**IBM – SUMMER INTERNSHIP REPORT**

**FULL STACK PROJECT REPORT ON**

**“TaskFlow: A Team-Based Task Management and Productivity Tracker”**

Submitted in fulfillment of the

Summer Internship Program at IBM (2025)

**Submitted By:**

Name: Viraj Keshwala

Enrollment No: 220280107057

Degree: B.E. Computer Engineering

College: L.D. college of engineering , Ahmedabad

**Guided By:**

Mr. Jaivik Panchal

Submission Date: July 2025

**Certificate:**

**Acknowledgment:**

I would like to express my sincere gratitude to **IBM** for providing me with the opportunity to undertake this summer internship and work on the project titled **“TaskFlow: A Team-Based Task Management and Productivity Tracker”** This internship has been a highly enriching experience and has contributed significantly to my practical knowledge and skills in full stack development.

I extend my heartfelt thanks to my project mentor **Mr. Jaivik Panchal** for their continuous guidance, valuable feedback, and support throughout the internship. Their insights and encouragement have been instrumental in the successful completion of this project.

I am also thankful to the entire IBM team and colleagues who made my internship experience smooth, collaborative, and intellectually stimulating.

I would also like to thank my college and the Department of Computer Engineering for their constant support and encouragement.

Lastly, I am deeply grateful to my family and friends for their unwavering support and motivation throughout this journey.

**CONTENT**

**1. Abstract .............................................................................. 1**

**2. Objective .......................................................................... 2**

**3. System Architecture ....................................................... 3**

3.1 Architecture Diagram

3.2 Component Interaction & Technology Flow

**4. Technology Stack ............................................................ 4**

**5. Modules / Features ....................................................... 5**

5.1 Module Descriptions

5.2 Technologies Used

**6. Frontend Development .................................................. 6**

6.1 Framework & Layout

6.2 UI/UX Design

6.3 Sample Pages / Screenshots

**7. Backend Development ................................................... 7**

7.1 Framework & API Structure

7.2 Routing, Controllers

7.3 Authentication / Authorization

**8. Database Design ............................................................ 8**

8.1 DBMS Used

8.2 ER Diagram & Schema

8.3 Sample Data & Relationships

**9. Data Flow Diagrams ....................................................... 9**

9.1 Level 0 and Level 1 DFDs

- Encryption, JWT

- Input Validation

**10. Limitations & Future Enhancement ................................................................. 12**

**11. Annexures .................................................................. 15**

16.1 Code Snippets

16.2 API Docs

16.3 GitHub Repo Link

**12. References ................................................................ 16**

**1. Abstract**

#### **TaskFlow** is a full-stack role-based task management web application developed as part of my summer internship. The core objective of this project is to streamline task delegation, progress tracking, and team collaboration within organizations by providing a centralized platform for administrators, managers, and team members.

#### In this system, **administrators** can manage users and projects, **managers** can create projects and assign tasks to multiple team members, and **team members** can view and update the status of their assigned tasks. The platform supports role-based access control, project-based task grouping, and priority-based status updates to ensure organized workflows and effective accountability.

#### Built with a modern tech stack—**HTML, CSS, JavaScript** for the frontend and **Node.js, Express, MongoDB** for the backend—the application ensures responsive design, secure authentication using JWT, and clean API-based communication. TaskFlow aims to boost productivity through structured task handling while maintaining simplicity, scalability, and security.

#### • **What the Project Is:**

#### This project is a **role-based task and project management platform** that allows:

#### **Admins** to manage users and oversee all projects and tasks,

#### **Managers** to create projects and assign tasks to team members,

#### **Team members** to view their assigned tasks and update task status.

#### The application provides a centralized system for organizing team activities, assigning responsibilities, tracking task progress, and improving workflow efficiency within organizations or teams.

#### • Why It Was Developed:

The system was developed:

* To understand the workflow of real-world Project Management.
* To gain hands-on experience in **full stack development** using core **JavaScript** for the frontend and **Node.js, Express.js, MongoDB** for the backend.
* To build a scalable, maintainable, and secure solution that addresses common project management problem.

#### • Key Features:

* **Role-based access:** Admin, Manager, and Team Member
* **Project and task management:** Create projects, assign tasks, and track status (To-do, In-progress, Done)
* **JWT-based authentication and password encryption** for secure user access
* **Task assignment to team members** with project linkage
* **Dynamic dashboards** for each role with customized views and controls
* RESTful API integration
* User-friendly interface built with HTML, CSS, and JavaScript

**2. Objective**

#### The objective of this project is to develop a full stack, web-based TaskFlow Management System that streamlines task coordination and execution among administrators, managers, and team members. The system aims to simulate a real-world project management environment where projects can be created, tasks can be assigned and tracked, and progress can be monitored efficiently. It enables structured workflows, improves team collaboration, and enhances transparency and accountability across all roles

#### **Primary Purpose:**

#### To automate and optimize the process of managing project tasks through a centralized role-based system that enables:

#### Admins to create and manage projects, assign tasks, and oversee platform operations.

#### Managers to initiate and track project progress, assign tasks to team members, and monitor performance.

#### Team members to view and update their assigned tasks with clear deadlines and statuses.

#### This project aims to enhance collaboration, increase productivity, and ensure accountability in team-based task execution while reducing manual coordination efforts.

