relationships between them. While this project also involved an ERD, it concerned the structure and linkages of employees, departments, tasks, time logs, and comments from the backend.

Our Employee and Task Management System involves different user roles such as Admin, Assigner, by Department, assigning tasks, keeping a progress track, how many hours each user logged in his tasks, and communicating through comments. The ERD is designed to perform these operations and ensure that data consistency, normalization, and scalability are achieved.

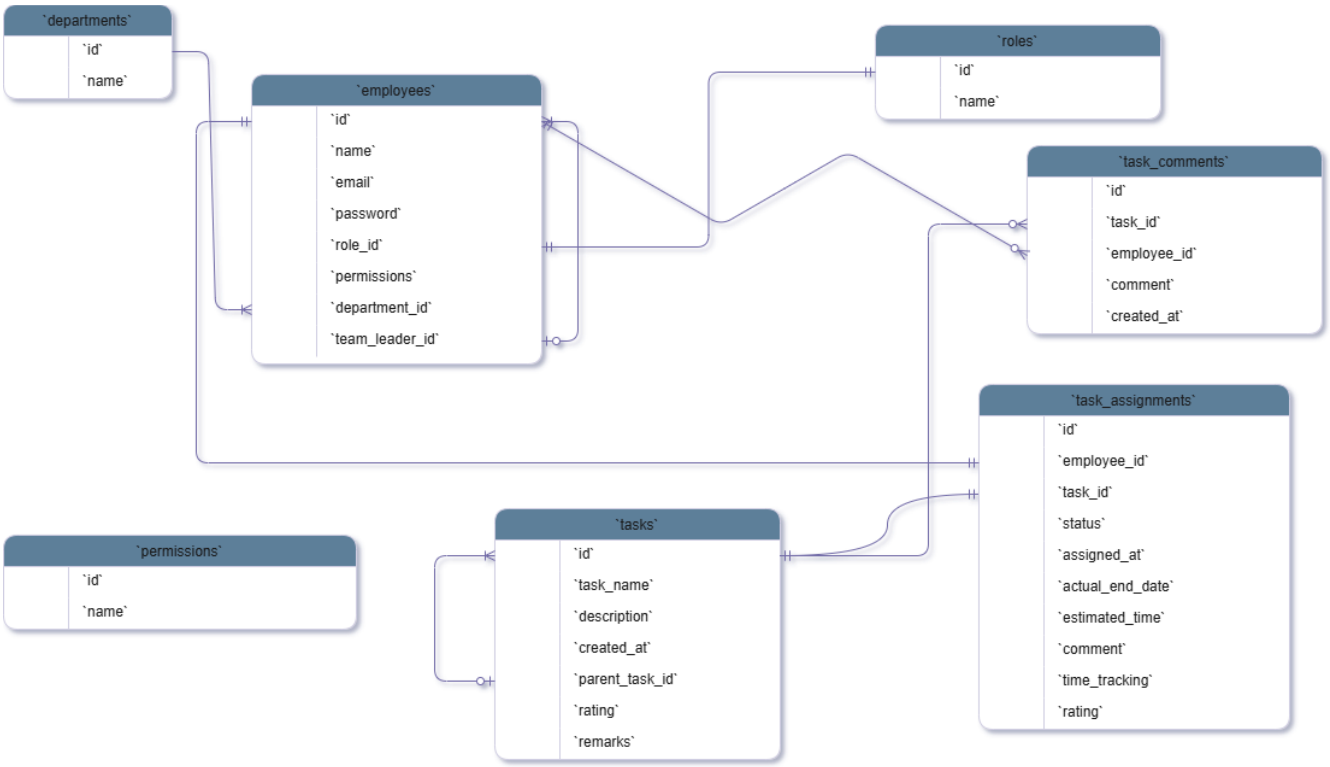

Fig.3.2 - ER Diagram of database

Core Entities & Their Attributes

User: All people interact through the system under this User entity.

- user_id (Primary Key)
- full_name
- email (unique)
- password_hash
- role (ENUM: 'admin', 'assigner', 'assignee')
- created_at
- updated_at
- Department
- This entity contains data about departments within the organization.
- department_id (Primary Key)
- department_name
- description
- created_at
- updated_at

UserDepartment: As a user can belong to many departments, a department can have many users; thus, a many-to-many relationship exists between User and Department. This was realized by way of a junction table:

- user_department_id (Primary Key)
- user_id (Foreign Key → User.user_id)
- department_id (Foreign Key → Department.department_id)

Task: Every task in the system will be assigned by an Assigner and worked on by the Assignee.

- task_id (Primary Key)
- title
- description
- status (ENUM: 'pending', 'in_process', 'completed')
- estimated_time (in hours or days)
- start_date
- calculated_end_date (start_date + estimated_time)
- actual_end_date (automatically filled when task marked completed)
- created_by (Foreign Key → User.user_id) - The Assigner
- assigned_to (Foreign Key → User.user_id) - The Assignee
- department_id (Foreign Key → Department.department_id)
- created_at
- updated_at

TimeLog: The times logged generally represent the everyday expenditure of time by an assignee when working on a particular task.

- time_log_id (Primary Key)
- user_id (Foreign Key → User.user_id)
- task_id (Foreign Key → Task.task_id)
- log_date
- hours_logged

  Comment: The Comment entity records all the communication that happens regarding specific tasks, thereby allowing users to be involved in discussions about task progress.
- comment_id (Primary Key)
- task_id (Foreign Key → Task.task_id)
- user_id (Foreign Key → User.user_id)
- message
- Timestamp

Relations Between Entities

- A User can belong to multiple Departments, and each Department can have multiple Users → many-to-many relationship hence handled by UserDepartment.
- A User (as Assigner) can create multiple Tasks, and a User (as Assignee) can have multiple Tasks assigned to him → two one-to-many relationships from User to Task.
- Each Task is assigned to a department → many-to-one.
- A Task can have multiple Comments → one-to-many relationship.
- A Task can have multiple TimeLogs → one-to-many relationship.
- A User can create many Comments and many TimeLogs → one-to-many relationship from User to Comment and TimeLog respectively.

  Normalization & Design Considerations
- All tables are normalized to 3rd Normal Form (3NF)—no redundancies, atomic fields, and transitive dependencies eliminated.
- Primary and Foreign keys shall be defined in order to keep Referential Integrity.
- Passwords shall be stored in hashed form (never plain text).
- While using the ENUM fields for role and status, invalid values are rendered impossible to have values.
- Created_at and updated_at timestamps are done to track changes made to record.
- With the junction table, which is UserDepartment, flexibility and scalability for multi-department support applications are assured.

