Uploading Big Files (AWS)

ERD (Engineering Requirements Document)

Desarrollado por	**Estado**
José Cabrera	Done

Introduction

This document describes the functional and non-functional requirements and the architectural and security considerations for implementing a file upload system based on pre-signed URLs to Amazon S3. The objective is to eliminate direct file uploads (images and videos) through the backend, reducing resource usage and improving system scalability.

Objective

Purpose

Implement a mechanism that allows clients (front-end) to obtain a pre-signed URL from the backend to upload files directly to Amazon S3.

References:

- **https://medium.com/@aidan.hallett/securing-aws-s3-uploads-using-presigned-urls-aa821c13ae8d**
- **https://docs.aws.amazon.com/AmazonS3/latest/userguide/using-presigned-url.html**
- **https://aws.amazon.com/blogs/compute/uploading-large-objects-to-amazon-s3-using-multipart-upload-and-transfer-acceleration/**

Expected Benefits

- **Resource Reduction**: Decrease memory and disk usage on the backend.
- **Improved Scalability**: Delegate file transfer to S3, which is optimized for handling large files and concurrent uploads.
- **Enhanced Security**: AWS credentials remain exclusively on the backend.

Scope

Functionality

A backend endpoint (using FastAPI) will be implemented to:

- Accept basic file parameters (name, type, size).
- Generate a pre-signed URL for uploading (using either PUT or POST) directly to S3.
- Return the pre-signed URL and the file key so the front-end can directly upload to S3.

Exclusions

- Integrity validations or file transformations on the backend.
- Handling callbacks or post-upload confirmation (this may be considered as optional functionality).

Functional Requirements

Pre-Signed URL Generation

FR1: The endpoint must require mandatory parameters such as `fileName`, `fileType`, and `fileSize`. (high priority)

FR2: A key must be generated to uniquely identify the object in S3 (e.g., by placing it under an `uploads/` folder). (high priority)

FR3: The endpoint will use Boto3 to invoke AWS S3's `generate_presigned_url` method. (high priority)

FR4: The URL must have a configurable expiration time (e.g., 3600 seconds). (high priority)

Endpoint Response

FR5: The endpoint must return the generated pre-signed URL and key in a structured JSON format. (high priority)

Example:

```json
{
  "url": "https://your-bucket.s3.amazonaws.com",
  "key": "123e4567-e89b-12d3-a456-426614174000",
  "status": "pending"
}
```

FR6: In case of an error (e.g., failure to connect to S3), the endpoint should return an HTTP 500 error with an appropriate message. (high priority)

Security and Validation

FR7: The endpoint must validate input parameters (e.g., ensuring accepted formats for `fileName`, `fileType`, and `fileSize`). (high priority)

FR8: The backend must ensure that pre-signed URL generation only occurs if the user is authorized (using appropriate authentication and authorization mechanisms). (low priority)

Non-Functional Requirements

Performance

The generation of the pre-signed URL should be completed in under 200 milliseconds.

Scalability

The solution must support multiple concurrent requests without degrading backend performance.

Security

AWS credentials must not be exposed on the front-end. The pre-signed URL should have a short expiration to limit its usage.