#### **Specific Goals:**

1. **Design a responsive, user-friendly frontend interface** using HTML, CSS, and JavaScript to ensure intuitive navigation for all user roles.
2. **Develop a secure and scalable backend** using Node.js and Express.js to handle authentication, routing, and business logic efficiently.
3. **Integrate MongoDB** as the primary database to store structured data .
4. **Implement Role-Based Access Control (RBAC)** to restrict features and views based on user types — Admin , Manager , Team Member
5. **Enable JWT-based authentication and password encryption** to maintain user security and data integrity.
6. **Build RESTful APIs** for seamless communication between the frontend and backend systems.
7. **Create an Admin dashboard** to monitor system activity, manage users, and perform administrative tasks.
8. **Ensure modularity and clean code structure** to support future enhancements, new feature integration, and scalability.
9. **Deploy the application locally or on a cloud-based platform** for testing and demonstration purposes.

**3. System Architecture**

1. **Architecture diagram**

┌────────────────────────────────────────┐

│ Frontend (Client) │

│ HTML, CSS, JavaScript │

│ │

│ • User Interfaces (UI) │

│ • Form Validation │

│ • API Requests (Fetch / Axios) │

└───────────────┬────────────────────────┘

│ HTTP Requests / Responses (JSON)

▼

┌────────────────────────────────────────┐

│ Backend (Server) │

│ Node.js, Express.js │

│ │

│ • RESTful APIs │

│ • Role-based Authentication (JWT) │

│ • Business Logic │

└───────────────┬────────────────────────┘

│ MongoDB Driver / Mongoose

▼

┌────────────────────────────────────────┐

│ Database (MongoDB) │

│ │

│ • Users Collection │

│ • Projects Collection │

│ • Tasks Collection │

└────────────────────────────────────────┘

### 2. Component Interaction and Technology Flow

#### **Frontend (Presentation Layer):**

* Built using **HTML, CSS, and Vanilla JavaScript**.
* Responsible for presenting the UI to the users based on their roles: **Team Member**, **Manager** , and **Admin**.
* Interacts with the backend via **AJAX/Fetch API** using **JSON data exchange**.
* Performs client-side validation before sending data to the server.

**Backend (Application Layer):**

* Built using **Node.js** with **Express.js** framework.
* Hosts **RESTful APIs** for managing:
  + User Authentication (Login/Register)
  + Project/Task Operations (Create, Update, View, Delete)
  + Role Management
* Secures endpoints using **JWT-based Authentication**.
* Applies **Role-Based Access Control (RBAC)** to ensure restricted access based on user type.

#### **Database (Data Layer):**

* Uses **MongoDB**, a NoSQL database that stores data in **JSON-like documents**.
* Collections:
  + **Users**: Stores user credentials, roles (Admin, Manager, Team), and basic profile information.
  + **Projects**: Stores project-related details such as project title, description, manager, and assigned team members.
  + **Tasks**: Contains task information including title, description, assigned project, team members, status, and deadlines.

**4. Technology Stack**

The project " TaskFlow: A Team-Based Task Management and Productivity Tracker " was built using a modern full stack technology architecture. Below is a detailed description of each component used:

### ****Frontend****

* **HTML5**: The structure of all web pages was built using HTML5, ensuring semantic markup and compatibility with modern browsers.
* **CSS3**: Used for styling and layout, CSS3 enabled a responsive, visually appealing, and user-friendly interface across various devices.
* **JavaScript**: Core client-side scripting language that handled dynamic interactions, DOM manipulation, form validations, and AJAX-based communication with the backend APIs.

### ****Backend****

* **Node.js**: A runtime environment that allows JavaScript to run on the server side. It was used to build a scalable and high-performance backend.
* **Express.js**: A minimal and flexible web application framework for Node.js. It was used to handle routing, RESTful API creation, middleware, and backend logic efficiently.

### ****Database****

* **MongoDB**: A NoSQL database used to store structured data in the form of collections and documents. It provided flexibility in managing unstructured data like tickets, users, comments, and categories. It also ensured high performance, scalability, and fast read/write operations.

### ****Tools & Utilities****

* **VS Code**: Visual Studio Code was used as the primary code editor due to its rich extension support, Git integration, and debugging features.
* **Git & GitHub**: Used for version control and code repository management. GitHub also enabled seamless collaboration, backup, and code sharing.
* **Postman**: Used for testing RESTful APIs during backend development. It helped in sending HTTP requests, analyzing responses, and debugging endpoints.
* **MongoDB Compass**: GUI tool to visually explore MongoDB data, manage collections, and monitor performance.

**5. Modules/ Features**

**1. Authentication Module**

* **Description**:  
  Handles user registration, login, and access control. Ensures that only authenticated users can access role-specific features ( Admin, Manager , Team member).
* **Key Features**:
  + Signup and login forms
  + Password encryption using bcrypt.js
  + Role-based access control
  + Token-based session handling with JWT
* **Technologies Used**:  
  HTML, CSS, JavaScript, Node.js, Express.js, MongoDB.

**2.Task Management Module**

• **Description:**  
Enables Admins and Managers to create and assign tasks to team members based on project requirements. Team members can view their assigned tasks and update their progress.

