

Assumptions

- Authentication and authorization are not detailed here, but all APIs must be protected with role-based access control
- Reports are generated from all available documents unless future requirements specify user or trial specific filtering.
- Each report is shared among all subscribed users
- Report generation involves long-running, resource-intensive processing, requiring parallelization and retry mechanisms.
- The system must be fault tolerant, with retries and dead-letter queues for failed jobs.
- Workers can access the s3 bucket and database
- The focus of this document is to deliver the conceptual solution, not to detail every possible scenario

Time log

- 1h concept and initial version
- 2h system diagram and structuring the document
- 1h review and rework
- 1 h review and rework

API Changes

New endpoints

POST /api/reports/registered-users

- Registers a user for a weekly report

DELETE /api/reports/registered-users/{email}

- Removes a user from a weekly report

GET /api/reports/registered-users

- Retrieve a list of all users registered for the weekly report, has to be a paginated response to be scalable

GET /api/documents

- Retrieve all documents, has to be paginated response in order to be able to handle high volume of documents

POST /api/reports

- Initiates report generation, needs to have logic to prevent sending duplicate emails

PATCH /api/reports/{id}/state

- Endpoint used to update state

PATCH /api/reports/{id}/result

- Endpoint to add the report creation result

GET /api/reports/{id}

- Get specific report

GET /api/reports

- Retrieve all existing reports, response should be paginated in order to handle number of reports

POST /api/reports/emails/send/{reportId}

- Initiates the sending of the reports to customers via email

POST /api/reports/emails

- Creates a record of a scheduled email sent to a customer

GET /api/reports/emails/{reportId}

- Get all emails that have been scheduled or sent for a report

PATCH /api/reports/emails/{id}

- Updates the status of the email send for a customer for a specific report

New components

Components that have been added to the existing system are following:

- **Report generation queue** and corresponding DLQ, handles messages with requests for report generation.
- **Report generation worker** service planned to handle report generation based on the existing documents.
- **Report customer send queue** and corresponding DLQ, handles messages with request to send report to users registered for weekly report.
- **Customer email scheduling worker** service is responsible for handling all requests for scheduling email send to users registered for weekly report.
- **Email queue** and corresponding DLQ, handles messages generated by **Customer email scheduling worker**
- **Send email worker** service is responsible for sending emails using a third party service to users registered for weekly report.
- **Bounced email queue** and corresponding DLQ, handles messages sent by Email service which contain emails that were not delivered to the users
- **Bounced email worker** is responsible for handling bounced user emails and updating the email status.

Solution summary

- Customer can subscribe via frontend app, using **POST /api/reports/registered-users**
- Customer can unsubscribe via frontend app, using **DELETE /api/reports/registered-users/{email}**
- Report generation is scheduled and triggered by a cron service that calls **POST /api/reports** endpoint to initiate the process. This endpoint records this in DB in a new table that is used to keep track of report generation and sends a message to **Report**

generation queue to initiate report generation. This process has to implement a logic to prevent excessive / unwanted report generation, with a possibility of override.

- **Report generation worker** monitors the **Report generation queue** and runs the report generation that involves processing text files and images to create a report. As soon as the message is taken from the queue, report status is updated via **PATCH /api/reports/{id}/state** to indicate that the report generation process has started.
- Once the report generation completes the report, the result is uploaded to S3 and the API endpoint **PATCH /api/reports/{id}/result** is called to update the report status and to return the path of the report on S3.
 - In case of report generation failure, report state is updated appropriately via **PATCH /api/reports/{id}/state**
- Once called **PATCH /api/reports/{id}/result** endpoint will also trigger **POST /api/reports/emails/send/{reportId}** (This process is shown in the simplified way as I don't think you wanted the implementation details in this assignment)
- **POST /api/reports/emails/send/{reportId}** endpoint will send a message to **Report customer send** queue
- **Customer email scheduling worker** will process messages sent to **Report customer send** by retrieving all the email that the email needs to be sent to (using **GET /api/reports/registered-users** endpoint), and retrieve report via **GET /api/reports/{id}** all information needed for actual email send will be prepared, registered via **POST /api/reports/emails** and then send as messages to the **Email queue**.
- **Send email worker** processes messages from **Email queue** and reports the result to **PATCH /api/reports/emails/{id}**. Splitting customer email scheduling worker and Send email worker is done so that email sending can be scaled separately.
- Bounced emails would be sent to **Bounced email queue** and then processed by **Bounced email worker** which would update the state of the email **PATCH /api/reports/emails/{id}**

Note: I focused on explaining the flow though the solution without making divergence for any faults and issues that could occur during report generation and email sending.

Improvements and Best Practices

- Parallelization of report generation tasks to handle large datasets efficiently.
- Idempotent API design to prevent duplicate processing and ensure reliability.
- Role-based access control, encryption in transit and at rest, and audit logging for compliance with healthcare regulations
- Comprehensive monitoring (metrics, logs, alerts) to ensure reliability and quick issue resolution.