Eötvös Loránd University

Faculty of Informatics

Dep. of Programming Languages and Compilers

Classic Movie Collection Website based on Microservices and Cloud Native

Supervisor:
Dr. Zsók Viktória
Assistant Lecturer

Author:
Guohao Lin
Computer Science BSc

Budapest, 2022

EÖTVÖS LORÁND UNIVERSITY
FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:
Student's Name: Lin Guohao
Student's Neptun code: IW3XV9

Course Data:
Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Dr. Zsók Viktória*
Supervisor's Home Institution: *ELTE IK PNYF ELTE Fac of Informatics, PLC*
Address of Supervisor's Home Institution: *H-1117 Budapest, Pázmány Péter sétány 1/C*
Supervisor's Position and Degree: *Assistant Professor, Ph.D.*

Thesis Title: Classic movie collection website based on microservices and cloud native

Topic of the Thesis:
(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

Today's era is the era of information, and the carrier of cultural dissemination has also undergone great changes. From the dictation in the past, to the words, and then to the movies of today. Each time is a huge breakthrough in the amount of information. But with the reduction of production costs and the maturity of technology, a large number of new movies are released every week, some may become classics, but most of them will be forgotten in a few weeks.

With the faster and faster pace of people's lives, people are now lazy to filter information. But film is a rare art form. So I want to build a website for collecting classic movies. People are free to comment under each movie. You can also recognize other people's reviews and opinions on the movie. The website also gathers ratings of this movie from major media platforms, the movie's OST and classic stills. Make classic movies more memorable.

As we all know, the traditional single system has many problems. Therefore, this website will be constructed with the idea of microservices. Use springboot to build each microservice, and then each microservice communicates based on grpc. We modularized the platform and subdivided multiple subsystems. The database uses MySQL. Use junit and in-memory database for testing. Component testing will also be used. Finally, use docker to containerize them to reduce the dependency and coupling of the operating system.

This thesis will describe a complete web-based cloud native product and it provides detailed related descriptions.

Budapest, 2021. 12. 13.

Contents

Chapter 1

Introduction

As a kind of moving image art, films have gradually developed from niche art to mass media since the beginning of the 20th century, and it has had a huge impact on the fields of art, technology and politics of mankind. As an ever-evolving film system, films have always recorded the issues generated by human thinking from one era to another. With the development of the times, the expression of the film has also become more and more mature from the black and white silence at the beginning. Especially, the classic movie lines, stills and music can often leave a deep and even unforgettable impression on people.

In this project, we aim to build a website that collects, curates, and discusses classic movies. Unlike the popular IMDb[1], which mainly provides movie commercial information and promotional pages, this website is mainly about "recording and sharing moments" for classic movies. Specifically, admins can upload touching or thought-provoking stills and lines of classic movies, as well as the irreproducible original soundtrack of the movie. Users can bookmark and comment on their favorite movie clips, and find other classic movie enthusiasts. We also implement a search bar that users can quickly retrieve movies by keywords. In addition, admins can manage user comments and modify all visual elements on this website.

At the technical level, this project adapts the idea of microservices, which is the separate development of sub-projects with different functions and finally integrated into a complete project. To achieve communication between the back-end and front-end, the most popular approach is RESTful (Representational state transfer) API, however, it suffers problems such as low efficiency and vulnerable security. To mit-

[1]https://www.imdb.com/

igate those problems, in this project, we make a breakthrough practice that uses gRPC (Google Remote Procedure) to conduct communication between them. As an emerging data transmission protocol, RPC (Remote Procedure) has the characteristics of fast serialization, small size, efficient transmission, and multiplexing[1], which is widely used in communication between back-end components of microservices.

Moreover, almost no one uses RPC to achieve communication between the back-end and the front-end of a website. With the release of gRPC-web, it is possible to replace RESTful API by RPC-based methods. Therefore, by building this classic movie collection website, we make a bold attempt that abandoning RESTful API but embracing the RPC-based communication between the back-end and front-end. We also verify the feasibility and practicality of this technology transformation.

Specifically, the front-end uses Vue framework and Vue-CLI for project development, the back-end uses SpringBoot, and the communication is implemented by gRPC. This means that our projects are built entirely using frameworks and are very easy to test and maintain. Moreover, Nginx is utilized as a reverse proxy to convert gRPC requests, MySQL is used to establish the database, and Alibaba Cloud OSS (Object Storage Service) is used for resource storage services. Overall, the whole project is also containerized by Docker to achieve cloud-native and can be deployed by Docker Compose with simply one-click. Therefore, this project is also extremely extensible and can be continuously developed. We also use multiple configuration files to meet the deployment of various complex situations such as deploying in different front-end and back-end servers, which means our project can meet almost all deployment requirements.

Chapter 2

User Documentation

This chapter is the user manual, where you can find the introduction of the main tools used in the front-end and back-end, the content and distribution of the web UI, system requirements, program installation and operation methods.

2.1 Interface Layout

This section contains detailed introduction and layout about home page, sign up page, login page, profile page, movie detail page and navigation bar.

The front-end is built with Vue and has been split into multiple relatively independent components.

Home page Displays all movies in the database as a card with a relative height of 400, including the movie name, director name, actors name and movie rating, as well as the movie cover, and also provides a link to the movie details page. [2]

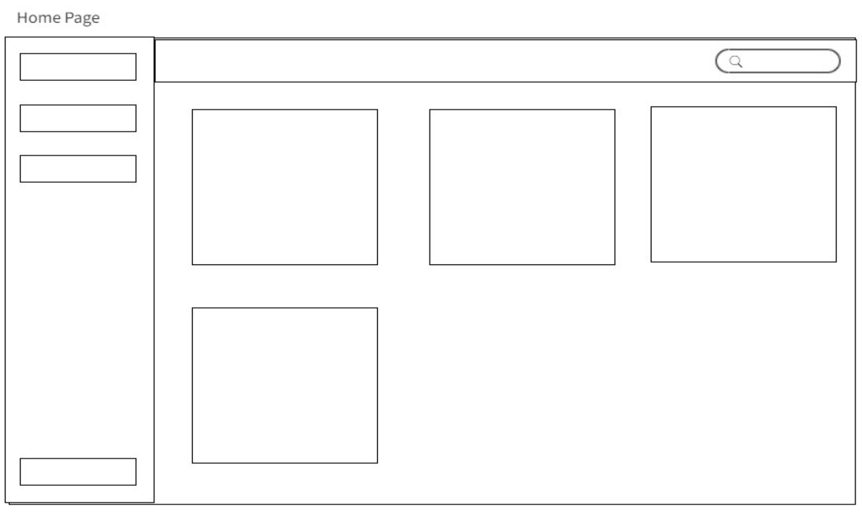


Figure 2.1: Home Page Layout

Sign up page Users can register on this page, and they need to provide a complete email address as username (with email address format detection, and the registered email address of all users must be different), full name (no need to be different from other users), password (provides a repeat password field to verify the password entered by the user). Click the sign up button to register. If there are invalid fields will be prompted until all the information inputted by the user is completely correct. After the sign up is successful, there will be a link to go to the login page. This page will also give a link when the user clicks, so that the registered user can jump to the login page.

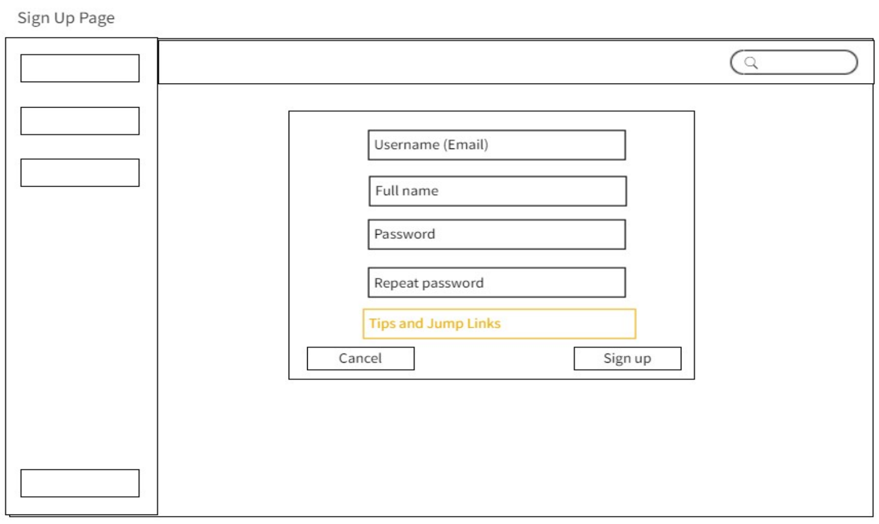


Figure 2.2: Sign up Page Layout

Login page The Login page will require the user to input the username (email address) and password. All input areas must be provided (otherwise give error messages). After clicking on the login, there will be a successful or unsuccessful message. When users jump to this page, the page will display a link to the sign up page to facilitate unregistered users to register. As shown in Figure 2.3:

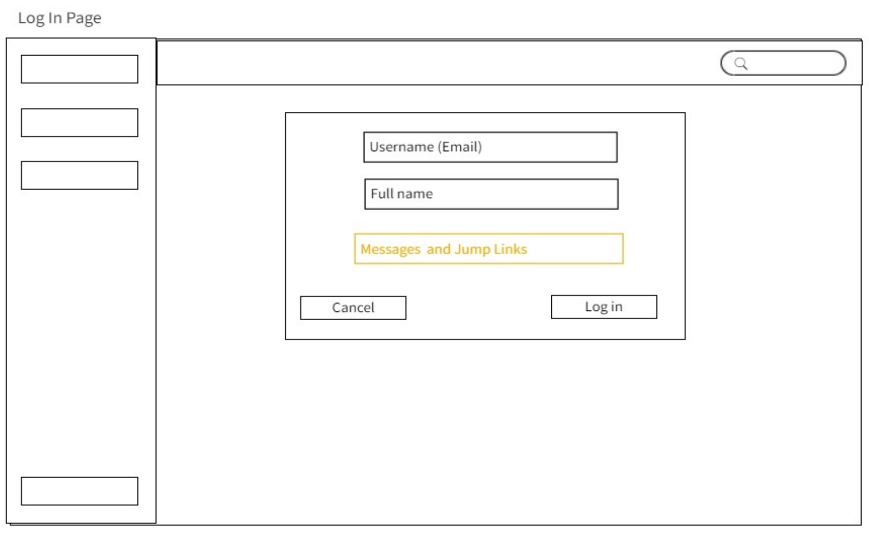


Figure 2.3: Log in Page Layout

Profile page The profile page will display the user's favorite movies and written comments, the user can delete comments, and the page also provides links to the favorite movie's details page.

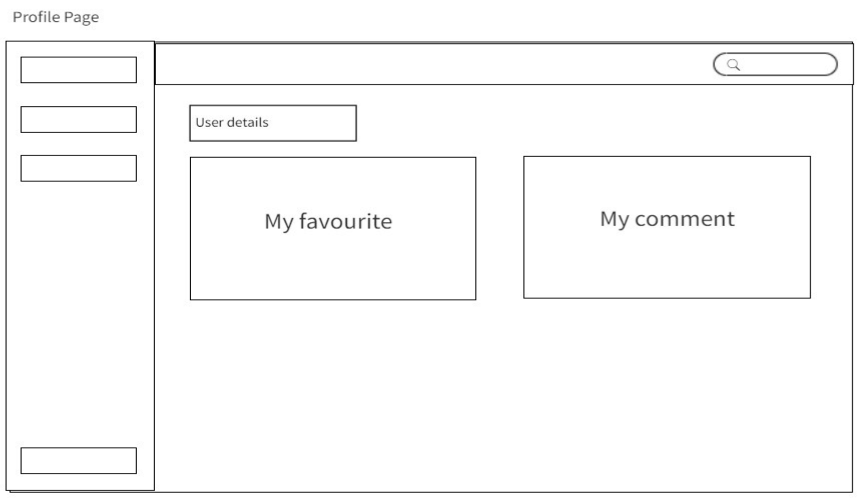


Figure 2.4: Profile Page Layout

Movie details page The movie details page is the showcase page for the movie. It contains movie introductions, ratings, director and actor names, stills, famous quotes, original soundtracks, and user comments. Users can also add this movie to Favorites.

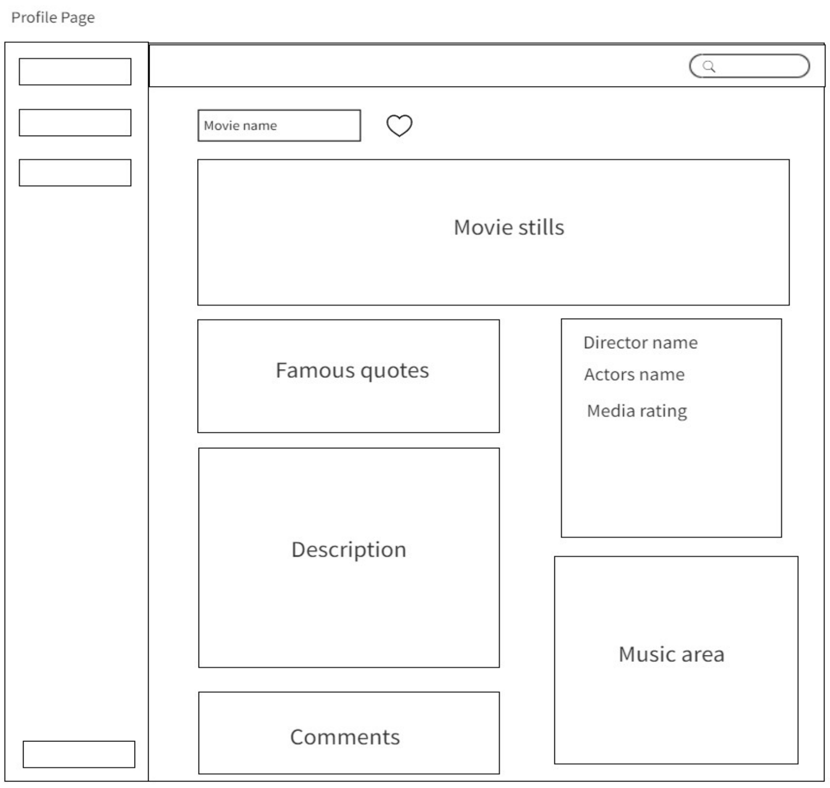


Figure 2.5: Movie Details Page Layout

Navigation bar The navigation bar on the left displays different content when the user is logged in and not logged in. When not logged in, it will display the Guest, home page link, sign up page link and login page link from top to bottom. When the user logs in, the user's full name will be displayed instead of the "Guest", and a link to the profile page will also be displayed. When the user is an administrator, there will be a link to add a movie. The upper navigation bar has a link button to the home page and a search bar that can filter the movie cards on the home page by movie title.