• **Key Features:**

* Create tasks with title, description, and due dates
* Assign tasks to individual or multiple team members
* Status updates: Pending, In Progress, Completed
* Task listing with filtering based on status/project

• **Technologies Used:**  
HTML, CSS, JavaScript, Node.js, Express, MongoDB

3**. Dashboard Module**

**• Description:**  
Displays customized dashboards based on user roles for streamlined access and management.

• **Role-based Views:**

* Admin: View overall system statistics, all users, projects, and tasks
* Manager: View assigned projects and team progress
* Team Member: View "My Tasks" assigned to them

• **Key Features:**

* Task and project summaries
* Quick stats and charts (optional enhancement)
* Recent activity and alerts

• **Technologies Used:**  
HTML, CSS, JavaScript, Node.js, MongoDB

4. **User & Role Management Module**

• **Description:**  
Admins can manage users and assign roles (Admin, Manager, Team) for secure access control.

• **Key Features:**

* Add/Edit/Delete users
* Assign roles during signup or via admin panel
* Role-based access control throughout the system

• **Technologies Used:**  
JavaScript, Node.js, Express, MongoDB, HTML/CSS

5. **Project Management Module**

• **Description:**  
Admins or Managers can create and manage projects, assign managers, and include team members.

**• Key Features:**

* Project creation with description
* Add or remove team members
* View project-specific tasks

• **Technologies Used:**  
HTML, CSS, JavaScript, Node.js, MongoDB

**6. Frontend Development**

**Technologies Used**

* **HTML5**: Used to build the structure and content of each page.
* **CSS3**: Applied for styling, layouts, responsiveness, and animations.
* **JavaScript**: Enables interactivity, DOM manipulation, client-side form validation, and dynamic content loading.

**Page Structure / Layout**

**• Header / Navbar**

* Displays the project logo and application title.
* Contains navigation elements such as user profile and logout option.
* Fixed at the top for easy access across all pages.

• **Sidebar (Role-Based Navigation)**

The sidebar dynamically renders options based on the authenticated user’s role:

* **For Admins:**
  + Dashboard
  + User Management
  + All Projects
  + All Tasks
  + Category Management
* **For Managers:**
  + Dashboard
  + My Projects
  + Assign Tasks
  + Team Progress
* **For Team Members:**
  + Dashboard
  + My Tasks
  + Task Progress

**• Main Content Area**

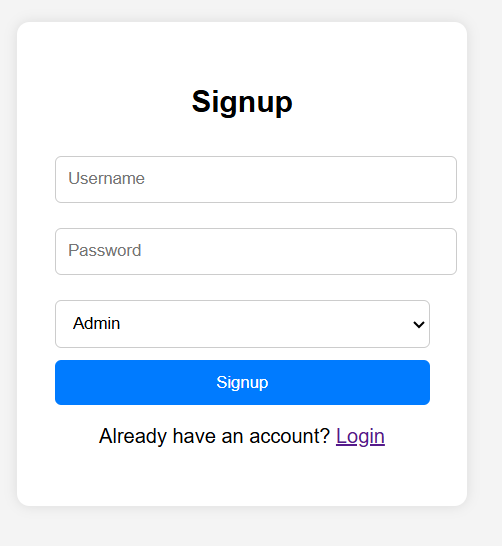
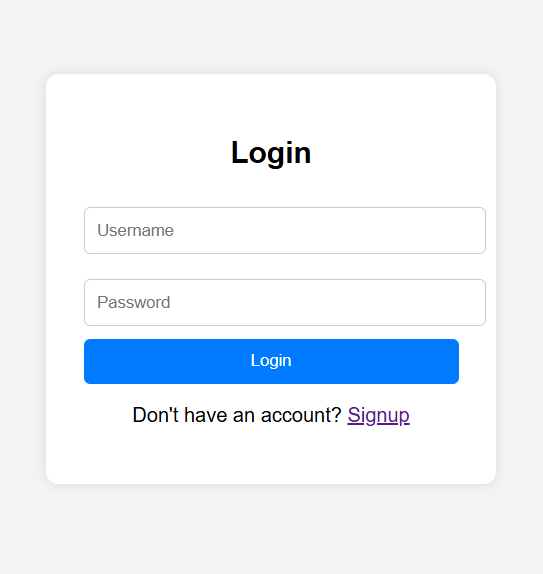
* Dynamically loads content according to the selected route or menu option.
* Each module or feature (e.g., create task, view project, user settings) is displayed here.
* Fully responsive and clean layout with reusable components.

