COLLEGE CODE: 8203

COLLEGE: AVC COLLEGE OF ENGINEERING

DEPARTMENT: INFORMATION TECHNOLOGY

STUDENT NM-ID:164F4C9B531C69F8C41498DE7B1C57F0

ROLL NO: 23IT64

DATE:15/09/2025

Completed the project named as Phase 2

TECHNOLOGY PROJECT NAME: Email Remainder System

SUBMITTED BY,

NAME: LOGESH R

MOBILE NO: 9344415970

Phase 2: Solution Design & Architecture

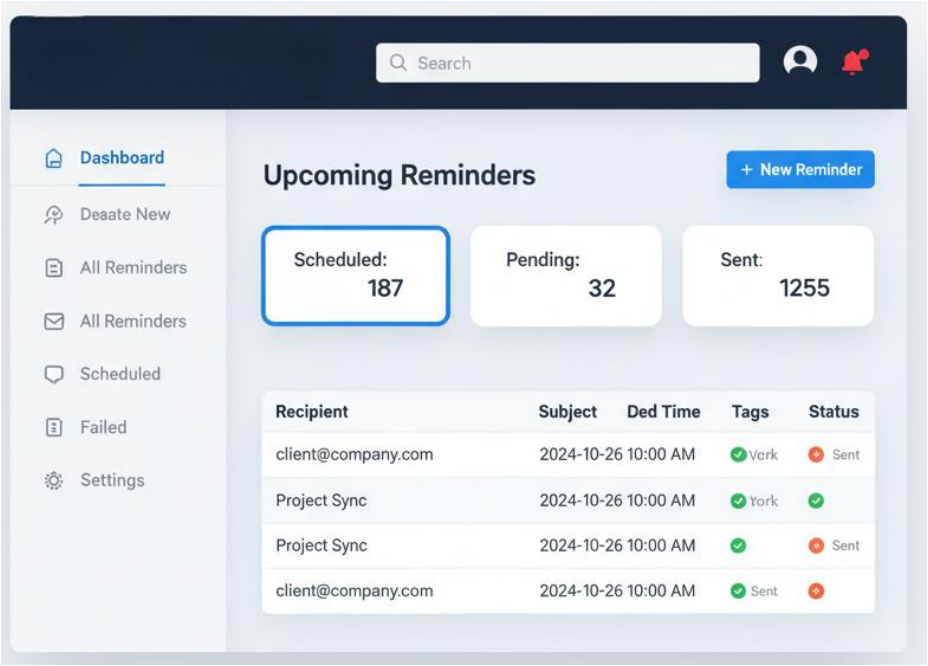
1.Tech Stack Selection – Email Reminder System

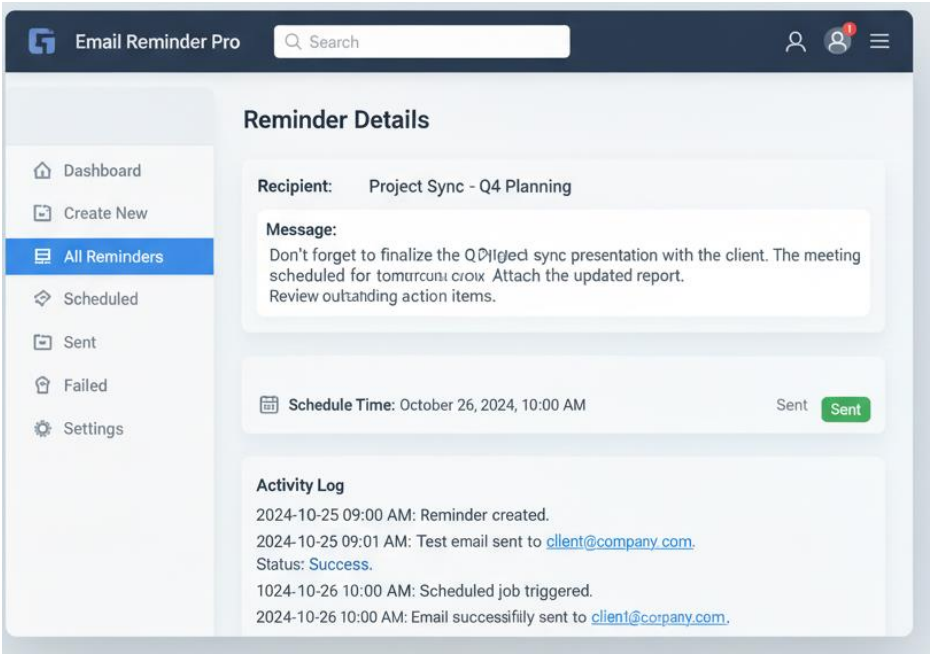
- **1. Frontend (Admin Interface - Optional)**
  - **HTML, CSS, JavaScript:** To build a simple, clean interface for creating and managing reminders.
  - (Optional) **React.js** → For modern, dynamic, and mobile-friendly interface
  - **Reason:** Lightweight and universal, perfect for a straightforward admin dashboard without complex state management.
- **2. Backend (Server Logic)**
  - **Node.js:** An efficient, non-blocking runtime ideal for handling API requests and scheduled tasks.
  - **Express.js:** A minimal and flexible Node.js framework for building the REST API and managing routes.
  - **Reason:** Fast, scalable, and part of the popular MERN/MEAN stack, making it a standard choice for modern web services.
- **3. Database**
  - **MongoDB:** A NoSQL document database used to store reminder information (recipient email, subject, body, schedule, and status).
  - **Reason:** Its flexible schema is perfect for storing varied data structures, and it integrates seamlessly with Node.js.
- **4. Scheduling & Emailing**
  - **node-cron:** A task scheduler to trigger jobs at the user-defined times for sending reminders.
  - **Nodemailer:** A module for Node.js applications to allow easy email sending.