Sample Use Case Flow (for better understanding of relationships)

- The admin creates Department A and includes in its User X (Assigner) and User Y (Assignee).
- User X creates Task T1, assigns it to User Y, and sets the estimated time to 5 days.
- Task T1 will be kept in the Task table, having linked in with Department A; assigned_to = Y; created_by = X.
- If User Y has started working, he will log in hours spent doing so with the status being changed to "in_process"→ all of these entries will be kept in TimeLog.

- User Y also leaves comments for clarification→ it will be shelved in Comment table.
- Once the task has been completed, status will be assigned to "completed", and actual_end_date will then auto-fill.

4. Implementation

The implementation phase is where design and theoretical understanding are converted into practical application. The project emphasized developing a robust and scalable backend in Nest.js while maintaining a lightweight frontend interface with Next.js. This chapter elaborates on how the system was implemented and brought up through a set of interrelated modules, development, and testing of the prototype, role-based functionality, task lifecycle, and API integration in testing.

4.1 Modules

According to the Nest.js design, the backend has developed in a modular way. Each module depicts a particular domain of logic and interacts with other modules through services and controllers.

Authentication Module:

Authentication was done using JSON Web Tokens (JWT) and password hashing (using bcrypt). At login, a JWT is provided to the client, and the token is then used to access all protected routes. Guards are put widely in the routes of the API to prevent unauthorized access.

Key Features:

- Register users and login
- Password hashing
- JWT-based access control
- Guards for role and permission validation

User Module

The user module takes care of user and role management. Three roles, Admin, Assigner, and Assignee can each be assigned to a user, who also must belong to at least one department.

The major functionalities include:

- Creation and update of user profiles
- Assign role and department
- Retrieve users for a specific department or role

Department Module

Departments are necessary for defining the organizational structure. The admins can create, modify, and delete departments. Every user should belong to at least one department so that tasks and permissions can effectively be scoped in the system.

The module provides for:

- Creating and deleting departments
- Assigning users to departments
- Checking for departmental memberships while assigning tasks

Task Module

This is the major module in the application. Assigners create tasks and assign them to users in their department. Every task is represented with a set of attributes: title, description, estimated time, start date, estimated end date, actual end date, and status.

The implementation includes:

- Creating and assigning tasks
- Changing task status: Pending, In Process, Completed
- Automatically calculating Estimated End Dates
- Recording Actual End Date for the task upon its completion

Comment Module

Collaboration on tasks is often needed handled through the comment module. Task users, either assigners or assignees, can post comments. The module makes sure that nobody sees or posts comments regarding a task unless they are authorized.

Features:

- Post comments
- Retrieve comments for given tasks
- Comment display based on role and relationship for the respective task.

Time Log Module

Timekeeping is pivotal to evaluating performance; thus, the module allows for logging daily task hours by individual Assignees. This, in turn, is used to estimate task efforts and efficiency.

This includes:

- Logging hours on a daily basis
- One entry per day per user-task combination
- Cumulative view of time logs for each task

4.2 Prototype

A prototype system was set up to test and show functionalities. The back-end was solely powered by Nest.js and exposed a suite of RESTful APIs, while the front end built in Next.js constituted a basic interface that allowed communication with these APIs.

The following screens were built into the user interface:

- User authentication (login/signup)
- Task dashboard showing assigned as well as created tasks

- Task detail view with a comment thread and time logging form

The front end issued HTTP requests against the back end using fetch requests. The JWT tokens obtained after logging in were stored on the client side and attached to each later request involving secure communication. While not fully styled, this UI provided sufficient interactivity to show the main logic of the back end.

The back end implemented MySQL as the database, while the ORM mapping was realized using TypeORM. Relations were well defined between the entities:

- Users and Departments: Many-to-Many
- Tasks and Users: Many-to-One (assignee)
- Tasks and Comments: One-to-Many
- Tasks and Time Logs: One-to-Many

Postman is used for the back end to help testing of the API endpoints and allow developers to interactively test the endpoints directly.

4.3 Role-Based Functionality

It is put in place the Model of Role-Based Access Control (RBAC) for action in a very secure and functional environment in a system. It enables a user of the system to act along specific lines of permission that pertain directly to his or her line of business and the organization. As of present application, there are three user roles: an Admin, Assigner, and Assignee. Each of which determines what a user can or cannot do and manage.

Architecture for Role Implementation

Roles are implemented using custom decorators and guards in Neast.js. A custom @Roles () decorator is used to attach metadata to the controller routes, and a corresponding Role Guard is used to filter incoming requests and to check:

- If the JWT token was retained and whether it was valid.
- What role the authenticated one had.
- If the role is allowed to access the requested resource.

Thus, almost all the access control and safety would be tackled at the central level to ensure security at the route level.

Admin Functionalities

This is the super user of the platform who has complete overview of everything.

- User Management: Create, read, update and delete any user account in the system.
- Department Management: May contain departments with users assigned.
- Role Assignment: Has the authority of transferring or changing roles among other users.
- Overview On Global Tasks: All tasks can be seen from all the departments, though they never assign or complete any task.

Thus, it is able to conduct order at the system level of all the activities of organizational hierarchies for the administration.

Assigner Functionality

Assigners are coordinated to the departmental level, and they plan and delegate work.

- Task Creation: They can create tasks and assign these tasks to specific users in their department.
- Task Management: View overall status of all tasks in the departmental level.
- Review Time and Performance: The assigners can review task logs and time entries entered by assignees.
- Restrictions: Cannot modify users and roles and cannot interact with other departments other than his own.

This function supports decentralized management where every department enjoys the full autonomy of its workflow performance.

Assignee Functionality

An assignee is any staff member who executes the tasks set before him/her.

- View Tasks: This is where users see the tasks assigned only to them.
- Update Statuses: The assignees can update the status of a task from Pending to In Process and finally to Completed.
- Time Tracking: It logs daily time worked on each task.
- Comment: Assignees can comment on the tasks for clarification, to report progress, or express issues.
- Restrictions: Cannot view or change other users, roles, and tasks that are not assigned to them.

This way, the system empowers every user based on his/her role, fostering autonomy while maintaining boundaries.

4.4 Task Lifecycle Implementation

The task lifecycle, as defined within this project, is a structured, state-based flow mechanism for creating, assigning, progressing, and completing tasks. Life cycle management ensures that all work within the organization can be held accountable, publicly visible, and traceable. The implementation primarily concerns the management of business rules and the technical enforcement of these rules.