Maintainability

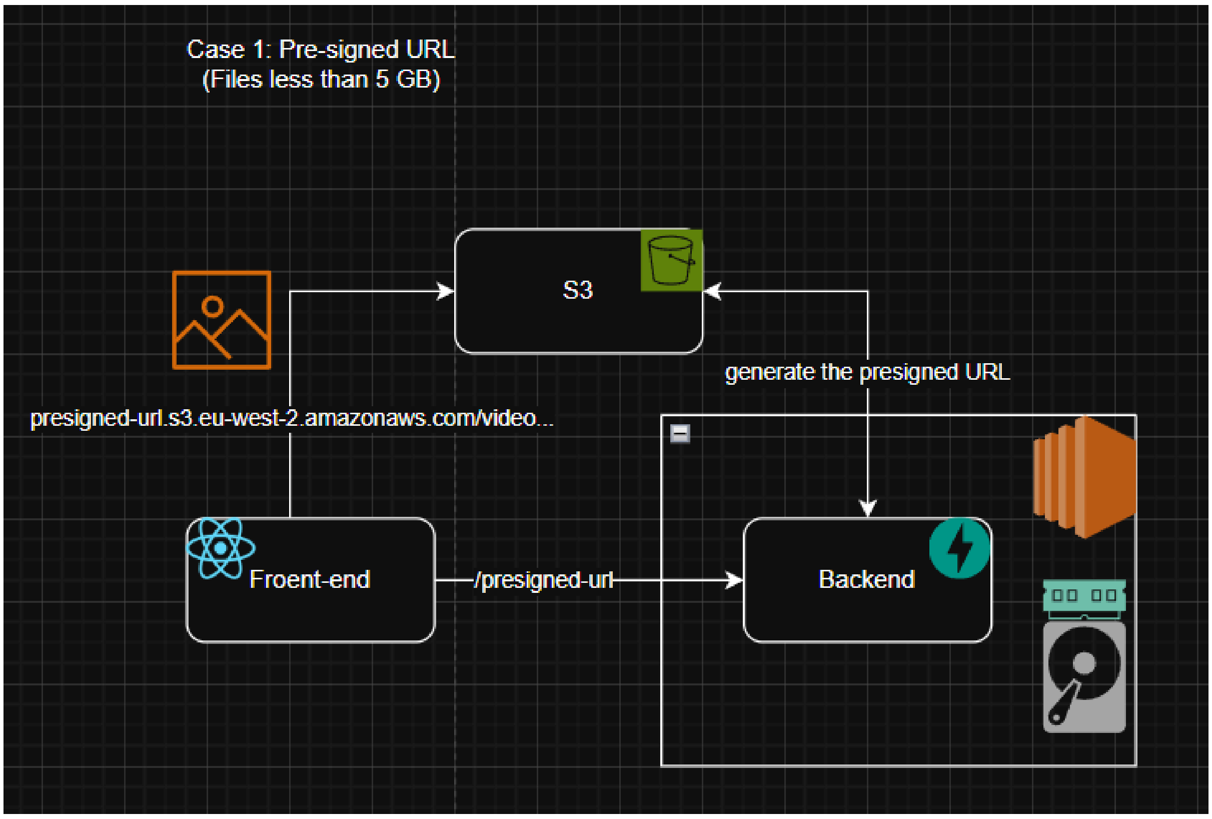
The code must be well-documented and adhere to Python coding standards.

Monitoring and Logging

Errors and key metrics related to pre-signed URL generation should be logged and monitored.

Architecture and Design

Flow Diagram



drawio

Description of the Flow

Request for the Pre-Signed URL

The front end (e.g., a web or mobile application) requests the back end to obtain a pre-signed URL. The request includes basic file information such as name and type.

URL Generation in the Backend

The backend communicates with S3 (using AWS credentials and configurations) to request the creation of a pre-signed URL. This URL contains a cryptographic signature and restrictions (such as expiration and HTTP method), allowing S3 to accept the file upload without exposing AWS credentials.

Delivery of the URL to the Front-End

The backend returns the pre-signed URL to the front-end. Thus, the file does not pass through the backend server; it is uploaded directly to S3.