2.2 Page Interaction

2.2.1 For Users

1. Users can only simply browse the page when they are not logged in.

2. After logging in, users can view their profile and manage their comments and favorite movies on this page. On the movie details page, they can click the "heart" button to mark the movie as a favorite (it will turn into a red heart after the mark) or remove out from favorites (the movie not marked as favorites displayed as the outline-only heart). You can also post comments and delete your own comments in the movie comment section. Users can log out by clicking at the bottom of the navigation bar.

2.2.2 For Administrators

1. The administrator user has all page interactive functions of a normal user.

2. It is able to modify or add (delete) any part of the movie details page. Movie detail pages are modularized, and when logged in as an administrator, an interactive bar will appear above each module.

3. Manage comments of all users.

4. The left navigation bar will display the "Add a movie" link, which requires the administrator to input the movie name, media ratings (IMDb and Rotten tomatoes), and movie cover. After that, the card of the newly added movie will be listed on the home page, and then the administrator can go to the detail page of the movie to edit each module.

2.3 Main Methods and Tools

In this section, we present the outline of main tools that need to be used to implement the website.

Front-End The front-end is built using the Vue framework and constructed using Vue-CLI. The user interface is written using Vuetify.

Back-End The back-end is written using SprintBoot, database is supported by relational database MySQL. We use OSS (Object Storage Service) storage for static resources.

Proxy Nginx as a reverse proxy to convert HTTP requests to gRPC requests.

Communication gRPC as the way of communication between the front-end and back-end

Deployment Project is split into parts and containerized by Docker, orchestrated using Docker Compose.

This section will introduce the used languages and tools in detail.

2.3.1 Front-End Tools

The front-end framework uses Vue, combined with HTML, CSS and JavaScript languages, and the project is used by the Vue-CLI framework for package management with npm. UI style adopts Vuetify.

HTML HyperText Markup Language [3]. The most basic standard markup language for creating web pages, which can be used with CSS and JavaScript to design web pages. Browsers can read HTML files and render them into visual web pages.

CSS Cascading Style Sheets (CSS) specify how HTML elements are displayed, enabling pixel-precise control over the typography of element positions in web pages.

JavaScript JavaScript is a programming language born to meet the needs of making dynamic web pages. It was originally created to "make web pages more vivid". It is a scripting language that is widely used in web pages to realize web pages and browsing. In this project, it combines with Node.js to exert its powerful capabilities. In Vue.js it is defined in the <script> block.

Vue Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS and JavaScript, and provides a declarative and component-based programming model that helps us efficiently develop user interfaces, be it simple or complex. [4] It can also easily obtain data updates, and realize the interaction between the view and the model through specific methods inside the component. Vue componentized, modular and responsive design greatly simplifies the process of web development.

Vue-CLI Officially released standard tool for Vue.js development. It's built on webpack [5], with full community support and a wide variety of plugins. Combined with npm packages, various complex deployment and management plugins can be easily and quickly implemented.

Vuetify The user interface is written using Vuetify, a complete UI framework built on top of Vue.js. Vuetify takes a mobile first approach to design and it is developed exactly according to Material Design specification with every component meticulously crafted to be modular, responsive, and performant[6]. It can be quickly installed by adding the Vuetify Vue-CLI package using the cli (Pay attention: Vuetify only support Vue 2).

2.3.2 Back-End Tools

A java microservice project based on the SpringBoot framework, which combines a variety of plugins to provide gRPC services. Use the mybatis plugin to connect to the MySQL server and wrap the interface for accessing OSS.

SprintBoot The back-end is written using SprintBoot, an open-source Java-based framework used to create a micro Service[7]. Dependency Injection and Inversion of Control (IoC) can greatly simplify configuration and coding. In the project, we use Maven as a package management tool, and SpringBoot manages jar packages through spring boot starter, which simplifies Maven configuration. Different life cycles define different configuration files that can fit a variety of development environments.

MySQL MySQL is open-source, small in size and fast[8]. Moreover, with its open-source code and no copyright restrictions, the use cost is very low, which is very suitable for this project.

OSS This project uses OSS (Object Storage Service) storage for static resources, which is supported by Alibaba Cloud. Alibaba Cloud's OSS has many advantages, it is secure, cost-effective, and highly reliable cloud storage service that allows users to store a large amount of data [9]. It provides standard RESTful API interfaces that make downloading, uploading and changing files very easy.

2.3.3 Proxy Tools

Nginx Because of the separation of the front-end and back-end of this project and the particularity of gRPC, we need to use reverse proxy as a reverse proxy to convert HTTP requests to gRPC requests. In this project, we choose to use Nginx. Nginx has the characteristics of high concurrent connections, low memory consumption, low cost, simple configuration files, etc.[10], which is a perfect choice for this project.

2.3.4 Communication Tools

gRPC As a new attempt to replace the REST API, we choose gRPC as the communication tool between the front-end and the back-end. gRPC-web[11] provides functionality to send requests to the back-end. The back-end uses the maven package of yidongnan/grpc-spring-boot-starter[12] to integrate with SprintBoot. gRPC messages are serialized using protobuf, an efficient binary message format. Serialization of Protobuf on server and client is very fast. The serialized message volume of Protobuf is very small, so the performance is very good.

2.3.5 Deployment Tools

Docker Docker is a well-known open-source application container engine that allows us to package applications and dependencies into a portable container. Containerization makes application deployment very easy[13].

Docker Compose Docker Compose is a Docker container orchestration tool for defining and running multi-container Docker applications[14]. It enables rapid deployment of multi-container projects.

2.4 Software Installation

In this section, we will describe how to deploy and run this application on your host. We will also show the outcome after deployment. Before starting deployment, please make sure your system meets our system requirements.

The project can be found on Github at the link:

https://github.com/LinGuohao/Movie-Collection

All versions of development are recorded on Github, and you can also browse the versions you need to download.

2.4.1 Basic System Requirement

Hardware Requirements

Program can run on both Linux and Windows. The reference computer performance should be at least as shown in the table below:

Requires a 64-bit processor and operating system	
OS:	Windows or Linux
Processor:	Haswell Architecture 2.0 Ghz or better
Memory:	2 GB
Storage:	3 GB available space
Network:	Broadband internet connection required
Additional Notes:	Keyboard

Environment Requirements

The program is based on Docker deployment and the source code adopts Git version control. The Docker, Docker Compose and Git environments are required to be installed in the system. If you do not use the Fast Installation way to install the project, you may need additional environments.

Additional Requirements

Benefiting from the power of Docker Compose, we can deploy all containers with one click. In the default configuration, we require certain ports of the target machine to be accessible and unoccupied. If you customize docker-compose.yaml or deploy all containers manually, please refer to your configuration to open specified ports.

The default ports that are required to be opened and the services provided by the ports are shown in the following table:

Port	Service provided
80	Web application
9080	gRPC reverse proxy listening port
8888	Tomcat
3307	MySQL

2.4.2 Fast Installation

We have used Docker Compose to orchestrate all containers, so in the Fast Installation step, please make sure that the target host 80 (web application), 9080 (gRPC reverse proxy listening port), 8888 (Tomcat) and 3307 (MySQL) ports are accessible and not occupied. If needed to change ports, please refer to section 2.4.4 Customized Installation.

After making sure that your server meets the basic system requirements shown in section 2.4.1, follow the below steps to install the project:

1. Create a new project directory, initialize the Git repository, and pull the desired version from the project address to the local (in fast installation, please choose the version which includes the deploy folder, if you need to install other versions, please refer to Manual Installation in section 2.4.4 Customized Installation).

2. Open a terminal and change the path to the deploy folder.

3. Run: `docker-compose up`

4. Wait for the installation to finish.

2.4.3 Step by Step Installation with Docker Compose (Source Level)

The Manual installation will allow you to change the source code and default configuration files, and generate recompiled jar and dist packages. In this chapter, you only need to focus on customizing the code and configuration, and the deployment is still done by Docker Compose.

Additional System Environmental Requirements

In addition to meeting the Basic System Requirement in section 2.4.1, this installation method also requires Node.js version 8.9 (or above), Vue CLI 3.X, Maven 4.X and Java 11 on your computer.

Installation Steps

1. You can change the back-end code in the movie-db folder. After that, run: `mvn clean package` to package back-end project. When finished, copy jar file from movie-db/target to deploy/movie-db.
2. You can change the front-end code in the movie-web folder. After that, run: `npm run build` to package front-end project. When finished, copy dist folder from movie-web to deploy/movie-web.
3. Run: `docker-compose up`
4. Wait for the installation to finish.

2.4.4 Customized Installation

This section will describe how to highly customize this project. You can customize the mapped ports by modifying the docker-compose.yml file under the deploy folder.

It is not recommended to change the mapping of port 9080. If you want to change it, you need to change the VUE_APP_PORT attribute in the env file under the movie-web folder and the listening port in nginx.conf file under the deploy/movie-web folder.

You can make the back-end match your MySQL configuration by changing the configuration file under movie-db/src/main/resources.

Different life cycles use different configuration files, you can refer to the table below to choose to modify the corresponding configuration.

Front-End Configuration Files

Life Cycle	Configuration File Name
development	.env.development
production	.env.production

Back-End Configuration Files

Life Cycle	Configuration File Name
development	application.yaml
production	application-dev.yaml
test	application-test.yaml

Manual Installation

1. Use movie_db.sql under our deploy/mysql to initialize your MySQL database. When the back-end starts, the data in the OSS server and the MySQL movie_db database will be automatically synchronized, and the intersection of the two will be taken to delete any redundant data from the OSS server and the MySQL movie_db database.
2. Use the nginx.conf file under deploy/movie-web to replace your Nginx configuration file (if you have customized the RPC port, please modify the listening port manual).
3. Start the Back-End service.
4. Start the Nginx proxy.
5. Start the Front-End service.

2.4.5 Install the Front-End and Back-End on Different Servers

If you are required to deploy front-end and back-end on servers that use different IPs. You need to fill out the VUE_APP_HOSTNAME attribute in the env file under the movie-web folder using your back-end server IP or domain name.

2.4.6 Running the Program

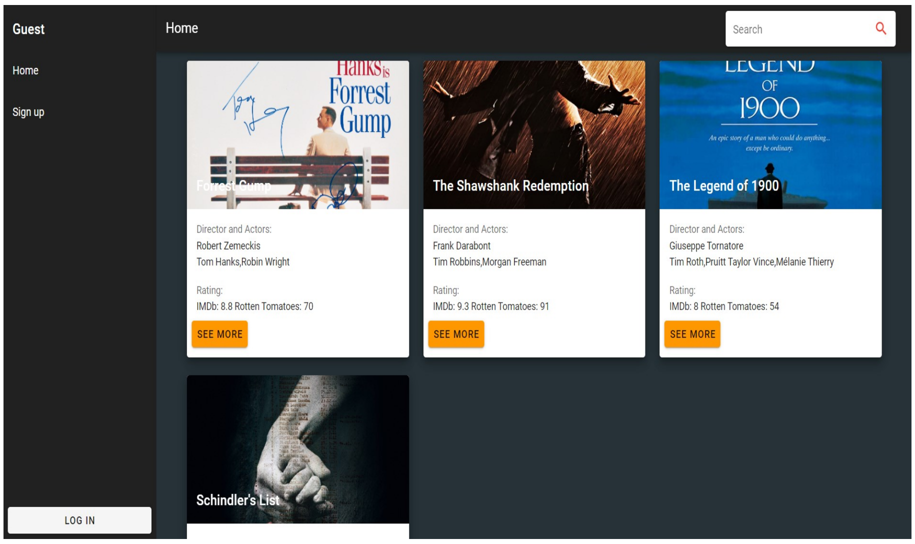
Any error messages will be printed to the console when deploying with Docker Compose. By default, after the back-end is deployed successfully, you can see the "Movie-db deploy successfully" message by accessing its port 8888 from the public network.

You can use grpcurl to test your RPC server is working or not as show in Code 2.1. For more usage of grpcurl, please refer to the github project [15].

```
lin@shadowsong:~$ grpcurl -plaintext localhost:9090 list
com.guohaohome.moviedb.proto.MoviedbService
grpc.health.v1.Health
grpc.reflection.v1alpha.ServerReflection
```

Code 2.1: Usage of grpcurl

By default, when the entire project is successfully deployed, you can access our project by accessing port 80 from the public network (port 8080 if deployed locally).