**UI/UX Design Strategy**

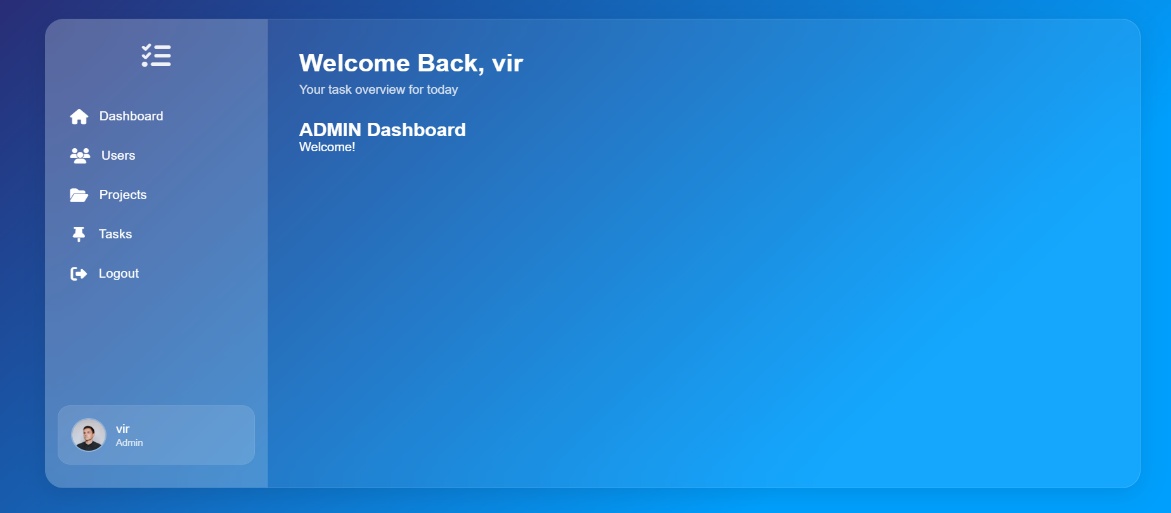
* **Responsiveness**: Ensured through media queries and flexible layouts for mobile, tablet, and desktop views.
* **Clean and Minimal UI**: Focused on intuitive navigation and clutter-free layout for different users.
* **Role-Based Interface**: Each user type sees only the relevant features, improving usability.
* **Feedback Mechanisms**: Success/failure alerts, loaders, and validation messages guide users effectively.
* **Accessibility Considerations**: Proper contrast, label associations, and button sizes for better accessibility.

**Sample Pages**

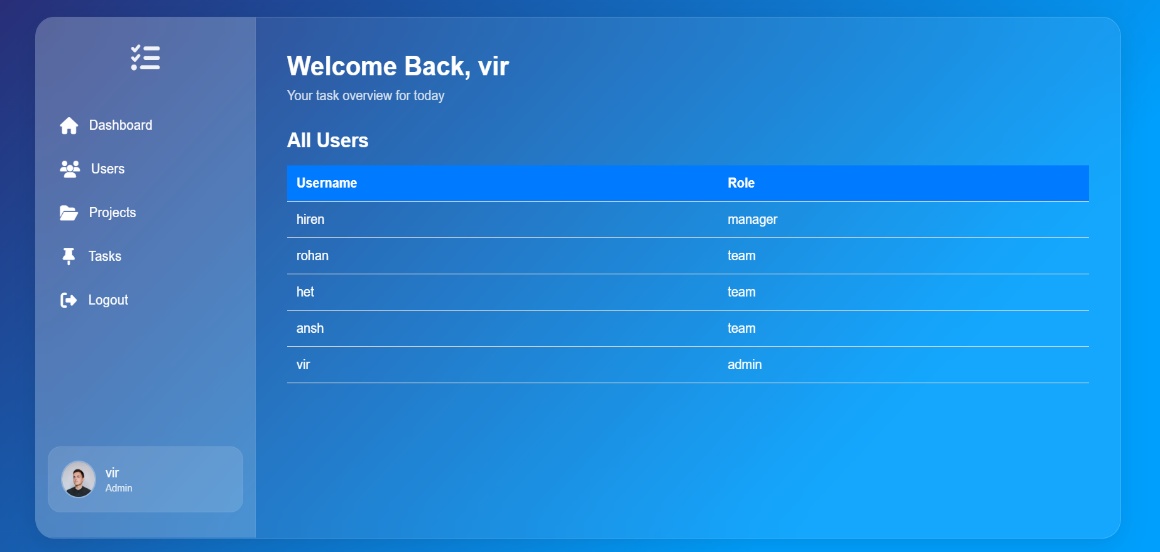
1. **Login & signup page**
   * Fields: Username , Password , Role
   * Action: Role-based redirection after login