A. Task States and Transitions

The task object is governed by well-defined states:

- Pending: The task has been created and assigned but has not yet started.
- In Process: The task is actively being worked on.
- Completed: The assignee has finished the task and marked it as done.

Each transition has to obey a logical constraint:

- The ONLY entity that can authorize a task from Pending → In Process is the Assignee.
- The ONLY entity that can authorize a task from In Process → Completed is the Assignee.
- Upon completion, the task is read-only and cannot be further edited unless done so by an Admin.

- All transitions will be timestamped and logged.

B. Task Metadata Management

Metadata-rich tasks are as follows:

- Title and description
- Department with respect to the task
- Assigned value which determines to whom the task is assigned
- Assigner which determines who created the task
- Estimated time in days or hours
- Start date set manually or by assignment
- Calculated end date (start date + estimated duration)
- Actual end date which is auto set when task is marked as Completed

This information is stored in a relational schema under MySQL and references user and department foreign keys.

C. Time Tracking Integration

- As a matter of discipline and to improve performance monitoring, the assignee should report every day the number of hours spent on each task. The time entries must include:
- A specific assignment to a task against a certain date
- A report by aggregation
- Evaluated by Admin and Assigner roles
- It is guaranteed that the only time entries that can be input either refer to are the current time or for activities whose status is already "In Process."

D. Comments and Collaboration

The task entity works in tandem with the commenting system. Each comment is tagged with a user ID and belongs to a task. Comments help to:

- Discuss tasks in real time by assigners and assignees
- Update progress or request clarifications
- Follow a threaded view in timestamp order

E. Backend implementation behind NestJS

The logic of capturing significant business operations regarding the task lifecycle is duly encapsulated in the TaskService class.

State transitions are permitted on behalf of respective roles via guards and interceptors only.

The general validators together with the custom pipes and decorators ascertain that the proper validation is made, so a status cannot be marked as Completed before it is marked In Process.

Changing the status to Completed would automatically invoke another side effect: the actual_end_date is automatically set.

F. Error Handling and Constraints

- Having duplicate task names within the same department is not allowed.
- A non-negative integer must be submitted for the estimated time.
- Transitioning from Completed to any other state is not permitted, except by an Admin.
- Role-Based Restriction Handling @Roles() decorators and custom guards.

4.5 API Integration and Testing

A robust backend API layer is integral to any task management system. This project consisted of RESTful APIs built on the NestJS framework, with API endpoints architecture around resource-based access to tasks, employees, departments, time logs, and comments. The API integration also implemented rigorous manual and automated testing to establish reliability and performance.

A. API Design and Routing

All routes are structured as per REST conventions and are categorized by resource.
For instance, /auth – for login and issuance of token
/users – for CRUD action on employees
/departments – create, update, list departments
/tasks – core task operations (create, update, change status, view)
For comments – create/view task comments
/time_logs – log hours and view timesheet
This means that authentication is done through JWT-based protection and role-based decorators. The routes are protected by middleware that checks on the role and department membership of a user.

B. Authentication & Authorization Middleware

It makes use of Passport.js for JWT validation on the backend, as well as a custom RoleGuard to maintain access:
No routes can be accessed for users without valid JWTs.
RoleGuard limits access to certain routes by @Roles() metadata.
DepartmentGuard ensures that tasks belonging to their department can be seen and/or worked on by users except Admins.

C. API Response Structure

Each API endpoint must have a common defined structure in the response: {success: boolean, message: string, data: any}
The same makes it easier to integrate this into the front end and makes guarantees for uniform error handling.

D. Integration into the Next.js frontend

Next.js frontend access their NestJS APIs using fetch and Axios. Each frontend page or component accesses the backend endpoints:

- Login page calls /auth/login from where the JWT is stored in localStorage.
- Applies to the Dashboard which calls /tasks to fetch its relevant task by role.
- A task detail page facilitates status updates, comments, and time logging.
- Token handling is done via a request interceptor that adds the Authorization header automatically.

E. Manual Testing via Postman

Postman collections were significantly used for:

- Testing every route with multiple user roles
- Simulating complex workflows such as task creation → time logging → status change → comment thread
- Checking error conditions like unauthorized access, absence of fields, or invalid transitions
- A few test cases would be:
- Attempting status change on a task without permissions
- Logging time on a finished task (should fail)
- Access to tasks in other departments (for Admins only)

F. Automated Testing (Unit and E2E)

All features were tested using NestJS testing utilities (Jest + Supertest) for unit and E2E tests:

- Unit tests covered service-level logic, such as status change or time calculation.
- E2E tests verify full API behavior by simulating requests and responses with actual HTTP calls.
- Examples of test cases:
- Create task → Assign to user → Change status → Verify actualEndDate
- Ensure role-based access blocks unauthorized endpoints.

G. API Documentation

Swagger creates API documentation automatically:

- Accessible through /api/docs
- It also contains all endpoints with input schemas and response formats.
- Helps frontend and other members of the team to understand backend behavior in no time.

H. Error Handling and Status Codes

Meaningful HTTP Status Codes are used by the API:

- 200 OK indicates successful operations.
- 201 Created shows resource creation.
- 400 Bad Request are validation errors.
- 403 Forbidden indicates unauthorized role access.
- 404 Not Found for missing resources.
- 500 Internal Server Error for unhandled failures.

The error responses have meaningful messages that would guide both front-end and end-users.

5. Results and Analysis

5.1 Overview of System Functionality

The Employee and Task Management System has been created to allocate tasks seamlessly, track their status, and have department-specific access control based on roles of users; it has tested and integrated the backend built on NestJS with a prototype frontend built using Next.js.
Critical functionalities were tested under the various user roles and within department specifications to validate business rules applicability and smooth inter-module interaction.
Major components validated:

- Authentication and role-based authorization.
- Creation of tasks, updating their states, and status transitions in the lifecycle.
- Department-based task visibility and user segregation.
- Time logging and task duration calculations.
- Commenting on tasks for team collaboration.

