EMIAL REMINDER SYSTEM

PHASE 5

PROJECT DEMONSTRATION AND DOCUMENTATION

Step 1: Start MongoDB

- Open **MongoDB Compass** or **mongod.exe** to start your database server.
- Ensure the **database `emailReminderDB`** and **collection `reminders`** are ready.

Step 2: Run the Node.js Server Open **PowerShell** or **Command Prompt** in your project directory (`C:\Users\ELCOT\Desktop\email remainder`).

1. Run the server:

```
node server.js
```

- Expected Output:

```
✅ Server running on port 3000
✓ Connected to MongoDB
```

Step 3: Insert a Reminder

- Go to **MongoDB Compass** → select `emailReminderDB` → `reminders` collection.
- Click **Insert Document** and fill it like this

- **Tip:** Convert your local IST time to UTC for correct scheduling.

Step 4: Wait for the Scheduled Time

- The **Node.js cron job** checks reminders every minute.
- When the **current time matches the `time` field**, the system sends an email.

Step 5: Check the Email

- Open your **Gmail inbox** (`jenistajones2006@gmail.com`) to verify the reminder.
- Expected Email:
  - **Subject:** Reminder Notification
  - **Message:** `Submit project report`

Step 6: Verify in MongoDB

- After sending, the `sent` field of the document updates to `true`.
- MongoDB Compass shows:

Step 7: Error Handling / Logs

- Node.js console shows cron job activity.
- If an email fails (e.g., wrong credentials), Nodemailer throws an error.
- Always check `.env` for correct `EMAIL_USER` and `EMAIL_PASS`.

Step 8: Optional – Adding Multiple Reminders

- You can insert multiple reminders in MongoDB.
- Cron job will send each email at its scheduled time automatically.

Step 9: Demo Conclusion

- Show the **email received**, **MongoDB document updated**, and **server logs**.
- Explain how it automates reminders and reduces manual work

Project Report

1. Project Title

Email Reminder System

2. Objective

To develop a system that automatically sends email reminders to users at specified times, ensuring they never miss important tasks or deadlines.

3. Technologies Used

- **Frontend:** None (console-based system; optional: React/HTML if needed)
- **Backend:** Node.js
- **Database:** MongoDB (to store reminders)
- **Libraries/Packages:**
  - `nodemailer` (for sending emails)
  - `node-cron` (for scheduling reminders)
  - `dotenv` (for environment variables)
  - `mongoose` (for MongoDB connection)

4. System Architecture

1. **User adds a reminder** → includes email, message, and scheduled time.
2. **Reminder stored** in MongoDB with a `sent: false` status.
3. **Server cron job** continuously checks the database for reminders whose time matches the current time.
4. **Email is sent** using Nodemailer.
5. **Database updated** (`sent: true`) after successful delivery.

5. Features

- Schedule reminders for specific times.
- Automatic email delivery.
- Status tracking of reminders (`sent` or `pending`).
- Works 24/7 while the server is running.

6. Implementation Steps

1. Install Node.js, MongoDB, and required npm packages.
2. Set up `.env` file with email credentials and MongoDB URI.
3. Create **Reminder model** in Mongoose.
4. Write a server script to:
    - Connect to MongoDB.
    - Use `node-cron` to periodically check reminders.
    - Send email using `nodemailer`.
5. Test by inserting sample reminders in MongoDB.

7. Database Design

Collection: `reminders`
Fields:

- `_id` : ObjectId
- `email` : String (recipient email)
- `message` : String (reminder content)
- `time` : Date (UTC timestamp for scheduled reminder)
- `sent` : Boolean (true if email sent)

8. Working

1. User or admin inserts a reminder into the database.
2. Server cron job runs every minute to check for pending reminders.
3. If the current time ≥ reminder time and `sent: false`, an email is sent.
4. Status is updated to `sent: true` to prevent duplicate emails.
5. Logs in the server console show successful email deliveries.

