To implement this custom email-sending application, follow these high-level steps to meet all the requirements:

**1. Set Up Your Development Environment**

* **Frontend:** Use a frontend framework such as React.js to build the dashboard for the user interface.
* **Backend:** Use Node.js with Express for handling API requests (e.g., for sending emails, fetching Google Sheets, or handling CSV uploads).
* **Database:** Use a database like MongoDB or PostgreSQL to store email scheduling data, email statuses, and analytics.
* **Authentication:** Implement OAuth2 authentication for users to securely connect their email accounts.
* **Email Service Provider (ESP):** Integrate an ESP such as SendGrid, Mailgun, or Amazon SES for sending emails and tracking delivery statuses.
* **LLM API:** Integrate with a content-generation API (e.g., OpenAI, Groq API) for email customization.
* **Scheduling:** Use a task scheduler like **Celery** (with a backend such as Redis) or **node-cron** to manage email scheduling and throttling.

**2. Detailed Steps to Implement Features**

**A. Data Connection (Google Sheets / CSV Upload)**

* **Google Sheets Integration:** Use the Google Sheets API to allow users to connect to their Google account and read data from a sheet (e.g., columns like Company Name, Email, etc.).
  + Use the googleapis library in Node.js to fetch data from Google Sheets.
* **CSV File Upload:** Provide an option to upload CSV files and read their contents using libraries like csv-parser or papaparse.

**Steps:**

* Set up Google Sheets API access (OAuth2).
* Fetch the data from the connected sheet or uploaded CSV and store it in memory or a database.

**B. Email Integration (OAuth2 for Email Account Connection)**

* **OAuth2 for Gmail/Outlook:** Implement OAuth2 authentication to allow users to securely connect their Gmail or Outlook accounts.
* **SMTP/ESP:** Provide options for connecting via SMTP for custom email accounts or use ESPs like SendGrid.

**Steps:**

* Use the google-auth-library and nodemailer (or equivalent) for connecting to the email provider.
* Integrate ESP APIs like SendGrid for managing email delivery and events.

**C. Customizable Prompt Box for Email Content**

* Create a prompt box in the frontend to allow users to input their email template with placeholders.
* Dynamically replace placeholders with values from the dataset (e.g., {Company Name}).

**Steps:**

* On the frontend, allow users to input placeholders in the prompt.
* On the backend, replace the placeholders with the actual data for each email.

**D. Column Detection and Dynamic Field Replacement**

* Automatically detect column names in the dataset and allow users to insert them as placeholders.

**Steps:**

* On CSV/Sheet data fetch, display column names in the frontend.
* Let users insert these dynamic columns into the email template.

**E. Email Customization and Sending**

* Use an LLM API like OpenAI’s GPT or Groq API for dynamic email content generation based on the user’s prompt.
* Use ESP APIs like SendGrid or Mailgun for email sending.

**Steps:**

* Call the LLM API to generate email content for each row in the dataset.
* Send emails using ESP integration.

**F. Email Scheduling and Throttling**

* Implement email scheduling (e.g., node-cron) to allow users to set a schedule for when emails should be sent.
* Implement throttling to ensure that emails are sent at a rate that complies with ESP limits.

**Steps:**

* Store scheduled time in a database.
* Use a cron job or task scheduler (like Celery or node-cron) to send emails based on the schedule.
* Allow the user to define throttling limits (e.g., send 50 emails per hour).

**G. Real-Time Analytics for Sent Emails**

* Display analytics on the dashboard, such as:
  + Total emails sent
  + Emails pending
  + Emails failed
  + Response rate

**Steps:**

* Use WebSocket or polling to show real-time analytics.
* Update email status regularly in the backend (sent, failed, pending) and reflect it in the frontend dashboard.

**H. Email Delivery Tracking with ESP Integration**

* Use ESP (e.g., SendGrid, Mailgun, or Amazon SES) to track email delivery status (e.g., delivered, bounced).
* Display the email delivery status in real-time on the dashboard.

**Steps:**

* Use ESP's API/webhooks to listen to delivery events (e.g., delivered, bounced, opened).
* Store and update email statuses in the database.

**I. Real-Time Dashboard for Email Status and Tracking**

* Build a dashboard with a real-time status indicator showing the progress of email sending.
* Display email statuses (e.g., Sent, Scheduled, Failed) and delivery tracking (e.g., Delivered, Opened, Bounced).

**Steps:**

* Use front-end technologies (e.g., React) to build a real-time dashboard.
* Use a backend API to fetch and update the status of emails.

**3. Technical Requirements & Libraries**

* **Backend:** Node.js, Express, nodemailer, googleapis (for Google Sheets integration), axios, celery (for Python-based scheduling), node-cron (for Node.js scheduling)
* **Frontend:** React.js, WebSocket (for real-time updates)
* **Database:** MongoDB or PostgreSQL to store email statuses, analytics, and schedules.
* **Authentication:** OAuth2 for Gmail/Outlook authentication.
* **ESP Integration:** SendGrid, Mailgun, Amazon SES for email sending and delivery tracking.
* **Content Generation:** OpenAI API, Groq API, or another LLM for generating email content.
* **Deployment:** Docker for containerization (optional), Heroku/AWS for deployment.

**4. Development Phases**

1. **Backend Setup:**
   * Implement API endpoints for Google Sheets/CSV reading, email sending, and analytics tracking.
   * Set up OAuth2 authentication for email integration.
   * Implement scheduling and throttling.
2. **Frontend Setup:**
   * Create the dashboard to display email statuses and real-time analytics.
   * Implement form inputs for custom email content and scheduling.
3. **Integration:**
   * Integrate the email sending logic with an ESP.
   * Implement email content generation via LLM API.
   * Set up real-time delivery tracking and analytics.
4. **Testing & Optimization:**
   * Test email sending, scheduling, and throttling functionalities.
   * Ensure the dashboard updates in real-time.

**5. Documentation**

In your README.md file:

* **Setup Instructions:** Explain how to configure OAuth2, API keys, and ESP settings.
* **Usage Instructions:** Detail how to upload data, configure the email content, and schedule emails.
* **Database Setup:** Instructions for configuring the database.
* **API Endpoints:** Document the backend API routes for sending emails, fetching statuses, and scheduling.

**6. GitHub Submission**

* Submit the code via GitHub.
* Include a README.md with setup and usage instructions.
* Optionally, include a short video demonstrating the solution.

OPEN AI API KEY : sk-proj-6-MyGEPuX-VI0MJlQFIq9mzqt0pRDu\_9oWiEl5f3Vyc8QQZHQfy9FCVUEWm92luHv4whOmqU52T3BlbkFJJVdY-TG1SyoSlTS8\_Pi-aEEZUENZZpq4wgNMFgxZy5fgzUbrvFvO1q2ptGlcFUYQ2v2wNP11YA