Dashboard view for different user roles:

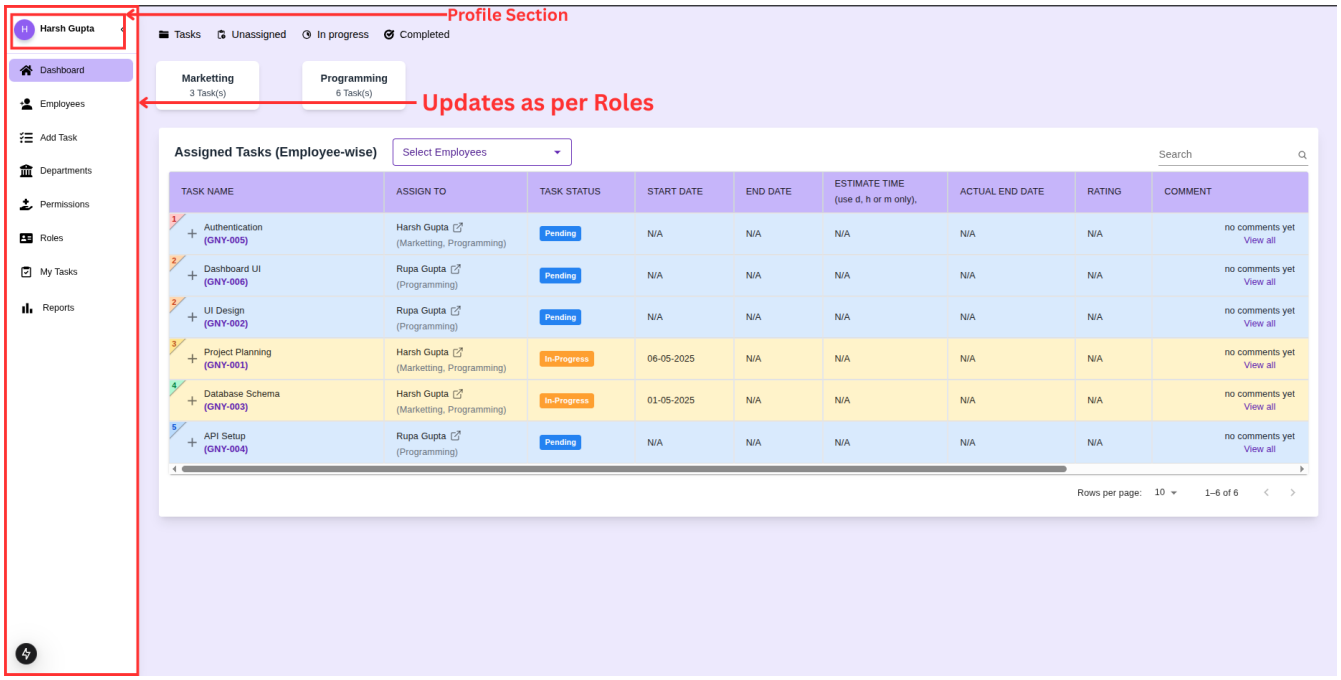

Fig.5.1 - Admin Dashboard

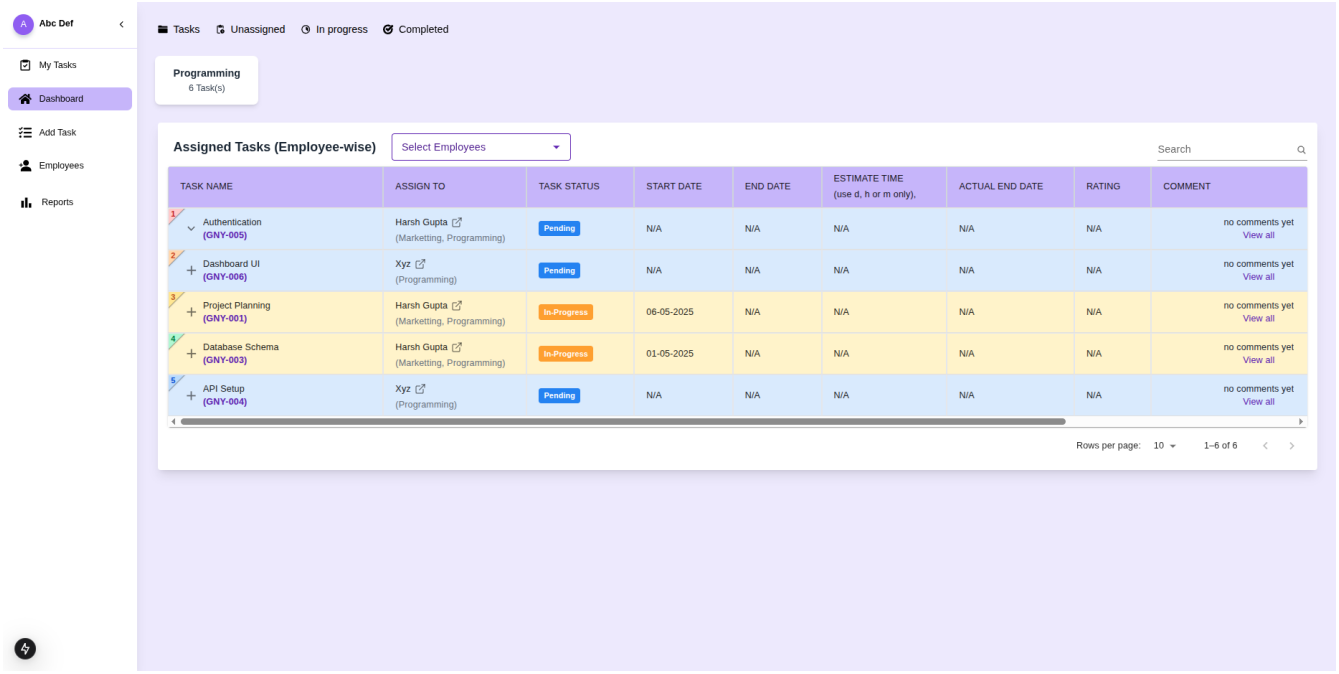

Fig.5.2 - Assigner Dashboard

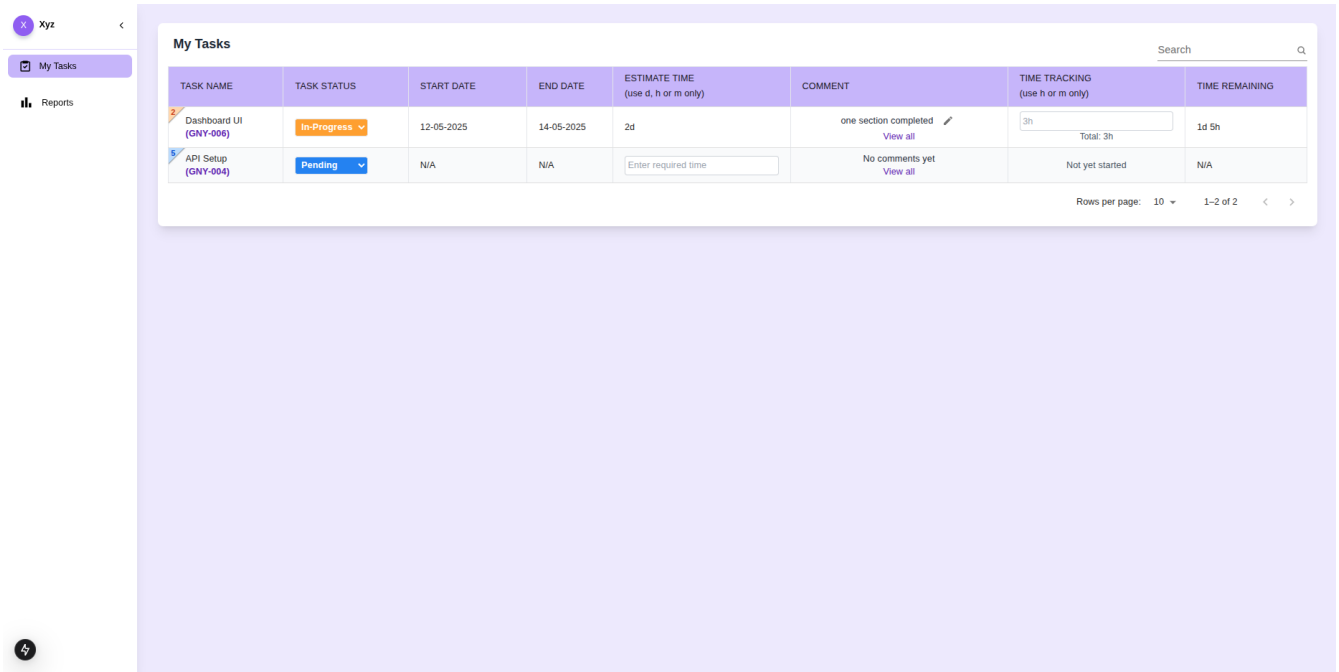

Fig.5.3 - Assignee Dashboard

Task detail view showing metadata (start date, estimated time, end date, actual end date, etc.)

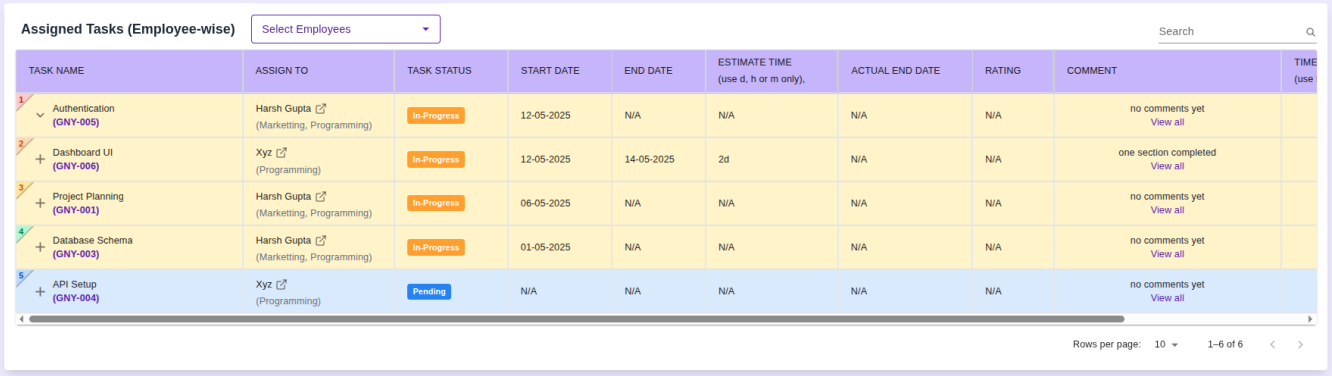

Fig.5.4 - Task Detail Table

5.2 Feature wise Testing and Outputs

The entire gamut of major system features were tested for functional correctness, access control, and stability with both manual and automated methods.

A. User Role and Permissions

Test Scenarios:

- Admin should be able to view and manage all users, departments, and tasks
- Assigner should only assign tasks to the users within his/her department
- Assignee should be able to update only his/her own tasks

Outcome:

All roles maintained the integrity of their boundaries. Successful unauthorized access attempts were effectively blocked with pertinent error messages.

