

Fig. 1: Movie recommender architecture

1. Testability:

Each layer in the application’s architecture has specific tasks as shown in Fig. 1. Therefore, it can be further specialized into isolated modules, which we can create specific unit tests. By this way, it would help to increase the testability of the system.

1. Persistence:

To implement the persistence in the application, we can create Data Access Objects (DAOs) in the Persistence Layer and map these objects to relational databases using JDBC API. Alternatively, we can use Hibernate, which has almost the same capability. I personally prefer JDBC because I have more experience with it.

1. Design Patterns of Microservices:

We can use following patterns in this application:

1. Decomposition Pattern:

This application can be decomposed into microservices based on business capability. For example, the movie and genre data are required to build suggestion data for user. So, we can have two microservices: GenreService and MovieService. The data from these two services are used to implement algorithms in another microservice, the SuggestionService.

1. Aggregator Pattern:

This is for Aggregator services, such as SuggestionService, to communicate with multiple microservices to implement a specific business.

1. Database Pattern:

Each microservice in large-scale systems should have its own database, but this requires more resources and between-database synchronization. From my point of view, we can use Shared Database per Service as an alternative if our database traffic is not too heavy.

1. API Gateway Pattern:

This pattern provides the single requesting-point for client.