1. INTRODUCTION

In the fast-paced world of academics, students often struggle to keep track of tasks, deadlines, and study materials efficiently. Traditional methods such as manual to-do lists, scattered digital notes, or relying on memory can lead to missed deadlines, disorganized study plans, and an overall decline in productivity. Without a structured system in place, students may find it difficult to prioritize tasks, collaborate effectively, and maintain an organized workflow.

To address these challenges, the app introduces a comprehensive, AI-powered productivity solution that helps students manage tasks, notes, and study materials efficiently. Built using modern web technologies like the MERN stack, the app offers a structured task management system where students can create, filter, edit, and categorize tasks into To-Do, In-Progress, and Completed sections. This ensures a clear workflow, allowing students to stay focused and manage their academic responsibilities effectively.

In addition to task management, the app includes an intuitive file and folder management system, enabling students to upload, organize, and access PDFs seamlessly using Multer. This feature helps students keep their notes in one place, eliminating the hassle of searching through multiple platforms for study materials. Whether it's lecture notes, reference materials, or project documents, everything is structured and easily accessible.

A key highlight of the platform is its integration of an AI-powered chatbot, leveraging Hugging Face models, designed to provide personalized productivity tips, task recommendations, and smart reminders. The chatbot helps students stay on track by offering insights on how to optimize their workflow, prioritize tasks, and maintain consistent study habits.

Beyond individual task management, the app also supports collaborative task assignments, allowing students to work efficiently in teams by delegating responsibilities, setting deadlines, and tracking progress in real-time. The built-in reminder system ensures that no task is overlooked, reducing the chances of missing important deadlines.

By combining intelligent automation, seamless task organization, and AI-driven **assistance**, the Student Productivity App transforms the way students manage their academic responsibilities. It replaces inefficient, manual workflows with a centralized, smart productivity system, empowering students to **stay** organized, maximize efficiency, and enhance their academic performance. This structured approach not only helps students maintain discipline but also ensures that they make the most of their time, reduce stress, and achieve their academic goals with confidence.

1.2 Objectives & Scope of the Project

The objective of this project is to develop a **comprehensive student productivity application** that enhances task and note management through smart automation and AI powered assistance. By implementing this system, students can efficiently organize their academic workload, prioritize tasks, and streamline their study materials. The main objectives include:

- Automate Task Management by allowing students to create, edit, and categorize tasks into To-Do, In-Progress, and Completed sections. Improved transparency and accountability by ensuring each approval step is documented.
- Enhance Productivity through an AI-powered chatbot that provides personalized productivity tips, reminders, and task suggestions. Enabling real-time tracking of approvals through a user-friendly dashboard.

- Improve Organization with a file and folder management system, enabling students to upload, save, and access study materials easily**.**
- Enable Seamless Collaboration by allowing users to assign tasks, set deadlines, and track progress in real time.
    - **Provide smart reminders** to help students stay on track with deadlines and upcoming tasks.
    - **Optimize File Handling** through **Multer**, ensuring seamless **Folder creation**, **PDF uploads and viewing** within the app.
- **Deliver Insights with Data Visualization** to help students track their task completion rate and productivity trends over time.

The project is designed to provide an intuitive and structured approach to academic productivity, ensuring that students can manage tasks, track progress, and access study materials efficiently. The system will include the following core modules :

- **Task & Note Management Module –** Allows students to organize, prioritize, and track tasks effectively.
- **File & Document Management Module** – Enables students to store and access study materials and PDFs effortlessly.

AI-Driven Productivity Assistance – Utilizes Hugging Face AI models to offer productivity insights, reminders, and task suggestions.

1.3 Problem Statement

Students often struggle with managing tasks, deadlines, and study materials efficiently, leading to disorganization, missed deadlines, and reduced productivity. Traditional methods such as manual to-do lists, scattered digital notes, and reliance on memory often result in inefficiencies, making it difficult for students to prioritize tasks

effectively. Without a structured workflow, tracking assignments, scheduling study sessions, and maintaining organized notes becomes a significant challenge.

One of the major issues is the lack of a centralized task management system, where students can systematically track their academic workload. Many students rely on multiple tools and applications to manage tasks and notes, which leads to fragmented information spread across different platforms. This scattered approach not only makes it difficult to retrieve study materials quickly but also increases the likelihood of forgetting important tasks and deadlines.

Another key challenge is inefficient time management. Without smart scheduling and prioritization, students often struggle to balance multiple assignments, projects, and personal tasks. This results in procrastination, last-minute stress, and decreased academic performance. Additionally, students who engage in group projects often face difficulties in task delegation, tracking progress, and ensuring accountability among team members. The absence of collaborative task management tools creates confusion and inefficiencies in teamwork.

Furthermore, students frequently deal with disorganized study materials, making it difficult to access important notes and resources when needed. Manual file storage and inconsistent folder structures often lead to misplaced or lost notes. Without a structured system for organizing and accessing study materials, students spend unnecessary time searching for essential resources, impacting their overall efficiency.

A critical aspect of productivity is staying motivated and receiving timely reminders. However, many students struggle with self-discipline and forget to review or complete tasks on time. Traditional task management tools often lack intelligent reminder systems and personalized insights that adapt to students' study patterns and workload. The lack of AI-powered assistance results in poor planning and ineffective study habits.

Additionally, the absence of a dedicated student-focused productivity tool that integrates task management, note organization, file handling, and AI-driven assistance makes it difficult for students to manage their academic responsibilities effectively. Existing task management apps are often generic and do not cater specifically to the needs of students, such as organizing assignments, setting study reminders, or managing academic deadlines.

The Student Productivity App addresses these challenges by offering a centralized, structured, and AI-powered productivity system tailored for students. By integrating features such as intelligent task management, seamless file organization, AI-based productivity insights, and smart reminders, the app ensures that students can stay organized, manage time efficiently, and maximize productivity throughout their academic journey.

- Lack of a structured task management system, leading to inefficiencies.
- Difficulty in prioritizing tasks due to the absence of intelligent scheduling.
- Disorganized study materials, making it hard to retrieve essential notes and PDFs.
- Lack of AI-powered productivity insights, resulting in poor time management.
- No dedicated student-focused productivity platform, leaving students dependent on generic tools.

The proposed system eliminates inefficiencies and enhances productivity by providing a smart, centralized, and AI-integrated solution. With task tracking, file management, AI driven assistance, and collaboration features, students can plan, prioritize, and stay on top of their academic tasks with ease

2. LITERATURE SURVEY

2.1 TASK AND PRODUCTIVITY MANAGEMENT SYSTEMS

Task and productivity management systems have evolved significantly over the years, aiming to improve workflow efficiency, time management, and collaboration. These systems help individuals and teams plan, organize, and execute tasks effectively by integrating features such as task tracking, scheduling, note-taking, and automation.

Traditional task management methods, such as manual to-do lists and standalone notetaking apps, often result in inefficiencies due to lack of integration, poor organization, and difficulty in tracking progress. To address these challenges, modern digital task management solutions leverage structured workflows, AI-based automation, and cloud storage to enhance productivity and streamline information access.

Several studies highlight the effectiveness of digital productivity applications in improving academic and professional efficiency. The integration of AI-powered recommendations, task prioritization, and real-time notifications has shown a significant increase in user engagement and time management skills. These advancements contribute to better decision-making, structured workflows, and reduced cognitive overload.

2.2 AI-POWERED PRODUCTIVITY ENHANCEMENT

Artificial Intelligence (AI) has played a crucial role in enhancing task management and productivity applications by providing intelligent insights, automation, and personalized assistance. AI-driven chatbots, such as those powered by Hugging Face

models, enable users to receive context-aware reminders, task suggestions, and productivity tips based on their work patterns and deadlines.

AI-based task management systems utilize natural language processing (NLP) and machine learning algorithms to analyze user behavior and suggest optimal task organization strategies. Studies in AI-powered productivity tools demonstrate their ability to reduce procrastination, improve focus, and provide personalized time management solutions.

By integrating an AI-driven chatbot, the proposed Student Productivity App enhances user engagement by providing real-time productivity recommendations, intelligent task prioritization, and personalized reminders. This ensures students can efficiently manage their academic workload without feeling overwhelmed.

2.3 FILE AND DOCUMENT MANAGEMENT SYSTEMS

The ability to efficiently store, organize, and retrieve documents is essential for effective learning and productivity. Traditional file storage methods, such as local device storage and physical notebooks, often lead to disorganization, loss of materials, and difficulty in access.

To overcome these limitations, cloud-based file management systems have gained popularity, offering seamless access, real-time synchronization, and structured organization of digital notes.

Modern file management solutions incorporate Multer-based file handling, which enables users to upload, view, and categorize study materials efficiently. Additionally,

PDF viewing capabilities allow students to access educational resources without needing external applications, improving the seamless study experience.

Studies indicate that structured file management systems significantly enhance retrieval efficiency, knowledge retention, and overall productivity. By integrating a folder-based file organization system, the Student Productivity App ensures that users can manage their documents efficiently, reducing the time spent searching for study materials.

2.4 IMPACT OF PRODUCTIVITY APPLICATIONS ON STUDENT PERFORMANCE

Research in student productivity and time management suggests that structured task management applications significantly enhance focus, reduce stress, and improve academic performance. Digital productivity tools help students develop self-discipline, better planning skills, and efficient workload management. By integrating AI-powered assistance, real-time notifications, and structured task organization, the Student Productivity App aims to bridge the gap between academic planning and execution.

3. SYSTEM ANALYSIS

3.1 Requirement Specification

The system is designed to provide an efficient task management and note organization system tailored for students. The platform ensures seamless usability, secure data handling, and AI-powered assistance to enhance productivity. The key system requirements include:

- User Roles and Access Control: The system supports multi-user access with role based functionalities, allowing students to manage their tasks and notes while ensuring data integrity and controlled permissions.
- Multi-User Authentication: Secure login mechanisms must be implemented to handle multiple users simultaneously, ensuring that each session is isolated and protected from unauthorized access.
- Data Security and Privacy: All user data, including tasks, notes, and uploaded files, must be securely stored and encrypted to prevent unauthorized access and data breaches.
- Task and Note Management: Users should be able to create, edit, delete, and organize tasks into categories such as To-Do, In-Progress, and Completed. Additionally, students should be able to store and access notes efficiently, including uploading and viewing PDFs.
- AI-Powered Assistance: The system integrates an AI chatbot to provide personalized task reminders, study tips, and productivity suggestions, enhancing time management and task prioritization.