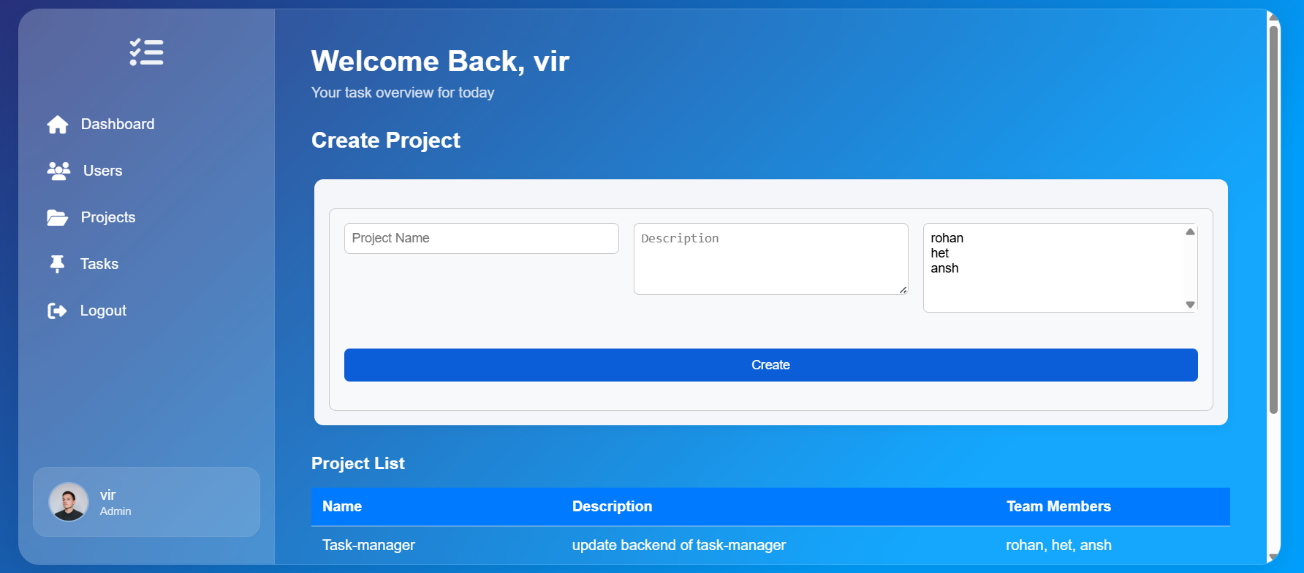


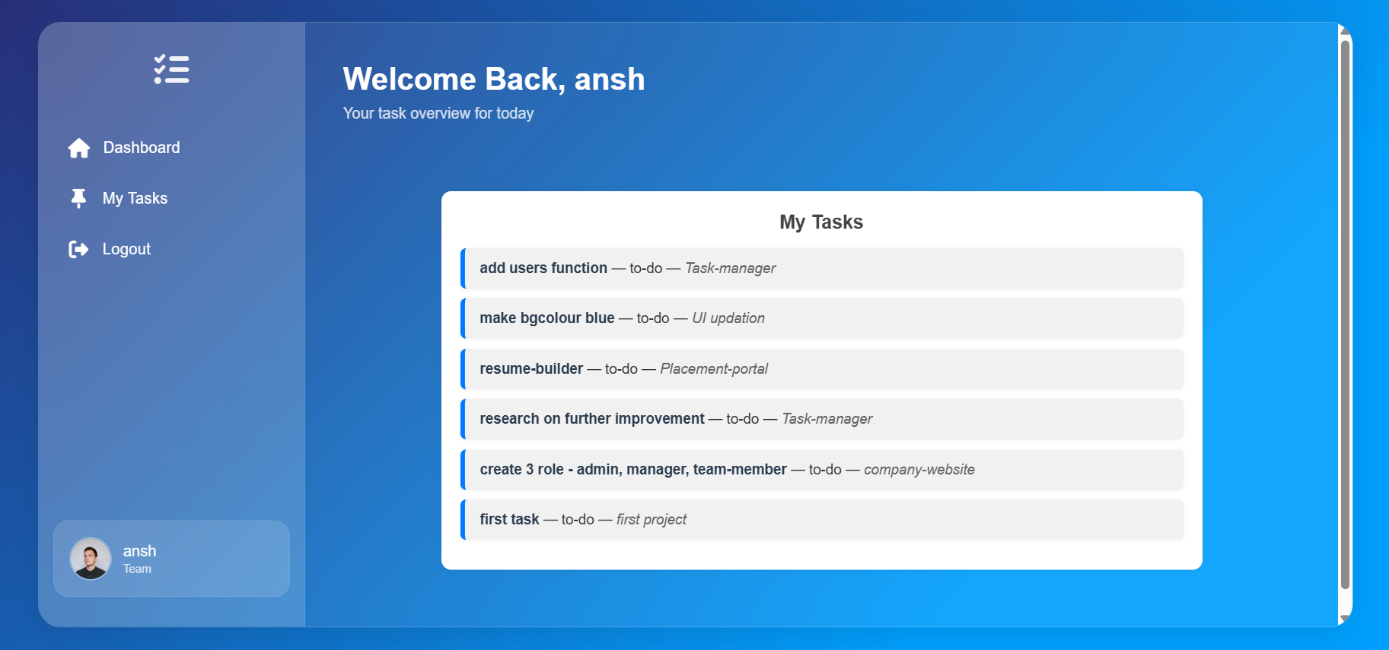
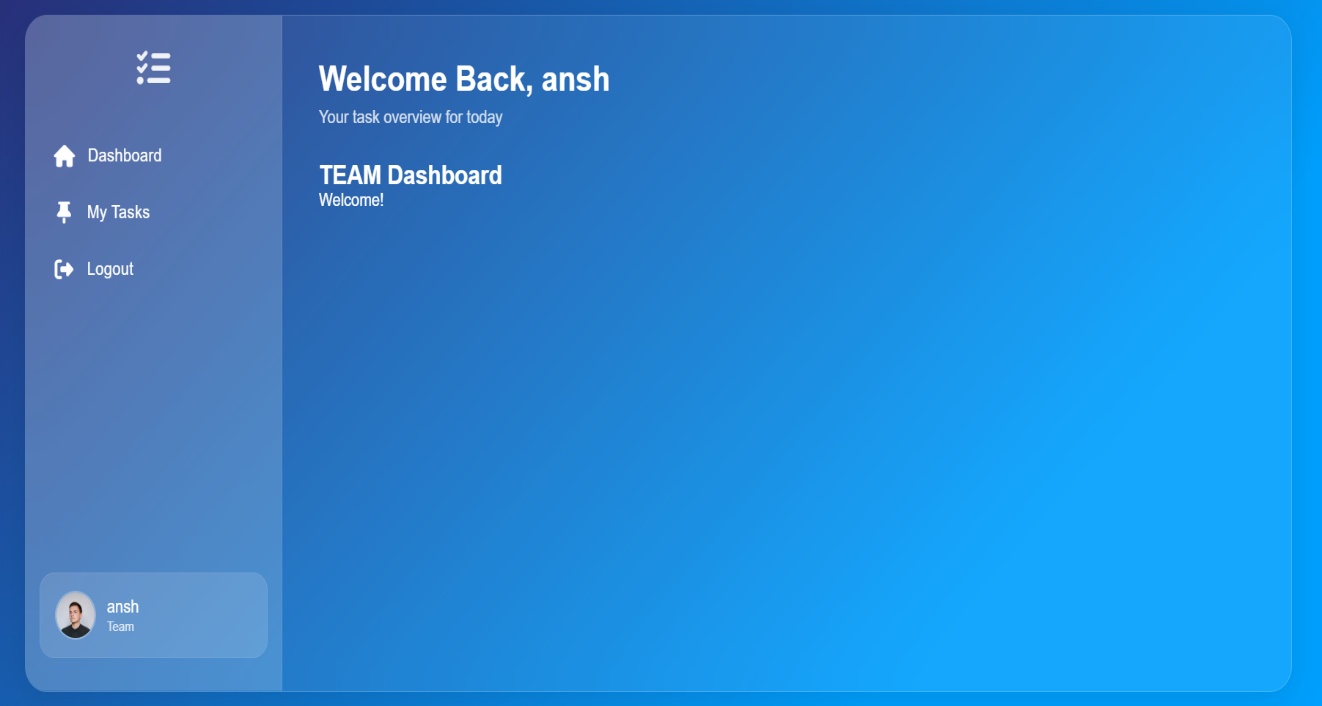
1. **Admin Dashboard**



