COLLEGE CODE: 8203

COLLEGE: AVC COLLEGE OF ENGINEERING

DEPARTMENT: INFORMATION TECHNOLOGY

STUDENT NM-ID:164F4C9B531C69F8C41498DE7B1C57F0

ROLL NO: 23IT64

DATE:08/09/2025

Completed the project named as Phase 1

TECHNOLOGY PROJECT NAME: Email Remainder System

SUBMITTED BY,

NAME: LOGESH R

MOBILE NO: 9344415970

Phase 1: Problem Understanding & Requirements

Problem Statement

In today's fast-paced world, individuals and professionals juggle multiple tasks, deadlines, and appointments. Forgetting important events like paying bills, attending meetings, or submitting assignments can lead to negative consequences. While calendar apps exist, they can be overly complex for the simple need of getting a timely notification.

There is a need for a **dedicated and reliable Email Reminder System** that:

- ❖ Allows users to easily schedule reminders for specific dates and times.
- ❖ Leverages robust backend technologies like **Node.js** and **Express** to handle requests.
- ❖ Uses a powerful scheduling tool like **node-cron** to ensure reminders are triggered precisely on time.
- ❖ Sends dependable email notifications using **Nodemailer**.
- ❖ Securely stores reminder information and delivery status in a **MongoDB** database.

This project aims to deliver a streamlined, API-driven solution for creating, scheduling, and delivering email reminders efficiently.

Users & Stakeholders:

Users:

- **Students:** To get reminders for assignment deadlines, exam schedules, and project submissions.
- **Professionals:** To schedule follow-ups with clients, remember meeting times, and track project deadlines.
- **General Public:** To remember to pay bills, attend personal appointments, or get notified about birthdays and anniversaries.

- **Developers:** To integrate a simple reminder functionality into their own applications via the REST API.

- **Stakeholders:**
  - **Project Development Team:** Responsible for designing the API, writing the code, testing, and deploying the system.
  - **System Administrators:** Ensure the server, database (MongoDB), and cron jobs are running smoothly and reliably.
  - **Email Service Provider (e.g., Gmail, SendGrid):** Their service uptime and reliability directly impact the system's success.
  - **Educational Institution / Faculty Guide:** As this is an academic project, they will oversee the project's progress and evaluate the final deliverables.

User Stories

❖ **As a student,** I want to schedule a reminder with a custom message for my assignment deadline, **so that** I receive an email a day before it is due.

❖ **As a professional,** I want to create a reminder to follow up on a client email, **so that** I can maintain good business communication.

❖ **As a general user,** I want to set a recurring monthly reminder for my credit card bill, **so that** I can avoid paying late fees.

❖ **As a developer,** I want to send a POST request to an API endpoint to create a reminder, **so that** I can easily integrate this service into my application.