- File and Folder Management: The app should allow students to upload, categorize, and retrieve documents, with Multer-based file handling ensuring smooth document management and PDF viewing.
- Performance Optimization: The system must be designed for high efficiency, supporting multiple users and handling large amounts of data without significant lag or downtime.
- Scalability: The application should be scalable, ensuring it can accommodate future enhancements, increased user activity, and additional AI features without performance degradation.
- User-Friendly Interface: The system must have an intuitive and modern UI/UX design, ensuring that students can easily navigate, organize their workload, and manage tasks efficiently.
- Automated Notifications: The system should provide real-time notifications to users regarding task deadlines, reminders, and status updates, ensuring that no task or note is overlooked.

3.2 Constraints

The Student Productivity App operates within several constraints to ensure security, efficiency, and smooth functionality for all users. These limitations help maintain a structured workflow, prevent unauthorized access, and optimize system performance.

User Authentication & Role Assignment.

- Users must be registered before accessing the platform.
-
- Only authorized users can create, edit, or delete tasks and notes.

Task & Note Management Restrictions:

- Tasks must be categorized into To-Do, In-Progress, or Completed sections.
- Notes and files can only be accessed or deleted by the respective user.

File & Storage Limitations:

- File uploads (PDFs, documents) are restricted to a maximum size to optimize storage usage.
- Users can only upload supported file formats to ensure compatibility.
- Notes and files are stored securely within the defined hardware and database limitations.

AI Chatbot Functionality:

- The AI assistant provides task recommendations, study tips, and reminders, but cannot modify user tasks directly.
- AI-generated responses are based on predefined models and cannot provide real time updates outside the system.

Task & Reminder Restrictions:

- Task reminders must be set within a specific timeframe to avoid excessive notifications.
- Users can only reschedule or update reminders

3.3 Software Specification

Operating System:

Windows 11 (64-bit) provides a stable and secure environment for development and deployment. It ensures compatibility with modern development tools and supports efficient multitasking for seamless project execution.

Backend Development Tools:

Node.js and Express.js are used as the primary backend technologies, enabling fast and scalable server-side operations.

Axios is implemented for handling API requests efficiently, ensuring seamless communication between the frontend and backend. Multer is used for file handling and secure data uploads within the system.

Multer is a Node.js middleware implemented for simplifying file uploads by handling multipart/form-data, which is a common format for sending files in web applications.

Frontend Development Tools:

React.js is used to build an interactive and dynamic user interface, allowing smooth navigation and real-time updates. Bootstrap is integrated to enhance UI responsiveness and ensure a modern design structure. JavaScript is used for handling client-side logic, improving user interactions and experience.

Database Management System:

MongoDB is chosen for its flexibility and scalability, providing a NoSQL approach for efficient data management. The database ensures secure and structured storage of task related information and user data.

AI Integration:

Hugging Face models are implemented to enhance the AI chat feature, providing intelligent responses and personalized recommendations for students. The AI component is integrated into the workflow to improve task prioritization and productivity.

Integrated Development Environment (IDE):

Visual Studio Code (VS Code) is used for development, offering real-time debugging, intelligent code suggestions, and seamless integration with Git for version control.

Browser Requirements:

Google Chrome and Microsoft Edge are recommended for running and testing the web application, ensuring compatibility with the latest web technologies and providing an optimized user experience.

Utility Tools:

WPS Office is used for documentation purposes, supporting structured report writing, formatting, and project documentation management.

3.4 Hardware Specification

Processor:

Intel Core i5 / AMD Ryzen 5 or higher ensures sufficient computational power for smooth application execution.

Memory (RAM):

8 GB RAM is recommended to handle multiple concurrent tasks, ensuring smooth operation of the application and database queries.

Storage:

A minimum of 50 GB free space is required to install development tools, store project files, and manage MongoDB database records securely.

System Type:

A 64-bit operating system running on an x64-based processor is essential to support modern development frameworks and ensure compatibility with Java, MySQL, and other software components used in the project.

Display:

The system operates on a standard monitor without pen or touch input, providing a clear and accessible interface for users to interact with the pricing approval system effectively.

3.5 Software Framework

This section outlines the key software components and frameworks that will be used for developing the student task management web application:

- **Framework(s):**

  Primary Framework: The application will be developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) to ensure a scalable and efficient workflow.

  Supporting Framework: Bootstrap will be used to enhance UI responsiveness and provide a clean, modern design.

- **Programming Languages:**

  JavaScript: The core programming language for both frontend and backend development, ensuring seamless interaction between components.

  HTML/CSS: Used for structuring and styling the web application, ensuring an intuitive and user-friendly interface.

- **Database Management:**

  MongoDB: A NoSQL database that securely stores user data, tasks, and chat history, providing flexibility for future scalability.

  Mongoose: A MongoDB Object Data Modelling (ODM) library for handling database operations efficiently.

- **Integrated Development Environment (IDE):**

Visual Studio Code (VS Code): The primary IDE for development, offering real time debugging, code suggestions, and seamless integration with Git for version control**.**

Version Control:

Git: Used for tracking changes and managing the project codebase efficiently. The repository will be hosted on GitHub for collaboration and version control.

- **Security Framework:**

  Role-Based Access Control: Ensures that users can only access their own tasks and chat history, maintaining data privacy.

  JWT (JSON Web Tokens): Used for authentication and session management to secure user data and prevent unauthorized access.

- **Testing Frameworks:**

Postman: Used for unit testing React components and backend API endpoints.

Cypress: For end-to-end testing to validate the functionality of the application.

Communication:

Axios: Used for interaction between the frontend and backend, enabling efficient data exchange.

Web Sockets: Implemented for real-time updates.

3.6 Feasibility Study

The feasibility study for the **Student Task Management Web Application** evaluates its viability from technical, operational, economic, and legal perspectives. This assessment ensures that the system is practical, efficient, and cost-effective while addressing user needs:

3.6.1. Technical Feasibility

The system is technically feasible as it is developed using the MERN stack (MongoDB, Express.js, React.js, Node.js), which ensures a modern and scalable web application. The chosen technology stack is widely supported and well-suited for handling dynamic data and real-time updates.

The frontend is built using React.js, providing a responsive and interactive UI that enhances the user experience. Bootstrap is used to ensure consistent styling and responsiveness.

The backend is powered by Node.js and Express.js, allowing efficient API handling and seamless communication between the frontend and database. MongoDB is used as the database, offering flexible data storage for tasks, user authentication, and AI chat history, while Mongoose ensures structured database management. Security is a priority, with JWT (JSON Web Tokens) used for authentication, alongside HTTPS and role-based access control to prevent unauthorized access. The system is designed as a single-user application, ensuring a personalized workflow without the complexities of multitasking. Additionally, an AI chat feature is integrated using Hugging Face APIs, providing intelligent task assistance. Multer is used for file uploads and file management.

3.6.2. Operational Feasibility

From an operational perspective, the system ensures smooth functionality with a structured workflow specifically designed for students. Users can create, edit, track, and delete tasks efficiently, with task prioritization and reminders to enhance productivity.

The AI chat assistant, integrated with Hugging Face APIs, helps users organize their tasks, answer queries, and provide productivity tips. The system maintains role-based access, allowing users to interact only with their data, ensuring security and a clutter-free experience.

The user interface is intuitive and distraction-free, providing seamless navigation and effective task tracking. Real-time updates keep users informed about their task progress, upcoming deadlines, and AI-generated recommendations.

3.6.3. Economic Feasibility

The project is economically feasible due to its reliance on open-source tools and frameworks, significantly reducing costs. React.js, Node.js, MongoDB, and Express.js are free to use, eliminating licensing expenses.

The application can be deployed on cloud services such as Vercel for the frontend and Render for the backend, minimizing infrastructure costs. Maintenance costs will mainly involve periodic updates, database optimization, and security enhancements.

By automating task management, the system saves users time that would otherwise be spent on manual organization, improving their overall productivity.

3.6.4 Legal Feasibility

The legal feasibility of the system is ensured through strict security measures to protect user data. Authentication is managed using JWT, preventing unauthorized access, while all communications are encrypted to safeguard sensitive information.

4. SYSTEM DESIGN

4.1 System Architecture

The web application follows a three-layered architecture, ensuring scalability, maintainability, and security. The system is structured into three primary layers: Client Layer (Frontend), Application Layer (Backend), and Database Layer, each responsible for different aspects of the application's functionality.

4.1.2 Client Layer (Frontend)

The Client Layer serves as the user interface, allowing students to manage their tasks efficiently. The frontend is developed using React.js, CSS with additional styling and responsiveness ensured by Bootstrap (for responsive design). Key functionalities include:

- **User Interface**: Provides a login/signup page and smooth UI.
- **Task Management:** Allows users to add, update, delete, tag, search, pin and prioritize tasks based on importance.
- **AI Chat Assistance:** Integrated with Hugging Face APIs, the AI assistant helps users organize tasks and provides productivity tips and study tips.
- **Dashboard:** Users can view tasks by their status, their task title, remainders.
- **Notes management:** Users can create folders, upload, delete and view files. The frontend communicates with the backend via **RESTful APIs**, ensuring smooth and efficient data exchange.

4.1.3 Application Layer (Backend)

The Application Layer manages the business logic and backend operations using Node.js with Express.js. It processes user requests, handles authentication, and interacts with the database. Major components include:

- **User Authentication:** JWT (JSON Web Token) authentication ensures secure login and session management**.**
- **Task Management API**: Provides endpoints for CRUD operations (Create, Read, Update, Delete) on tasks**.**
- **AI Chat API:** Communicates with Hugging Face APIs to generate intelligent responses based on user queries.
  - **File Management:** Multer streamlines the process of file uploads in Node. js by providing middleware for handling multipart/form-data, which allows easy uploading of files, handling of file size limits, defining file filters, and organizing uploaded files efficiently.
- This layer ensures **data validation, error handling, and security measures** to protect user information.

4.1.4 Database Layer

The Database Layer is responsible for securely storing and managing all essential application data, ensuring efficient retrieval and integrity. MongoDB is used as the primary database due to its scalability and ability to handle structured and unstructured data. Mongoose is implemented for schema enforcement and validation.

Schemas and Data Storage

- **User Schema:** Stores user information, including usernames, hashed passwords, roles, and authentication details**.**
- **File Management Schema:** Manages file uploads, storing metadata such as file paths, types, and associated user references.
- **Task Schema:** Maintains task-related information, including task descriptions, deadlines, priorities, and user assignments.
- **Reminder Schema:** Tracks reminders linked to tasks, with timestamps and notification settings.
- **Chat Schema:** Manages messages, user information, timestamps of messages and everything.

Data Relationships and Constraints

- **User-Task Relationship:** Users can create and manage multiple tasks, establishing a one-to-many relationship**.**
- **Chat Session Storage:** Conversations are stored per user, allowing retrieval of past interactions.
- **File Management:** File paths and metadata are stored, ensuring seamless file retrieval and organization.
- The system design incorporates constraints, such as **foreign keys**, **default values**, and **auto-incrementing primary keys**, to ensure consistency and integrity of the data.