1. **Users**
   * List of Users



1. **Project** 
   * Create and view project
2. **Team Member**

****

**7. Backend**

The backend of the **Task-Flow** is designed using a scalable and secure RESTful architecture. It serves as the core engine of the application by handling requests, managing data, and enforcing business logic.

### ****Framework Used****

* **Node.js**: JavaScript runtime environment used to build a fast, non-blocking, and scalable server-side application.
* **Express.js**: A lightweight Node.js web application framework used to define API routes, handle middleware, and structure server logic efficiently.

### ****API Structure****

The application follows a **RESTful API** design, which enables the frontend to communicate seamlessly with the backend using HTTP methods.

* **GET**: Fetch tickets, users, categories, comments, etc.
* **POST**: Create new tickets, users, categories, comments, etc.
* **PUT/PATCH**: Update ticket status, user details, etc.
* **DELETE**: Remove users, categories, or tickets (admin only)

### ****Routing and Controllers****

**Express Router**: Used to modularize the codebase by separating route files like:

* **authRoutes.js**
* **userRoutes.js**
* **projectRoutes.js**
* **taskRoutes.js**

**Controllers**: Each route is linked to a controller function that handles the logic:

### **authController.js**

### **userController.js**

### **projectController.js**

### **taskController.js**

### ****Authentication and Authorization****

* **Role-Based Access Control**:
  + Users are assigned roles: admin, manager , team member
  + Middleware checks roles to allow or deny access to certain endpoints.
  + Example:
    - Only admin can view user list
    - Only admins can view all project and tasks

**8. Database**

The backend database plays a critical role in managing login, project details and task details. A NoSQL database system was used for its scalability, flexibility, and performance.

### ****DBMS Used****

* **MongoDB**  
  MongoDB is a NoSQL document-oriented database used to store data in the form of **collections** and **documents (JSON-like)**. It allows for dynamic schemas, making it well-suited for applications with frequently changing data structures like a ticketing system.

### ****Entity-Relationship (ER) Diagram****

### **[User]**

### **├── (One-to-Many) ──▶ [Project]**

### **└── (One-to-Many) ──▶ [Task] (as assignedTo)**

### **[Project]**

### **└── (One-to-Many) ──▶ [Task]**

### **[Task]**

### **└── (Many-to-One) ──▶ [Project]**

### **└── (Many-to-One) ──▶ [User] (assignedTo)Schema / Collections**

### ****Users Collection****

{

"\_id": ObjectId,

"username": "John Doe",

"password": "hashed\_password",

"role": "team member", // or "manager", "admin"

"createdAt": ISODate

}

### ****Project Collection****

{

"\_id": ObjectId,

"name": "task-flow",

"description": "update backend of task manager."

"createdby": objected,

"members": array,

"status": active,

"createdAt": ISODate,

"updatedAt": ISOdate

}

### ****Task Collection****

{

"\_id": ObjectId("6878ae694215653a5c3f9a24"),

"title": "add users function",

"description": "all users list will display",

"projectId": ObjectId("6878ad4e4215653a5c3f99ff"),

"assignedTo": [

ObjectId("6878a4ea4215653a5c3f99b2") // Team member ID

],

"createdBy": ObjectId("6878a4ea4215653a5c3f99a1"), // Admin or Manager ID

"priority": "medium", // Enum: low, medium, high

"status": "to-do", // Enum: to-do, in-progress, done

"history": [],

"createdAt": ISODate("2025-07-17T08:03:53.287Z"),

"updatedAt": ISODate("2025-07-17T08:03:53.287Z"),

"\_\_v": 0

}

**9. Data Flow Diagram**

## **Data Flow Diagrams (DFD)**

## **Level 0 DFD – Context Diagram**

This diagram represents the system as a single process and shows how it interacts with external entities.

**Entities & Processes:**

* **Users** (Admin, Manager , Team Member)
* **Task-Flow** (Main system)
* **Database** (MongoDB)

#### **Data Flows:**

* Users → View My Task → System
* Admin → Create Project Create Task→ System
* System ↔ Reads/Writes → MongoDB

**Level 0 DFD**

[user] ------------> (view task) ------------> [System]

[Admin] ---------> (create/view project/tasks) ------> [System]

[System] ---------------> (Store/Retrieve Data) -----> [MongoDB]

### ****Level 1 DFD****

This level provides a more detailed breakdown of internal components.