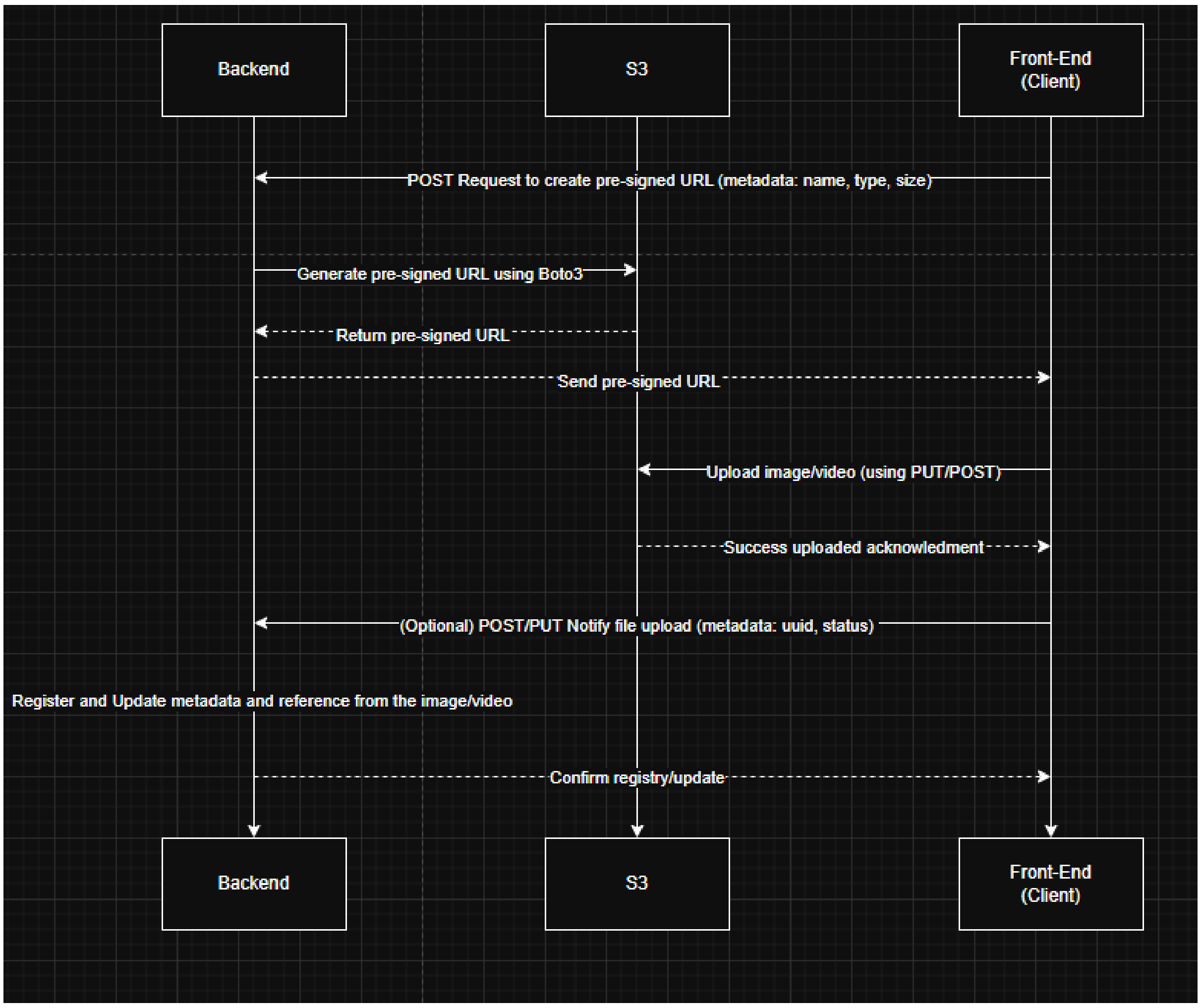
Direct Upload to S3

The front end uses the pre-signed URL to send (PUT) the file to S3. S3 validates the signature and parameters. The file is accepted and stored in the designated bucket if everything is correct.

Storage in S3

The file becomes available in S3, and the backend (or other components) can access it using the key or public URL (if configured).

Request-Response Diagram


Backend
S3
Front-End (Client)
POST Request to create pre-signed URL (metadata: name, type, size)
Generate pre-signed URL using Boto3
Return pre-signed URL
Send pre-signed URL
Upload image/video (using PUT/POST)
Success uploaded acknowledment
(Optional) POST/PUT Notify file upload (metadata: uuid, status)
Register and Update metadata and reference from the image/video
Confirm registry/update
Backend
S3
Front-End (Client)

draw.io

Steps Request-Response

1. The client requests the pre-signed URL from the backend.
2. The backend generates the URL using Boto3 after contacting S3.
3. The backend returns the pre-signed URL and file key to the client.
4. The client uploads the file directly to S3 using the URL.
5. S3 sends an acknowledgment to the client upon successful upload.

Design Considerations

Decoupling

The backend only generates the URL; it does not handle the actual file transfer.

Flexibility

The solution must allow modifications to parameters like expiration time or key path without significant refactoring.

Validation

Implement input validations in the backend to prevent malicious or incorrect requests.

Assumptions and Dependencies

Assumptions

- The S3 service is assumed to be correctly configured and accessible from the backend environment.
- Front-end clients must know and handle the file upload flow using pre-signed URLs.

Dependencies

- **Boto3:** For integration with AWS S3.
- **FastAPI:** For developing the API endpoints.
- **Authentication and Authorization Mechanisms:** Must already be in place in the system.
- **Database and ORM (Ormar with Postgres):** For managing file status and metadata.

Security Considerations

The endpoint must be protected using authentication methods (e.g., JWT).

The pre-signed URL must have a short expiration and be restricted to a specific action (PUT or POST).

Controls should be implemented to prevent abuse (e.g., rate limiting, validating file size and type).