4.2 Detailed Workflow Process

User Registration & Authentication:

- **New users register** with details such as name, email, and password.
- Upon successful registration, user details are stored in the **User Schema**.
- Users log in using **JWT-based authentication**, ensuring security.
- Upon login, users access their **personalized dashboard** with tasks, reminders, and everything.

Task Management:

- Users can **create, edit, and delete tasks**, specifying:
- Task title, description, deadline, priority, and status (Pending, In Progress, Completed)
- Tasks are stored in the **Task Schema** in MongoDB.
- Tasks can be filtered and searched.
- Tasks can be deleted and updated.

File Management System:

- Users can **upload and manage files** related to tasks or notes.
- Users can create folders and store files inside it.
- Users can view, delete, or organize files and folders.

AI Chat Integration:

- Users can interact with an **AI-powered chat assistant** for help with:
- Task suggestions.
- Productivity tips.

General queries related to studies or work

Chat sessions are stored in the **Chat Session Schema**.

Reminder System:

- Users can set **reminders for tasks** based on deadlines.
- Reminders are stored in the **Reminder Schema**.

Task list:

- Provides a list of tasks fetching from the task component titles.
- By tapping on it the task status is changed from to-do or in-progress to completed status

4.3 Security & Access Control

The system incorporates multiple security measures to protect user data, prevent unauthorized access, and ensure data integrity. The following security strategies are implemented:

Password Security & Authentication: **Passwords are securely stored** being saved in MongoDB. JWT (JSON Web Token) authentication is used to verify user identities, enable secure session management, Protect against session hijacking. Token expiration ensures that users **re-authenticate periodically** to maintain security.