Forrest Gump
Director and Actors:
Robert Zemeckis
Tom Hanks,Robin Wright
Rating:
IMDb: 8.8 Rotten Tomatoes: 70

The Shawshank Redemption
Director and Actors:
Frank Darabont
Tim Robbins,Morgan Freeman
Rating:
IMDb: 9.3 Rotten Tomatoes: 91

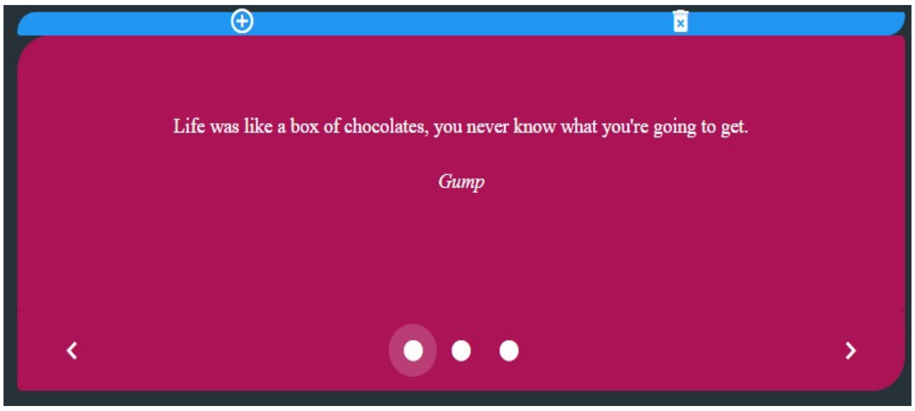
The Legend of 1900
Director and Actors:
Giuseppe Tornatore
Tim Roth,Pruitt Taylor Vince,Mélanie Thierry
Rating:
IMDb: 8 Rotten Tomatoes: 54

Schindler's List

Figure 2.6: Home Page without Registration

In the movie Forrest Gump, we show a rendering of the movie detail page with all the modules well filled out.

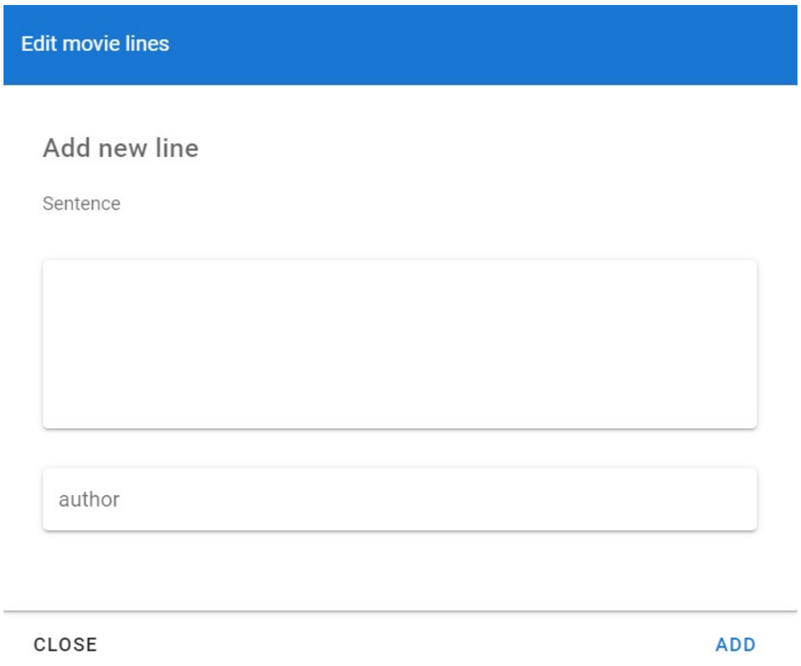
All sections on the page are modular and they can be edited and modified. Here is an example. The movie lines module will display all the sentences by cards, one sentence per card (the blue bar above the module only appears when you are logged in as an administrator) like Figure 2.7. Click the "plus" on the blue bar to add a sentence, and the text in the author will appear in italics below the sentence. Only when both "sentence" and "author" text boxes have text input, the page will respond after clicking the add button in Figure 2.8. If we click the "Trash Can" button, we can delete the sentence as shown in Figure 2.9.



Life was like a box of chocolates, you never know what you're going to get.

Gump

Figure 2.7: Movie Lines Module



Edit movie lines

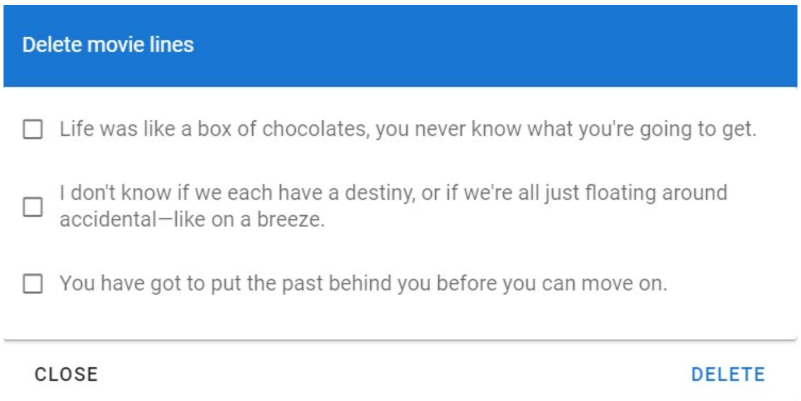
Add new line

Sentence

author

CLOSE ADD

Figure 2.8: Edit Movie Lines Window



Delete movie lines

☐ Life was like a box of chocolates, you never know what you're going to get.

☐ I don't know if we each have a destiny, or if we're all just floating around accidental—like on a breeze.

☐ You have got to put the past behind you before you can move on.

CLOSE DELETE

Figure 2.9: Delete Movie Lines

Specifically, in the music module, you can click the lower right corner of the card to expand the music list. By clicking the music name in the list to switch to the specified music, you can also pause and drag the progress bar at will to adjust the music playback progress. When the current music is ended, the module will automatically select the first music in the list to play.

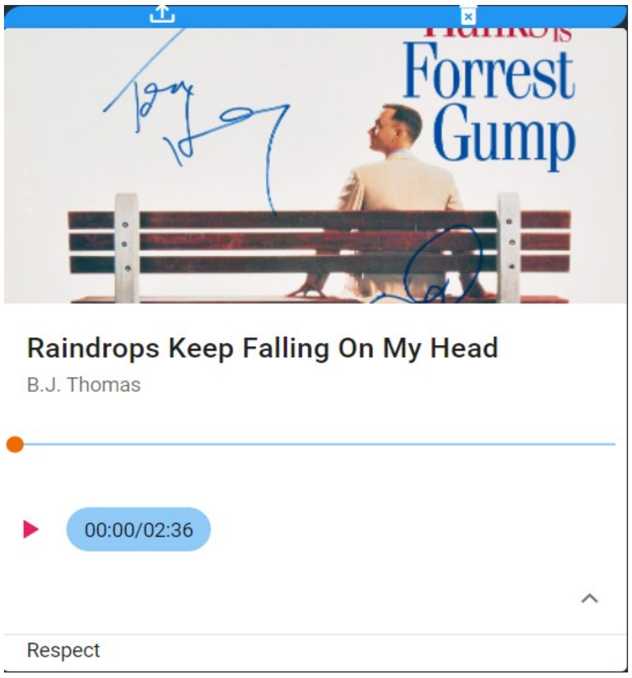
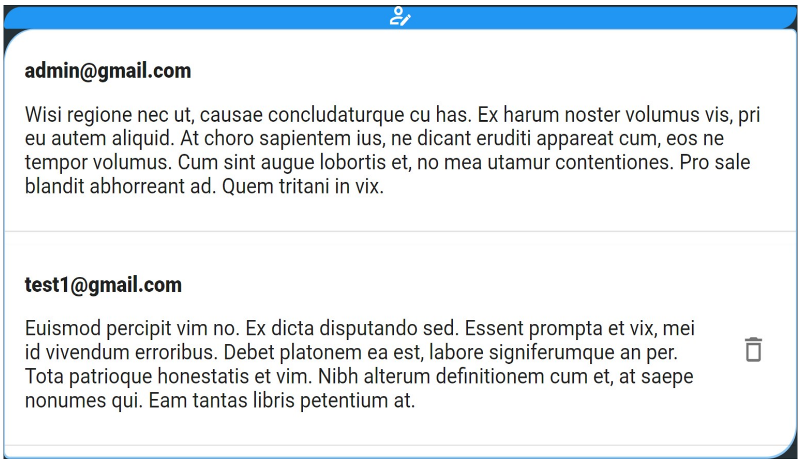


Figure 2.10: Music Card

After logging in, you can click the "Write" button on the blue bar in the comment area to add a comment. You can also delete your comment by clicking the "Trash Can" on the right of your comment shown in Figure 2.11. You can also see and delete your comments in the right half of your profile as Figure 2.13. If you are logged in as admin, you can delete any user's comment.



admin@gmail.com

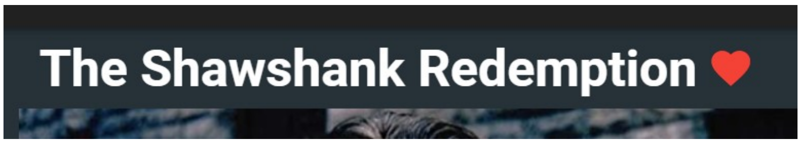
Wisi regione nec ut, causae concludaturque cu has. Ex harum noster volumus vis, pri eu autem aliquid. At choro sapientem ius, ne dicant eruditi appareat cum, eos ne tempor volumus. Cum sint augue lobortis et, no mea utamur contentiones. Pro sale blandit abhorreant ad. Quem tritani in vix.

test1@gmail.com

Euismod percipit vim no. Ex dicta disputando sed. Essent prompta et vix, mei id vivendum erroribus. Debet platonem ea est, labore signiferumque an per. Tota patrioque honestatis et vim. Nibh alterum definitionem cum et, at saepe nonumes qui. Eam tantas libris petentium at.

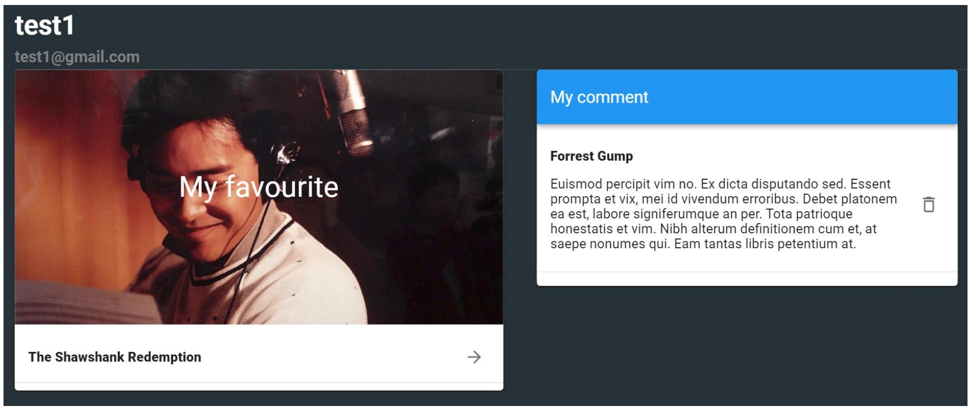
Figure 2.11: Comment Area Logged In as test1@gmail.com

You can also click the "heart" icon right of a movie name to add it to your favorite list (click the "heart" icon again to remove it) like Figure 2.12. All your favorite movies will be listed in the left half of your profile such as Figure 2.13.



The Shawshank Redemption

Figure 2.12: A Movie Was Added to Favorite List



test1

test1@gmail.com

My favourite

The Shawshank Redemption

My comment

Forrest Gump

Euismod percipit vim no. Ex dicta disputando sed. Essent prompta et vix, mei id vivendum erroribus. Debet platonem ea est, labore signiferumque an per. Tota patrioque honestatis et vim. Nibh alterum definitionem cum et, at saepe nonumes qui. Eam tantas libris petentium at.

Figure 2.13: Profile Page for test1@gmail.com User

If you are logged in as an administrator, you can find the "Add a Movie" link on the right navigation bar, in the popup window you only need to input the most basic movie information as shown in Figure 2.14. After the movie is created, you can edit each module in detail on the movie's details page. You can also delete a movie completely by clicking the "Trash Can" icon below the movie's name in the movie details page like Figure 2.15.

