API FLOW

1. **/email/send**
    - **Phase One (Without queue)**
        - Client hit a POST request to **/email/send**
        - First Global Middleware & Guard will be validated
        - Then route level rate limit will be check
            - If failed
                - Thrown error
        - After that DTO validation
            - If failed
                - Thrown error
        - Then Come to the controller
            - Then service will call
                - Create **PENDING** log in DB
                    - If failed
                        - Thrown error
                - Enqueue Job in Redis Queue Name “email”
                    - If failed
                        - Thrown error
                        - Update log to **FAILED** in database
                    - If success
                        - Return 201 Create (Pending)
    - **Phase Two** (Inside Queue)
        - Job Picked by Worker
            - Send Email via SMTP
                - If success
                    - Update log status to **SENT** in DB
                        - Job Complete (Done)
                - If failed
                    - Retry by the worker
                        - Worker will try till the max attempt set (3)
                            - After 3 times try
                                - If failed

        - Update log to FAILED in DB” (Update log to FAILED in DB”)
    - If success
        - Update log to SENT in DB” (Job complete)

Does It scalable?

Answer: I think /email/send API is scalable one because, I use a queue based architecture, where worker are handling the sent process

But, there have a drawback that can be solve, The problem is the queue just pass it to another SMTP server but default the SMTP server is not reply any response that the email has sent or not. The solution I find out that, there need to create a webhook for that and in that webhook the response will sent by SMTP server based on that the database update will be happen.

Currently, Only nodemailer related error comes as a reason of failed message sent.

2. **/email/logs**
    - Client Send GET request
        - With page && limit
    - Then Global Middleware and Guard will hit
    - Then global rate limit check
        - If failed
            - Thrown Too many request error
    - Then controller will call
    - Then Service will call
        - Sanitize Input (safePage, safeLimit, skip)
        - Find time (startUtc, endUtc)
        - Then start Parallel DB Operations
            - Operation
                - Fetch paginated Logs details
                    - Body
                    - To
                    - From
                    - Create_at
                    - Failed_at
                    - Sent_at
                    - Subject
                    - Error message
                - Count total logs
                - Aggregate today's email sent details
                    - Total create today
                    - Sent today
                    - Failed today
            - If failed
                - Thrown error
            - If success
                - Return response with
                    - Pagination
                    - Today's email details

- All email items sort by ASC

Does it Scalable?

Answer: I think, **/email/logs** is scalable for moderate traffic with pagination which is a OFFSET technique.

But, When the traffic increase than need to implement indexes, caching, read replica and need to change the logic of pagination. It need to switch to either cursor based or hybrid where offset technique will apply for small range like page 1 – 10 and cursor for endless scrolling.

However, I prefer the cursor technique where, first request client will ask with a limit like /email/logs?limit=10 and server will return 10 latest item including a cursor which is basically the the last items's id

In Addition, When user hit second time the cursor will be passed and that time server will give only those data which is create before the cursors's item. And will return next 10 data as the limit is 10 including a new cursor which is the last item id.

Also, there can be currently to get data query I used parallel query by using Promise.all() but, there some drawback of it will create multiple connection pool as a solution I prefer to convert it to transaction mode and as isolation layer I do prefer Repeatable Read because, each time the query will happen from a fix snapshot sothat all query will maintain a fix dataset.

However, There have a another approach to deal with this situation by using caching layer by using Redis. There will be a separate api for get today's total email, sent, failed count and that data will be fetch from DB using transaction Repeatable Read isolation layer and keep it cache in Redis. When a new logs will be attach only that time database pool connection will setup and get cache into redis.

In addition when user hit to get /email/logs api it will fetch only the current logs list and give the report based analysis for example: total email sent count, successful, failed count from cache server.

But this use case will be good if email sent rarely or not by too many user.

Lastly, I want to say that, for moderate traffic, I think parallel approach which is a simple and good one, using caching layer which is good if email has send rarely or get request

applied rarely, but if traffic raised highly then, using transaction with Repeatable Read isolation layer would be nice approach.

Why do I choose PostgraSQ?

Answer: The service what I had to build which store a structured logs for every email. Each record contained a unique ID, subject, body, status, various kind of timestamp for creation, sending and failure and any error message.

I think, postgraSQL is a well balanced relational database that can storing and querying this kind of structured data. It supports strong consistency and ACID transaction, which are import for ensuring log integrity, though I used Parallel process here because of independent task.