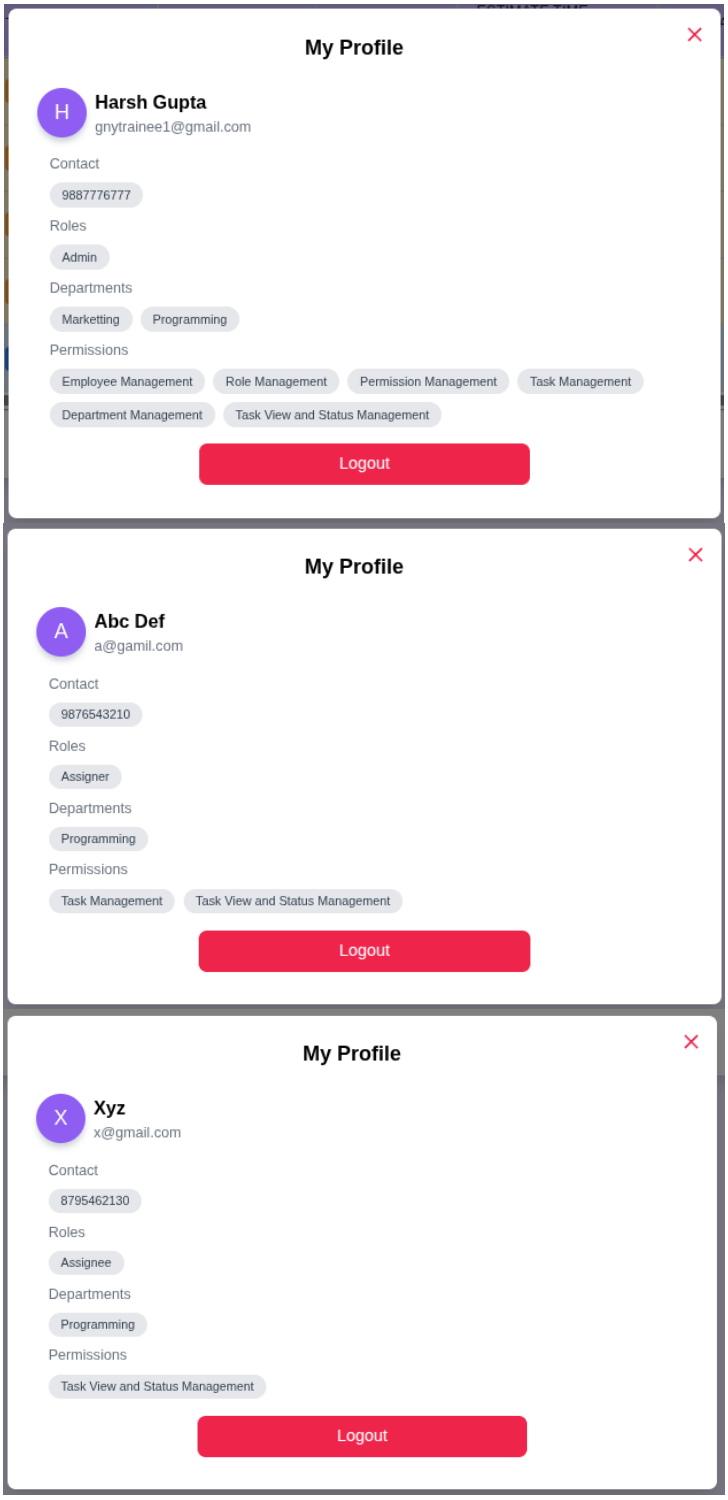

Fig.5.5 - Permissions as per different Roles

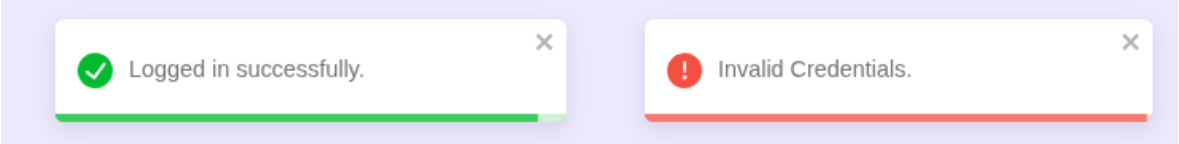

Fig.5.6 - Authorized and Unauthorized Login Alerts

B. Task Lifecycle Transitions

Test cases:

- Pending-In Process by an Assignee
- In Process-Completed automatic setting of actual_end_date
- Invalid transitions like Completed-Pending disallowed

Result:

Lifecycle logic held as expected. Timestamps were updated correctly. Backend validations rejected the invalid transitions successfully.