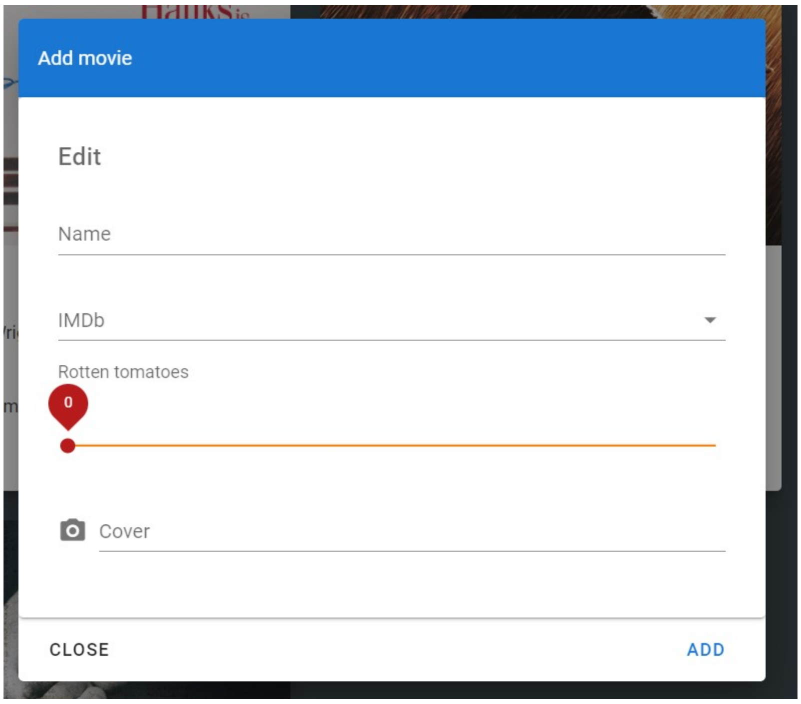
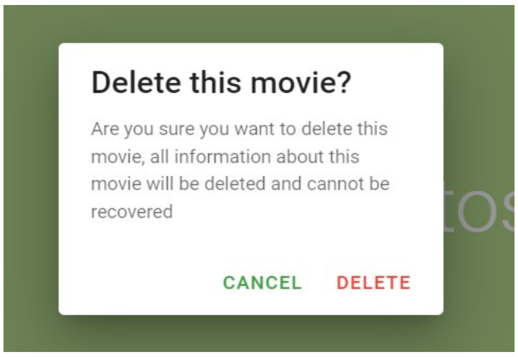


Figure 2.14: Add Movie Window



Delete this movie?

Are you sure you want to delete this movie, all information about this movie will be deleted and cannot be recovered

Figure 2.15: Confirmation Message about Deleting a Movie.

Currently, the default admin@gmail.com user is admin. To add more administrators, you can modify the roles column of the corresponding user of the movie_user table in the database.

The above is our documentation for users. After reading it you will be able to deploy our application in various ways and understand the layout of the web and how to interact with it. Developers please refer to the following chapters.

Chapter 3

Developer Documentation

In this chapter, we will discuss the architecture of our microservices and introduce our main components of the project separately. The understanding and use of configuration files and testing methods will also be covered.

3.1 Microservice Architecture

3.1.1 Why Choose the Microservices Architecture

Microservice is an architectural style that divides a huge application into small service units and uses API for resource access and operation between each unit.

Microservice is a design idea, which is the process of transforming application services from a single large-scale design to multiple small units. The early web application architecture design from a single architecture to the separation of front-end and back-end can be considered as today's prototype micro-services (or can also be considered as a simple microservice design).

This design brings us a lot of convenience[16], such as

- Individual microservice modules are language agnostic, so we can take advantages of each language.
- Each microservice module is independent of the other, they have their own life cycle and development cycle, and can be maintained and operated independently.
- Each module can be upgraded independently without uninstalling the entire application for maintenance, which ensures the iterability of the entire service.

The concepts of agile development and continuous integration can be well implemented by microservice architecture.

- In the development of large-scale projects, each unit of microservices can be independently developed and managed by a separate team, reducing management costs.
- Each unit of microservices can be integrated with other projects. This upgrades the concept of code reuse to component reuse.
- A problem with a microservice will not affect the operation of the application.
- Microservices are more suitable for distributed deployment, and can make dynamic expansion easier.

3.1.2 Application Structure Diagram

Our application design architecture diagram is shown below Figure 3.1.

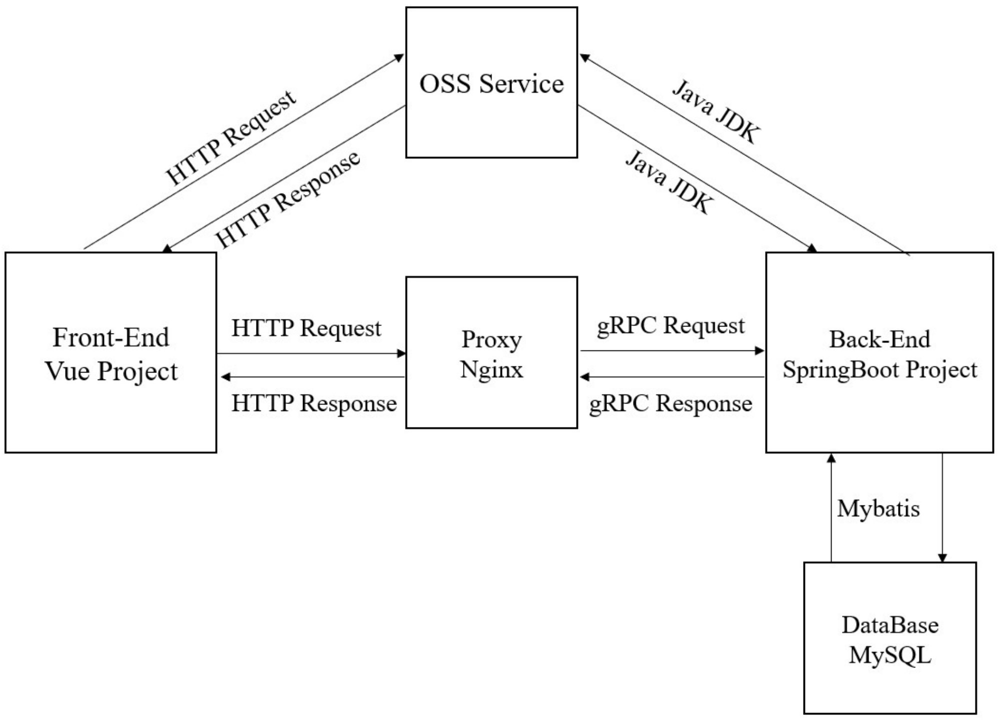


Figure 3.1: Application Architecture Diagram

The details of each part will be introduced in the following sections.

3.2 Data and the Way to Store Data

This section describes the data structure of our project and how we store and use data.

3.2.1 How to Store

We use ID to keep track of any new items that are added, which can be movies, reviews, stills, etc. The ID is generated using the current time as UTC milliseconds from the epoch plus a five-digit random number.

Data storage details can be found below.

MySQL

In the MySQL, we use the movie_db database to store our data. The introduction of each table can refer to the following:

Table Name	**Illustration**
info	Stores the movie name (name: varchar) corresponding to the movie ID (id: varchar), IMDb score (IMDb: double), Rotten Tomatoes Score (Tomatoes: int)
movie	Stores the name of the movie (name: varchar) and the corresponding id (id: varchar)
movie_lines	This table is used to store the lines of the movie that has the sentence (sentence: varchar), the author's name (author: varchar), the id of the sentence (line_id: varchar), and the id of the movie this sentence belongs to (id: varchar).
movie_user	Stores the data of all registered users information, with registered email (username: varchar), user's password (password: varchar), user's full name (fullname: varchar), user's role (roles: varchar)
user_comment	Stores the username who wrote this comment (username: varchar), the movie id that this comment belongs to (movieID: varchar), the content of the comment (content: varchar), we will also generate a unique id for this comment (commentID: varchar)
user_like	Stores the user's username (username: varchar) and the id of the item the user likes (ID: varchar)

Table 3.1: List of MySQL Table and Illustration

OSS

The OSS we choose is provided by Alibaba Cloud. Alibaba Cloud's OSS is affordable and provides very comprehensive documentation and technical support.

Alibaba Cloud OSS uses the "bucket" to store static files, so we created a new bucket named movie-db to store all the data of our project. In this bucket, we use the id of the movie as the root path to all the data for this movie.

The storage paths of various files can refer to the following table:

Path	**Illustration**
id/info.json	Information about the movie's actor names and director name.
id/cover.jpg	The cover of the movie, which will be displayed on the movie card of the Home Page and on the Music Component of this movie.
id/description.json	The content displayed on the description component of the Movie Detail page.
id/screenshot	This path is used to store the stills of the movie used on the movie details page scroll bar.
id/OST	This path is used to store the movie's soundtracks, which are displayed on the Music Components of the Movie Details page.

Table 3.2: List of OSS File Path and Illustration

Specifically, the music file is stored using this named format: {music name}_{artist name}.{music format}. For example, the music file named Respect_Aretha Franklin.mp3 means music name is Respect, the artist name is Aretha Franklin and the music format is MP3.

3.2.2 How to Use

We use Mybatis to access Java and MySQL, and request resources from OSS through HTTP protocol.

Mybatis

Mybatis is used to extract database objects into entity classes, and use the methods written in the Mapper file to operate the database[17]. Entity classes are in the back-end with path movie-db/src/main/java/sqlEntity.

Entity class	**Attributes**
Comment.java	String: commentID, String: username, String: movieID, String: content
Info.java	String: id, String: name, double: IMDb, int: Tomatoes
Line.java	String: id, String: sentence, String: author, String: line_id
Movie.java	String: id, String: name
User.java	String: userName, String: password, String: fullName, String: roles
UserLike.java	String: username, String: id

Table 3.3: List of Entity Classes

The Java interface files of Mapper are stored in movie-db/src/main/java/dao. Multiple function mappings based on business requirements are defined.

OSS

Regarding the static files stored in OSS, our back-end uses the officially provided JAVA JDK to implement CRUD. The front-end can make HTTP resource requests directly through the Axios library to get files.

3.3 Communication and RPC File

We use gRPC to implement the communication between the front-end and the back-end, so they need to share the same Protocol Buffers file, and then use the corresponding proto RPC compiler to compile the proto file into Java or JavaScript language.

Generate RPC File

In the back-end, benefit from protobuf-maven-plugin plugin[18], we just need to put the proto file into the movie-db/src/main/proto folder and run the compile and compile-custom of the plugin to automatically generate Java classes.

On the front-end, we need to manually compile proto files to generate JavaScript, so I made a script file to simplify the process. The corresponding JavaScript file can be generated by putting the proto file under the movie-web/protoGen folder and run the proto-gen.bat script. But it should be noted that we can still only send HTTP requests using the generated JavaScript file, so we need a reverse proxy to convert the request.

The schematic diagram is as follows:

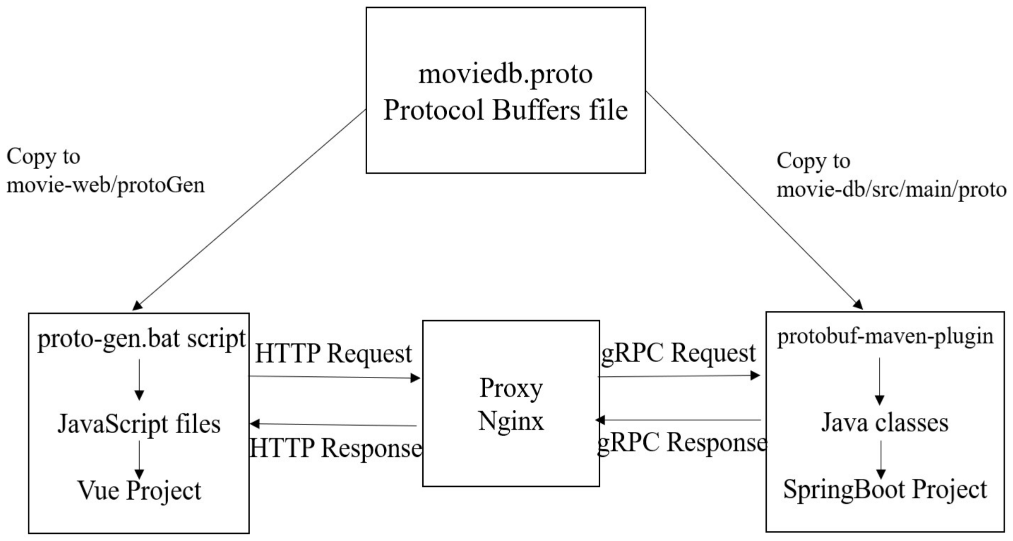


Figure 3.2: Compilation and Usage of Protocol Buffers File

Nginx Configuration

The Nginx reverse proxy configuration file nginx.conf is stored in the deploy/movie-web folder. The attribute is configured in the http server code block, where the code: `client_max_body_size 1G;` is indispensable. It specifies that nginx can proxy up to 1G of data (the default size is 3M). If this line of code is missing, we will not be able to upload the original sound track.

Proto File Considerations

In Protocol Buffers file, the BooleanResponse message type we created is used to represent boolean variables.

```
  message BooleanResponse{
    int32 isTrue = 1;
  }
```

Code 3.1: BooleanResponse Message Type

The reason why the boolean type provided by proto3 is not used is that in the proto3 protocol, the transmission default value (false) will be ignored, and the front-end will cause trouble when processing the ignored data, so we use the int value of

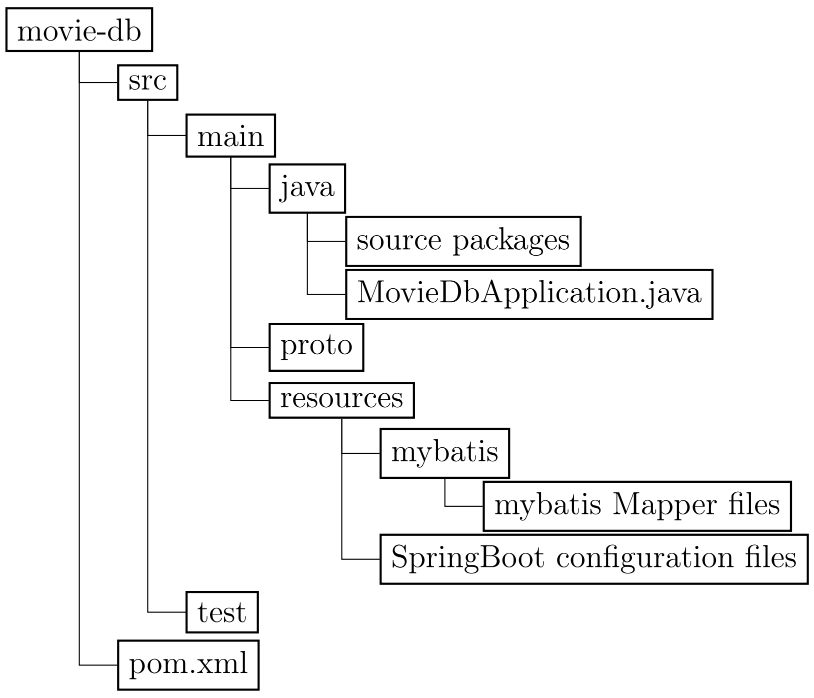
"1" to represent true, "-1" represents false to facilitate front-end processing like Code 3.2.

```
        builder.setIsTrue(user == null ? -1 : 1);
        BooleanResponse response = builder.build();
```

Code 3.2: An Example Usage of BooleanResponse Message Type

3.4 Back-End Manual

The back-end is built with SprintBoot and Maven. The project structure is shown as below.



