Assessment Task: Queued Email Delivery & Logging Service

You are tasked with building a small backend service in **NestJS** that allows users to send emails via an API. Emails must **not** be sent immediately — instead, they should be **queued and processed in the background**. Every email, whether sent successfully or failed, must be logged in the database with its status and timestamps.

Requirements

- The application must have **two endpoints**:
  - **POST** `/send` – Send an email to a specific recipient.
  - **GET** `/logs/email` – Retrieve a **paginated** list of email logs.
- All email records must be stored in a database (**PostgreSQL** or **MongoDB**).
  - Use **Prisma** or **Mongoose**, depending on your preferred database.
- The logs should provide a clear report of emails sent through the system, including:
  - Total emails sent today
  - How many were successful
  - How many failed
  - `createdAt`, `sentAt`, and `failedAt` timestamps (where applicable)
  - Error details for failed deliveries
- Use **Bull** or **BullMQ** for job management (queue-based processing).

Bonus Features *(Optional – if you have time after completing the core requirements)*

- **Rate limiting** on the `/send` endpoint.
- **Request and response logging** to track the lifecycle of each request in the console (no need to store logs). You may use **Winston**, **Pino**, or any other logging library.
- A **Dockerfile** to containerize the NestJS application, and a **docker-compose.yml** to run all containers.

Resources

- **SMTP Server:** Brevo or any other, as you prefer
- For the database, you may use Docker images, or if you prefer cloud, then check below:
  - PostgreSQL: Neon
  - MongoDB: Atlas
  - **Redis (for Bull/BullMQ):** Upstash

Deliverables

- Publicly accessible **GitHub** (or other platform) repository link.
- **Postman** documentation (public link or exported JSON collection).
- Any necessary secrets (`.env` file) required to run the application.
- A brief explanation of your system design process, approach, and structure. This should cover the following terms:
    - Why have you chosen this database
    - Is your system scalable? If yes, then explain how, and if no, then what approach can make it scalable?

Deadline

Last submission time: 12:30 PM, 17th August 2025

Note: Please aim to complete all requirements, but even if you cannot finish everything, submit what you have. Your approach, problem-solving, and effort will also be part of the evaluation.