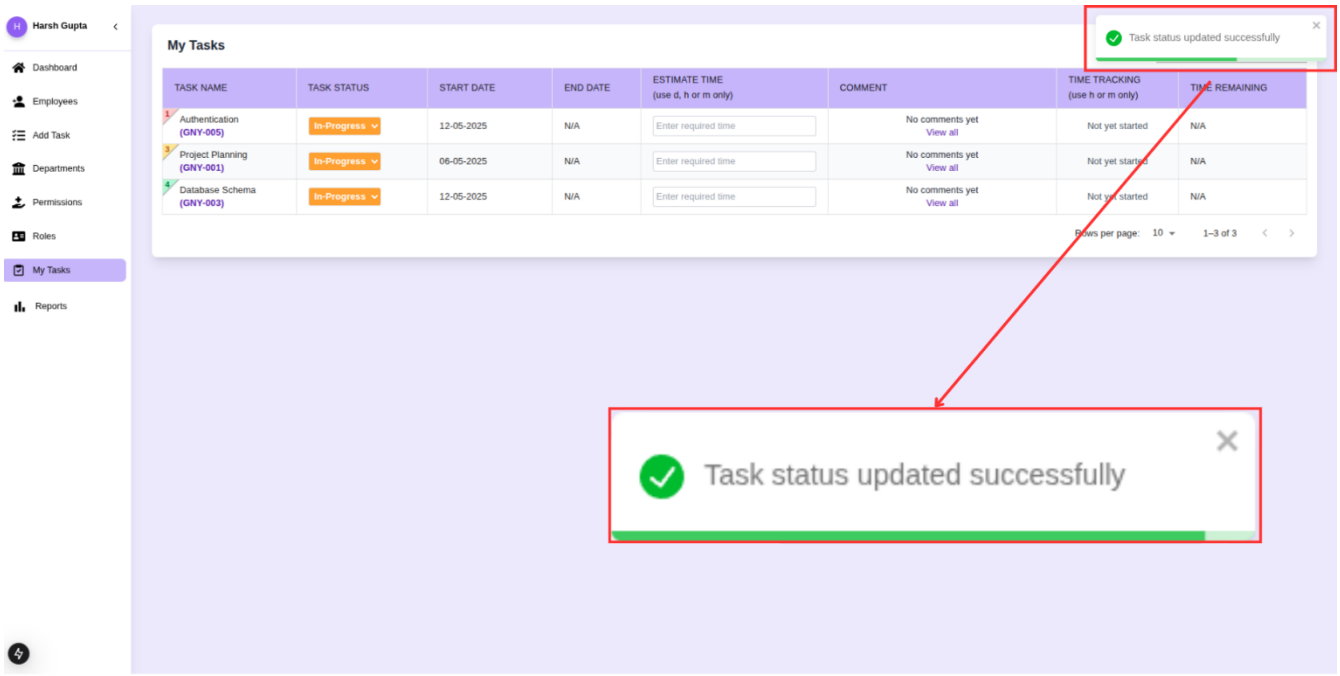

Fig.5.7 (i) - Successful Status Change

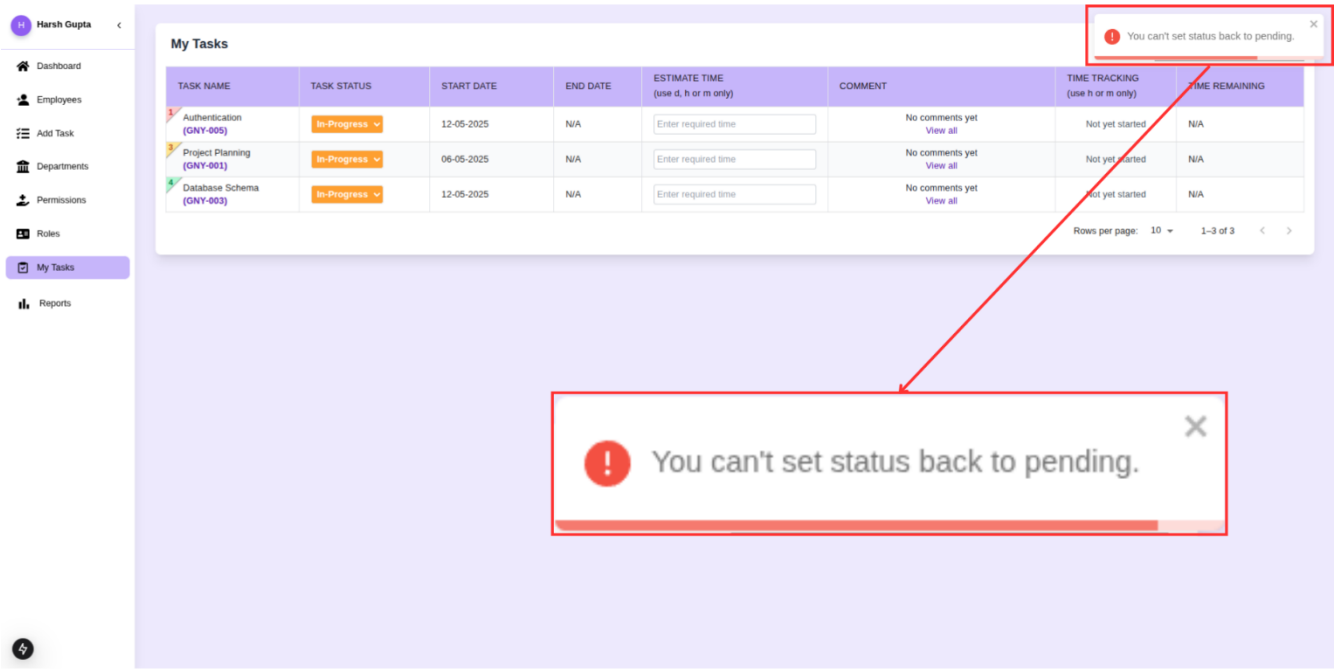

Fig.5.7 (ii) - Task Status can be set to "Pending" again

5.3 Performance Analysis and Improvements

A. API Response Time

Average API response times for main endpoints were measured using Postman and browser dev tools:

- /tasks (GET all for current user): around 120-180ms
- /tasks/:id (GET single task with logs and comments): around 200-250ms
- /login (POST): around 70ms
- /comments (POST): around 150ms

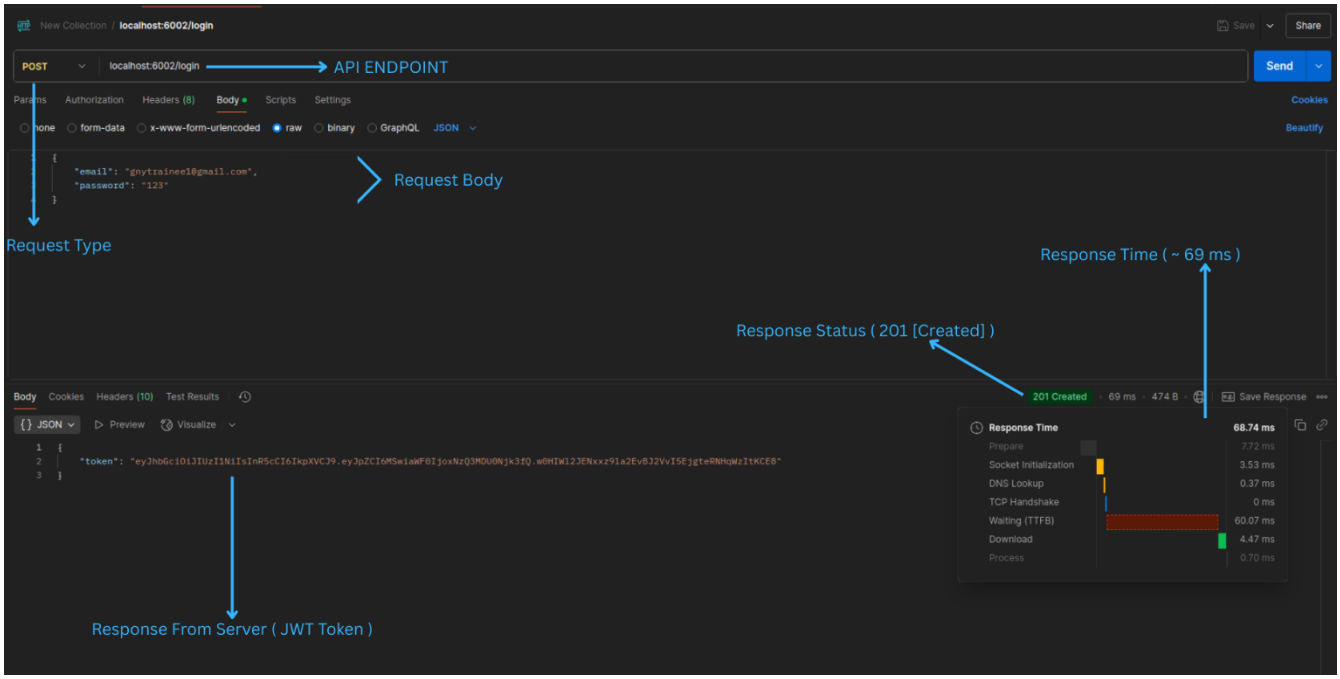

Fig.5.8 - API Response and Statistics

B. Database Performance

- Indexing had been applied to frequently queried fields such as userId, departmentId, and taskId. Thus:
- Task queries filtered by user and department run under 100ms.
- JOINs for the logs and comments were optimized with eager loading.

C. Scalability and Load Handling

Despite being deployed in a local environment, the modular architecture and layered service pattern guarantee that the application will be horizontally scalable with minimal changes. This allows for planned additions, including WebSocket-based real-time updates and background workers for notifications, without alarming the present code.

D. Limitations Observed

- No pagination implemented yet on task or comment lists, which could affect performance at scale.
- No integration of WebSocket or push notification yet.
- The UI is plain and mainly for testing-a real UX design is under development.

6. Conclusions & Future Scope

6.1 Conclusions.

The Employee and Task Management tool developed during this project will adequately serve the need for a structured, scalable, and collaborative platform in managing employee roles and organizational tasks. The central principle of the system - having employees in specific roles (Admin, Assigner, Assignee) and putting them together with departments - has been proven correct and supported by a modular backend implementation created by NestJS. It allows feature addition flexibility, privacy of concern, and secure handling of user permission and data.

These key conclusions are derived from the project:

- ***Role-Based Architecture Confers Clear Permissions***
  The system thus divides users into three roles (Admin, Assigner, and Assignee), whereby the responsibilities and access to data are dichotomized with respective statuses in the organization. Each of these roles has a preset function that is enforced through middleware and guards in the NestJS backend. For instance, while Admin can manage users and departments, he/she may not do all those things with assignments. Meanwhile, while it can create and operate tasks in the department, assigners cannot do those tasks, and an assignee can only act on assigned tasks.

- ***Organizational Benefits of Departmental Segmentation***
  By segmentation into departments, grouping of users as well as tasks logically assigns one's access to the tasks and channels of communication relevant to teams. This becomes particularly useful when the organization is large, with teams to work independently or even across various functions.

- ***Task Lifecycle Management Provides Transparency of Workflow***
  It covers a whole task life from its creation to completion with a set of attributes that will track time and progress. Every task registers a starting date, estimated time, calculated end date, and an actual end date, thereby enabling project managers and team leads in tracking deadlines, identifying delays, and understanding in a data-driven manner workload sharing across teams.

- ***Time Logging and Comments Promote Accountability and Collaborative Actions***
  Through this logging of time spent by the assignees on the assigned tasks daily, users are becoming more accountable and productive thus aiding to measure productivity. And let team members communicate on a task's thread through the comment feature instead of digging through dozens of external communication tools, improving traceability.

- ***Scalable, Modifiable Backend***
  Of the several technical landmarks of the part, an inner scalable modular backend was achieved in NestJS. The basic function encapsulates one of the core features (for User, Department, Task, Comment, TimeLog

at least) into its own angularized modularization with independent services, DTOs, and controllers. Such a design exposed itself to the extensibility and the ease of maintenance and upgrade in the future.

- ***Front Prototype as Validation Layer.***
  The front end, although minimal and constructed mostly for testing, acts as a very valuable validation layer of all backend features. Key interfaces such as logging-in, displaying tasks, and updates about the status were quickly implemented using Next.js. This lays the ground for potentially more extensive development on a full-blown frontend.

- ***Backend validation of security and data integrity***
  All the core functionalities—task creation, role-enforcement, department validation, and task status transitions—will all be server-sided validations. Therefore, unauthorized actions cannot be performed if the movable/immutable part of the application is tampered with even if the frontend was unsuccessfully attacked and compromised.

In furnish, the system suffices almost all requirements that a contemporary task and employee management tool should have. The backend support created through this project becomes trustworthy, modular, and extensible, ready for accommodating improvements in the areas of analytics, mobile support, and integrations.

6.2 Future Scope of Work

While the project satisfies its fundamental functional requirements, a great number of enhancements and extensions could convert it into an enterprise level platform from a purely simple task management tool.

Below are some of the important avenues for future work:

- ***Advanced Reporting & Analytics Dashboards***
  - At this point, data generation is in place, but not really deep within the analysis and visualizations. Future analytics might include the following features:
  - Task statuses, departmental workloads, and performance of users visualized on dashboards.
  - Report generation of time spent by tasks, departments, and users.
  - Graphs for trend lines of task completion, burn-down charts, and productivity heat maps.
  - Reports exportable in PDF and Excel formats to be used by HR and project managers.

- ***Notification and Reminder System***
  - Both active and scheduled notification can drastically improve task compliance and engagement.
  - Upcoming improvements could feature:
  - Emails about new tasks assigned, about tasks coming up to their end dates, and overdue ones;
  - In-app notifications and push notifications in mobile applications;
  - Weekly summary emails for admins and assigners to view progress;
  - User preference-modifiable notification settings.

- ***Full-Featured Frontend with Responsive UI/UX***
  - The current frontend is functional, but a more advanced and user-friendly front end would significantly enhance the user experience:
  - Dashboard-style UI with different charts, recent tasks, and calendar views.
  - Implement task prioritization and progress tracking via drag-and-drop (e.g.: Kanban boards).
  - Responsive design compatible in desktop, tablet, and mobile devices.
  - Dark/light mode and accessibility features to enable inclusive usage.

- ***Mobile Application Development***
  - Mobile access building a mobile app as most users tend to use their phones when on-the-go is a plus in general accessibility as well as app usage. The mobile app is promising to have the following:
  - Quick updating and changing of status for tasks.
  - Push-notifications regarding important updates.
  - Offline mode with syncing capabilities when connected again.
  - Camera/photo upload integration for tasks that require the use of media.
  - Development through technologies like React Native or Flutter should be considered for cross-platform building.

- ***Customizable Roles and Permission Matrix***
  - Three current roles are supported by the system. The future versions would allow admins to:
  - Define custom roles wherein permissions may be configured.
  - Assign fine-grained permissions (say: view-only, edit, and comment-only).
  - Create templates for permission settings-per-department or team.
  - This will better tailor the system for larger and more complex organizations with unique workflows.

- ***Integration with External Platforms***
  - To enhance the value of this tool integration on third-party tools, there are quite a number of them. Some of them are:
  - Google Calendar, Outlook, to be specific task scheduling and reminders.
  - Slack or Microsoft Teams for communication and notifications.
  - Integrate with Jira, Trello, or Asana to import/export tasks.
  - Link to tasks with GitHub or GitLab for developers.
  - Improved Audit Log and Version Control
  - Comprehensive audit trail on actions performed by users: changing or creating tasks, editing time logs, updating permissions, and so forth.
  - Kept historical versions of tasks or comments, in the event that rollback or review is warranted.
  - Performance Optimization and Load Testing

- ***Scale of the systems:***
  - Query optimization of the databases while providing indexation for the handling of data in large numbers.

- Implement caching mechanisms (for instance, Redis) for frequently accessed data.
- Load Testing using tools like Apache JMeter or Locust. Deploys cloud infrastructure with autoscaling (e.g., AWS, GCP) to make it highly available.

References

References and Bibliography

- *https://www.npmjs.com/package/react*
- *https://mui.com/material-ui/react table/?srsltid=AfmBOoqvHDhrlZJ4J6hkPO4zEkfbigLLkM4G1LD4lVSK50XklmmQXQo-*
- *https://nextjs.org/docs/pages/building-your-application/routing/dynamic-routes*
- *https://nextjs.org/docs/pages/building-your-application/data-fetching/client-side*
- *https://docs.nestjs.com/*
- *https://quilljs.com/docs/installation*
- *https://tailwindcss.com/docs/colors*
- *https://codesandbox.io/examples/package/react-big-calendar*
- *https://mui.com/material-ui/material-icons/*
- *https://lucide.dev/icons/categories#math*
- *https://react-tooltip.com/docs/getting-started*