Implementation Plan / MVP

For the Minimum Viable Product (MVP), the following changes will be implemented in the backend:

/presigned-url Endpoint:

- **Method:** POST
- **Parameters:** `filename(str)`, `type(str)`, `file_size(int)`
- **Function:** Generates a pre-signed URL and returns it in the response.

/file-status Endpoint:

- **Method:** PUT
- **Function:** Updates the status of the resource (file) with possible states: `pending`, `uploaded`, `processing`, `completed`, `failed`.

Technical Details

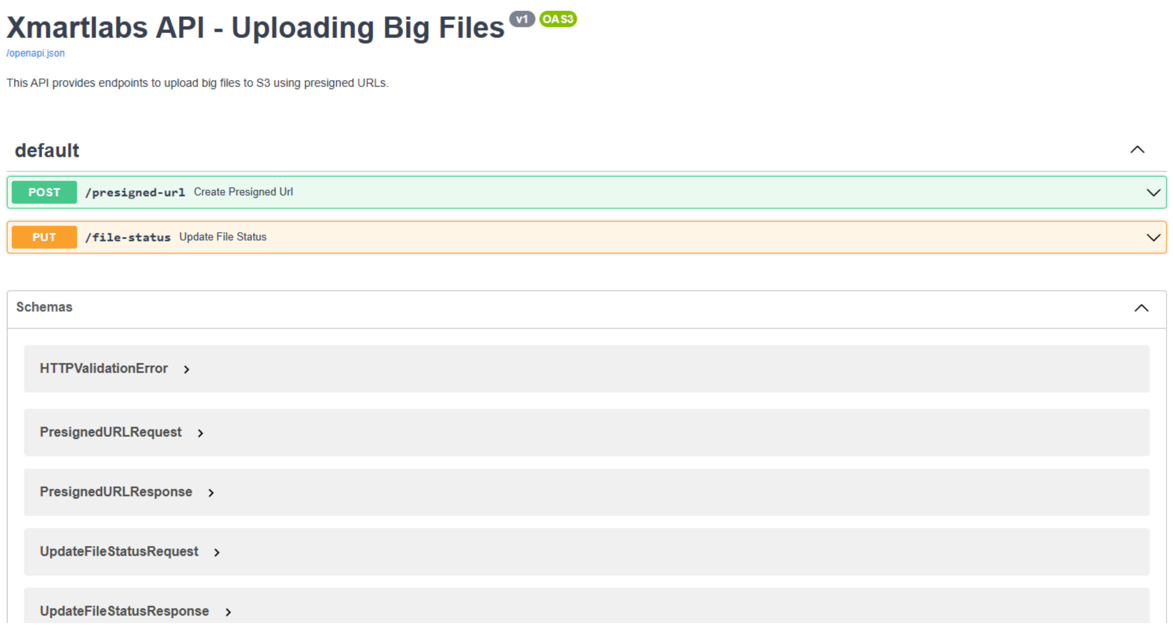
Framework and Tools:

FastAPI for API creation.

Ormar ORM for data management with Postgres.

Docker for containerizing the project and its dependencies (specified in `requirements.txt`).

Swagger for auto-generated API documentation.


Xmartlabs API - Uploading Big Files v1 OAS3

/openapi.json

This API provides endpoints to upload big files to S3 using presigned URLs.