9. Conclusion

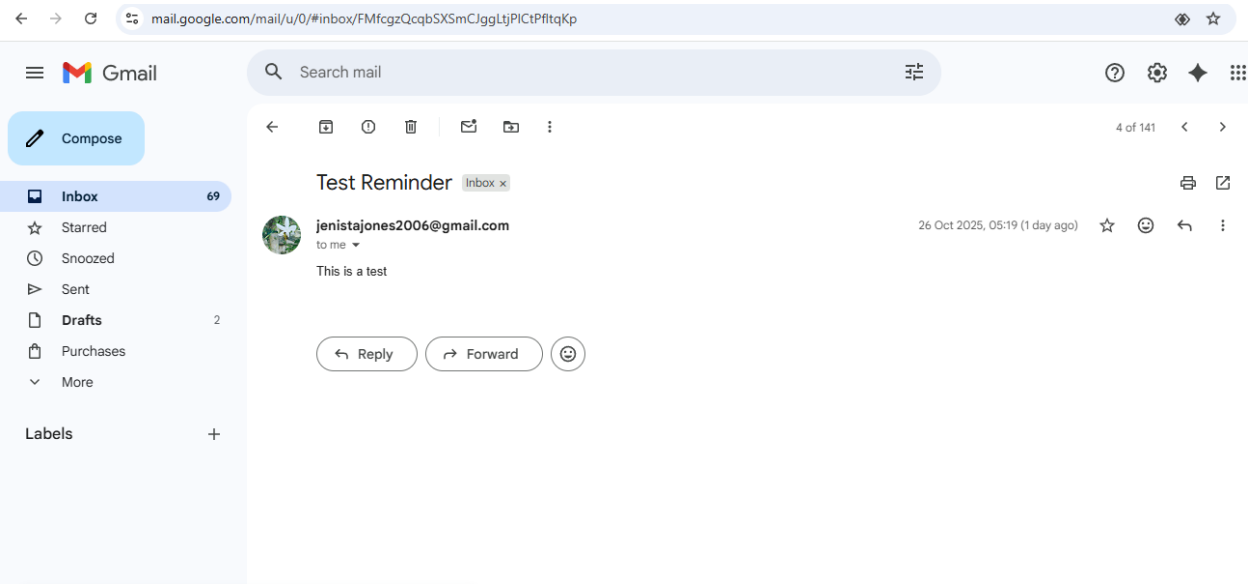
The Email Reminder System is a reliable tool to automate task reminders. It eliminates manual follow-ups, improves productivity, and ensures timely notifications via email.

10. Future Scope

- Add a **web interface** for easier reminder management.
- Include **recurring reminders**.

- Add **SMS notification** alongside email.
- User authentication for multiple users.

SCREENSHOT



Challenges and Solutions

1. Challenge: Time Zone Handling

- Users may be in different time zones, causing reminders to send at incorrect times.
  Solution: Store all times in **UTC** in the database and convert to the user's local time when sending reminders.

2. Challenge: Email Sending Failures

- Emails may fail due to network issues, invalid credentials, or server downtime.
  Solution: Implement **retry logic** and proper **error handling**. Use reliable email services like Gmail, SendGrid, or SMTP with secure credentials.

3. Challenge: Server Downtime

- If the server is not running, scheduled reminders won't be sent.
  Solution: Deploy the server on a **cloud platform** (Heroku, AWS, or Azure) or run it as a **background service**.

4. Challenge: Duplicate Emails

- If reminders are checked multiple times, users may receive duplicate emails.
  Solution: Update the database with a **`sent: true` flag** immediately after sending an email.

5. Challenge: Handling Large Number of Reminders

- Many reminders can slow down the server or cause delays.
  Solution: Use **efficient queries** in MongoDB and **cron jobs** to process reminders in batches.

6. Challenge: Securing Email Credentials

- Storing email passwords in code can be insecure.
  Solution: Use a **`.env` file** or **environment variables** to store credentials securely.

Github README and setup guide

Features

- Schedule email reminders by adding documents to MongoDB.
- Automatically sends emails at the specified time.
- Tracks whether a reminder has been sent to avoid duplicates.
- Easy to configure with environment variables.

Technologies Used

- **Node.js** – Backend runtime environment
- **Express.js** – Server framework
- **MongoDB** – Database to store reminders
- **Node-cron** – Task scheduler
- **NodeMailer** – Sending emails
- **dotenv** – Managing environment variables

Setup Guide

1. **Clone the repository**: Download the project from GitHub using the repository URL.
2. **Navigate to the project folder** in your terminal.
3. **Install dependencies** by running `npm install`.
4. **Create a `.env` file** in the root directory and add the following variables:
    - `EMAIL_USER` – your email address
    - `EMAIL_PASS` – your email password or app password (for Gmail with 2FA)
    - `MONGO_URI` – your MongoDB connection string
    - `PORT` – the port number for the server (e.g., 3000)
5. **Start MongoDB** locally or connect to a cloud MongoDB instance.
6. **Start the server** by running `node server.js`.
7. **Add reminders** to the `reminders` collection in MongoDB. Each reminder must include `email`, `message`, `time` (in UTC), and `sent` (false).
8. **The system automatically sends emails** at the scheduled time and updates the `sent` status to true.

How it Works

- The server continuously checks the database for reminders that are not yet sent.
- When the scheduled time is reached, NodeMailer sends the email.
- The database is updated so that each reminder is sent only once.

Future Improvements

- Add user authentication for personalized reminders.
- Create a front-end interface for easier management.

- Use a queue system like RabbitMQ for handling large-scale reminders.

Final Submission

Repository link:

https://github.com/jones2605/Email-Rimender-System.git