4.4 Flow Chart Diagram

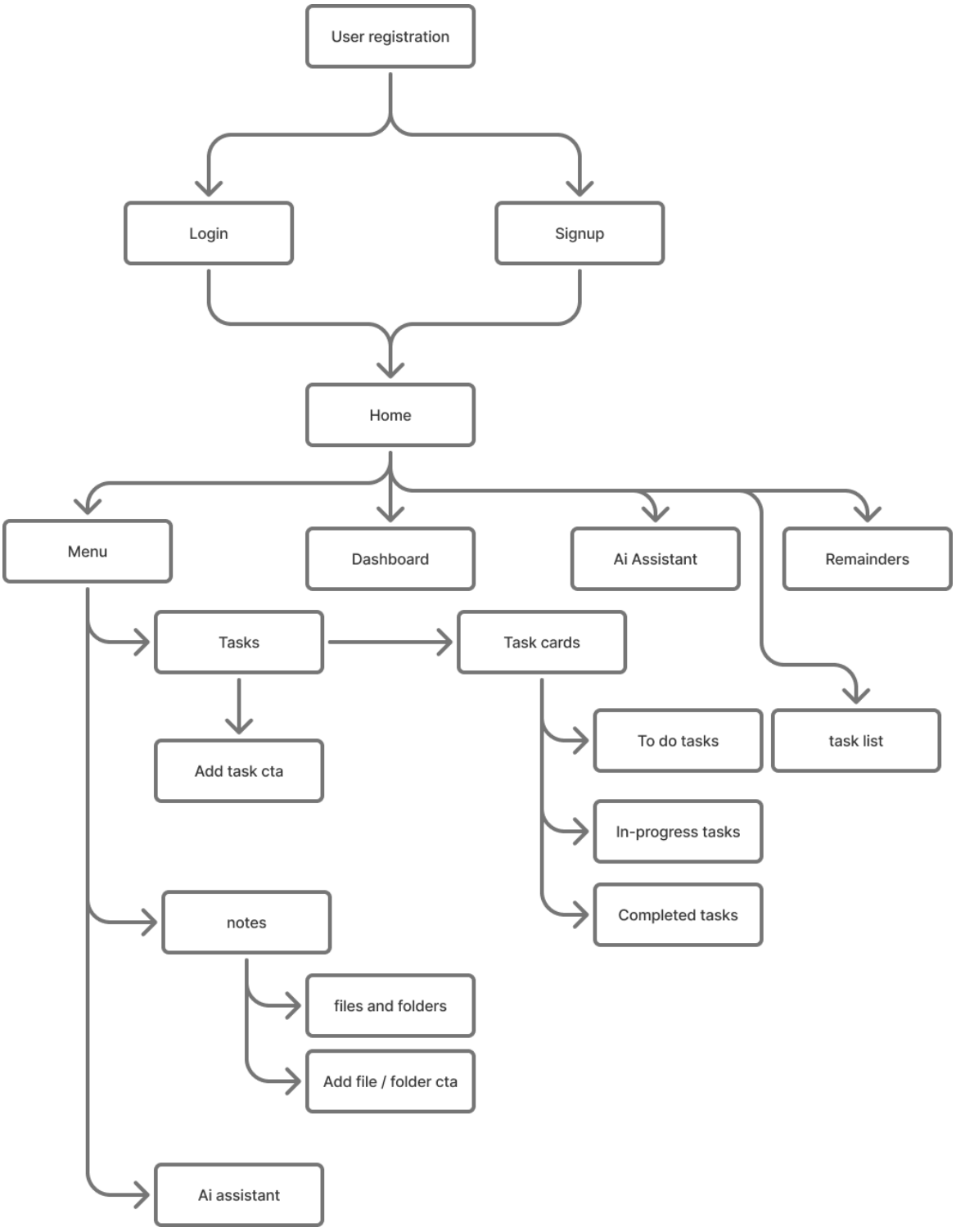

FIGURE 4.4

4.5 Entity Relationship Diagram

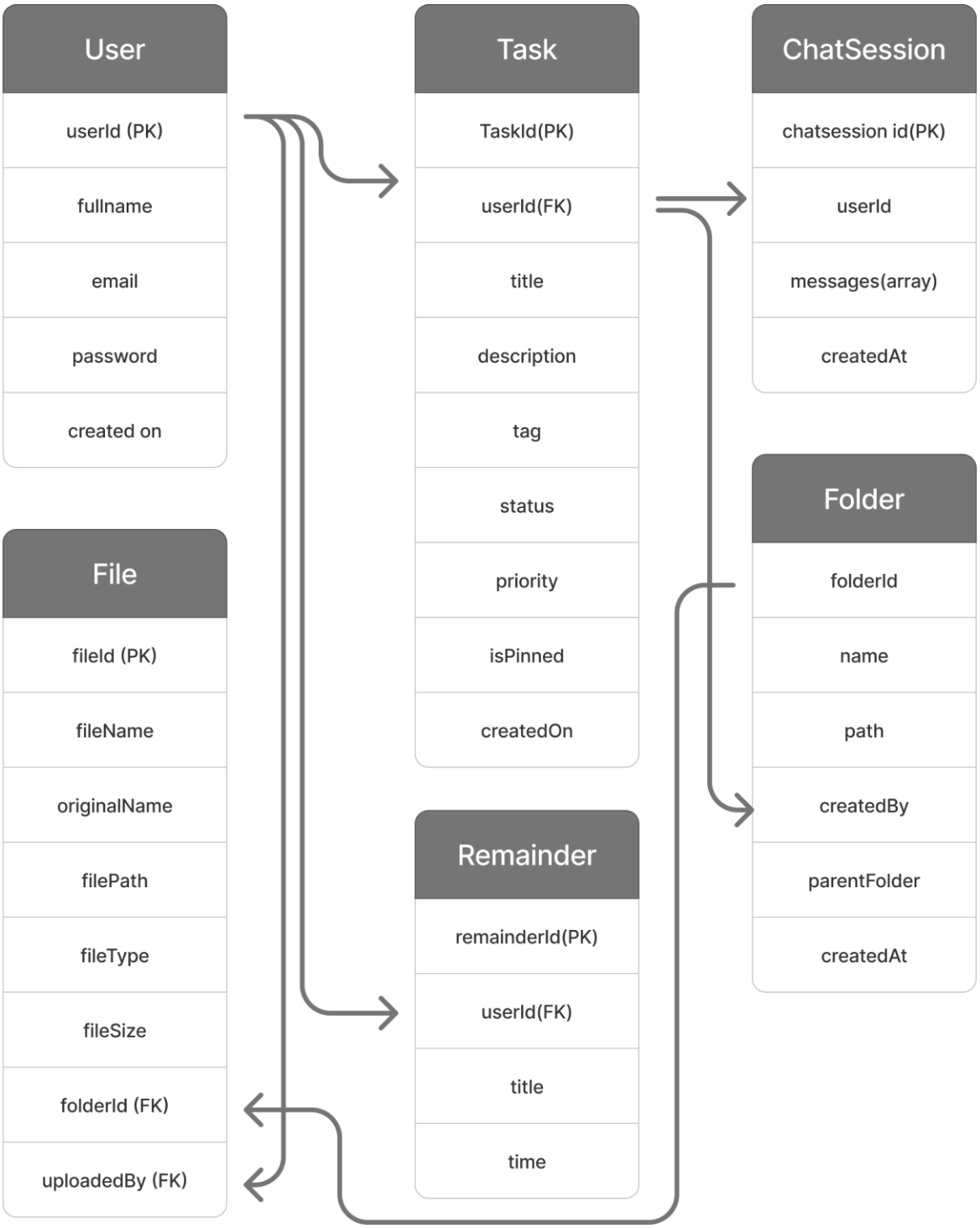

FIGURE 4.5

4.6 Class Diagram

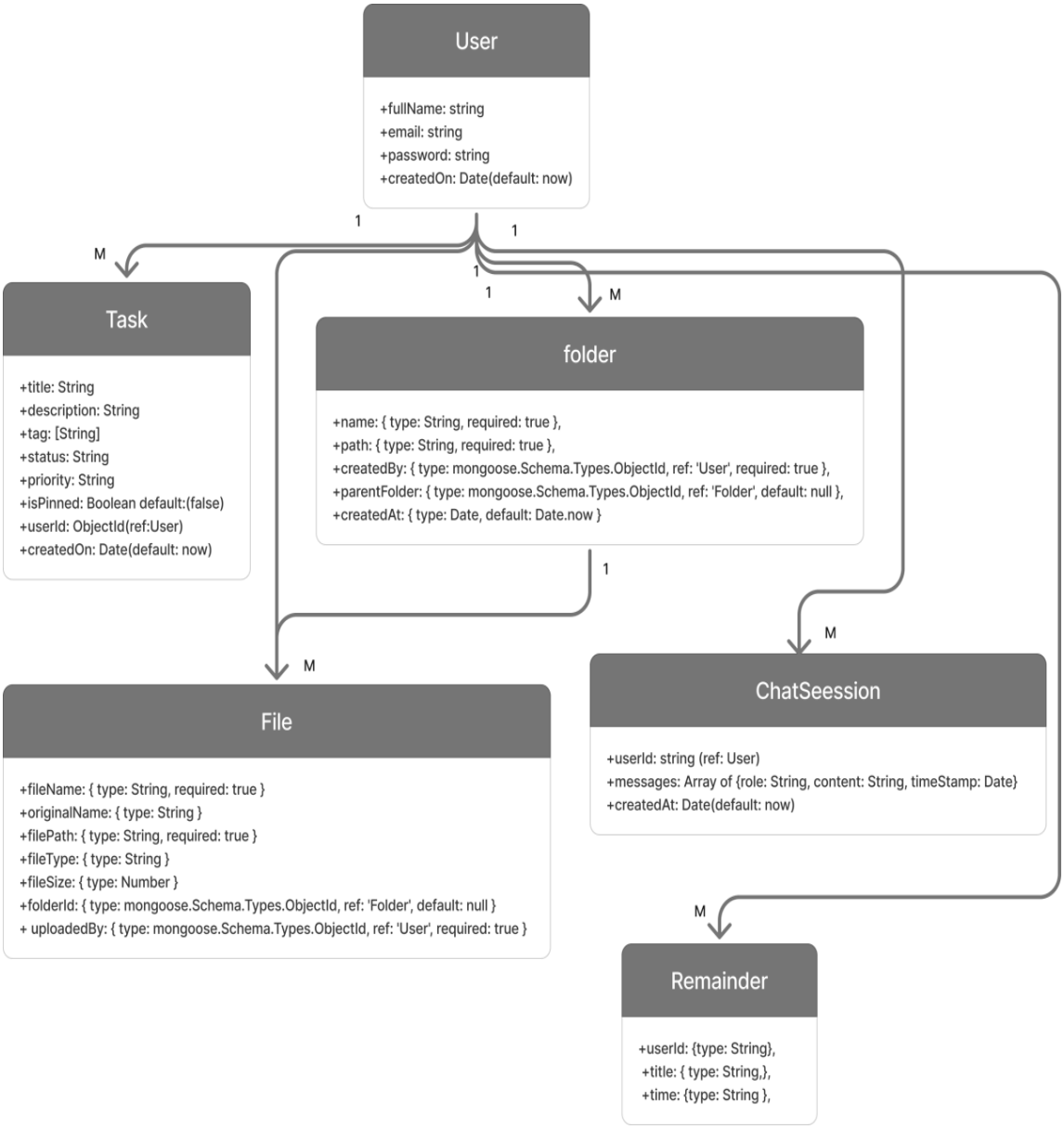

FIGURE 4.6

4.7 Database Design

Users Schema:

The users table stores user-related information and handles authentication, ensuring secure login and role-based access.

- user_id: ObjectId, Primary Key, Unique identifier for each user.
- fullName: String, Stores the full name of the user.
- email: String, Unique, Stores the user's email address for login.
- password: String, Stores the encrypted password for authentication.
- createdOn: Date, Default: now, Stores the account creation date.

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const userSchema =
new Schema({
fullName: {type:
String}, email:
{type: String},
password:
{type:String},
  createdOn: {type: Date, default: new Date().getTime()},
```

```
})

module.exports = mongoose.model("User", userSchema);
```

Task schema:

The task schema stores user tasks with details such as priority, status, and tags.

task_id: ObjectId, Primary Key, Unique identifier for each task.

user_id: ObjectId, Foreign Key, Links the task to specific user.

title: String, Required, Stores the task title.

description: String, Stores task details.

tag: Array[String], Stores associated tags.

status: String, Defines task status (e.g., Pending, Completed).

priority: String, Defines task priority (e.g., High, Medium, Low).

isPinned: Boolean, Default: false, Defines whether a task is pinned.

createdOn: Date, Default: now, Stores task creation date.

```
const mongoose =
require("mongoose");
```

```
const Schema =
mongoose.Schema;

const taskSchema = new
Schema({

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

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

   tag: {type:[String], default: []},

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

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

   isPinned: {type:Boolean, default:false},

   userId: {type: String, required: true},
   createdOn: {type: Date, default: new
Date().getTime()}

})

module.exports = mongoose.model("Task",
taskSchema);
```

Folder and file schema

- **folder_id**: `ObjectId`, Primary Key, Unique identifier for each folder.
- **name**: `String`, Required, Stores folder name.
- **path**: `String`, Required, Stores folder path.

- **createdBy**: `ObjectId`, Foreign Key, Links folder to a user.
- **parentFolder**: `ObjectId`, Nullable, References parent folder.
- **createdAt**: `Date`, Default: `now`, Stores folder creation timestamp.
- **file_id**: `ObjectId`, Primary Key, Unique identifier for each file.
- **fileName**: `String`, Required, Stores file name.
- **originalName**: `String`, Stores original file name.
- **filePath**: `String`, Required, Stores file location.
- **fileType**: `String`, Defines the file type (e.g., `image/png`).
- **fileSize**: `Number`, Stores file size in bytes.
- **folderId**: `ObjectId`, Foreign Key, Links file to a folder.
- **uploadedBy**: `ObjectId`, Foreign Key, Links file to a user. const mongoose = require('mongoose');

// Schema for Files const fileSchema = new mongoose.Schema({ fileName: { type: String, required: true }, // Unique index automatically created originalName: { type: String }, filePath: { type: String, required: true },

fileType: { type: String }, fileSize: { type: Number }, folderId: { type: mongoose.Schema.Types.ObjectId, ref: 'Folder', default: null }, uploadedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }, })

```
// Schema for Folders const folderSchema = new mongoose.Schema({

  name: { type: String, required: true }, path: { type: String, required: true }, createdBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }, parentFolder: { type: mongoose.Schema.Types.ObjectId, ref: 'Folder', default: null }, createdAt: { type: Date, default: Date.now }

});
// Indexes for better query performance fileSchema.index({ folderId: 1, originalName: 1 }); fileSchema.index({ uploadedBy: 1 }); fileSchema.index({ fileName: 1 });

folderSchema.index({ parentFolder: 1 }); folderSchema.index({ path: 1 }); folderSchema.index({ createdBy: 1 }); const File = mongoose.model('File', fileSchema); const Folder = mongoose.model('Folder', folderSchema); module.exports = { File, Folder };
```

Chat schema

The chat session table stores AI chat history for each user.

- **chatSession_id**: `ObjectId`, Primary Key, Unique identifier for each chat session.
- **user_id**: `ObjectId`, Foreign Key, Links chat to a user.
- **messages**: `Array`, Stores an array of chat messages.
- **createdAt**: `Date`, Default: `now`, Stores session creation timestamp.
- const chatSessionSchema = new mongoose.Schema({

```
 const mongoose =
require('mongoose');
const chatSessionSchema =
new
mongoose.Schema({
 userId: {
  type:
mongoose.Schema.Types.Obj
ectId,
  required: true,
  ref: 'User'
 },
 messages: [{
  role: {
   type: String,
   enum: ['user', 'assistant'],
   required: true
```