#### **Processes:**

1. **User Authentication**
2. **Project and Task Management**
3. **Task Assignment**
4. **status**
5. **Admin Monitoring**

#### **Data Stores:**

* Users Collection
* Project Collection
* Task Collection

### ****Level 1 DFD (Text Description)****

 **Users →** Login / Submit Tasks → System

 **Admins/Managers →** Assign Projects & Tasks, Monitor Users → System

 **Team Members →** View Assigned Tasks, Update Status, Comment → System

 **System ↔** Stores & Retrieves all entities → **MongoDB**

## **Data Flow Between Components**

Your project has 3 major layers:

* **Frontend** (HTML, CSS, JavaScript)
* **Backend** (Node.js + Express)
* **Database** (MongoDB)

It involves 3 types of users: **Team Member**, **Manager**, and **Admin**.

### 1. ****User Authentication Flow****

#### Actions:

* Sign Up / Log In

#### Data Flow:

Frontend → Backend (POST /login or /signup) → MongoDB (Users Collection)

#### Explanation:

1. A user fills the login/signup form on the frontend.
2. Form data is sent to the backend via a REST API endpoint (e.g., /api/login).
3. The backend verifies the user (checks hashed password for login or stores a new user for signup).
4. If successful, backend sends back a JWT token or success message.
5. The token is stored in browser storage (local/session) for authenticated requests.

### 2.Task Creation (Admin/Manager Side)

### Actions:

### • Admin or Manager creates a new task within a selected project and assigns it to one or more team members.

### Data Flow:

### Admin/Manager → Frontend Task Form → Backend (POST /api/tasks) → MongoDB (Tasks Collection)

### Explanation:

### Admin/Manager selects a project, enters task details (title, description, priority, assigned members).

### Data is sent via an HTTP POST request to the backend.

### The backend validates and stores the task in the tasks collection, linked to the project.

### A success message is sent back to the frontend.

### 3. Project Creation (Admin/Manager Side)

### Actions:

### • Admin or Manager creates a new project and assigns team members.

### Data Flow:

### Admin/Manager → Frontend Project Form → Backend (POST /api/projects) → MongoDB (Projects Collection)

### Explanation:

### Project name, description, and members are submitted.

### The backend stores this project and links the members.

### Members assigned are now visible when tasks are created within the project.

### 4. Team Dashboard – My Tasks View

### Actions:

### • Team member logs in and views only tasks assigned to them.

### Data Flow:

### Team → Frontend Dashboard → Backend (GET /api/tasks/my-tasks) → MongoDB → Response with Task Data

### Explanation:

### Upon login, the frontend uses the JWT token to fetch user-specific tasks.

### The backend filters tasks where assignedTo includes the user's ID.

### The result is returned and rendered as a task board (e.g., To-do, In Progress, Done).

### 5. Task Status Update

### Actions:

### • Team member updates the status of their task (e.g., To-do → In Progress → Done).

### Data Flow:

### Team → Frontend Task Card → Backend (PUT /api/tasks/:id/status) → MongoDB (Tasks Collection)

### Explanation:

### The user selects a new status from a dropdown or drag-and-drop board.

### A PUT request updates the status field of the task.

### A response confirms the update.

### 6. Admin Panel – User & Project Oversight

### Actions:

### • Admin views users, all projects, all tasks.

### Data Flow:

### Admin → Frontend Dashboard → Backend (GET /api/admin/\*) → MongoDB → JSON Responses

### Explanation:

### Multiple GET calls fetch user lists, project metadata, and task statuses.

### The data is returned and visualized as tables/charts.

### Admin can also delete users or edit project memberships.

### 7. Role-Based Sidebar Navigation

### Behavior:

### • Navigation and dashboard content are dynamically adjusted based on the logged-in user’s role.

### Role Access Logic:

### Admin: /dashboard/admin

### Manager: /dashboard/manager

### Team: /dashboard/team

### 8. Data Validation & Security Flow

### Features:

### Validation: Required fields (e.g., task title, project members)

### Prevent XSS, SQL injection using express-validator / sanitize-html

### Authentication: JWT-based login with protected routes

### Authorization:

### Only Admins can manage users/projects.

### 10. Limitations & Future Enhancement

**Limitations:**

While the Task Flow achieves its core functionalities, there are certain **limitations and areas for improvement**:

### 1. ****Limited Role-Based Access Control****

* Currently, the system provides basic access separation for Team member , manager and Admins.
* Granular permission levels are **not yet implemented**.

### 2. ****Basic Security****

* JWT-based authentication is used, but:
  + No token refresh mechanism is implemented.
  + No brute-force protection or rate-limiting on login routes.
* Input validation exists but **could be enhanced further** with a centralized validation library.

### 3. ****UI/UX Constraints****

* The frontend, built using HTML, CSS, and JavaScript, may lack dynamic responsiveness on all devices.
* Real-time features like live chat or notifications are **not integrated**.

### 4. ****Testing Coverage****

* Unit tests and API tests are **limited or not fully implemented**.
* Manual testing has been used, but automated testing pipelines are **yet to be developed**.

### 5. ****Data Analytics****

* The admin dashboard offers basic statistics.
* Advanced analytics are **not available**.