- **Reason:** Both are robust, highly popular, and purpose-built libraries that simplify complex tasks like scheduling and email delivery.

- **5. Tools & Utilities**
  - **Git/GitHub:** For version control and code management.
  - **Postman:** Essential for testing the REST API endpoints.

2.UI Structure & API Schema Design:





UI Structure – Email Reminder System Admin Panel

- **Header:** Title "Email Reminder System".
- **Create Reminder Form:**
  - Input box: Recipient Email
  - Input box: Subject
  - Text Area: Message Body
  - Date/Time Picker: Schedule Time
  - Button: "Schedule Reminder"
- **Reminders List:** A table displaying scheduled reminders with columns for:
  - Scheduled Time
  - Recipient
  - Subject
  - Status (e.g., "Pending," "Sent," "Failed")
  - Actions (Edit/Delete buttons)
- **Status/Log Area:** Displays confirmation messages ("Reminder scheduled successfully") or error messages.

API Schema – Email Reminder System

- **Endpoint: /reminders**
  - **Method:** POST
  - **Request Body:**

    {

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

      "subject": "Project Deadline",

      "body": "Submit the final report by 5 PM.",

      "scheduleTime": "2025-10-28T17:00:00Z"

    }

    Success Response (201 Created):

{ "message": "Reminder scheduled successfully",

"reminderId": "632b...e4f" }

Endpoint: /reminders

- **Method:** GET
- **Success Response (200 OK):**

{

"count": 2,

"data": [

{

"_id": "632b...e4f",

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

"subject": "Project Deadline",

"status": "Pending",

"scheduleTime": "2025-10-28T17:00:00Z"

},

{

"_id": "632b...e5a",

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

"subject": "Team Meeting",

"status": "Sent",

"scheduleTime": "2025-09-25T10:00:00Z"

}

]

}

Error Response (e.g., 400 Bad Request):

{

"error": "Invalid email address provided."

}

3. Data Handling Approach

- **1. User Input Handling (Reminder Creation)**
  - A user sends a POST request to the /reminders endpoint with the recipient's email, subject, body, and scheduleTime.
  - The backend validates the input (e.g., checks for a valid email format, ensures scheduleTime is in the future).
- **2. Backend Data Handling & Scheduling**
  - **Step 1: Database Storage**
    - Upon successful validation, the reminder data is saved as a new document in the MongoDB reminders collection with a default status of "Pending".
  - **Step 2: Job Scheduling**
    - A node-cron job is created and scheduled to run exactly at the scheduleTime provided by the user. The job is linked to the unique ID of the reminder document in MongoDB.
- **3. Scheduled Job Execution**

    - When the node-cron job triggers at the scheduled time, it retrieves the full reminder details from MongoDB using its ID.
    - It then uses **Nodemailer** to construct and send the email to the recipient.
    - **On Success:** The reminder's status in MongoDB is updated from "Pending" to "Sent".
    - **On Failure:** The status is updated to "Failed", and the error from Nodemailer is logged for debugging.
- **4. Error Data Handling**
    - **API Errors:** If a user provides invalid data, the API immediately responds with a 400 Bad Request and a clear error message (e.g., { "error": "scheduleTime must be in the future" }).
    - **Email Sending Errors:** If Nodemailer fails to send the email (e.g., due to incorrect SMTP credentials or a temporary network issue), the failure is logged, and the status is updated in the database, but the API does not crash.

