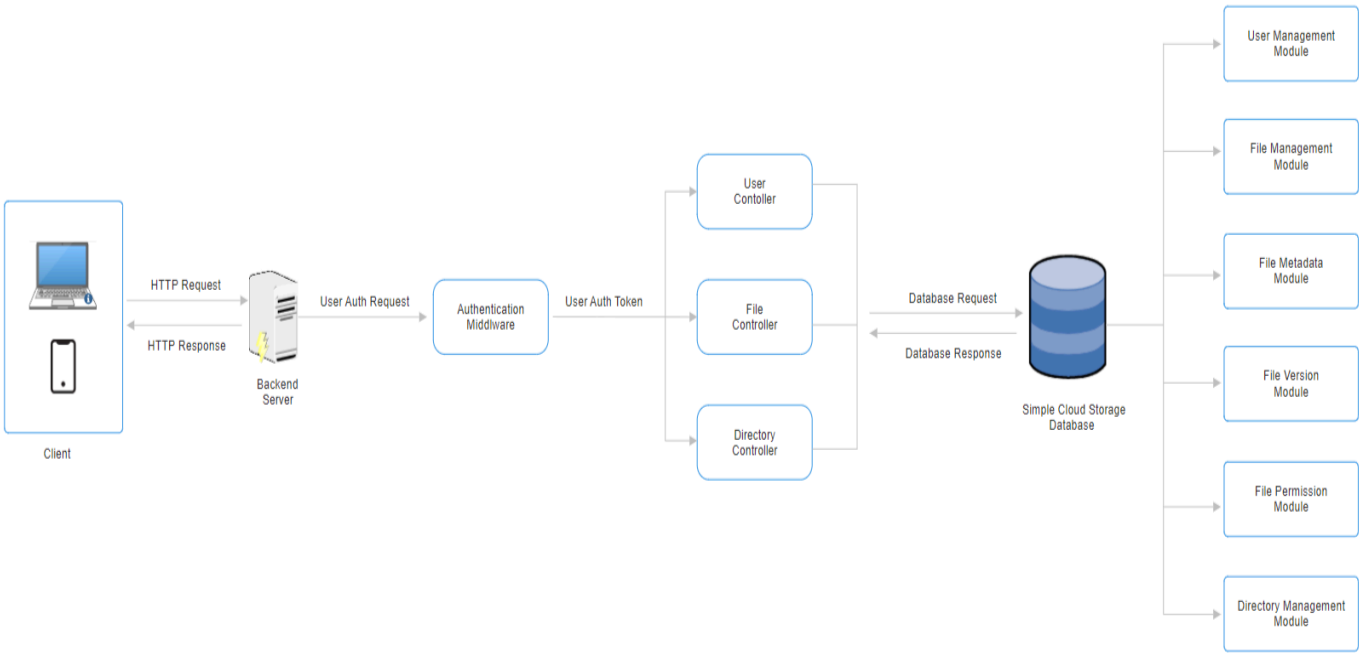
Amazon S3: Simple Cloud Storage (SCS)

Submission By: Siddharth Soni
(Cohort 5 Backend)

1. System Design:



Description of Architecture:

The architecture of the Simple Cloud Storage (SCS) application consists of three main components: Frontend, Backend, and Database.

- **Frontend:** Represents the user interface accessed via web browsers. It allows users to interact with the application, performing actions such as authentication, file management, and directory navigation. The frontend sends HTTP requests to the backend API and updates the UI based on the responses received.
- **Backend:** As the intermediary between the front end and the database. Built using Node.js and Express.js, it provides a RESTful API for communication with the front end. The backend handles incoming requests, processes them, interacts with the MySQL database, and sends back responses to the frontend.
- **Database:** Utilizes MySQL Workbench as the relational database management system. It stores all persistent data the SCS application requires, including user information, files, directories, file versions, file metadata, and other relevant data.

Key Components:

- Authentication Middleware
- User Controller
- File Management Module
- Directory Management Module
- File Version Controller
- File Metadata Controller

2. Database Design:

The database schema includes the following tables with their respective structures:

→ **User Table**

- user_id (Primary Key)
- username
- email
- password_hash
- registration_date

→ **File Table**

- file_id (Primary Key)
- user_id (Foreign Key referencing User Table)
- directory_id (Foreign Key referencing Directory Table)
- system_generated_file_name
- file_name
- file_path
- file_size
- upload_date
- file_type
- download_count

→ **Directory Table**

- directory_id (Primary Key)
- user_id (Foreign Key referencing User Table)
- parent_directory_id (Foreign Key referencing itself to represent nested directories)
- directory_name
- directory_path

→ **FilePermissions Table**

- permission_id (Primary Key)
- file_id (Foreign Key referencing File Table)
- user_id (Foreign Key referencing User Table)
- Permission_type

→ **FileVersion Table**

- version_id (Primary Key)
- file_id (Foreign Key referencing File Table)
- version_number
- file_path
- upload_date

→ **FileMetadata Table**

- metadata_id (Primary Key)
- file_id (Foreign Key referencing File Table)
- metadata_key
- metadata_value

3. Key APIs:

→ **User Authentication APIs:**

- **Endpoint:** /api/auth/register
  - ➔ **Method:** POST
  - ➔ **Description:** Registers a new user.
  - ➔ **Request:** JSON object containing username, email, and password.
  - ➔ **Response:** Success message or error details.

- **Endpoint:** /api/auth/login
  - ➔ **Method:** POST
  - ➔ **Description:** Logs in an existing user.
  - ➔ **Request:** JSON object containing email and password.
  - ➔ **Response:** JWT token on successful authentication or error details.