3.4.1 Configuration

This section describes the main and key configuration files and how they affect our application.

Maven Configuration

In addition to the default SprintBoot configuration, the project also introduces many external dependencies and plugins.

Dependency Name	**Explanation**
mysql-connector-java	Provides the connection between Java and MySQL.
mybatis-spring-boot-starter	Persistence layer framework that extracts SQL statements.
h2	In-memory database for testing.
grpc-testing	gRPC test module.
grpc-server-spring-boot-starter	yidongnan gRPC SpringBoot dependencies.
aliyun-sdk-oss	Alibaba Cloud OSS service SpringBoot dependency.
commons-codec	Apache encoding and decoding package for converting Base64.
protobuf-maven-plugin (plugin)	Compile protobuf to Java source code.
grpc-bom	grpc dependency management.
wiremock	For component testing, testing HTTP requests.
junit-jupiter-api	Provide API required for junit test.
junit-jupiter-engine	Test engine for junit5.

Table 3.4: List of Dependencies and Plugins

In the following code block, we specify the location where the jar package packaged by maven is stored. We choose the JAVA 11, and the project version is 0.0.1. The packaged jar package will be generated in the target folder under the project directory.

```
  <properties>
    <java.version>11</java.version>
    <version>0.0.1</version>
    <target>../target</target>
  </properties>
```

Code 3.3: Maven Properties

gRPC kernels will conflict with each other when we use grpc-testing and grpc-server-spring-boot-starter, so we introduce maven bom[19] to be compatible with them:

```
<dependencyManagement>
  <dependencies>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-bom</artifactId>
    <version>1.44.0</version>
    <type>pom</type>
    <scope>import</scope>
  </dependency>
  </dependencies>
</dependencyManagement>
```

SpringBoot Configuration

Different life cycles use different configuration files, refer to the following table:

Back-End Configuration Files

Life Cycle	Configuration File Name
development	application.yaml
production	application-dev.yaml
test	application-test.yaml

Mybatis Mapper file is defined in the mybatis folder under the resource directory, and this path can be modified by changing this configuration:

```
mybatis:
  mapper-locations: classpath:mybatis/*.xml
```

The lifeCycle property block is used to selectively inject beans when SprintBoot starts:

```
lifeCycle:
  stage: dev
```

OSSConfiguration.java class under ossClient folder is OSS configuration. It is written as a configuration class and injected automatically when SprintBoot runs. The following initOSSClient method can creates an OSS client:

```
    public OSS ossClient(){
        return initOSSClientBuilder().build(endpoint, accessKeyId,
            accessKeySecret);
    }
```

Code 3.4: Init OSS Client

3.4.2 Methods and Implementation

This section introduces the methods that have typical characteristics and some specific implementations.

Synchronize OSS and MySQL Init.class in controller package will sync data in MySQL and OSS, it will keep movies that they have in common and then delete everything else from both locations. This class also provides detection of database connections by call `dataSource.getConnection()`, it will throw `SQLException` if database is not available. For implementation, we first use all movie id in MySQL with false pair to create HashMap (Code 3.5), then iterate all paths in the OSS. If HashMap has the same id, we change the key corresponding to the id to true. If the id is not found, we delete it from the OSS, and finally we iterate the HashMap and delete id from MySQLwhich the corresponding value is false.

Specially, Alibaba Cloud OSS JAVA SDK only provides a method to obtain all paths, so as shown in Code 3.6, we need to split all paths and get the movie id information using the paths.

Annotation method `@ConditionalOnExpression()` used to control the injection of this Component in the specified life cycle.

```
            List<String> movieList = movieMapper.getAllID();
            HashMap<String, Boolean> movieJudgeMap = new
                HashMap<String, Boolean>();
            for (String id : movieList) {
                movieJudgeMap.put(id, false);
            }
```

Code 3.5: Init HashMap

```
            ObjectListing objectListing = ossClient.listObjects
                (ossConfiguration.getBucketName());
            List<OSSObjectSummary> sums = objectListing.
                getObjectSummaries();
            for (OSSObjectSummary s : sums) {
                String link = s.getKey();
                String ossId = s.getKey().split("/")[0];
                if (movieJudgeMap.containsKey(ossId)) {
                    movieJudgeMap.put(ossId, true);
                } else {
                    ossClient.deleteObject(ossConfiguration.
                        getBucketName(), s.getKey());
                    System.out.println("OSSFile: "+s.getKey()+
                        " has deleted");
            }
        }
```

Code 3.6: Get Movie Ids of OSS and Delete Redundant Items

Base64 to file conversion `base64ToFile` method in `Utils` class. We use the Base64 file stream format to transmit the files uploaded from the Front-End. Therefore, we need to receive the file stream from the Back-End and convert it into a file before uploading it to OSS. Note that because of the Code 3.7, this method can currently only handle files of type image and audio.

```
        if (base64.contains("data:image")||base64.contains("
            data:audio")) {
            base64 = base64.substring(base64.indexOf(",") + 1);
        }
        base64 = base64.replace("\r\n", "");
```

Code 3.7: Base64 Stream Cutting

Generate music information `generateMusicInformation` method in `Utils` class will helps us to generate music infomation. The music path we obtained from OSS contains the music name and artist information, so as shown in Code 3.8, we need to process the information recorded in the path to acquire this information. It's worth noting that movie titles or artist name may also contain ".", so we need to take that into account when dealing with movie title and movie suffixes. Only the final "." will be ignored.

```
        String[] temp = fullName.split("\\.");
        fullName = "";
        for (int i = 0; i < temp.length - 1; i++) {
            fullName = fullName.concat(temp[i]);
            if (i != temp.length - 2) {
                fullName = fullName.concat(".");
            }
        }
```

Code 3.8: Music Path Processing

Generate id `generateId` method in Utils class will create ids for all our new items, we use the current time as UTC milliseconds from the epoch plus a random number less than 100000 to generate the ids.

To avoid potentially inserting a lot of information at the same time, we seed it with Calendar's time value in milliseconds plus a random number like Code 3.9.

```
Random r = new Random();
r.setSeed(time+new Random().nextInt());
```

Code 3.9: The Seed for Generate Id

SHA256 encryption `SHA256Encryption` method in `MoviedbService` class will help us encrypt the string with SHA256, we encrypt the users' passwords information to keep safe. Here we use Apache's tool class for this purpose. For more information, refer to the official website[20].

Delete movie `deleteMovieByID` method in `MoviedbService` class will delete a movie and all information corresponding to this movie. Because OSS file storage has no concept of folders, we need to delete all files that the paths contain the movie id. Such as Code 3.10, we generate a path iterator using the id name of the movie which we want to delete as a prefix.

```
ListObjectsRequest listObjectsRequest = new
    ListObjectsRequest(ossConfiguration.getBucketName())
        .withPrefix(request.getId() + "/")
        .withMarker(nextMarker);
```

Code 3.10: Generate Path Iterator Using Movie Id

As shown in Code 3.11, we use the `getKey` method to get all the eligible paths, then we use the SDK method of `deleteObjectsRequest` to delete all files.

```
DeleteObjectsRequest deleteObjectsRequest = new
    DeleteObjectsRequest(ossConfiguration.getBucketName()).
    withKeys(keys).withEncodingType("url");
DeleteObjectsResult deleteObjectsResult = ossClient.
    deleteObjects(deleteObjectsRequest);
List<String> deletedObjects = deleteObjectsResult.
    getDeletedObjects();
```

Code 3.11: Detete All Suitable Files from OSS

`objectListing.getNextMarker();` and `objectListing.isTruncated()` used to continue to start a new round of deletion at the last position when we need to delete too many files to be completely listed at one time.

To completely remove the movie's information, we also need to remove all contents about it in MySQL. So we need the Code 3.12 to delete all of them.

```
movieMapper.deleteMovieByID(request.getId());
infoMapper.deleteInfoByID(request.getId());
userLikeMapper.deleteUserLikeByID(request.getId());
commentMapper.deleteCommentByMovieId(request.getId());
```

Code 3.12: Delete All Contents about the Movie from MySQL

Determine if username is unique `judgeUsername` method in `MoviedbService` class will returns whether the same username exists in the database. We try to get the information corresponding to the specified username in the database through `getUserByUserName`. If there is no such information, we can determine that the username does not yet exist in the database.

```
BooleanResponse.Builder builder = BooleanResponse.
    newBuilder();
User user = userMapper.getUserByUserName(usernameRequest.
    getUsername());
builder.setIsTrue(user == null ? -1 : 1);
```

Code 3.13: Determine if Username is Unique

Authenticate user `authenticateUser` method in `MoviedbService` class will verify that the user's password is correct or not, we compare the password inputted by the user with SHA256 encryption and the encrypted string stored in the database to determine the correctness of the password like Code 3.14.

```
if (!SHA256Encryption(verificationRequest.getPassword()).
    equals(user.getPassword())) {
     builder.setUsername("wrong");
} else {
     builder.setUsername(user.getUserName()).setFullName(
         user.getFullName())
              .setPassword(user.getPassword()).setRoles(user.
                  getRoles());
}
```

Code 3.14: Determine the Correctness of the Password

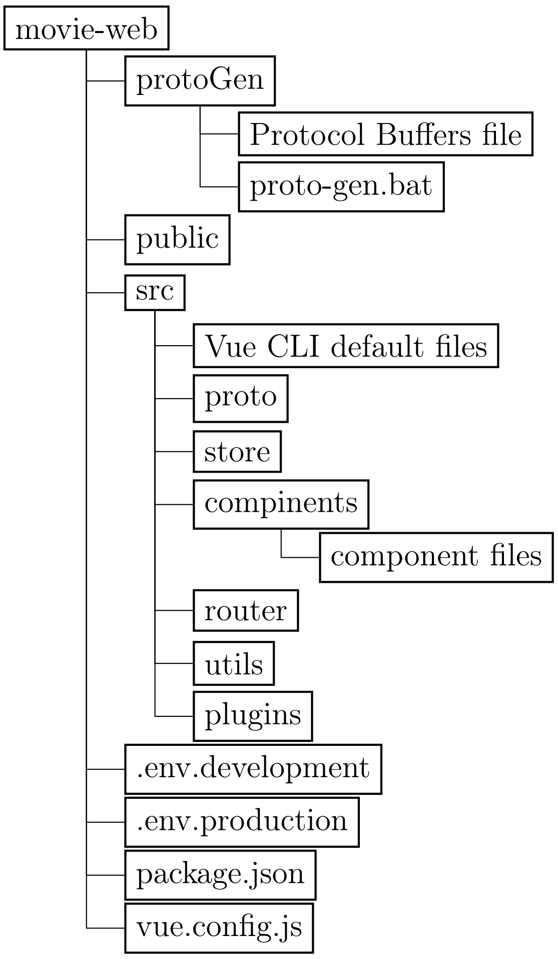
Upload file to OSS `uploadFileToOSS` method in `MoviedbService` class will help us upload files to OSS. Because sometimes we care about the filename (e.g. a music filename contains artist information) and sometimes not (e.g. uploading a movie still), we can do this by setting the `ObjectName` value in the `FileUploadRequest` when we request. As shown in Code 3.15, when this value is set to -1, our`uploadFileToOSS` service will automatically generate the file name (using the generateId method in the Utils class plus File Suffix), if not -1 then we will use the `ObjectName` in the request as the file name.

```
        if(request.getObjectName().equals("-1")){
            filename =  utils.generateFileName(request.getType
                ());
        }else{
            filename = request.getObjectName()+"."+request.
                getType();
        }
```

Code 3.15: Determine Filename

3.5 Front-End Manual

The front-end is built with Vue and constructed by Vue CLI. The project structure is shown in the below:



3.5.1 Configuration

This section describes the main or key configuration files and dependencies or packages and how they are used in our project.

Npm Package or Plugin

We use many external npm JavaScript packages and plugins for various purposes and to enrich our project.

Dependency Name	**Explanation**
ali-oss	Alibaba Cloud OSS JavaScript SDK.
google-protobuf	C# runtime library for Protocol Buffers - Google's data interchange format.
grpc-web	gRPC-Web provides a Javascript library that lets browser clients access a gRPC service.
vue-router	Vue Router is part of the Vue Ecosystem, which provides routing support for Vue.
vuetify	Vuetify is a complete UI framework built on top of Vue.js.
vuex	It serves as a centralized store for all the components in an application.
axios	A promise-based network request library.
protobufjs	TypeScript support for node.js and the browser.