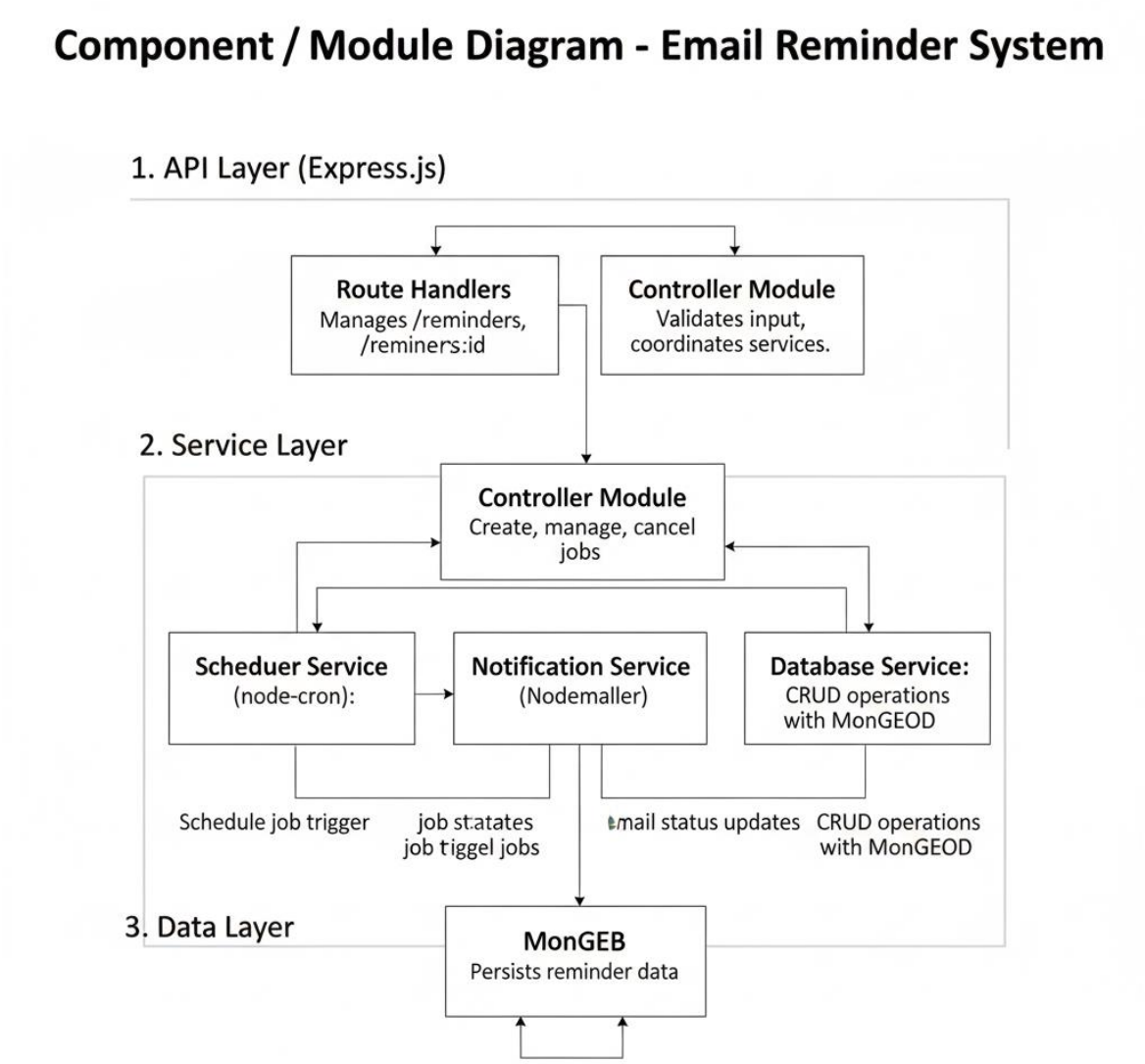
4. Component / Module Diagram

The system can be broken down into these core components:

- **1. API Layer (Express.js)**
    - **Route Handlers:** Manages the API endpoints (/reminders, /reminders/:id).
    - **Controller Module:** Contains the core logic for handling requests—validating input and coordinating with other services.
- **2. Service Layer**
    - **Scheduler Service (node-cron):** Responsible for creating, managing, and canceling scheduled tasks.
    - **Notification Service (Nodemailer):** Solely responsible for sending emails.
    - **Database Service:** Handles all communication (create, read, update, delete) with the MongoDB database.

- **3. Data Layer**
    - **MongoDB:** The database where all reminder data is persisted.

Module Diagram:



5. Basic Flow Diagram

This diagram shows the lifecycle of a single reminder from creation to completion

Flow Diagram: