Template for Airtribe’s Capstone Project (Engineering Launchpad)

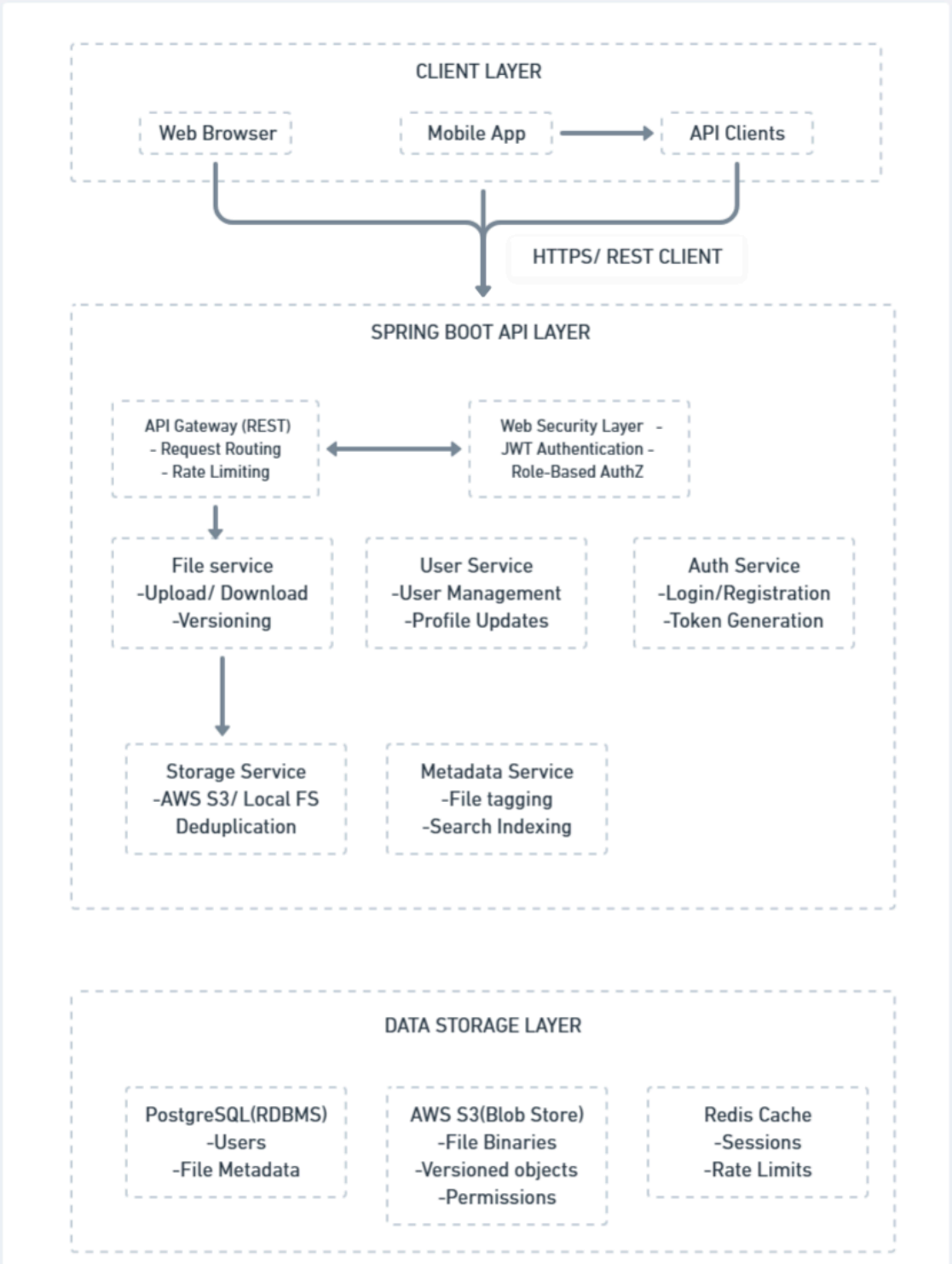
Simple Cloud Storage (SCS) - Complete Project Documentation

1. Project: Simple Cloud Storage (SCS)

A simplified Amazon S3 clone that provides secure cloud file storage with user authentication, file management, version control, and access permissions.