Table 3.5: List of Dependencies and External Package

- vue-router configuration file under src/router.
- vuex configuration file under src/store.
- vuetify configuration file under src/plugins.
- The JavaScript generated according to the proto file is under src/proto.
- vuetify configuration file is vue.config.js.

Protocol Buffers

The protoGen folder is used for us to generate the corresponding JavaScript files according to the Protocol Buffers file. The protoGen/include folder is the necessary library required for compilation, protoc.exe and protoc-gen-grpc-web.exe are the compilers downloaded from grpc-web project on github[11].

We wrote proto-gen.bat script to simplify the compilation process. The script is used to specify how to use the compiler, specifying the compiled language, mode and the path to the compiled files.

Project Global Configuration

Different life cycles use different configuration files, refer to the following table:

Front-End Configuration Files

Life Cycle	Configuration File Name
development	.env.development
production	.env.production

In src/main.js defines the global variables used by the project. Including defining the axios variable, initializing the OSS client and RPC client, and loading the vuetify, router, and vuex modules.

```
new Vue({
  vuetify,
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

Code 3.16: Loading Modules

You can configure your own OSS in this area shown in Code 3.17.

```
let client = new OSS({
  endpoint: "Endpoint Address",
  accessKeyId: "accessKeyId",
  accessKeySecret: "accessKeySecret",
  bucket: "movie-db"
})
```

Code 3.17: Initializing OSS

Specially, in order to adapt to different development environments, we can load the RPC service address from the corresponding configuration file. VUE_APP_HOSTNAME is used for deployment when the external addresses of the front-end and back-end are different. The project will default set the back-end to the same IP address as the front-end if that value is empty. To achieve this, we used code like Code 3.18, extract the IP from the URL accessed by the user and use this IP as the destination IP to send the RPC request.

```
if (process.env.VUE_APP_HOSTNAME != '') {
  let address = "http://" + process.env.VUE_APP_HOSTNAME + ":"
     + process.env.VUE_APP_PORT
  Vue.prototype.$backend = new MoviedbServiceClient(address,
     null, null);
} else {
  let address = ""
  if (process.env.NODE_ENV == 'development') {
    address = "http://" + window.location.href.split("//")[1].
       split(":")[0] + ":" + process.env.VUE_APP_PORT
  } else {
    address = "http://" + window.location.href.split("//")[1].
        split("/")[0] + ":" + process.env.VUE_APP_PORT
  }
  ...
}
```

Code 3.18: Initialize the RPC Client

3.5.2 Componentization

We divide the web page into multiple independent components, and then we can realize complex functions by referring components to each other.

The components and functions currently available in the project are described in the following table:

Component Name	**Explanation**
App.vue	Control the navigation bar and "Add a movie" card
DescriptionPage.vue	Provides display and modification functions for the "Description" area of the "Movie Details" page
DetailPage.vue	Display movie detail pages, layout and manage multiple components
DetailRight.vue	Shows and provides editorial support for media ratings and director and actor name cards in the right half of the Movie Details page.
HomePage.vue	It has the function of displaying all movies by cards, and has the relevant code for searching bar.
LogInPage.vue	Rendering OF User login page.
MovieLines.vue	Rendering and editing support for the Movie Lines module in the Movie Details page.
MusicComponent.vue	Editing and rendering support for music cards on the Movie Details page.
ProfilePage.vue	Rendering of the user profile page.
SignUpPage.vue	Rendering of User login page.
UserComment.vue	Editing and rendering of the user comments section of the movie details page.

Table 3.6: List of Components and Functions

The logical relationship diagram of the mutual refer of the components is as follows Figure 3.3:

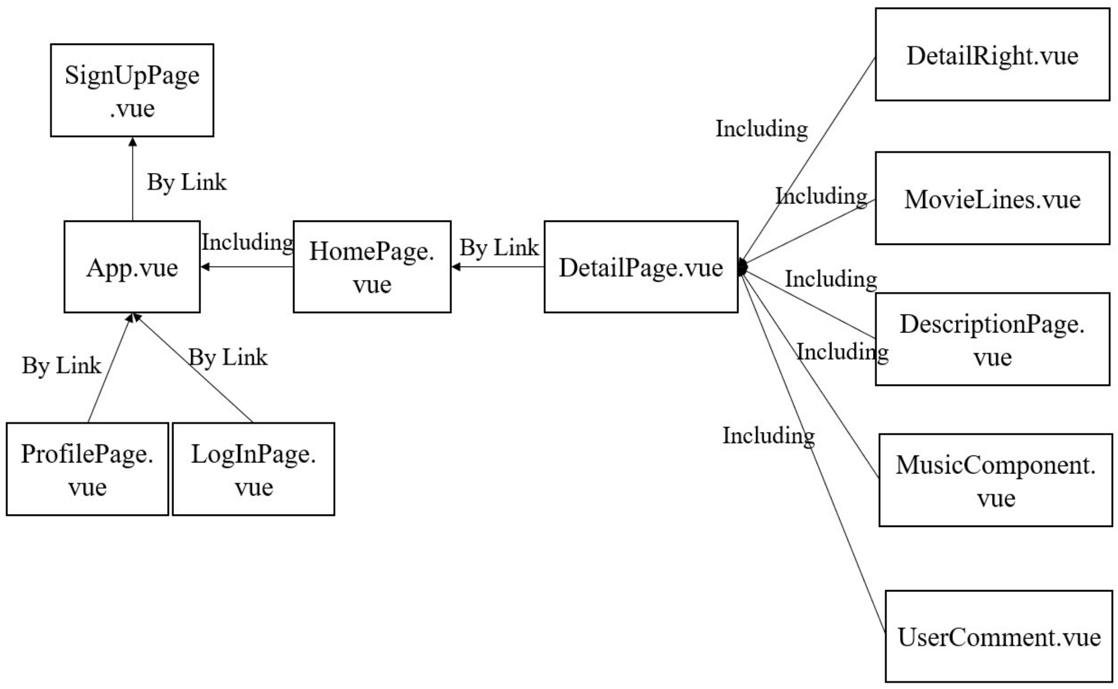


Figure 3.3: Component Relationship Structure

3.5.3 Methods and Implementation

This section introduces the methods that have typical characteristics and some specific implementations.

Common

Get login information We use localStorage to store user login information and define Vue computed variables to automatically obtain and use the user information. An example is shown in Code 3.19.

```
  userInfo() {
    if (localStorage.getItem("username") == null) {
      return null;
    }
    ...
```

Code 3.19: Usage of LocalStorage

Rule matching for input field Using the rules property provided by the v-text-field API, we can match input rules and display errors in a nice format like Code 3.20. For example, `addressEmail()` will take the input and return true or false depend on the input meet our requirement or not. `@blur` can help us remove the space after user input such as Code 3.21.

```
  <v-text-field
  ref="username"
  v-model="username"
  :rules="[
    () => !!username || 'This field is required',
    addressEmail() || 'Must be in Email format',
  ]"
  ...
  @blur="removeSpaces()"
></v-text-field>
```

Code 3.20: Usage of Rule Matching

```
  removeSpaces() {
    if (this.username) {
      this.username = this.username.trim();
    }
    if (this.fullname) {
      this.fullname = this.fullname.trim();
    }
  }
```

Code 3.21: Remove the Input Area Space

Activate input box The rules API in our v-text-field is only activated when the input box is clicked, so the page will not display an error message when the user clicks the Sign Up button without clicking the input box. In order to avoid this problem, as shown in Code 3.22, we first check whether all the input boxes meet the requirements when the user submits. If they do not meet the requirements, then we focus and then blur each input box in turn to activate the error of the message input box.

```
      if (!this.repassword) {
        this.$refs.repassword.focus();
        this.$refs.repassword.blur();
      }
      if (!this.password) {
        this.$refs.password.focus();
        this.$refs.password.blur();
        }
        ...
```

Code 3.22: Activate Input Boxes Error Messages

Type of file to upload The `accept` API in `<v-file-input>` can ask the user to upload a file of a specified type. An example like Code 3.23.

```
      <v-file-input
        accept="image/jpeg"
        placeholder="Pick a cover"
        ...
      ></v-file-input>
```

Code 3.23: Type of File to Upload

Get the Base64 content of a file We use target.result in the callback function of the onload method in the FileReader class to get the Base64 of the file. The back-end obtains files by processing Base64 and uploads them to OSS.

```
var reader = new FileReader();
reader.readAsDataURL(file);
reader.onload = (e) => {
  this.$backend.uploadFileToOSS(
    new FileUploadRequest()
      .setObjectpath(this.newid + "/")
      .setType("jpg")
      .setContent(e.target.result)
      .setObjectname("cover"),
    {},
    ...
```

Code 3.24: Get the Base64 Content of a File and Upload

Dynamic refresh Vue can make it easier for us to update the content without refreshing the entire page. Vue detects element changes and helps us automatically updates the elements on the page dynamically.

Use v-mode to control the display We can bind a boolean value to the `v-model` in some elements to dynamically show or hide the elements. Such like `<v-dialog>` and `<v-alert>` in Code 3.25. This greatly facilitates us to control whether the element is displayed or not.

```
  <v-dialog
    max-width="600"
    v-model="showDelete"
    ...
  />
  <v-btn text @click="showDelete = false">Close</v-btn>
  del() {
    this.showDelete = true;
  }
```

Code 3.25: V-Mode to Control the Display

Preparation of a large amount of callback data When we request multiple pieces of data from the back-end or OSS, these operations are asynchronous, but we sometimes care about the order in which elements are returned to correspond multiple sets of data to each other. In this case, we can use the `Promise.all` method like Code 3.26.

```
let pList = [];
let infoList = [];
let ret = response.toObject().replyList;
ret.forEach((element) => {
  pList.push(this.getInfoJson(element.id));
  infoList.push(this.getInfo(element.id));
  this.covers.push(this.ossPrefix + element.id + "/cover.jpg"
     );
});
Promise.all(pList).then((res) => {
  this.movies = res.map((item) => item.data);
});
Promise.all(infoList).then((res) => {
  this.info = res.map((item) => item);
});
```

Code 3.26: Usage of Promise.all

Search bar Because the element of the Search Bar is in the App.vue component and the display of the movie card is in the HomePage.vue, we achieve this by maintaining a vuex variable[21], monitoring the value change in the Search

Bar in real-time through the watch block, and then in the HomePage. This value is used to decide whether to show the movie card.

```
  watch: {
   searchText(newValue) {
     this.$store.state.searchText = newValue;
   },
 }
   <template v-for="(n, index) in movies">
       <v-col cols="4" v-if="searchMethod(info[index][1])" :
          key="index">
       ...
   searchMethod(movieName) {
     if (this.searchText == "") {
       return true;
     } else {
       if (
         movieName.toLowerCase().indexOf(this.searchText.
            toLowerCase()) !== -1
       ) {
         return true;
       ...
```

Code 3.27: Search Bar

Passing values in components Benefiting from Vuex, it becomes easier for us to pass values in multiple components. For example in Code 3.28, DetailPage.vue contains multiple components, but we only need to get the movie information once to use it for other contained components.

```
const store = new Vuex.Store({
    state: {
        detailInfo: [],
        detailInfoJson: [],
        searchText:""
    },
    ...
this.$backend.getInfoByID(
          ...
          this.$store.state.detailInfo = response.array;
          ..
```

Code 3.28: Passing Values in Components

Record user's multiple selections Bind variables to `v-model` in the `v-item-group` element can record the elements which user clicked on as an array: `<v-item-group v-model="selected" multiple>`.

Web typography Benefiting from Vuetify, we can quickly and easily layout web pages to arrange the positions of various containers.

```
<v-row>
  <v-col cols="7">
   ...
  </v-col>
  <v-col cols="5" style="padding-left: 50px">
    ...
  </v-col>
</v-row>
```

Code 3.29: Web Typography

Control rendering Because Vue helps us control the rendering of the view layer, this allows us to focus only on our own project logic without worrying about how and when the DOM is rendered. When data changes, Vue will help us automatically re-render the changed areas. We can also prepare all the data in advance and display it through the v-if API like Code 3.30 without worrying about the data not being fully loaded.

```
<v-row v-if="info.length == movies.length">
...
</v-row>
```

Code 3.30: Control Rendering

SignUpPage.vue

Password format check We specify that the user's password input must be 6-16 digits and contain numbers and letters. So we define a regular expression to check the user's input used Code 3.31, if the input does not meet this requirement we will display an error message to the user.

```
checkPassword() {
  var reg = /^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]{6,16}$/;
  if (this.password) {
    if (reg.test(this.password)) {
      this.isPassword = true;
      return true;
    } else {
      this.isPassword = false;
      return false;
    }
  }
}
```

Code 3.31: Check Password Method

Email format check We use a regular expression to match if the user has inputted the correct email format.

UserComment.vue

Display of delete icon We only allow users to delete their own comments, so we need to decide whether to show the delete icon or not based on the author of the comment.

```
<v-list-item-action v-if="showDelete(item[1])">
  <v-btn icon @click="deleteComment(item[0])">
    <v-icon>mdi-delete-outline</v-icon>
  </v-btn>
</v-list-item-action>
```

Code 3.32: Decide whether to Show the Delete Icon

LogInPage.vue

Store user login information When all the input entered by the user meets our requirements and the back-end verification is successful, we use localStorage such as Code 3.33 to store the user's information and redirect the webpage to the homepage.