```
    },
    content: {
      type: String,
      required: true
    },
    timestamp: {
      type: Date,
      default: Date.now
    }
  }],
  createdAt: {
    type: Date,
    default: Date.now
  }
});
const ChatSession = mongoose.model('ChatSessio
n', chatSessionSchema);
module.exports = ChatSession;
```

Data Integrity and constraints

- **Primary Keys**: Each table has a unique identifier (`ObjectId`).
- **Foreign Keys**: Tables reference `user_id` to link user-related data.
- **Indexes**: Used for frequently queried fields (e.g., `email`, `task_id`).

- **Default Values**: Fields like `createdAt` ensure proper data initialization.

Relationship and normalization

- **1NF**: Ensures atomic values in each column.
- **2NF**: No partial dependencies on composite keys.
- **3NF**: No transitive dependencies.

Security and access control

- **Authentication**: User credentials stored securely with hashing (e.g., bcrypt).
- **Role-Based Access**: The system distinguishes between different user roles.
- **Data Privacy**: Encrypted user passwords prevent unauthorized access.

This structured database design ensures efficiency, security, and maintainability, supporting future enhancements with minimal impact on existing functionality.

5. SYSTEM IMPLEMENTATION

5.1 Modules Description:

The **system** is divided into three key modules:

- **User Module** – Handles authentication, user data retrieval, and profile management.
- **Task Module** – Manages task creation, editing, filtering, and retrieval.
- **AI Chat Module** – Provides intelligent task-related assistance.

User Module:

Authentication & User Management:

- **Registration (/create-account):** Allows users to create an account with their full name, email, and password.
- **Login (/login):** Authenticates users and returns an access token for secure API interactions.
- **Get User Data (/get-user):** Retrieves the authenticated user's details**.**

Task Module:

Task CRUD Operations

Add Task (POST /add-task)

- Allows users to create a task with the following fields:
- title (required)
- description (required)
- status (required) – e.g., **TO DO, IN PROGRESS, COMPLETED**
- priority (required) – e.g., **Low, Medium, High**

- tag (optional) – An array of tags
- The task is linked to the authenticated user.

Edit Task (POST /edit-task/:taskId)

- Updates a specific task by allowing changes to:
- title, description, status, priority, tag, isPinned.
- Only the owner of the task can update it.

Delete Task (DELETE /delete-task/:taskId)

- Allows a user to delete a specific task.
- Only the task owner can delete it.

Get All Tasks (GET /get-all-tasks)

- Retrieves all tasks created by the user.
- **Pinned tasks appear first**, followed by other tasks.
- Sorted by isPinned (descending) and then by creation date.

Task Status Filtering

Get To-Do Tasks (GET /get-all-todo-tasks)

- Retrieves all tasks with the status **"TO DO"**.
- Tasks are **sorted by isPinned (descending) and then by priority**.

Get In-Progress Tasks (GET /get-all-inprogress-tasks)

- Retrieves all tasks with the status **"IN PROGRESS"**.
- Tasks are **sorted by isPinned (descending) and then by priority**.

Get Completed Tasks (GET /get-all-completed-tasks)

- Retrieves all tasks with the status **"COMPLETED"**.
- Tasks are **sorted by completion date (most recent first)**.

Additional Task Features

Update Task Pin Status (PUT /update-task-pinned/:taskId)

- Allows users to pin or unpin a task.
- isPinned must be a boolean (true or false).

Search Tasks (GET /search-tasks?query=...)

- Allows users to search for tasks based on title or description.
- Searches are **case-insensitive**.

File Management Module Documentation:

Folder Management

Create Folder (POST /create-folder)

Allows users to create a new folder.

Request Body:

- **name *(required)* – Name of the folder**
- **parentFolder *(optional)* – ID of the parent folder (if creating a subfolder)**
-

Behavior:

- **If parentFolder is provided, the folder is created inside the specified parent folder.**
- **If parentFolder is not provided, the folder is created at the root level.**

Get Subfolders (GET /get-folders/:folderId)

- Retrieves subfolders within a specified folder.

Parameters:

- folderId – ID of the parent folder (use root for top-level folders)

Behavior:

- If folderId is "root", returns all root-level folder
- Otherwise, retrieves subfolders inside the specified folder.

Delete Folder (DELETE /delete-folder/:folderId)

- Deletes a folder and all its contents (subfolders and files).

Parameters:

- folderId – ID of the folder to delete

Behavior:

- Recursively deletes all subfolders and files within the folder.
- Only the owner of the folder can delete it.

File Management

Upload File (POST /upload-files)

Allows users to upload a file to a specific folder.

Request Body:

- title *(optional)* – Custom title for the file
- description *(optional)* – Additional details about the file
- folderId *(required)* – ID of the folder where the file will be uploaded
- file *(required)* – The file to upload

Behavior:

- The file is initially uploaded to a temporary folder and then moved to the appropriate directory.
- If the folder does not exist, the upload is rejected. **Get Files in a Folder (GET /get-files/:folderId)** Retrieves all files within a specified folder.

Parameters:

- folderId – ID of the folder containing the files

Behavior:

- Only files belonging to the authenticated user are retrieved.
- If the folder does not exist, an error is returned.
- **Delete File (DELETE /delete-file/:fileId)** Deletes a specific file.

Parameters:

- fileId – ID of the file to be deleted

Behavior:

- The file is removed from both the database and the file system.
- Only the file owner can delete it.
- If the file is missing in the file system but exists in the database, the database entry is still removed.

Reminder Module:

Reminder CRUD Operations

Get All Reminders (GET /reminders)

Retrieves all reminders for the authenticated user.

Behavior:

- Fetches all reminders belonging to the logged-in user.
- Returns an array of reminders**.**
- Create Reminder (POST /reminders) **Allows users to create a new reminder.**

Request Body:

- title *(required)* – Title of the reminder

- time *(required)* – Date and time for the reminder (ISO format)

Behavior:

- The reminder is linked to the authenticated user.
- The completed field is set to false by default.
- Update Reminder (POST /reminders/:id) **Updates the details of an existing reminder.**

Parameters:

- id – ID of the reminder to update

Request Body:

- title *(optional)* – Updated title for the reminder
- time *(optional)* – Updated time for the reminder

Behavior:

- At least one field (title or time) must be provided.
- Only the owner of the reminder can update it**.** Delete Reminder (DELETE /reminders/:id) **Deletes a specific reminder.**

Parameters:

- id – ID of the reminder to delete

Behavior:

- Only the owner of the reminder can delete it. ○ If the reminder does not exist, an error is returned.

Chat Module:

Chat Session Management

Create a Chat Session (POST /create-chat-session)

Creates a new chat session for the authenticated user.

Request Body:

- **initialMessage *(optional)* – First message from the user**

Behavior:

- If an initialMessage is provided, the AI generates a response.
- Messages are stored within the chat session.
- Send a Message (POST /send-message)

Sends a message to the AI and retrieves a response.

Request Body:

- sessionId *(optional)* – ID of an existing chat session
- message *(required)* – User's message

Behavior:

- If no sessionId is provided, a new chat session is created.
- The AI generates a response using Hugging Face's Mistral model.
- Conversation history is formatted for better AI responses.
- Response is added to the chat session.

5.2 User Interface design

User Module:

The **User Module** is the primary interface for students, allowing them to register, log in, create and manage tasks, set reminders, and track their progress. This module ensures a smooth and intuitive experience for students to stay organized and productive:

Registration

- New users can sign up through a Register Page.
- Fields required: Full Name, Email, Password, Confirm Password.

Validation Checks:

- Email format validation.
- Password strength check (minimum 8 characters, including a number and special character).
- email ID should be unique**.**

Login

- User log in using their Email & Password.
- if the credentials are valid, they are redirected to their Dashboard.
- Forgot Password? Option allows students to reset their password via email verification.

Dashboard

- **The** dashboard is the main interface where user can:
- View an overview of tasks (pending, in progress, completed).
- Access reminders.
- View task history**.**
- Access ai assistant
- Can access their task titles as to do list

Includes count cards displaying:

- Total Tasks Created
- Pending Tasks
- Completed Tasks
- **Upcoming Reminders**
- A search bar and filters help in sorting and finding tasks quickly.

Task Management

- Users can add, edit, delete, and mark tasks as completed.
- Create New Task:
- Task Title
- Description
- Tags for tasks
- Due Date & Time
- Priority Level (Low, Medium, High)
- **Task Status Updates:**
- Users can change the status to "Pending", "In Progress", or "Completed".
- They can also edit their task status.

Reminders

- Users can set reminders for important tasks**.**
- Notifications appear when:
- A task is due soon.
- A reminder is triggered.
- A new feature or update is available.

Task Collaboration (Future Scope)

Users can invite peers to collaborate on tasks.

Shared tasks will show the contributors.

Chatbot Integration

- Users can ask an AI-powered chatbot for task suggestions, productivity tips, and reminders.
- The chatbot assists with time management strategies.

File Management Module

This module ensures students can efficiently store and access important academic files within the platform**.**

Features

Upload Files

- Users can upload supporting documents for their tasks.
- Accepted formats: PDF, DOCX, PNG, JPG, TXT.
- Maximum file size: 5MB per file**.**

Organized File Storage

Files are categorized based on:

Task Attachments

Personal Notes

Shared Files (if collaboration is enabled in the future**).**

Folder structure for better organization**.**

File Preview & Download (Future scope)

Users can preview PDFs and images before downloading.

File Deletion

- Deleted files move to a Trash Bin where they remain for 7 days before permanent deletion.
- Users can restore accidentally deleted files.

Reminder Module

The Reminder Module helps students stay on track with their academic and personal tasks.

Features

Set Reminders

- Users can set reminders for:
- Task deadlines
- Study sessions
- Personal events (e.g., project submissions)
- Reminders can be one-time.

Reminder Dashboard

- Displays all upcoming reminders in a list format.

Chat Module

The Chat Module enables students to interact with an AI assistant for productivity support.

Features

Start a Chat Session

Users can start a new conversation with the AI.

AI can help with:

- Task prioritization
- Study techniques
- Productivity tips

Message History

- Past conversations are saved.
- Users can revisit previous chats**.**

AI Responses

- The AI assistant provides detailed responses.
- If an answer is unclear, users can request clarifications**.**

- Students can use Ai to improve their productivity by asking for smart task titles and setting productive reminders.

Chat UI Enhancements

- Wider chat window for better readability.
- Clock icon to view chat history.
- Star button to open AI chat, turning into a close button (✕) when active.

Overall System Features:

- Dashboard Insights
- Displays student productivity statistics:
- Tasks completed
- Tasks pending
- Reminder trends

6. SYSTEM TESTING

System testing ensures that the entire **UpNext Web App** functions as expected. It verifies the integration of all components, evaluates system performance, and ensures compliance with requirements. The testing process includes **test cases, unit testing, integration testing, performance testing, and security testing** to validate different aspects of the system.

6.1 Test Cases

- Test cases are designed to validate different functionalities of the Student Task Management Web App. Each test case includes test inputs, expected outcomes, and actual results.
- Sample Test Cases:
- User Registration: Verify if a new user can register successfully with valid details.
- Login Authentication: Ensure users can log in with valid credentials. • Task Creation: Verify that users can create a new task with a title, description, and due date.
- Task Editing: Ensure that users can modify task details and save changes. • Task Deletion: Test whether users can delete tasks and restore them from the Trash.