2. System Design

Architecture Diagram



Made with Whimsical

Key Components

1. **Spring Boot Application Layer**
    - REST Controllers: Handle HTTP requests/responses
    - Service Layer: Business logic implementation
    - Repository Layer: Database interactions (Spring Data JPA)
2. **Security Layer**
    - Spring Security with JWT authentication
    - Role-based access control
3. **Storage Service**
    - AWS S3 integration for production
    - Local filesystem storage for development
4. **Database**
    - PostgreSQL relational database
    - Hibernate ORM for object-relational mapping

3. Database Design

Schema Overview

Diagram

Table Structures

users

Column	Type	Description
id	SERIAL	Primary key
username	VARCHAR(50)	Unique username

email	VARCHAR(100)	User email
password_hash	VARCHAR(255)	BCrypt hash
created_at	TIMESTAMP	Account creation time
is_active	BOOLEAN	Account status (active/banned)
last_login	TIMESTAMP	Last login timestamp

files

Column	Type	Description
id	SERIAL	Primary key
user_id	INTEGER	Owner
folder_id	INTEGER	Parent folder
name	VARCHAR(255)	Filename
s3_key	VARCHAR(512)	S3 object key
size_bytes	BIGINT	File size
file_type	VARCHAR(100)	File type(eg img/png)
is_public	BOOLEAN	Public access flag
created_at	TIMESTAMP	Upload time
updated_at	TIMESTAMP	Last modified time

file_versions

Column	Type	Description
id	SERIAL	Primary key
file_id	INTEGER	Parent file
version	INTEGER	Version number
s3_key	VARCHAR(512)	S3 object key
created_at	TIMESTAMP	version timestamp

access_permissions

Column	Type	Description
id	SERIAL	Primary key
user_id	INTEGER	Recipient user
file_id	INTEGER	Shared file
folder_id	INTEGER	Shared folder
permission	VARCHAR(20)	READ/WRITE/ADMIN
granted_at	TIMESTAMP	Permission grant time

folders

Column	Type	Description
id	SERIAL	Primary key
user_id	INTEGER	Owner of the folder
name	VARCHAR(255)	Folder name
parent_id	INTEGER	Parent folder (for nesting)
created_at	TIMESTAMP	Creation time

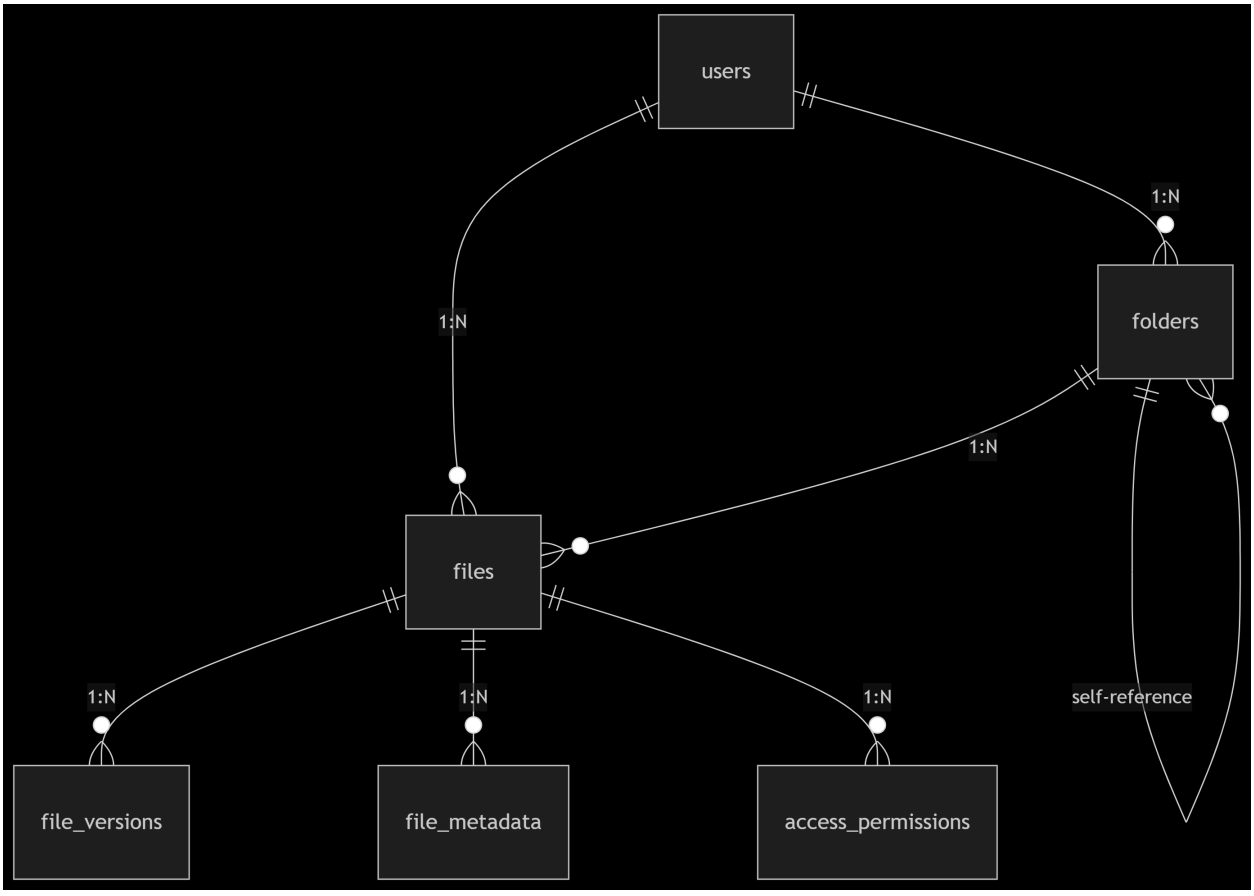
file_metadata

Column	Type	Description
id	SERIAL	Primary key
file_id	INTEGER	Associated file
key	VARCHAR(100)	Metadata key (e.g., author)
value	TEXT	Metadata value (e.g., Alice)