```
localStorage.setItem('username',response.array[0]);
localStorage.setItem('fullname',response.array[2]);
```

```
    localStorage.setItem('roles',response.array[3]);
    this.goHome();
    window.location.reload();
```

Code 3.33: Store Logged in User Information

DescriptionPage.vue

User input to Json file We sometimes need to use JSON files to store content, so we need to convert the String type data input by the user into the text content supported by JSON. Because we want to display Json content with web elements, and JSON does not have good support for line breaks and colons, so we need to replace these elements with HTML elements as shown in Code 3.34.

```
      editedContent = editedContent
        .replace(/\r\n/g, "<br>")
        .replace(/\n/g, "<br>")
        .replace(/"/g, '\\"');
```

Code 3.34: Process String to Fit JSON Format

MusicComponent.vue

Drag the progress bar to adjust the music progress We bind two mouse events for `<v-slider>` to monitor the mouse down and up respectively in Code 3.35. By binding `v-model` to `<v-slider>` we can get the current position of the progress bar. Based on this value we calculate where the music time should be placed when the mouse is up from the progress bar.

```
    <v-slider
    v-model="progress"
    @mousedown="down"
    @mouseup="up"
    :color="'orange darken-3'"
    ></v-slider>
      up() {
      this.isDown = false;
      this.audio.currentTime =
      (this.musicLength / 100) * this.progress;
```

```
        ...
    }
```

Code 3.35: Drag the Progress Bar to Adjust the Music Progress

Automatically change We manage the music by setting the currently playing song as clicked and maintaining a musicList. As shown in Code 3.36, when the currently playing song finishes playing we add the song to the end of the musicList and remove the first song from the list to play.

```
    endMusic() {
      this.musicList.push(this.clicked);
      this.clicked = this.musicList.shift();
      this.prepareMusic(this.$ossPrefix + this.clicked[1]);
      ...
    },
```

Code 3.36: Automatically Change

Update progress bar We bind the `update` method to the timeupdate event of the audio element to update the progress bar in real-time, and calculate the position of the progress bar based on the song length and the current time.

Timestamp update We use Vue's computed block to dynamically calculate and update the timestamp through the change of the progress bar value.

```
     timestamp() {
          ...
          this.formatTime((this.musicLength / 100) * this.
             progress) +
          "/" +
          this.formatTime(this.musicLength)
          ...
```

Code 3.37: Timestamp Update

DetailPage.vue

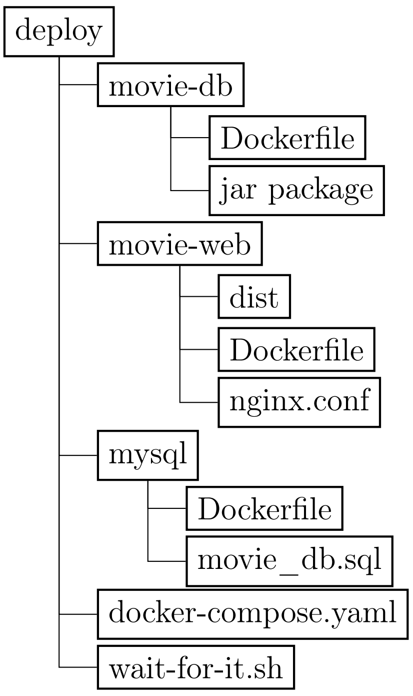
Bulk delete operations Users can delete multiple photos at once. We use `Promise.all` to wait for all operations to complete before refreshing the page.

```
Edit() {
  this.selected.forEach((e) => {
    this.pList.push(this.deleteOperation(this.imagesPath[e][0]));
  });
  this.pList.push(this.editNameOperation());
  Promise.all(this.pList).then(window.location.reload(true));
}
```

Code 3.38: Bulk Delete Operations

3.6 Deployment Manual

We used Docker and Docker Compose to simplify the deployment of the application, the project structure is shown below:



We deploy the entire application through three microservice parts, each packaged separately into a Docker project. They are stored in three folders under the deploy folder (`movie-db`, `movie-web`, `mysql`).

Each microservice is created from other published base images. You can refer to the following table:

Docker Image	**Base Image**	**Explanation**
movie-db	openjdk:11	Back-end project that will provide RPC services and connections to MySQL.
movie-web	nginx	Front-end web serving and reverse proxy.
mysql	mysql:5.7	Deployment and initialization of MySQL database

Table 3.7: Docker Images and Corresponding Base Images

Because the front-end and back-end of our project are separated, and the back-end has database connectivity detection, the installation order of each image is very important. So in `docker-compose.yaml` we use the `depend_on` property to control the order of startup. But this is not enough, because this property can only control the startup order to ensure that our Docker network is successfully built and the reverse proxy can detect the front image, but it cannot guarantee that the front image has been deployed before starting the subsequent image. (This still causes checks such as database connectivity probes to fail).

In order to solve this problem, we introduced the wait-for-it.sh script. As shown in Code 3.39, after putting this script into the image, it can detect whether the image of the specified IP is deployed, and then trigger other operations.

```
  depends_on:
    - "database"
  command: ./wait-for-it.sh database:3306 -t 30 -- java -jar /app
      .jar --spring.profiles.active=dev
```

Code 3.39: Usage of wait-for-it.sh

Regarding the wait-for-it.sh script, we use the vishnubob project. For more information, please refer to "wait-for-it" script[22].

You can change the default database password in the MySQL Dockerfile:

`ENV MYSQL_ROOT_PASSWORD="Your Password"`

The database is case-sensitive by default, which will cause many problems. You need to add the following code to solve it:

`CMD ["-lower_case_table_names= 1"]`

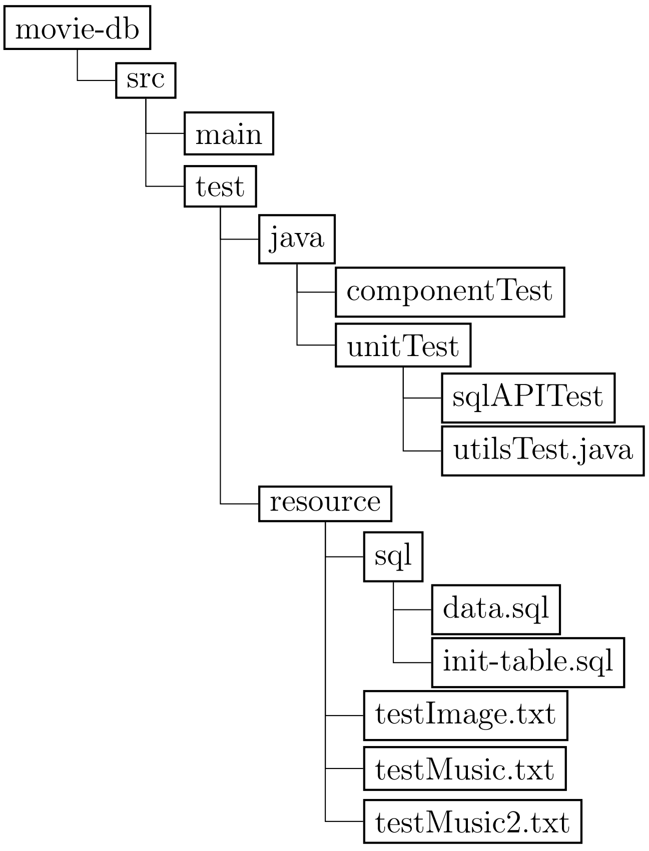
3.7 Testing

As we all know, as an important life cycle of application development, testing is an essential and significant part. The tests ensure the usability and stability of the interface, cover a variety of extreme cases, and lay the foundation for the program to run in various situations.

We divide the test into two parts, the `unit test` and the `component test`. The unit test mainly tests the usability of Mybatis, and the component test is used to test whether the API of the entire back-end is working properly.

This section will introduce how we tested our application, as well as ideas, methods, configuration instructions, etc.

The test is integrated in SpringBoot, and the structure is as follows:



3.7.1 Dependencies and Tools

Our back-end project mainly helps us connect to the database and provide RPC services. Therefore, we need to simulate the database environment. In this case, we use the h2 in-memory database[23] to simulate the real usage environment and use the grpc-testing dependency[24] to simulate RPC requests. The back-end also involves CRUD to the OSS server, therefore, we need to simulate the HTTP request to check the correctness of the operations, so we use wiremock dependencies[25] to implement it. All tests are based on Junit5, Junit4 has compatibility issues with our maven so it was deprecated by us.

The tools or dependencies involved for testing can refer to the following table:

Tool Or Dependency	**Explanation**
h2	In-memory database for simulating a database environment.
junit-jupiter-api	Provide API required for junit test.
junit-jupiter-engine	Test engine for junit5.
grpc-testing	Used to test RPC services, providing easy-to-write requests and easy-to-handle RPC return values.
wiremock	Open source testing tool that supports HTTP response stubs.

Table 3.8: Tools or Dependencies Involved for Testing

3.7.2 Configuration and Ideas

The `data.sql` and `init-table.sql` files in the `resource/sql` folder are used to initialize the h2 in-memory database, where you can find the simulated data.

In the `resource` folder, there are three files, `testImage.txt`, `testMusic.txt`, and `testMusic2.txt`, which store the Base64 data of the test image and the two test music in txt format. They are used as temporary files to test the feasibility of the OSS interface and the Base64 stream-to-file method.

Maven Configuration

Junit4 is not compatible with our maven. As shown in Code 3.40, we need to exclude SpringBoot's default junit to configure and use junit5.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Code 3.40: Exclude Default Junit

The test class must be placed in the `test` folder. The matching rules of the test class path included by default in maven's test are as follows: `**/*Test.java, **/Test*.java, **/*TestCase.java`. If you want to include classes with other names, you need to manually configure the surefire plugin.

Annotation Explanation

`@ActiveProfiles("test")` It specifies the configuration file used when testing starts and in the project its defaults to `application-test.yaml`. In it, we specify the storage location of the initial file of the h2 in-memory database.

```
datasource:
  url: jdbc:h2:mem:test;DB_CLOSE_DELAY=-1
  ...
  schema: classpath:sql/init-table.sql
  data: classpath:sql/data.sql
```

Code 3.41: H2 Configuration

`@DirtiesContext` Ensures that the grpc-server is properly shutdown after each test. Without this annotation, our tests would not be able to run. There will be port occupancy problems.

For more information, please refer to the yidongnan gRPC unit testing instruction[26].

Test Ideas

Our initial data only needs to ensure that the operations of obtaining information from the database can be fully tested. Such as `testGetInfoByID()`, `testGetAllID()` methods. The reason for this is that when multiple test classes are running, we cannot really guarantee the running order of the junit test. Using too much data initialization will cause operations such as insertion and deletion to affect each other and get uncontrollable test results. So we can test for insertion and deletion methods at the same time, and restore data to default values after testing methods such like update() to maintain data consistency.

This idea also applies to component testing. We create an OSS temporary folder and delete this folder after testing to maintain the consistency of data for each test case.

3.7.3 Unit Test

Our unit tests are divided into two parts, the first part is located under `sqlAPITest` folder and is used to test the database API. The second part is `utilsTest.java` for testing our tool class `Utils.java`.

We show some typical implementations below.

Test SHA256 encryption The stability of the method is judged by encrypting the same string twice to obtain the same result.

```
public void SHA256EncryptionTest() {
    String plain = "SHA256EncryptionTestPlain";
    String cipher = Utils.SHA256Encryption(plain);
    String cipher2 = Utils.SHA256Encryption(plain);
    assertEquals(cipher, cipher2);
}
```

Code 3.42: Test SHA256 Encryption

Test id generation Test this method by generating 100 ids at the same time to see if they are all different. Such as Code 3.43, we use a hash table to determine if there are duplicate elements.

```
HashMap<String, Integer> hashMap = new HashMap<>();
for (int i = 0; i < 100; i++) {
    String id = utils.generateId();
    assertFalse(hashMap.containsKey(id));
    hashMap.put(id, 1);
}
```

Code 3.43: Test ID Generation

Test insertion and deletion at the same time Maintain the original data in the database by testing the insertion and deletion methods at the same time.

```
User user = new User(...);
userMapper.insertUser(user);
user = userMapper.getUserByUserName("...");
assertEquals ...
userMapper.deleteUserByUserName("...");
assertNull(userMapper.getUserByUserName("..."));
```

Code 3.44: Test Insertion and Deletion at the Same Time

Update testing We test the upgrade method with the original data in the database, and restore the data after the method is finished.

```
Info currentInfo = infoMapper.getInfoByID("testMovieId");
Info newInfo = new Info(currentInfo.getId(),"
    testUpdateInfoByIDName3"
        ,9.4,94);
infoMapper.updateInfoByID(newInfo,"testMovieId");
Info updateInfo = infoMapper.getInfoByID(currentInfo.getId
    ());
assertEquals ...
...
infoMapper.updateInfoByID(currentInfo,currentInfo.getId());
```

Code 3.45: Update Testing

3.7.4 Component Testing

Component testing is also an integral part of application testing, which can completely test the stability and reliability of the entire application.

In our program, component testing mainly tests the program's interactivity with the OSS server and the accuracy of the RPC service.

Here we continued the idea of unit testing. We cannot guarantee the execution order of a large number of composite tests, so we need to ensure that the data is consistent for each test case.

For each test involving the OSS server, we create a temporary `test` folder to execute and store temporary files, and then send HTTP requests through wiremock to test the correctness of OSS operations. For tests on the database, using the same strategy for unit testing, changes to the database need to be reverted after the test is over.

Benefiting from the power of the grpc-testing dependency, which we can use to simulate real RPC requests and responses.

We show some typical implementations below.

Use grpc-testing to simulate requests and corresponding Code 3.46 is an example of using the grpc-testing dependency, which we use heavily in our component tests.