- Task Completion: Verify that users can mark tasks as completed.
- Reminder Notifications: Ensure users receive notifications for upcoming deadlines.
- Task Filtering: Validate that users can filter tasks based on categories like "Pending," "Completed," or "Overdue."
- Dashboard Statistics: Confirm that the dashboard accurately displays total tasks, completed tasks, and upcoming deadlines.
- Task Sharing: Test whether users can share tasks with team members (if applicable).

6.2 Unit Testing

- Unit testing focuses on testing individual components of the system separately to verify their correctness.

- Tested Components:
- User Module: Testing functions related to user registration, login, password updates, and profile management.
- Task Management Module: Testing task creation, editing, deletion, and status updates. To test that the tasks are retrieved efficiently from the database.
- Reminder & Notification Module: Verifying that notifications trigger at the correct times.
- Notes management module: Testing files and folders are uploaded and retrieved from the database.
- Database Integration: Ensuring correct data storage and retrieval for user tasks and progress tracking.
- Tools Used: Postman.

6.3 Integration Testing

- Integration testing ensures smooth interaction between different modules. The following tests were conducted:
- User to Task Management: Ensuring users can create, update, and delete tasks.
- Task to Notification System: Verifying that task deadlines trigger notifications.
- User to Dashboard: Checking whether dashboard statistics update correctly based on user activity.
- Task Sharing (if applicable): Ensuring that shared tasks appear correctly for assigned users.

6.4 Performance Testing

- Performance testing ensures that the system operates efficiently under various conditions.

- Load Testing: Simulating multiple concurrent users adding and updating tasks to measure response time.
- Stress Testing: Testing system performance under peak loads to identify potential bottlenecks.
- Database Performance: Evaluating query execution times to optimize task retrieval and updates.

6.5 Security Testing

- Security testing ensures data protection and system integrity.
- User Authentication: Verifying role-based access control to prevent unauthorized access.
- Data Encryption: Ensuring passwords and sensitive data are securely stored and transmitted.
- Injection Attacks: Testing against SQL injection and cross-site scripting (XSS) vulnerabilities.
- Session Management: Ensuring proper handling of user sessions to prevent unauthorized access.

7. CONCLUSION

7.1 Results

The UpNext Web App was successfully developed and tested, ensuring efficiency, security, and a seamless user experience. Below are the key outcomes:

Secure User Management: The system enables secure user registration and login, ensuring that only authenticated users can access and manage their tasks.

Efficient Task Management: Users can easily create, edit, delete, and track their tasks, improving personal productivity and time management.

Seamless Task Filtering and Organization: The app allows users to categorize tasks as Pending, Completed, or Overdue, providing a structured workflow.

Real-time Notifications: Users receive timely notifications for upcoming deadlines and overdue tasks, reducing missed deadlines and improving productivity.

User-Friendly Interface: The app is designed with a modern UI/UX approach, ensuring an intuitive and seamless experience for students.

Data Security and Integrity: Sensitive user data is encrypted, and proper session management techniques are implemented to prevent unauthorized access.

Mobile-Friendly and Responsive Design: The interface is optimized for both desktop and mobile devices, ensuring accessibility across different screen sizes.

7.2 Future Enhancements

The system can be further improved with the following enhancements:

Automated Email & Push Notifications: Implementing email and push notifications for task reminders will enhance user engagement.

Task Collaboration Feature: Adding functionality for students to share tasks with friends or group members for collaborative projects.

AI-Powered Task Prioritization: Using machine learning to suggest priority levels for tasks based on deadlines, workload, and past user behavior**.**

Offline Mode: Allowing users to view and edit tasks offline, with automatic syncing when an internet connection is restored.

Dark Mode Support: Introducing a dark mode toggle for better accessibility and reduced eye strain.

Task Import & Export: Providing options to import tasks from CSV files or export task reports for better tracking**.**

Integration with Calendar Apps: Syncing with Google Calendar or Outlook for better task scheduling and deadline management**.**

Gamification for Motivation: Adding streaks, badges, and rewards for task completion to keep students motivated.

With these future enhancements, the UpNext Web App can become an even more powerful and engaging productivity tool for students. The current implementation successfully streamlines task organization, deadline tracking, and personal productivity, making it a valuable solution for student task management.

8. REFERENCE

Frontend Development

React.js

React.js is a JavaScript library used for building interactive user interfaces. Developed by Facebook (Meta), it follows a component-based architecture, allowing developers to create reusable UI components. React's virtual DOM improves performance, making it an ideal choice for building fast and scalable web applications.

React.js Documentation: https://react.dev/

Bootstrap

Bootstrap is a popular front-end framework that provides pre-designed components and a responsive grid system, making web development faster and easier. Created by Twitter, Bootstrap includes CSS and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components. It allows developers to build mobilefirst and responsive web applications efficiently.

Bootsrap- **Bootstrap Documentation**

CSS (Cascading Style Sheets)

CSS is a style sheet language used to define the visual appearance of web applications. It controls layout, colors, fonts, and animations, enabling responsive design across different devices.

CSS Documentation: https://developer.mozilla.org/en-US/docs/Web/CSS

Node.js Documentation: https://nodejs.org/en/docs

Express.js

Express.js is a minimal and flexible web framework for Node.js. It simplifies backend development by providing built-in routing, middleware handling, and API support.

Express.js Documentation: https://expressjs.com/

JWT (JSON Web Token) for Authentication

JWT is a secure token-based authentication method used to ensure that users can safely log in and manage their sessions. It provides a stateless authentication system by generating tokens that verify user identities.

JWT Guide: https://jwt.io/introduction/

Database & Data Storage

MongoDB

MongoDB is a NoSQL database used for storing structured and semi-structured data. It offers flexibility, scalability, and high performance, making it ideal for dynamic applications like task management systems.

MongoDB Documentation: https://www.mongodb.com/docs/manual/

Stack Overflow: https://stackoverflow.com/

MDN Web Docs

MDN Web Docs provides official documentation and tutorials for web technologies like HTML, CSS, JavaScript, and Web APIs.

MDN Web Docs: https://developer.mozilla.org/en-US/

GeeksforGeeks

GeeksforGeeks is an educational platform offering tutorials on data structures, algorithms, web development, and competitive programming. It is widely used for coding practice and interview preparation.

GeeksforGeeks: https://www.geeksforgeeks.org/

9. APPENDICES

9.1 SOURCE PROGRAM

Homepage.js:

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import Sidebar1 from '../../components/Sidebar/Sidebar1';
import Chat from '../../components/Chat/Chat';
import Navbar from '../../components/Navbar/Navbar';
import axiosInstance from '../../utils/axiosInstance';
import moment from 'moment';
import ToDolist from '../../components/Todo list/ToDolist';
import "../../App.css";
import Remainder from '../../components/Remainder/Remainder';
const Home = () => {
    const [userInfo, setUserInfo] = useState("");
    const [taskStats, setTaskStats] = useState({
        total: 0,
        todo: 0,
        inProgress: 0,
        completed: 0
    });
    const navigate = useNavigate();
    const currentDate = new Date();

    const getUserInfo = async () => {
        try {
            console.log("Fetching user info...");
```

```
        console.log("Token:", localStorage.getItem("token"));

        const response = await axiosInstance.get("/get-user");
        console.log("User response:", response.data);

        if(response.data && response.data.user) {
            setUserInfo(response.data.user);
        }
    } catch(error) {
        console.error("Error details:", error);
        console.error("Response status:", error.response?.status);
        console.error("Response data:", error.response?.data);

        if(error.response?.status === 401) {
            localStorage.clear();
            navigate("/login");
        }
    }
};

const AllTasks = async () => {
    try {
        const response = await axiosInstance.get("/get-all-tasks");
        console.log("Raw tasks response:", response); // Debug raw response
        console.log("Tasks data:", response.data); // Debug data

        // Check the actual structure of your response
        let tasks = [];
```

```
        if (response.data && response.data.tasks) {
            tasks = response.data.tasks; // If tasks are nested under 'tasks' key
        } else if (Array.isArray(response.data)) {
            tasks = response.data; // If response.data is directly the array
        } else {
            console.error("Unexpected response structure:", response.data);
            return;
        }

        console.log("Processed tasks array:", tasks); // Debug processed array

        // Now we're sure tasks is an array
        setTaskStats({
            total: tasks.length,
            todo: tasks.filter(task => task.status === 'TO DO').length,
            inProgress: tasks.filter(task => task.status === 'IN PROGRESS').length,
            completed: tasks.filter(task => task.status === 'COMPLETED').length
        });

    } catch (error) {
        console.error("Error fetching tasks:", error);
    }
};
useEffect(() => {
    AllTasks();
    getUserInfo();
}, []);
const getGreeting = () => {
```

```
    const hour = currentDate.getHours();
    if (hour < 12) return 'Good Morning';
    if (hour < 18) return 'Good Afternoon';
    return 'Good Evening';
  };
  return (
    <div className='d-flex flex-row' >
      <Sidebar1 bg="home"/>
      <div className='d-flex flex-column w-100'>
        <Navbar name="Home" userInfo={userInfo}/>
        <div className='d-flex flex-column px-4 pt-4 pb-2' style={{ fontFamily: "'Poppins', sans-serif" }}>
          <h1 className='fw-bold lh-1'>{getGreeting()}, {userInfo?.fullName || ''}</h1>
          <p className='text-secondary'>{moment(currentDate).format('dddd, Do MMMM YYYY')}</p>
        </div>
        <div className='d-flex flex-column w-100 gap-2'>
          <div className='d-flex flex-row w-100 gap-4 px-4'>
            <div className=' p-3 py-4 d-flex flex-column align-items-center rounded gap-2' style={{ width: '18rem', backgroundColor:"#7161EF" }}>
              <h3 className='text-light'>Total tasks</h3>
              <h1 className='text-light'>{taskStats.total}</h1>
            </div>
            <div className=' p-3 py-4 d-flex flex-column align-items-center rounded gap-2' style={{ width: '18rem', backgroundColor:"#7161EF" }}>
              <h3 className='text-light'>To-do tasks</h3>
              <h1 className='text-light'>{taskStats.todo}</h1>
            </div>
```

```
                <div className=' p-3 py-4 d-flex flex-column align-items-center rounded gap-2' style={{ width: '18rem', backgroundColor:"#7161EF" }}>
                    <h3 className='text-light'>In-progress tasks</h3>
                    <h1 className='text-light'>{taskStats.inProgress}</h1>
                </div>
                <div className=' p-3 py-4 d-flex flex-column align-items-center rounded gap-2' style={{ width: '18rem', backgroundColor:"#7161EF" }} >
                    <h3 className='text-light'>Completed tasks</h3>
                    <h1 className='text-light'>{taskStats.completed}</h1>
                </div>
            </div>

            <div className='d-flex flex-row gap- flex-wrap'>
                <div className='p-4'>
                    <ToDolist />
                </div>
                <div>
                    <Remainder />
                </div>
            </div>
        </div>
        <Chat />
      </div>
    </div>
  );
};

export default Home;
```