default

POST `/presigned-url` Create Presigned Url

PUT `/file-status` Update File Status

Schemas

HTTPValidationError

PresignedURLRequest

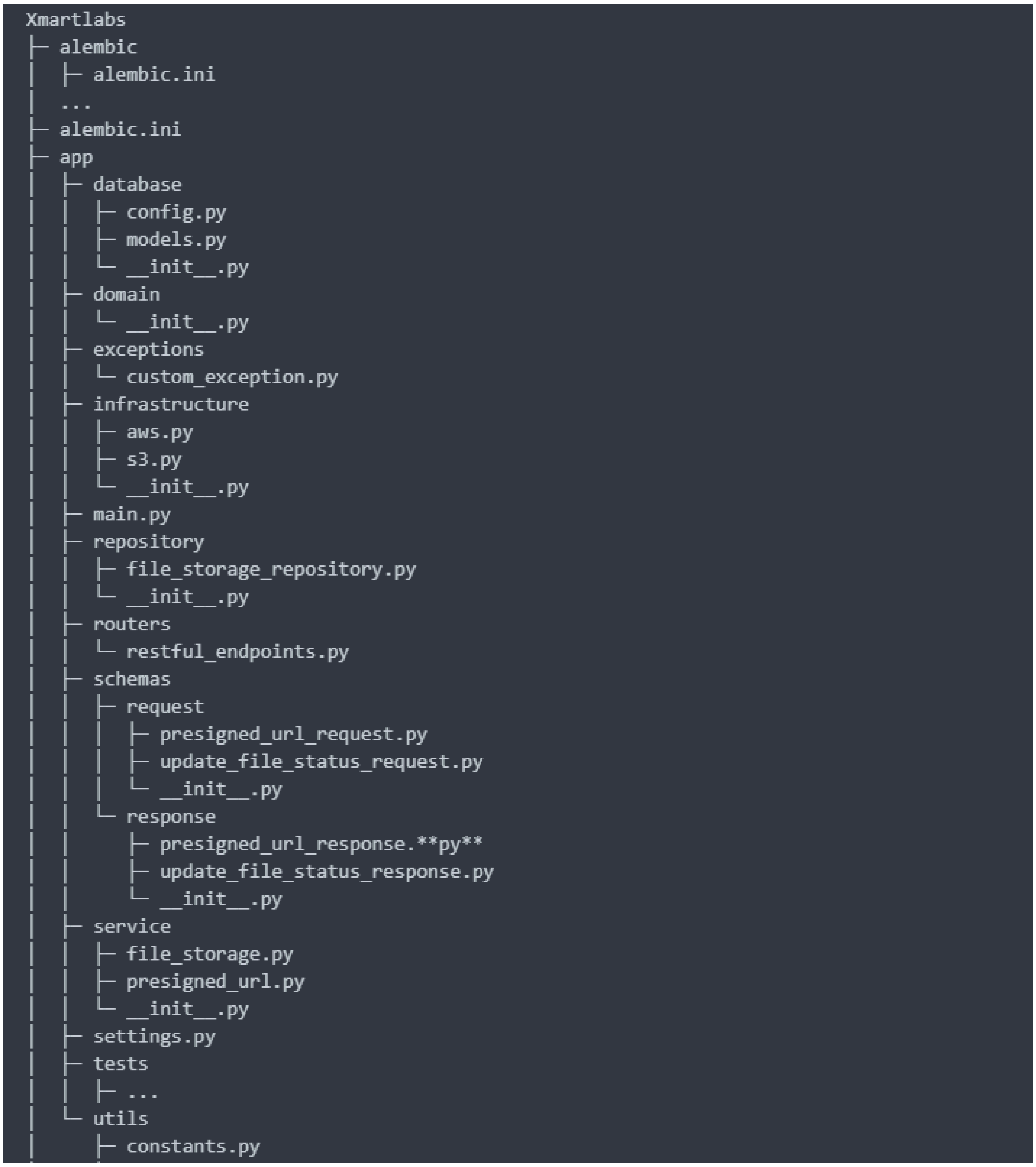
PresignedURLResponse

UpdateFileStatusRequest

UpdateFileStatusResponse

Architecture:

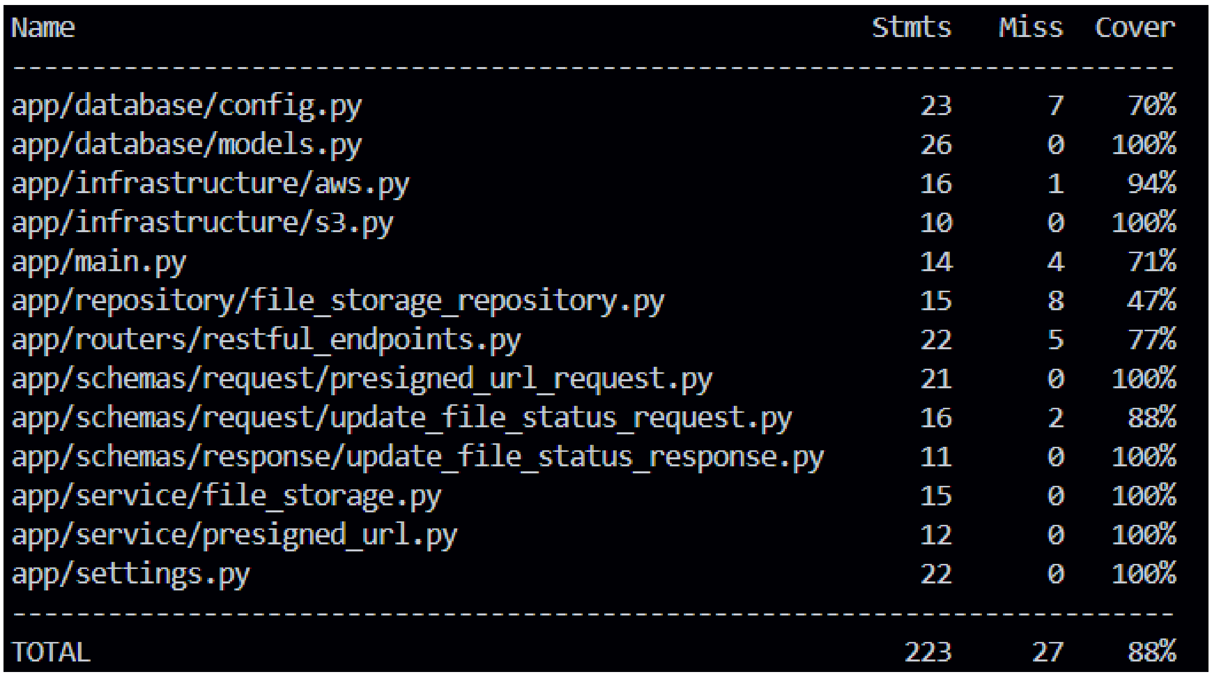
The project follows a Clean Architecture (Onion Architecture) variant with Hexagonal Architecture principles to enhance maintainability and scalability.



```
Xmartlabs
├─ alembic
│  ├─ alembic.ini
│  ...
├─ alembic.ini
├─ app
│  ├─ database
│  │  ├─ config.py
│  │  ├─ models.py
│  │  └─ __init__.py
│  ├─ domain
│  │  └─ __init__.py
│  ├─ exceptions
│  │  └─ custom_exception.py
│  ├─ infrastructure
│  │  ├─ aws.py
│  │  ├─ s3.py
│  │  └─ __init__.py
│  ├─ main.py
│  ├─ repository
│  │  ├─ file_storage_repository.py
│  │  └─ __init__.py
│  ├─ routers
│  │  └─ restful_endpoints.py
│  ├─ schemas
│  │  ├─ request
│  │  │  ├─ presigned_url_request.py
│  │  │  ├─ update_file_status_request.py
│  │  │  └─ __init__.py
│  │  └─ response
│  │     ├─ presigned_url_response.**py**
│  │     ├─ update_file_status_response.py
│  │     └─ __init__.py
│  ├─ service
│  │  ├─ file_storage.py
│  │  ├─ presigned_url.py
│  │  └─ __init__.py
│  ├─ settings.py
│  ├─ tests
│  │  ├─ ...
│  └─ utils
│     ├─ constants.py
```

Test Coverage:

The application currently has an 88% test coverage.



```
Name                                                  Stmts   Miss  Cover
-------------------------------------------------------------------------
app/database/config.py                                   23      7    70%
app/database/models.py                                   26      0   100%
app/infrastructure/aws.py                                16      1    94%
app/infrastructure/s3.py                                 10      0   100%
app/main.py                                              14      4    71%
app/repository/file_storage_repository.py                15      8    47%
app/routers/restful_endpoints.py                         22      5    77%
app/schemas/request/presigned_url_request.py             21      0   100%
app/schemas/request/update_file_status_request.py        16      2    88%
app/schemas/response/update_file_status_response.py      11      0   100%
app/service/file_storage.py                              15      0   100%
app/service/presigned_url.py                             12      0   100%
app/settings.py                                          22      0   100%
-------------------------------------------------------------------------
TOTAL                                                   223     27    88%
```

Acceptance Criteria

The endpoint must return a valid pre-signed URL for file upload to S3.

The URL must have a configured expiration and be limited to the specific HTTP method.

The system should reject requests with missing or invalid parameters.

AWS credentials must not be exposed anywhere in the flow.

Risks and Mitigations

Risk	Mitigation
Potential abuse of the endpoint for generating multiple URLs.	Implement rate limiting and validate user requests.
Failures in communication with S3.	Implement robust exception handling and retries during pre-signed URL generation.
Incorrect configuration of S3 permissions.	Perform audits and tests in controlled environments before deploying to production.
Hardcoded configuration values in environment variables.	Use a Secret Manager to handle necessary credentials and migrate to the AWS SDK.

Conclusion

Implementing a system based on pre-signed URLs for file uploads to S3 significantly improves scalability and reduces backend load. This document outlines the essential requirements for its development, emphasizing critical aspects such as performance, security, and flexibility, while establishing an actionable plan for MVP implementation and validation.