```java
StreamRecorder<AllMovieIDListResponse> responseObserver =
    StreamRecorder.create();
moviedbService.getAllID(null, responseObserver);
if (!responseObserver.awaitCompletion(10, TimeUnit.SECONDS)) {
     throw new TimeoutException();
}
assertNull(responseObserver.getError());
List<AllMovieIDListResponse> result = responseObserver.
    getValues();
assertEquals(1, result.size());
```

Code 3.46: Use grpc-testing to Simulate Requests and Corresponding

Get the Base64 stream for testing In actual use, the front-end sends Base64 file streams to the back-end, and the back-end needs to convert them into files and upload them to OSS. Therefore, we also need to simulate this process when testing. The following Code3.47 shows how we read the Base64 stream in the `resources` folder into String.

```java
final ClassPathResource classPathResource = new
    ClassPathResource("testImage.txt");
InputStream inputStream = classPathResource.getInputStream();
String content = StreamUtils.copyToString(inputStream,
    StandardCharsets.UTF_8);
inputStream.close();
```

Code 3.47: Get the Base64 Stream for Testing

Test files exist in OSS We use the package provided by wiremock to send HTTP requests and determine the return value such as Code3.48.

```java
     RestTemplate restTemplate = new RestTemplate();
```

```
    ResponseEntity response = restTemplate.getForEntity(
        ossPrefix + "/test/test.jpg", String.class);
    assertEquals(response.getStatusCode(), HttpStatus.OK);
```

Code 3.48: Test Files Exist in OSS

Test file does not exist in OSS Because the `getForEntity` method cannot actually return a 404 error code but throws an exception when the file does not exist, we need to check for the exception.

```
assertThrows(RestClientException.class, () -> {
    restTemplate.getForEntity(ossPrefix + "/test/test.jpg",
        String.class);
});
```

Code 3.49: Test File does not Exist in OSS

Test file content Using the getBody method of the ResponseEntity class like Code3.50, we can assert the file content.

```
RestTemplate restTemplate = new RestTemplate();
ResponseEntity response = restTemplate.getForEntity(ossPrefix +
    "/test/test.json", String.class);
assertEquals(response.getStatusCode(), HttpStatus.OK);
assertEquals("test:testUploadTextToOSS", response.getBody());
```

Code 3.50: Test File Content

StreamRecorder is not reusable problem The object instantiated by StreamRecorder cannot be repeatedly passed into RPC requestss, otherwise it will only retain the value when it was assigned for the first time. As shown in Code3.51, the class needs to be re-instantiated when using the same type of StreamRecorder multiple times.

```
        //Insert
        StreamRecorder<BooleanResponse> insertBoolean =
            StreamRecorder.create();
        moviedbService.insertLine(lineList, insertBoolean);
        assertEquals(1, insertBoolean.getValues().get(0).
            getIsTrue());
        //Delete
        StreamRecorder<BooleanResponse> deleteBoolean =
            StreamRecorder.create();
        moviedbService.deleteLine(deleteLineRequest,
            deleteBoolean);
        assertEquals(1, deleteBoolean.getValues().get(0).
            getIsTrue());
        ...
```

Code 3.51: Re-Instantiated StreamRecorder

Test InsertUser When testing `InsertUser`, we also need to test that the existing username cannot be inserted.

Test AuthenticateUser When we test `AuthenticateUser`, we need to test that the authentication is successful, the password is wrong and the user does not exist.

Test DeleteUserLike When testing `DeleteUserLike` we also need to test the UserLike record does not esixt.

Test DeleteMovieByID Because a movie involves OSS and multiple tables in the database. As shown in Code 3.52, it is necessary to check whether the content has been erased from all tables in the database and the OSS server when deleting.

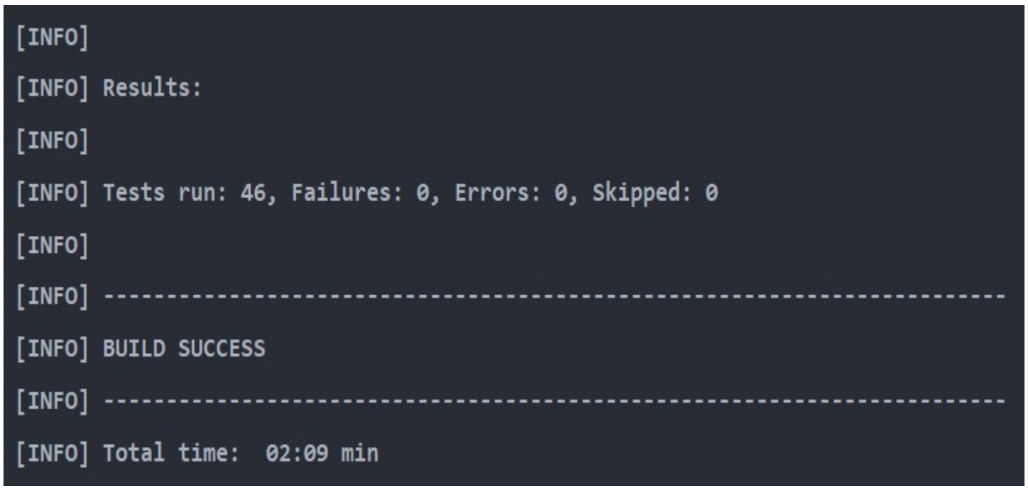
```
//Test no data in movie and info table
assertNull(movieMapper.getNameByID(info.getId()));
assertNull(infoMapper.getInfoByID(info.getId()));
//Test no data in UserLike table
assertEquals(0, userLikeMapper.getUserLikes("
    testInsertMovieAndDeleteMovieByIDUsername").size());
//Test no data in comment table
assertEquals(0, commentMapper.getCommentByMovieID(info.getId())
    .size());
//Test no data in OSS
moviedbService.getOssObjectList(ObjectListRequest.newBuilder().
    setBucketName(ossConfiguration.getBucketName())
        .setKeyPrefix(info.getId() + "/").build(),
            objectListResponseStreamRecorder);
assertEquals(0, objectListResponseStreamRecorder.getValues().
    get(0).getReplyCount());
```

Code 3.52: Multiple Checks for DeleteMovieByID Method

3.7.5 Run of the Test

You can run all tests by simply using the command: `mvn clean test`. Because the testing will repeatedly start and shut down multiple services and run with relatively complex dependencies and configurations, so it needs some time to finish.

When it finishes running, you can see the test results in the console:



```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 46, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:09 min
```

Figure 3.4: Testing Result Example

As shown in Figure 3.4, the results show that our website successfully passed all tests.

This chapter concludes with all the developer and testing instructions, and we will summarize our project in the following chapters, discussing the development experience and the places that can be improved.

Conclusion

This project is a breakthrough practice of using RPC for front-end and back-end communication. As mentioned earlier, RPC communication has many advantages, but it is currently only used heavily for backend-to-backend communication. gRPC-web is a newly released product, so there is very little practice on the Internet. As a protocol with many strengths, it is possible to replace the RESTful API in the future. Through this project, we have well proved the compatibility, controllability, efficiency and connectivity with the back-end of gRPC in the front-end.

A notable feature for future applications is that they require a lot of manpower and multi-party cooperation, and may require long-term continuous delivery development. Therefore, developments will be towards multi-module, cross-platform, easy-to-deploy and sustainable. So the use of containerization and modularity is indispensable. However current mainstream development approach fails to deliver those characteristics. In this project, we not only separate the front-end and back-end, reverse proxy and database, but also use Docker for cross-platform deployment and rapid deployment, which is implemented as a simple microservice and cloud-native model. Through Docker Compose and Script coordination of multiple containers are two highlights of this project.

Today's applications are becoming more and more complex, and the use of frameworks will greatly speed up the development progress. To adapt the website development processing with this feature, we used SpringBoot as the back-end framework and started from scratch and learned to use Vue and Vue CLI. In addition, the version iteration is very fast and the practice of many npm packages on the Internet has not been updated, so we have to read a lot of official documents and practice this project's architecture and construction methods. Therefore, this project is completely built using various frameworks, which we have thus greatly improved the efficiency of development, moreover, the security, maintainability and effective testing of the project have also been guaranteed.

“Vuetify” combined Vue to provide UI for the front-end is what we came up with after browsing a lot of related projects. This UI is highly integrated with Vue and provides a well-designed interface. Since every website is different, it is difficult to refer to others’ ready-made implementation on the internet. Therefore, we flipped through and read the documentation and tested it step by step, finally rendering my own web page.

The most difficult part of this project is the compatibility and configuration issues. Each fully modular containerized project uses different dependencies and components, and most of the time, the compatibility issues of dependencies can only be solved by reading the various official documents and trying again and again. We have solved many compatibility issues such as RPC SprintBoot kernel and grpc-testing, junit and maven, gRPC-web and Vue, protobuf and Spring Boot, npm package version and Vue CLI version, etc. The writing of configuration files is also a challenging point because many places need to be configured separately according to this specific project. For example, the configuration of Nginx reverse proxy is completed through a large amount of tests.

Considering the relative complexity of the project, we use many external dependencies to solve testing problems, and the test ideas are also summed up in repeated attempts.

To summarize, this project is a complete, instantly deployable containerized web application with front-end and back-end separation, with a simple microservice model and a practice of RPC for front-end and back-end communication. By reading numerous official documentations, we successfully combined a large number of external dependencies. Designing and building each part of the project independently improves my development ideas and debugging capabilities of the software.

Future Work

Although we have simplified the deployment and migration steps of the project, we cannot allow users to deploy from the source code level with one click. The advantage of deploying from source code is that users can continuously modify the code according to their needs without manually packaging the front-end and back-end projects each time, which will realize continuous integration in the true sense. The solution to this problem is that we can develop scripts that are deployed from source code to the remote server, but the workload is not small. We need to write multiple versions according to the system and automatically check and complete the system environment that the user lacks.

Also, due to the time limit, there is no traffic monitoring and reliable security protocols for our prototype website. Exposing the HTTP interface of the OSS server directly to the public network may increase unnecessary costs, and the backend gRPC using plaintext transmission will be vulnerable to flooding attacks. Therefore, traffic monitoring is very useful, and it will be added in the future.

Moreover, the project can only be deployed by one user at present, which is because the OSS service is a bucket maintained by a private account. Every deployment is based on this bucket. The way to improve it is to maintain a different bucket when deploying for each user by using a private OSS account or using the user's own OSS when deploying.

Bibliography

[1] Jeremy H. “gRPC vs REST: How Does gRPC Compare with Traditional REST APIs?” In: 2022. URL: https://blog.dreamfactory.com/grpc-vs-rest-how-does-grpc-compare-with-traditional-rest-apis/.

[2] Wireframes are drawn online using the mockflow website. URL: https://mockflow.com/.

[3] HTML Wikipedia. URL: https://en.wikipedia.org/wiki/HTML.

[4] “What is Vue?” In: chap. Introduction. URL: https://vuejs.org/guide/introduction.html.

[5] Vue CLI Introduction In: Chap. Overview. URL: https://cli.vuejs.org.

[6] “Why vuetify?” In: chap. Introduction. URL: https://vuetifyjs.com/en/introduction/why-vuetify/#why-vuetify3f.

[7] Spring Boot introduction. URL: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm.

[8] “What is a Database?” In: chap. MySQL - Introduction. URL: https://www.tutorialspoint.com/mysql/mysql-introduction.htm.

[9] “What is OSS?” In: chap. Product Introduction. URL: https://www.alibabacloud.com/help/en/object-storage-service/latest/what-is-oss.

[10] DevopsCurry. “What is Nginx? Understanding the basics of Nginx in 2021 !” In: 2021. URL: https://medium.com/devopscurry/what-is-nginx-understanding-the-basics-of-nginx-in-2021-f8ee0f3d3d54.

[11] Hein Meling. grpc-web project on github. URL: \url{https://github.com/grpc/grpc-web}.

[12] Project address for yidongnan grpc-spring-boot-starter on github. URL: https://github.com/yidongnan/grpc-spring-boot-starter.

[13] Docker overview. URL: https://docs.docker.com/get-started/overview/.

[14] Overview of Docker Compose. URL: https://docs.docker.com/compose/.

[15] Joshua Humphries. grpcurl project on github. URL: \url{https://github.com/fullstorydev/grpcurl}.

[16] Jeremy H. “7 Key Benefits of Microservices”. In: 2020. URL: https://blog.dreamfactory.com/7-key-benefits-of-microservices/.

[17] Mybatis official documentation. URL: https://mybatis.org/mybatis-3/.

[18] Maven Protocol Buffers Plugin official documentation. URL: https://www.xolstice.org/protobuf-maven-plugin/.

[19] baeldung. “Spring with Maven BOM”. In: 2020. URL: https://www.baeldung.com/spring-maven-bom.

[20] Apache Commons Codec official website. URL: https://commons.apache.org/proper/commons-codec/.

[21] Vuex official documentation. URL: https://vuex.vuejs.org/.

[22] Project wait-for-it script address on github. URL: https://github.com/vishnubob/wait-for-it.

[23] H2 Database Tutorial. URL: https://www.tutorialspoint.com/h2_database/h2_database_introduction.htm.

[24] Maven dependency grpc-testing. URL: https://mvnrepository.com/artifact/io.grpc/grpc-testing.

[25] baeldung. Introduction to WireMock. 2020. URL: https://www.baeldung.com/introduction-to-wiremock.

[26] Testing instruction for yidongnan gRPC. URL: https://yidongnan.github.io/grpc-spring-boot-starter/en/server/testing#integration-tests.

List of Figures

List of Tables

List of Codes