Bank Statement Aggregator

Detailed Problem Statement

Objective:

The goal of this project is to develop a system that aggregates bank statements for users from multiple branches of various banks. The system will handle the generation of bank statements, upload these statements to an AWS S3 bucket, and later allow users to download and convert these statements into Java objects for storage in a database. The end-users should be able to log in, view their bank statement details, and store them in the database.

Key Functionalities:

1. **User Authentication and Authorization:**

   ```
   Users can register and log in to the system.
   User credentials (username, email, password) are verified.
   Secure authentication using Spring Security.
   ```

2. **Bank Statement Generation:**

   ```
   Generate dummy bank statements for users from various
   branches.
   Use a predefined format for these statements (e.g., JSON,
   CSV).
   Save the generated statements with a proper naming
   convention.
   ```

3. **Uploading to AWS S3:**

   ```
   Connect to AWS S3 and create a bucket for storing bank
   statements.
   Upload the generated bank statement files to the S3
   bucket.
   Create folders in the S3 bucket based on user and company
   identifiers.
   ```

4. **Downloading and Processing Bank Statements:**

```
Allow users to download their bank statements from the S3
bucket.
Convert the downloaded statements into Java objects.
Store these objects in the database.
```

5. **Database Management:**

```
Design and implement a relational database schema.
Maintain tables for Companies, Users, Branches, and Bank
Statements.
Ensure proper relationships between tables.
```

Detailed Steps

Step 1: Database Schema Design
Tables to be created:

Companies Table

```
company_id (Primary Key): Unique identifier for each company.
company_name: Name of the company.
```

Users Table

```
user_id (Primary Key): Unique identifier for each user.
username: Name of the user.
email: User's email address.
password: Encrypted password for the user.
company_id (Foreign Key): References the company the user belongs to.
```

Branches Table

```
branch_id (Primary Key): Unique identifier for each branch.
branch_name: Name of the branch.
company_id (Foreign Key): References the company the branch belongs
to.
```

BankStatements Table

```
statement_id (Primary Key): Unique identifier for each bank
statement.
user_id (Foreign Key): References the user associated with the
statement.
company_id (Foreign Key): References the company associated with the
statement.
```

```
branch_id (Foreign Key): References the branch associated with the
statement.
statement_date: Date of the bank statement.
statement_data (Blob or Text): Contains the actual bank statement
data, such as transactions.
```

Transactions Table

```
 For storing the bank statements into java objects
```

Database Relationships:

```
A company can have multiple users.
A company can have multiple branches.
A user can have multiple bank statements.
A branch is associated with a single company but can have multiple
bank statements.
```

Step 2: Generating Dummy Bank Statements

Define a standard format for dummy bank statements (eg CSV) with columns like

```
Transaction_id
Date
Amount
Description
Company_name
```

Generate bank statements based on this format and save them locally.

Step 3: Implementing the Application

User Verification and Fetching Data:

```
Verify user email, username, and password.
Fetch user_id based on login credentials.
Using user_id, retrieve company_id and associated branch_ids.
```

Bank Statements Generation and Upload:

```
Generate bank statements file using user and company data.
Save the file with a proper naming convention
(user_id/company_id/...).
Connect to AWS S3.
Create a folder in the S3 bucket and save the file.
```

Retrieving and Processing Bank Statements:

Download the bank statements from the S3 bucket.
Convert the statements into a list of Java objects.
Store the objects in the database.
Ensure that the statements are merged appropriately.

Example Scenario

1.User Registration and Login:

User registers with their email, username, password, and associated company.
User logs in using their credentials.
System verifies the credentials and fetches user details from the database.

2.Generating and Uploading Bank Statements:

System generates bank statements for the user's company branches.
Generated statements are saved with a naming convention including user_id and company_id.
System uploads the statements to an AWS S3 bucket, creating necessary folders.

3.Downloading and Storing Bank Statements:

User requests to view their bank statements.
System downloads the statements from the S3 bucket.
Statements are converted into Java objects and stored in the database.
User can view their statements from the database.

Implementation Details
Folder Structure

```
bank-statement-aggregator/
├── src/
│ ├── main/
│ │ ├── java/
│ │ │ ├── com/
│ │ │ │ ├── example/
│ │ │ │ │ ├── aggregator/
│ │ │ │ │ │ ├── controllers/
│ │ │ │ │ │ ├── models/
│ │ │ │ │ │ ├── repositories/
│ │ │ │ │ │ ├── services/
│ │ │ │ │ │ ├── utils/
│ │ ├── resources/
│ │ ├── application.properties
├── pom.xml
└── README.md
```

Design Details

1. Controllers:
    - UserController: Handle user login and data retrieval.
    - StatementController: Manage bank statement generation, upload, and download.
2. Models:
    - User: Represent the user entity.
    - Company: Represent the company entity.
    - Branch: Represent the branch entity.
    - BankStatement: Represent the bank statement entity.
3. Repositories:
    - UserRepository: Interface for user data operations.
    - CompanyRepository: Interface for company data operations.
    - BranchRepository: Interface for branch data operations.
    - BankStatementRepository: Interface for bank statement data operations.
4. Services:
    - UserService: Business logic for user-related operations.
    - StatementService: Business logic for bank statement operations.
    - AWSService: Handle interactions with AWS S3.

5. Utils:
    - Utility classes for common operations (e.g., file handling, AWS integration).