**Objective:** Create a simple Django REST API that allows users (or frontend apps) to send email reminders. The API accepts the recipient’s email address, a subject, and a message, then sends the email and logs the details in a PostgreSQL database.

# 1. Project Setup & PostgreSQL Integration

- Django project created with the default structure.  
- PostgreSQL is used instead of SQLite for better performance, concurrency, and production readiness.  
- Database configured in settings.py with the correct credentials, host, port, and database name.

# 2. Model: EmailSend

- Defined in reminders/models/reminder\_model.py.  
- Fields:  
 \* to\_email: EmailField to store recipient email.  
 \* subject: CharField for email subject.  
 \* message: TextField for email content.  
 \* sent\_at: DateTimeField auto-set to when the email is saved.  
- Purpose: Keep a record of all emails sent by the API for auditing or future reference.

# 3. Serializer: EmailSendSerializer

- Converts between Python model instances and JSON.  
- Used by Django REST Framework to validate incoming API data and serialize responses.  
- Ensures to\_email, subject, and message are passed correctly in the API requests.

# 4. Email Configuration

- Configured SMTP email backend in settings.py for Gmail:  
 \* Host: smtp.gmail.com  
 \* Port: 587 with TLS encryption  
 \* User and password: your Gmail address and app password.  
- This allows Django to send real emails via Gmail’s SMTP server.  
- Alternative backends:  
 \* Console backend for local testing without sending real emails (prints emails to console).

Remind-me-later Project Explanation

# 5. API View: send\_email\_reminder - Decorated with @api\_view(['POST']) to accept only POST requests. - Flow: 1. Accepts JSON input with to\_email, subject, and message. 2. Validates input using EmailLogSerializer. 3. Calls Django’s send\_mail() with provided data. 4. If sending succeeds, saves the email log in the database. 5. Returns a success response with the saved email log data. 6. If sending fails, returns an error message with HTTP 500 status. 7. If validation fails, returns errors with HTTP 400 status.

# 6. URLs Configuration

- The API endpoint is /api/reminders/send-email/.  
- The main project URLs include reminders.urls so the endpoint is reachable.  
- This separation makes the project modular and scalable.

# 8. Migrations

- Used makemigrations and migrate commands to create the database tables for the new model.  
- Ensures the PostgreSQL database has the correct schema.

# 9. Testing the API

- Use Postman or curl to send POST requests with JSON body.  
- Example request body:  
 {  
 "to\_email": "recipient@gmail.com",  
 "subject": "Test Reminder",  
 "message": "This is a test reminder message."  
 }  
- Expected response confirms email was sent and logged.

# Notes & Best Practices:

- Gmail accounts with 2FA require an App Password for SMTP authentication.  
- For production, consider using environment variables to store sensitive info (EMAIL\_HOST\_USER, EMAIL\_HOST\_PASSWORD).  
- Use secure email sending practices and monitor for failures.  
- The app currently only sends emails and logs them; scheduling the reminders or sending SMS is outside scope but can be added later.  
- PostgreSQL is a robust choice for production compared to SQLite which is mainly for development/testing.