### 6. ****Email/Notification System****

* There's **no automated email** or in-app notification system for:
  + Updates related to project and task
  + Assigned project
  + Assigned task

### 7. ****Scalability****

* The system is optimized for a **small to medium** team size.
* At high scale , performance optimization and database indexing would be needed.

**Future Enhancement:**

**1. Real-Time Notifications**

Implement WebSockets using technologies like Socket.IO to enable live updates when a task is assigned, updated, or commented on. Add push notifications or email alerts for important user actions.

**2. Advanced Analytics Dashboard**

Enhance the admin panel with advanced analytics such as task completion rates, user performance trends, and project workload distribution. Use chart libraries like Chart.js or Recharts for data visualization.

**3. Email Integration**

Introduce automated email functionality to notify users about task assignments, status changes, deadlines, and project updates. Use services like Nodemailer, SendGrid, or Mailgun for integration.

**4. Enhanced Security**

Strengthen system security with role-based access control (RBAC), account lockout mechanisms after multiple failed login attempts, and two-factor authentication. Sensitive data should be encrypted beyond just passwords.

**5. Multi-language Support**

Add internationalization support to allow users to choose their preferred language. This will make the platform more accessible to global teams or users with diverse language preferences.

**6. Task Prioritization and SLA Tracking**

Introduce task priority levels such as high, medium, and low, along with deadlines. Monitor SLA (Service Level Agreement) compliance and implement auto-escalation of overdue tasks to supervisors or admins.

**7. Responsive Design and Mobile App**

Ensure full mobile responsiveness for better usability on smartphones and tablets. Consider developing a dedicated mobile application using frameworks like React Native or Flutter.

**8. Automated Testing and Continuous Integration**

Implement automated unit and API testing using tools like Jest, Mocha, or Postman. Set up CI/CD pipelines using GitHub Actions, Travis CI, or Jenkins to streamline development and deployment.

**9.AI-Powered Task Assistant**

Add a chatbot powered by AI technologies such as Dialogflow or GPT APIs to assist users with common queries or help team members automate task updates and suggestions.

**10. Third-Party Integrations**

Enable integrations with popular communication and productivity platforms like Slack, Microsoft Teams, or WhatsApp for real-time updates. Connect with CRM platforms like Salesforce or HubSpot for seamless project coordination.

**11.Annexures**

**Annexure A: Project Modules Overview**

| Module Name | Description |
| --- | --- |
| Authentication | Handles login, signup, JWT-based session management |
| Role-Based Dashboard | Separate dashboards for Admin, Manager, and Team roles |
| Project Management | Create, assign, and list projects with team member associations |
| Task Management | Create tasks, assign team, update status, track progress |
| User Management | Admin can manage all users; roles can view relevant users |
| Reports & Summary | View analytics and summaries (tasks per status, user workload) |

**Annexure B: Technology Stack**

| Category | Technologies Used |
| --- | --- |
| Frontend | HTML, CSS, JavaScript |
| Backend | Node.js, Express.js |
| Database | MongoDB |
| Authentication | JWT (JSON Web Tokens) |
| Tools | VS Code, Postman, Git, GitHub |
| Deployment | (Optional: Mention Render, Vercel, etc.) |

**Annexure C: MongoDB Collection Structures (Simplified)**

**Users Collection**

{

"\_id": ObjectId,

"name": "John Doe",

"email": "john@example.com",

"password": "hashed\_password",

"role": "admin | manager | team",

"createdAt": ISODate,

"updatedAt": ISODate

}

**Projects Collection**

{

"\_id": ObjectId,

"name": "TaskFlow Web App",

"description": "Project management tool",

"createdBy": ObjectId,

"members": [ObjectId],

"status": "active",

"createdAt": ISODate,

"updatedAt": ISODate

}

**Tasks Collection**

{

"\_id": ObjectId,

"title": "Build Login Feature",

"description": "User login and JWT token setup",

"projectId": ObjectId,

"assignedTo": [ObjectId],

"createdBy": ObjectId,

"priority": "low | medium | high",

"status": "to-do | in-progress | done",

"history": [],

"createdAt": ISODate,

"updatedAt": ISODate

}

**Annexure D: Sample API Endpoints**

| Method | Endpoint | Description | Access |
| --- | --- | --- | --- |
| POST | /api/users/signup | Register a new user | Public |
| POST | /api/users/login | Login and receive JWT token | Public |
| POST | /api/projects | Create a new project | Admin, Manager |
| GET | /api/projects | Get all projects | All Roles |
| POST | /api/tasks | Create a task | Admin, Manager |
| GET | /api/tasks/my-tasks | Get logged-in user’s assigned tasks | Team |
| PUT | /api/tasks/:id/status | Update task status | Assigned Users |
| DELETE | /api/tasks/:id | Delete a task | Admin, Manager |

**12.GitHub Repository:**

**Repository Link**: <https://github.com/Viraj-keshwala/Task-Flow.git>

**13. References**

This section lists the books, online libraries, tools, and other resources referred to during the development of the **Task Flow project**.

**Online Resources & Libraries:**

Online resources- IBM career Education program

Course Link- https://ibmcep.cognitiveclass.ai/

**Tools & Platforms:**

**Visual Studio Code :**Code editor used for development

**Git & GitHub -** Version control and code repository

**Postman** For testing RESTful APIs

**MongoDB Atlas**-Cloud-hosted MongoDB for database storage