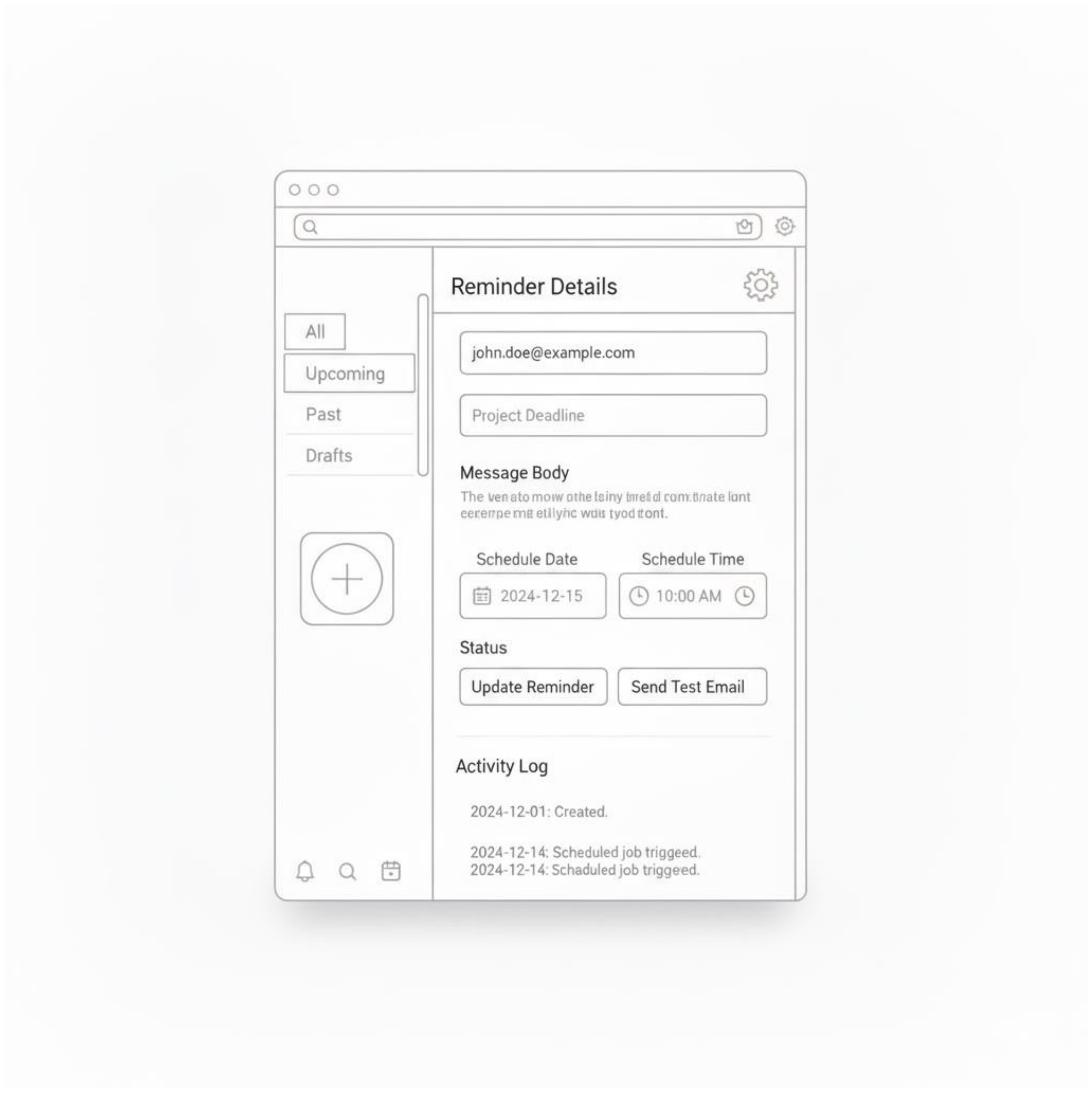
MVP (Minimum Viable Product) Features

1. **API-Based Reminder Creation:**
   - Users can create a reminder by sending a request to a REST API endpoint.
   - The request will include the recipient's email, subject, message body, and the specific date/time for the reminder.
2. **Automated Job Scheduling:**
   - The backend uses **node-cron** to schedule a job for each reminder at the specified time.
3. **Email Notification Delivery:**
   - At the scheduled time, the system uses **Nodemailer** to automatically send the email to the specified recipient.

4. **Database Storage:**
   - Reminder details and their status (e.g., "scheduled," "sent," "failed") are stored in a **MongoDB** database.
5. **Basic Logging:**
   - The system maintains logs for the success or failure of each email delivery.