Task page

```
import React, { useEffect, useState } from 'react';
import Sidebar1 from '../../components/Sidebar/Sidebar1';
import Navbar from '../../components/Navbar/Navbar';
import AddEditTask from '../AddEditTasks/AddEditTask';
import Modal from 'react-modal';
import { useNavigate } from 'react-router-dom';
import axiosInstance from '../../utils/axiosInstance';
import moment from "moment";
import TodoTaskCard from '../../components/TaskCard/TodoTaskCard';
import InProgressTaskCard from '../../components/TaskCard/InProgressTaskCard';
import Completed from '../../components/TaskCard/Completed';
import { Toast } from '../../components/ToastMessage/Toast';
import '../../App.css';
import Chat from '../../components/Chat/Chat';
// Required for accessibility
// Modal.setAppElement('#root');


const Tasks = () => {
  const [openAddEditModal, setOpenAddEditModal] = useState({
    isShown: false,
    type: "add",
    data: null,
```

```
});

const [isSearch, setIsSearch] = useState(false);

const [showToast, setShowToast] = useState({
  isShown : false,
  message: "",
  type : ""
})

const [userInfo, setUserInfo] = useState("");
const [allTasks, setAllTasks] = useState([]);
const filteredTasks = (status) => allTasks.filter(task => task.status === status);
const navigate = useNavigate();
const handleEdit = (taskDetails) => {
  setOpenAddEditModal ({ isShown: true, data: taskDetails, type : "edit" });
}
const ShowToastMsg = (message, type) => {
  setShowToast({
    isShown: true,
    message,
    type
  })
}
```

```
const handleCloseToast = () => {
  setShowToast({
    isShown: false,
    message: "",
    type:""
  })
//get user info
const getUserInfo = async () => {
  try {
    console.log("Fetching user info..."); // Debug log
    console.log("Token:", localStorage.getItem("token")); // Check if token exists
    const response = await axiosInstance.get("/get-user");
    console.log("User response:", response.data); // Debug log
    if(response.data && response.data.user) {
      setUserInfo(response.data.user);
    }
  } catch(error) {
    console.error("Error details:", error); // More detailed error logging
    console.error("Response status:", error.response?.status);
    console.error("Response data:", error.response?.data);
    if(error.response?.status === 401) {
      localStorage.clear();
      navigate("/login");
    }
```

```
  }
}
//get all tasks
const getAllTasks = async() => {
  try{
    const response = await axiosInstance.get("/get-all-tasks");
    if(response.data && response.data.tasks){
      setAllTasks(response.data.tasks);
    }
  }catch(error){
    console.log("An unexpected error occurred. Please try again");
  }
}
//delete tasks
const deleteTask = async (data) => {
  const taskId = data._id;
  try {
      const response = await axiosInstance.delete(`/delete-task/${taskId}`);
      if (response.data && !response.data.error) {
        ShowToastMsg("Task Deleted Successfully!", "delete")
          getAllTasks();
          onClose();
      } else {
          setError(response.data.message || "Failed to delete task");
```

```
        }
    } catch (error) {
        console.error("Error deleting task:", error);
        if (error.response?.data?.message) {
            setError(error.response.data.message);
        } else {
            setError("An unexpected error occurred while deleting the task");
        }
    }
}
const handleStatusChange = (taskId, newStatus) => {
  const updatedTasks = allTasks.map(task =>
     task._id === taskId ? { ...task, status: newStatus } : task
  );
  setAllTasks(updatedTasks); // Update the state with the new task status
};
  const onSearchTask = async (query) => {
    try{
      const response = await axiosInstance.get("/search-tasks", {
        params: {query},
      })
      if(response.data && response.data.tasks) {
        setIsSearch(true);
        setAllTasks(response.data.tasks);
```

```
    }
  }catch(error){
    console.log(error);
  }
 }
 const handleClearSearch = () => {
  setIsSearch(false);
  getAllTasks();
 }
 const updateIsPinned = async (taskData) => {
  const taskId = taskData._id
  try {
    const response = await axiosInstance.put("/update-task-pinned/"+taskId, {
      "isPinned" : !taskId.isPinned,
    });
    if (response.data && response.data.task) {
      ShowToastMsg("Task Updated Successfully!", "edit")
      getAllTasks();
      onClose();
    }
   } catch (error) {
    console.log(error);
   }
};
```

```
useEffect(() => {
  getAllTasks();
  getUserInfo();
}, []);
// Function to open the modal
const handleOpenModal = () => {
  setOpenAddEditModal({
    isShown: true,
    type: "add",
    data: null
  });
};
// Function to close the modal
const handleCloseModal = () => {
  setOpenAddEditModal({
    isShown: false,
    type: "",
    data: null
  });
};
return (
  <div className="d-flex flex-row " style={{ background: '#f4f7fa' }}>
    {/* Sidebar Section */}
    <Sidebar1 bg="tasks"/>
```

```
      {/* Main Content Section */}

     <div className='d-flex flex-column w-100'>

       <div  style={{ background: '#fff' }}><Navbar name="Tasks" userInfo={userInfo}
onSearchTask={onSearchTask} handleClearSearch={handleClearSearch} /></div>

       {/* Title and Add Button Section */}

      <div className='d-flex flex-row justify-content-between align-items-center px-4 pt-4 p-0'>

        <h2 className='fw-bold'>All Tasks</h2>

        <button

          className='btn-custom'

          onClick={handleOpenModal}

          style={{ width: '8rem' }}

        >

          Add Task

        </button>

      </div>

      {/* Tasks Display Section */}

      <div className=' p-3 px-3 d-flex flex-row gap-3'>

        <div className='d-flex flex-column gap-3 p-3 rounded mw-75 ' style={{ background: "
}}>

          <div className='p-2 px-3 rounded-1 fw-bold fs-6 '  style={{ width: '18rem',
backgroundColor:"#", color:"#7161EF", border: "1px solid #C2C0FF" }}>

TO DO

          </div>

           <div className='d-flex flex-column gap-3  '>

           {filteredTasks("TO DO").map(item => (
```

```
            <TodoTaskCard
              key={item._id}
              taskId={item._id}
              title={item.title}
              date={moment(item.createdOn)}
              description={item.description}
              tags={item.tag}
              status={item.status}
              priority={item.priority}
              isPinned={item.isPinned}
              onEdit={() => handleEdit(item)}
              onDelete={() => deleteTask(item)}
              onPinTask={() => updateIsPinned(item)}
              onStatusChange={handleStatusChange}
            />
          ))}
          </div>
        </div/>
        <div className='d-flex flex-column gap-3 p-3 rounded mw-75 ' style={{ background: "
}}>
        <div className='p-2 px-3 rounded-1 fw-bold fs-6 '  style={{ width: '18rem',
backgroundColor:"#", color:"#7161EF", border: "1px solid #C2C0FF" }}>
        IN PROGRESS
          </div>
           <div className='d-flex flex-column gap-3 h-100 '>
```

```
            {filteredTasks("IN PROGRESS").map(item => (
              <InProgressTaskCard
              key={item._id}
              taskId={item._id}
              title={item.title}
              date={moment(item.createdOn)}
              description={item.description}
              tags={item.tag}
              status={item.status}
              priority={item.priority}
              isPinned={item.isPinned}
              onEdit={() => handleEdit(item)}
              onDelete={() => deleteTask(item)}
              onPinTask={() => updateIsPinned(item)}
              onStatusChange={handleStatusChange}
              />
            ))}
          </div>
        </div>
        <div className='d-flex flex-column gap-3 p-3 rounded mw-50 ' style={{ background: "
}}>
          <div className='p-2 px-3 rounded-1 fw-bold fs-6 '  style={{ width: '18rem',
backgroundColor:"#", color:"#7161EF", border: "1px solid #C2C0FF" }}>
            COMPLETED
```

```
    </div>
     <div className='d-flex flex-column gap-3 h-100 '>
      {filteredTasks("COMPLETED").map(item => (
         <Completed
           key={item._id}
           title={item.title}
           date={item.createdOn}
           description={item.description}
           tags={item.tag}
           status={item.status}
           priority={item.priority}
           isPinned={item.isPinned}
           onEdit={() => handleEdit(item)}
           onDelete={() => deleteTask(item)}
           onPinTask={() => updateIsPinned(item) }
         />
       ))}
     </div>
  </div>
</div>
{/* Modal Section */}
<Modal
  isOpen={openAddEditModal.isShown}
  onRequestClose={handleCloseModal}
```

```
  style={{
    overlay: {
      backgroundColor: "rgba(0,0,0,0.5)",
    },
    content: {
      borderRadius: "10px",
      padding: "20px",
      maxWidth: "500px",
      margin: "auto",
    },
  }}
  contentLabel="Add/Edit Task Modal"
>
  <AddEditTask
    name = "New task"
    type={openAddEditModal.type}
    taskData={openAddEditModal.data}
    onClose={() => {
      setOpenAddEditModal({isShown: false, type: "add", data: null});
    }}
    getAllTasks={getAllTasks}
    ShowToastMsg={ShowToastMsg}
  />
</Modal>
```

```
        <Toast
          isShown={showToast.isShown}
          message={showToast.message}
          type={showToast.type}
          onClose={handleCloseToast}
        />
      </div>
      <div className='chat-component'>
        <Chat />
      </div>
    </div>
  );
}
export default Tasks;
```

