Meet5 Junior Backend Dev Task

Candidate: Hongye An

Project Github Repository: https://github.com/huyhi/meet5-task

Project Setup

1. This project relies on **MySQL** relational database to run. Before running the project, you need to use the `src/main/resources/init_table.sql` SQL script to initialize the data schema.
2. Navigate to `src/main/resources/application.properties` and set database connection information. The following three configuration items need to be configured.

   ```
   spring.datasource.username=
   spring.datasource.password=
   spring.datasource.url=
   ```

3. Run this project. If you use **IntelliJ IDEA**, you could use **IntelliJ IDEA** to open the project and run the `main` function in `src/main/java/com/hongye/App.java` that is the endpoint of the spring boot server.

   Or you could just use maven and run following command to start the spring boot server.

   ```
   mvn spring-boot:run
   ```

Project Overview

Directory tree explanation:

```
src
└ main
    ├ java
    │   └ com.hongye
    │       ├ App.java  Entry point of Spring application where the main method resides
    │       ├ configure
    │       │   ├ ExceptionResponseAdvice.java  Formatting exceptions for API response
    │       │   └ RestResponseAdvice.java  Formatting REST API responses
    │       ├ constant
    │       │   ├ UserActionTypeEnum.java  Enumerate user actions, including visit and like actions
    │       │   └ UserStatusEnum.java      Enumerate user status, including valid and fraud
    │       ├ controller
    │       │   └ UserActionController.java  API definition endpoint
    │       ├ dao
    │       │   ├ UserActionDAO.java  DB access methods related user action(visit or like)
    │       │   └ UserDAO.java        DB access methods related user
    │       ├ dto
    │       │   └ UserActionDTO.java  API interface data definition
    │       ├ model                   Entities model floder
    │       │   ├ ResponseData.java
    │       │   ├ User.java
```

```
    │        │    └─ UserAction.java
    │        └─ service
    │             └─ UserActionService.java  Business logic and operations related to user actions
    └─ resources
        ├─ application.properties  Project configuration
        ├─ init_table.sql          Initialization SQL script for DB table schema.
        └─ mapper                  Mybatis SQL implemention floder
             ├─ UserActionMapper.xml
             └─ UserMapper.xml
```

Explanation and Interpretation of Key Content

init_table.sql explanation:

This SQL script file is used to initialize database tables. Running this script will create two tables: `user` and `user_action_record`.

The `user` table stores various basic user information, while the `user_action_record` table keeps track of interactions between users, including visit and like. The `type` field is used to differentiate whether a record is a visit or a like.

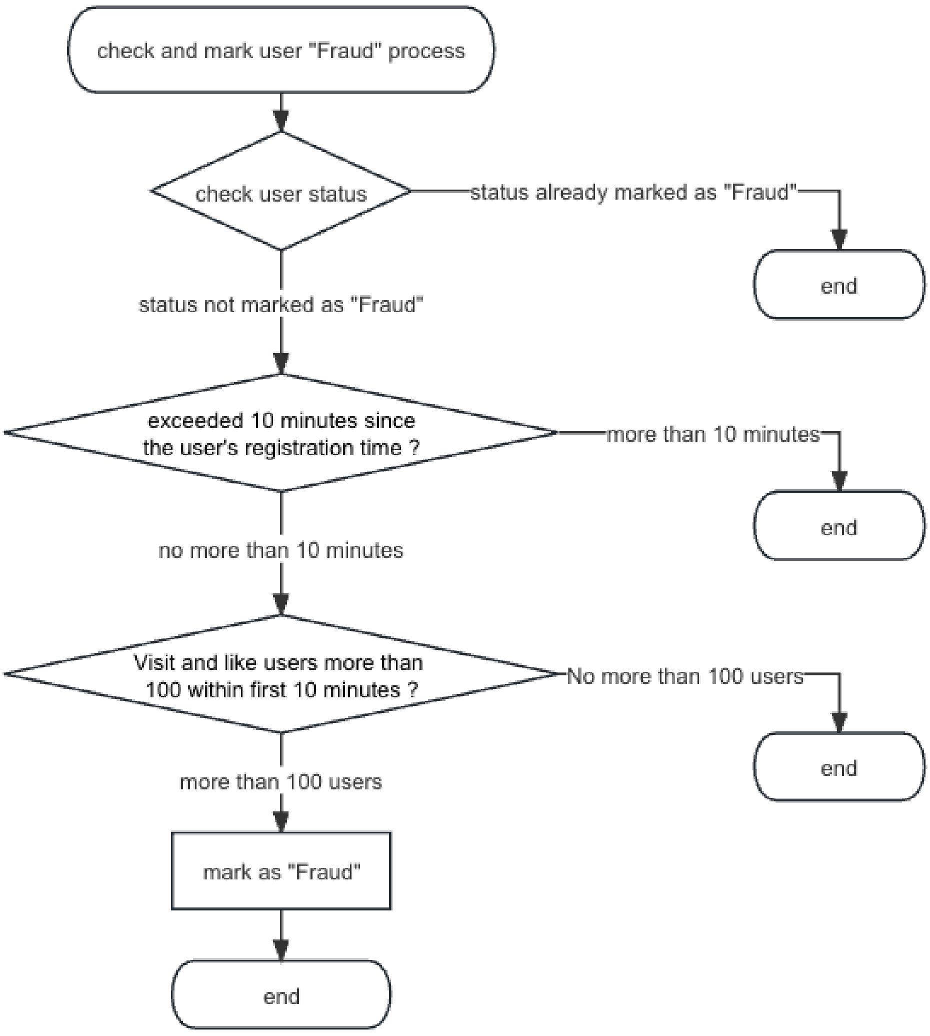
Requirement 2 (retrieves all profile visitors of a user) explanation:

The implementation of requirement 2 (retrieves all profile visitors of a user) was located in `com.hongye.dao.UserActionDAO#allVisitors` And the corresponding SQL implementation was in the file: `src/main/resources/mapper/UserActionMapper.xml`

This SQL perform a table join and sorts based on the time of the `UserActionRecord` entry. As a combined index of `to_id` and `recorded_at` fields is established when creating `user_action_record` table, the query and sorting are quite efficient relatively.

Fraud detection explanation:

When the API `/user/visit` or `/user/like` was called, "Fraud" detection will be executed. The process of Fraud detection is illustrated in the diagram below.


check and mark user "Fraud" process
check user status
status already marked as "Fraud"
end
status not marked as "Fraud"
exceeded 10 minutes since the user's registration time ?
more than 10 minutes
end
no more than 10 minutes
Visit and like users more than 100 within first 10 minutes ?
No more than 100 users
end
more than 100 users
mark as "Fraud"
end

Requirement 8 (bulk data insertion) explanation:

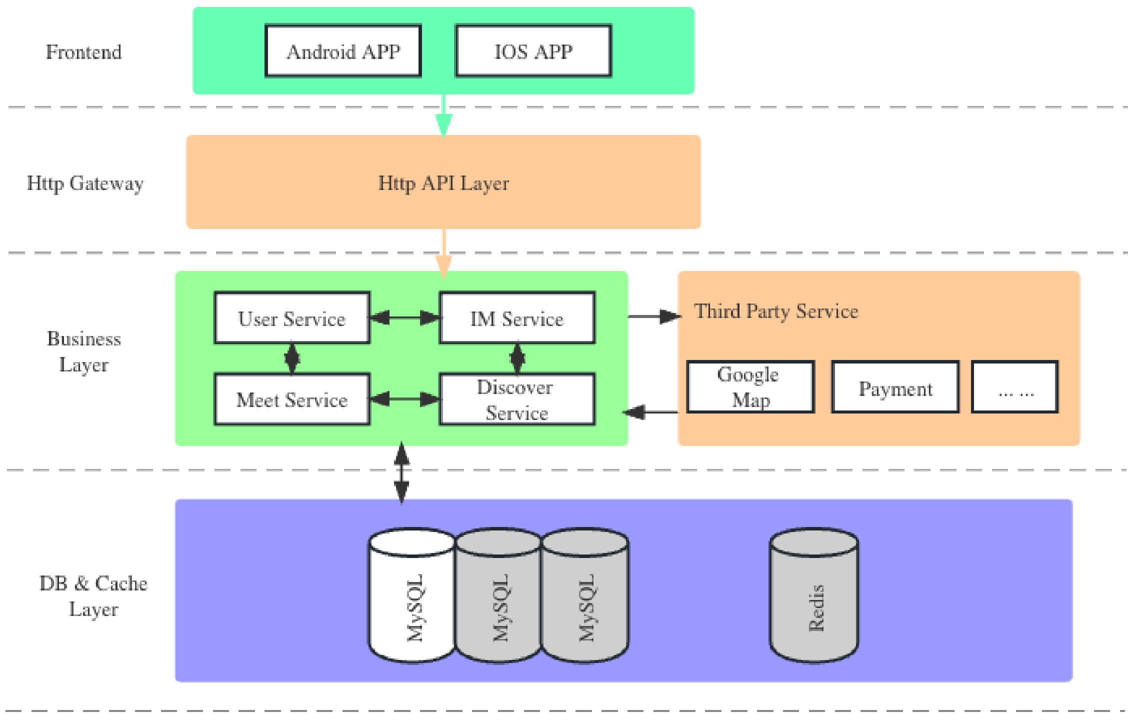
The implementation of requirement 8 (bulk data insertion) is `com.hongye.service.UserActionService#bulkInsert` This function accepts a list of `UserAction` object and is designed to insert these data into the database and returning the number of records.

The specific approach involves splitting this list into segments of a certain size (e.g. 500 rows per segment) and using batch insert SQL to insert each segment of data into the database. This helps to avoid excessive database load caused by inserting a large number of records in a single operation.

(P.S. This project use MyBatis framework to perform SQL operations on the database. MyBatis is **neither** part of JPA **nor** Hibernate)

Microservices architecture

Architecture Diagram



The above architecture diagram is based on my personal understanding of the current meet5 APP.

For the business layer, I have divided internal services into four parts based on service boundaries — User service, IM (instant messaging) service, Discover service and Meet service. The various services can communicate with each other as needed using microservices communication protocols such as `Thrift` or `gRPC` etc.

- User service: Provide user-related APIs such as registration, login, and user information retrieval etc.
- IM service: Provide a service for sending instant messages between users.
- Meet service: Offer APIs for creating, viewing, participating in, and bookmarking meets, as well as any other meet-related actions.
- Discover service: Offer APIs for discovering other users, such as viewing user profiles, liking users, and other related functionalities.