Wireframes & API Endpoint List

Wireframe



john.doe@example.com

Project Deadline

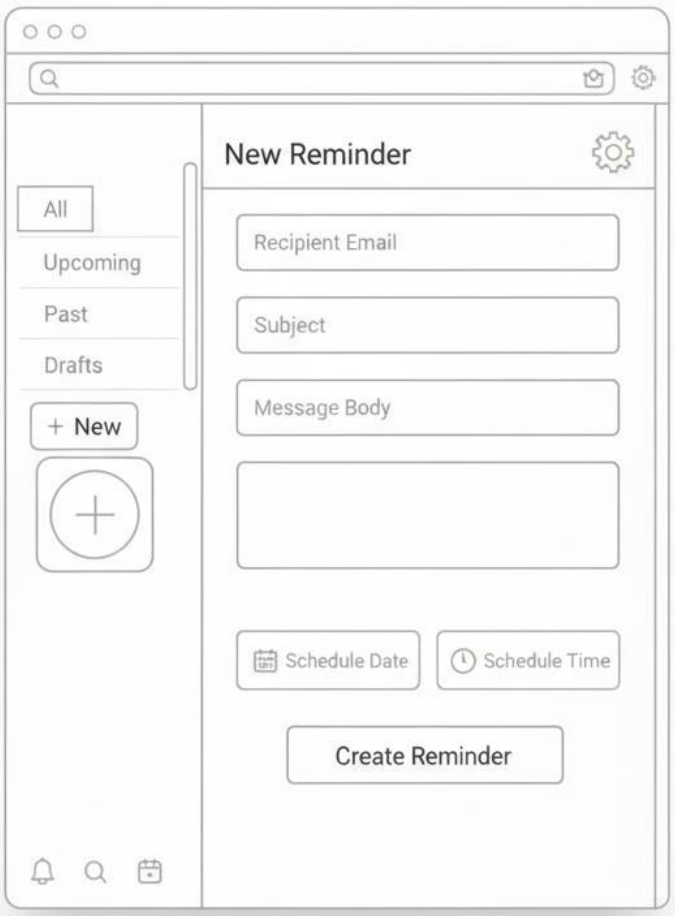
The ven ato mow othe lsiny bmetd com.tnate lont eeremepe me etllyhc wdis tyod tont.

2024-12-15

10:00 AM

2024-12-01: Created.

2024-12-14: Scheduled job triggeed.
2024-12-14: Schaduled job triggeed.



The interface includes:

- **Recipient Email:** Input field for the destination email address.
- **Subject:** Input field for the email's subject line.

- **Message:** A text area for the body of the reminder email.
- **Schedule Date & Time:** A date and time picker to set the reminder.
- **Button:** A "Schedule Reminder" button to submit the form.
- **Status Area:** A space to show confirmation or error messages.
- Of course. Here is the API endpoint list for the Email Reminder System.

API Endpoint List:

Endpoint	Method	Description	Request Body (for POST/PUT)	Response (Success)
`/reminders`	POST	Creates and schedules a new reminder.	`{ "email": "...", "subject": "...", "body": "...", "scheduleTime": "ISO_Date" }`	`{ "message": "Reminder created", "data": { ... } }`
`/reminders`	GET	Retrieves a list of all reminders.	(None)	`{ "count": 2, "data": [ {...}, {...} ] }`
`/reminders/:id`	GET	Retrieves a single reminder by its ID.	(None)	`{ "data": { ... } }`
`/reminders/:id`	PUT	Updates an existing reminder.	`{ "subject": "New Subject", "scheduleTime": "New_Date" }`	`{ "message": "Reminder updated", "data": { ... } }`
`/reminders/:id`	DELETE	Deletes/cancels a scheduled reminder.	(None)	`{ "message": "Reminder deleted", "data": null }`
`/health`	GET	Checks if the backend service is running.	(None)	`{ "status": "ok", "message": "Service is running" }`

Acceptance Criteria

1. **Reminder Creation:**

   ❖ A user can successfully schedule a reminder by sending a valid POST request to the /reminders endpoint.
   ❖ The API must return an error if the request is missing required fields (email, subject, scheduleTime) or if the data is invalid (e.g., past date).

2. **Email Delivery:**

   ❖ The system must send the email at the exact scheduled time (with a tolerance of +/- 15 seconds).
   ❖ The recipient must receive an email with the correct subject and body content as provided in the API request.

3. **Database & Logging:**

   ❖ Every new reminder is successfully saved in the MongoDB collection.
   ❖ After an email is sent successfully, its status in the database should be updated from scheduled to sent.
   ❖ If an email fails to send, the status should be updated to failed and the error should be logged.

4. **Error Handling:**

   ❖ If the email service (Nodemailer) fails, the system should catch the error gracefully without crashing.
   ❖ The API must return clear, user-friendly JSON error messages with appropriate HTTP status codes (e.g., 400 for bad requests, 500 for server errors).