9.2 OUTPUT SCREEN

9.2.1 LOGIN PAGE

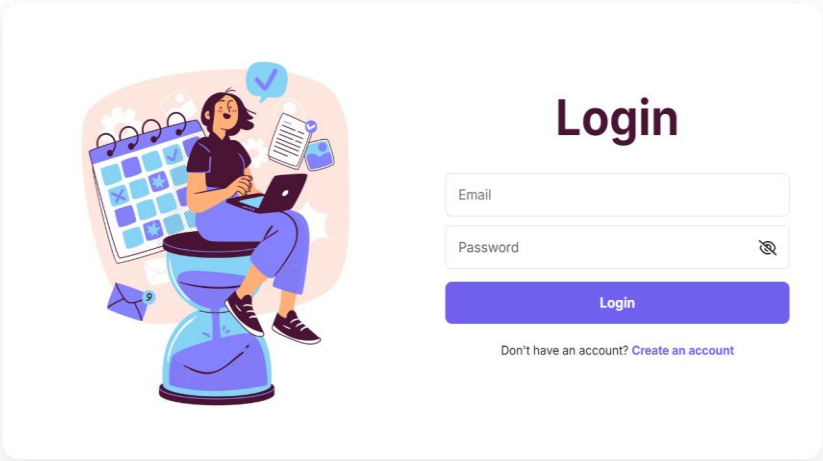

Figure 9.2.1

9.2.2 SIGNUP PAGE

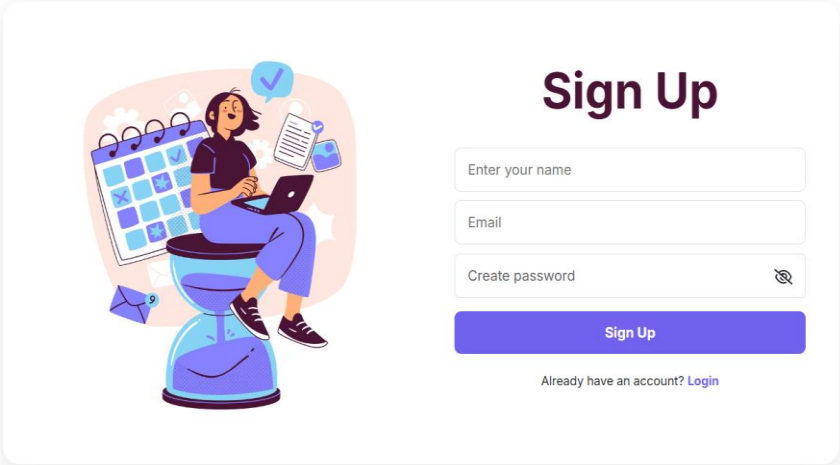

Figure 9.2.2

9.2.3 HOME PAGE

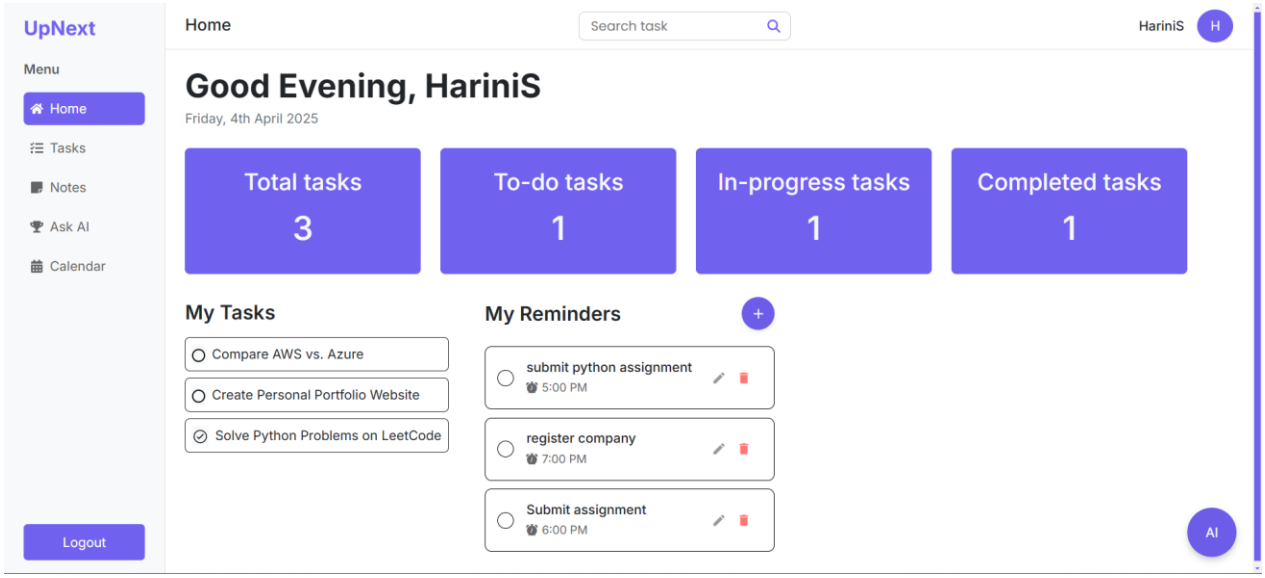

Figure 9.2.3

9.2.4 TASK PAGE

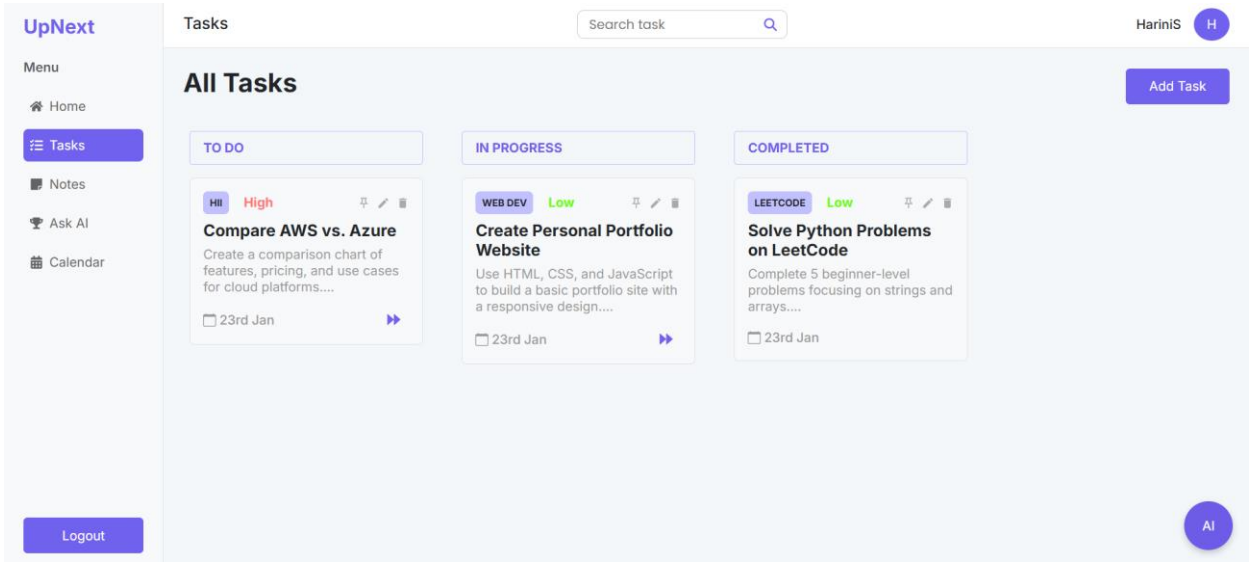

Figure 9.2.4

9.2.5 NOTES PAGE

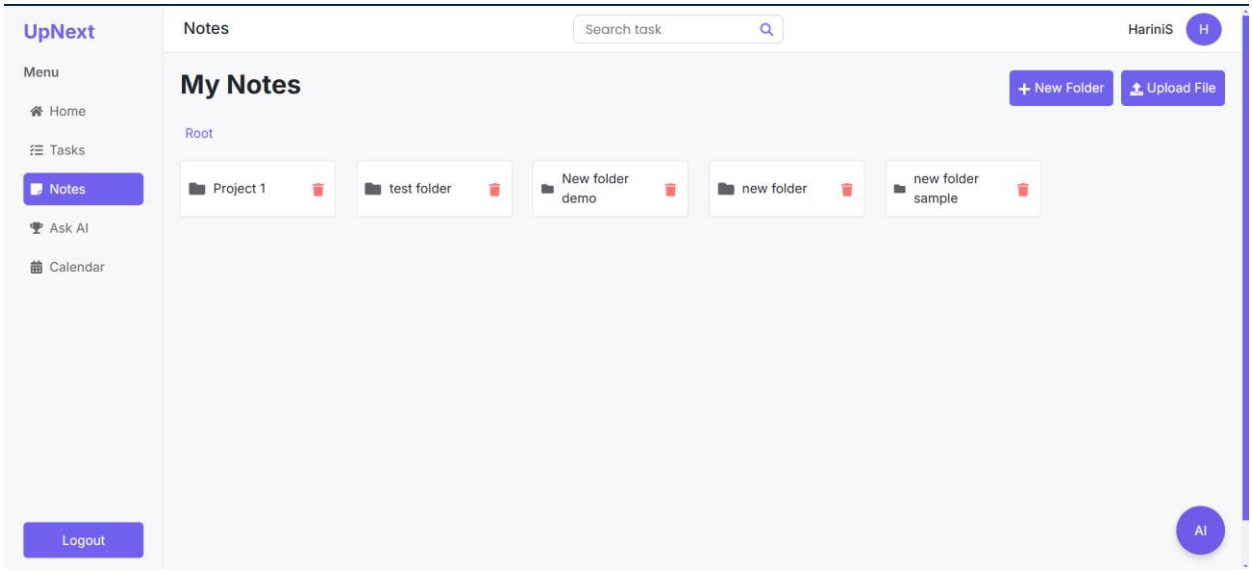

Figure 9.2.5

9.2.6 AI MODAL

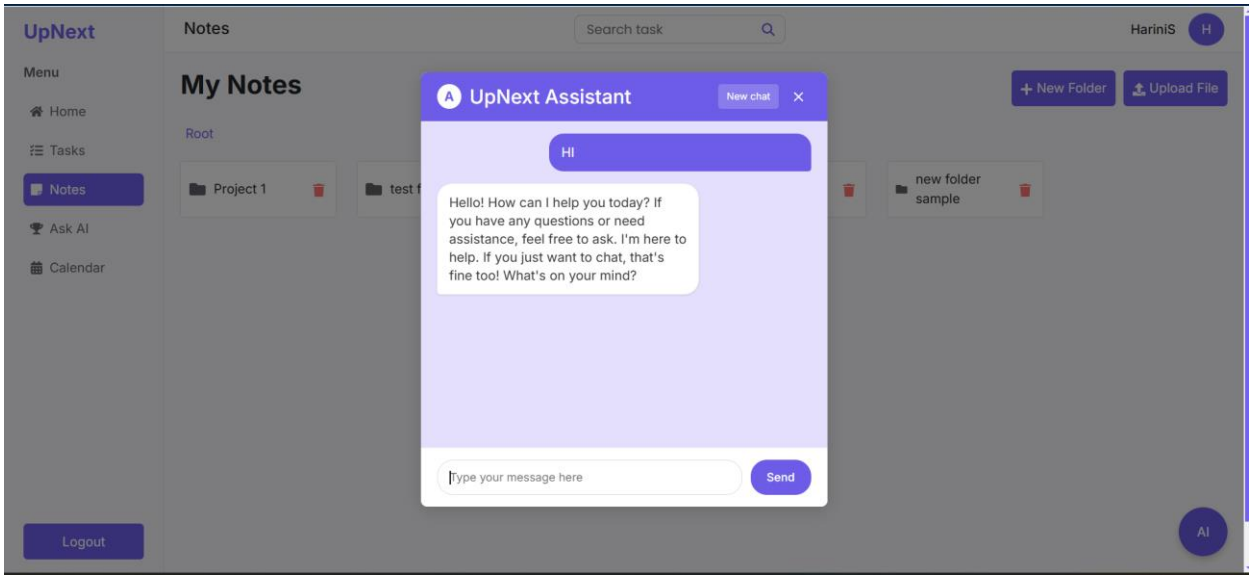

Figure 9.2.6

9.2.7 ADD TASK MODAL

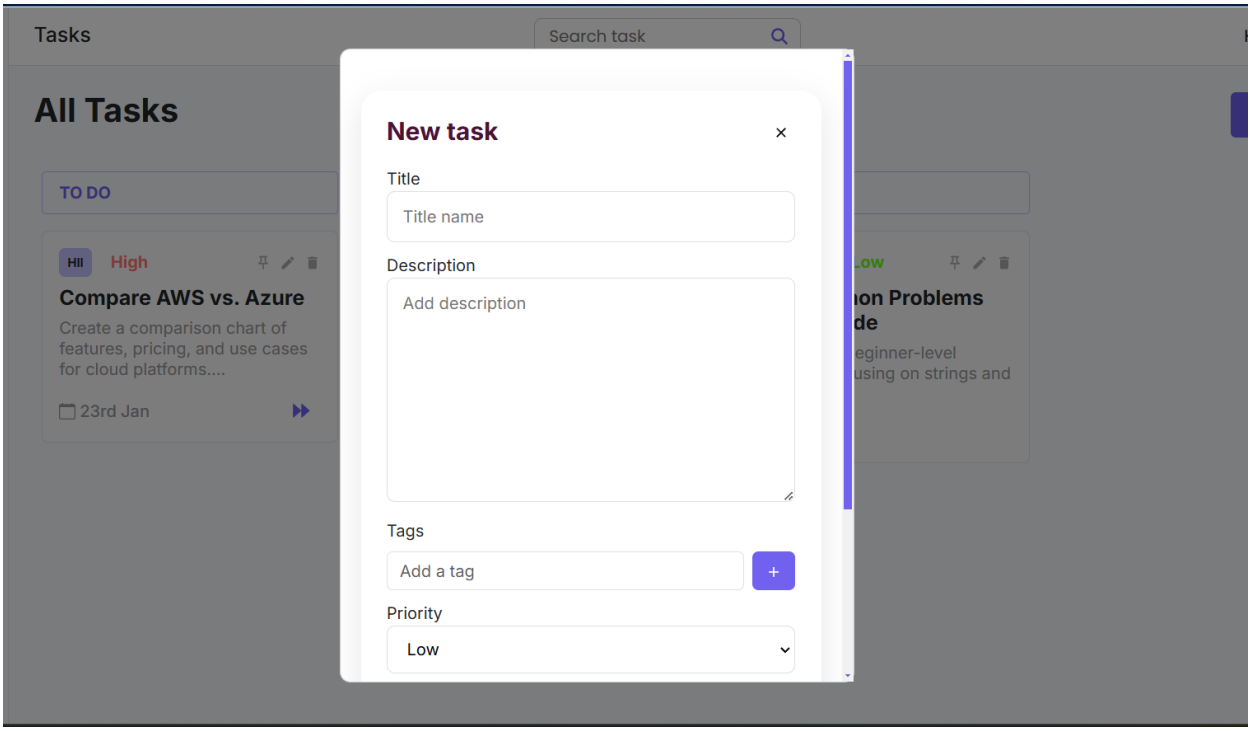

Figure 9.2.7