- **Endpoint:** /api/auth/logout
  - ➔ **Method:** POST
  - ➔ **Description:** Logs out the currently authenticated user.
  - ➔ **Request:** JWT token (in the Authorization header).
  - ➔ **Response:** Success message or error details.

→ **File Management APIs:**

- **Endpoint:** /api/files/upload
  - ➔ **Method:** POST
  - ➔ **Description:** Uploads a file to the user's storage space.
  - ➔ **Request:** FormData containing the file to be uploaded.
  - ➔ **Response:** Success message or error details.

- **Endpoint:** /api/files/download/:fileId
  - ➔ **Method:** GET
  - ➔ **Description:** Download a file by its ID.
  - ➔ **Request:** File ID in the URL parameter.
  - ➔ **Response:** File download or error message.

- **Endpoint:** /api/files/list
  - ➔ **Method:** GET
  - ➔ **Description:** Retrieves a list of the user's uploaded files.
  - ➔ **Request:** JWT token (in the Authorization header).
  - ➔ **Response:** JSON array of file objects containing file details.

- **Endpoint:** /api/files/search
  - ➔ **Method:** GET
  - ➔ **Description:** Searches for files based on filenames or metadata.
  - ➔ **Request:** Query parameter for search keyword.
  - ➔ **Response:** JSON array of matching file objects.

- **Endpoint:** /api/files/:fileId/permissions
  - ➔ **Method:** POST
  - ➔ **Description:** Sets permissions for a specific file.
  - ➔ **Request:** JSON object containing user IDs and permission types.
  - ➔ **Response:** Success message or error details.

- **Endpoint:** /api/files/:fileId/versions
  - ➔ **Method:** GET
  - ➔ **Description:** Retrieves all versions of a file identified by fileId.
  - ➔ **Request:** File ID in the URL parameter.
  - ➔ **Response:** JSON array of file version objects.

- **Endpoint:** /api/files/:fileId/versions/:versionId
  - ➔ **Method:** GET
  - ➔ **Description:** Retrieves a specific version of a file identified by both fileId and versionId.
  - ➔ **Request:** File ID and Version ID in the URL parameters.
  - ➔ **Response:** File version object.

- **Endpoint:** /api/files/:fileId/versions/:versionId/restore
  - ➔ **Method:** PUT
  - ➔ **Description:** Restores a previous version of a file identified by both fileId and versionId.
  - ➔ **Request:** File ID and Version ID in the URL parameters.
  - ➔ **Response:** Success message or error details.

- **Endpoint:** /api/files/:fileId/metadata
  - ➔ **Method:** POST
  - ➔ **Description:** Adds metadata to a file.
  - ➔ **Request:** JSON object containing metadata key-value pairs.
  - ➔ **Response:** Success message or error details.

- **Endpoint:** /api/files/usage-analytics
    - ➔ **Method:** GET
    - ➔ **Description:** Retrieves usage analytics related to storage usage, types of files stored, and file download frequency for the authenticated user.
    - ➔ **Request:** JWT token (in the Authorization header)
    - ➔ **Response:** Success message or error details.

- **Endpoint:** /api/files/delete
    - ➔ **Method:** DELETE
    - ➔ **Description:** Deletes a file or directory.
    - ➔ **Request:** JSON object containing type (file or directory) and ids (file IDs or directory IDs) in array.
    - ➔ **Response:** Success message or error details.

→ **<u>Directory Management APIs</u>:**

- **Endpoint:** /api/directories/create
    - ➔ **Method:** POST
    - ➔ **Description:** Creates a new directory.
    - ➔ **Request:** JSON object containing directory name parent directory ID, and directory path.
    - ➔ **Response:** Success message or error details.

- **Endpoint:** /api/directories/list
    - ➔ **Method:** GET
    - ➔ **Description:** Retrieves a list of user's directories.
    - ➔ **Request:** JWT token (in the Authorization header).
    - ➔ **Response:** JSON array of directory objects containing directory details.

- **Endpoint:** /api/directories/:directoryId/files
    - ➔ **Method:** GET
    - ➔ **Description:** Retrieves files within a directory.
    - ➔ **Request:** Directory ID in the URL parameter.
    - ➔ **Response:** JSON array of file objects within the directory.

- **Endpoint:** /api/directories/:directoryId/move
    - ➔ **Method:** PUT
    - ➔ **Description:** Moves a file or directory to another directory.
    - ➔ **Request:** JSON object containing file or directory ID and destination directory ID.
    - ➔ **Response:** Success message or error details.

4. Overall Approach:

For the Simple Cloud Storage (SCS) project, the chosen approach involves using **Node.js** as the backend framework and **MySQL** Workbench as the database. Node.js provides a non-blocking I/O model, making it suitable for handling multiple concurrent connections efficiently. MySQL Workbench offers a reliable and scalable relational database solution, ensuring data integrity and consistency.

Key decisions and technologies used include:

- **Node.js:** Chosen for its non-blocking I/O, a vast ecosystem of libraries, and ease of use in building scalable server-side applications.
- **Express.js:** Used as the web application framework for Node.js, providing essential features for building RESTful APIs and handling HTTP requests.
- **MySQL Workbench:** Selected as the relational database management system due to its stability, performance, and robust features for storing and managing data.
- **Sequelize ORM:** Employed for interacting with the MySQL database, providing an abstraction layer over raw SQL queries and simplifying database operations.
- **JWT (JSON Web Tokens):** Utilized for secure user authentication, providing stateless authentication without the need for server-side session management.

Design patterns and principles such as separation of concerns, modularity, and scalability have been considered to ensure the robustness and maintainability of the SCS application. The chosen technologies and approaches aim to fulfil the project requirements effectively while providing a secure and efficient cloud storage solution.