Key Relationships

1. **User → Files**
    - One user owns many files (1:N).
2. **User → Folders**
    - One user owns many folders (1:N).

3. **Folder → Files**
    - One folder contains many files (1:N).
4. **Folder → Subfolders**
    - Self-referential (nested folders).
5. **File → Versions**
    - One file has many versions (1:N).
6. **File → Metadata**
    - One file has multiple metadata entries (1:N).



Indexes for Performance

Table	Indexed Columns	Purpose
uses	email,username	Fast login lookups
files	user_id, folder_id	Quick user file listings
file_versions	file_id	Efficient version history
file_meatadata	file_id, key	Optimized metadata searches
access_permissions	file_id, user_id	Fast permission checks

---

4. Key APIs

Authentication APIs

Register User

Endpoint: /api/auth/register
Method: POST
Description: Register a new user
Request: { "username": "user1", "email": "user@test.com", "password": "pass123" }
Response: { "token": "jwt.token", "userId": 1 }

Login User

Endpoint: /api/auth/login
Method: POST
Description: Authenticate user and return JWT token
Request: { "email": "user@test.com", "password": "pass123" }
Response: { "token": "jwt.token", "userId": 1 }

File Management APIs

Upload File

Endpoint: /api/files/upload
Method: POST
Description: Upload a file to user's storage
Headers: Authorization: Bearer <token>
Request: Multipart form-data (file + metadata)
Response: { "fileId": 123, "name": "doc.pdf" }

Download File

Endpoint: /api/files/download/{fileId}
Method: GET
Description: Download a file
Headers: Authorization: Bearer <token>
Response: File binary stream

List Files

Endpoint: /api/files
Method: GET
Description: List all files for authenticated user
Headers: { "Authorization": "Bearer jwt.token.here" }
Response: [{ "id": 123, "name": "document.pdf", "size": 1024, "isPublic": false }]

Search Files

Endpoint: /api/files/search
Method: GET
Description: Search files by name or metadata
Headers: { "Authorization": "Bearer jwt.token.here" }
Query Params: ?query=report&tag=finance
Response: [{ "id": 123, "name": "annual_report.pdf", "tags": ["finance", "report"] }]

List File Versions

Endpoint: /api/files/{fileId}/versions
Method: GET
Description: List all versions of a file
Response: [{ "version": 1, "createdAt": "2023-01-01" }, ...]

Restore File Version

Endpoint: /api/files/:fileId/versions/:version/restore
Method: POST
Description: Restore a specific version of a file
Headers: { "Authorization": "Bearer jwt.token.here" }
Response: { "message": "Version restored successfully" }

5. Overall Approach

Technology Choices

1. **Spring Boot**:
    - Robust framework for enterprise applications
    - Built-in security features

- Easy integration with databases and storage systems

2. **PostgreSQL**:
   - ACID-compliant relational database
   - Excellent for structured data like user accounts and file metadata
   - JSON support for flexible metadata storage
3. **AWS S3**:
   - Industry-standard cloud storage
   - Highly durable and scalable
   - Versioning support out-of-the-box
4. **JWT Authentication**:
   - Stateless and scalable
   - Secure token-based authentication
   - Works well with mobile and web clients

Key Design Decisions

1. **Layered Architecture**:
   - Clear separation between controllers, services, and repositories
   - Better testability and maintainability
2. **Storage Abstraction**:
   - Interface-based design allows switching between AWS S3 and local storage
   - Easy mocking for testing
3. **Permission System**:
   - Three-tier access control (private/public/shared)
   - Granular permissions through ACLs
4. **Versioning Implementation**:

- Each version stored as separate S3 object
- Metadata tracking in database
- Efficient rollback capability

Implementation Phases

1. **Core Functionality** :
   - User authentication
   - Basic file upload/download
   - Database schema implementation
2. **Advanced Features**:
   - File versioning
   - Folder organization
   - Permission management
3. **Optimizations**:
   - File deduplication
   - Caching with Redis
   - Usage analytics

Deployment Strategy

- **Containerization**: Docker for consistent environments
- **CI/CD**: GitHub Actions for automated testing and deployment
- **Monitoring**: Spring Boot Actuator for health checks