International Journal of Multidisciplinary Research and Publications
ISSN (Online): 2581-6187

Microservices-based Dating Platform

Jian Xiang
School of Information and Electronic Engineering, Zhejiang University of Science and Technology, HangZhou, 310023, China
Email address: freenyspi(at)gmail.com

Abstract— *We designed a microservices-based front-end and back-end separated dating platform, which provides a series of services such as dating services and recruitment services for programmers. The platform is divided into three main parts: back-end microservices, website front page, and website management backend.*

Keywords— *Front and back-end separation, microservices, dating platform.*

I. INTRODUCTION

With the continuous development of Internet development technology, front-end and back-end separation has become the industry standard usage for Internet project development [7]. The use of front- and back-end separation allows for maximum decoupling of front- and back-end code. Front-end developers can focus on compatibility, user experience, page performance, etc. Back-end developers can focus on high concurrency, high availability, business, security, etc. With the technical support of front-end and back-end separation, the front-end and back-end can be developed in parallel, which improves the development efficiency to a certain extent and increases the maintainability and readability of the code. Through the corresponding front-end technology and back-end technology, the performance of the server can be greatly improved.

The concept of microservices emerged in 2012 as a way to accelerate the development process of Web and mobile applications. Its main role is to decompose functions into discrete services, thus reducing system coupling and providing more flexible service support. The design of microservices-based architecture can effectively split applications, achieve agile development and deployment, and facilitate distributed management and a high degree of fault tolerance to enable rapid evolution and iteration.

With the continuous development of the Internet, people are communicating more and more frequently online, and there are many dating platforms such as Lily.com and other dating platforms, as well as recruitment platforms such as Boss Direct.

Complete a microservices architecture based on the front-end and back-end separation of the dating platform, the dating platform for the majority of programmers to provide dating services as well as recruitment services and a range of services. The system as a whole is divided into three major parts: back-end microservices, website front page, and website management backend. The functional modules include articles, Q&A, tweeting, dating, recruitment, search center and third-party login, etc.

II. RESEARCH CONTENT

Basic content of the research: overall architecture design of the dating platform, database design of the dating platform, front-end page design of the dating platform, back-end microservice design of the dating platform, various business designs of the dating platform

The main problem to be solved: to provide a dating platform for programmers, where users can ask questions, tweet, make friends and recruit.

In the application of microservice architecture, docker container technology plays a crucial role. docker is an open source engine that makes it easy to create a lightweight, portable and self-sufficient container for any application [3]. the main uses of docker, currently, are in three main categories.

(1) Provide a one-time environment. For example, local testing of other people's software, continuous integration when providing unit testing and build environment.

(2) Provide elastic cloud services. Because Docker containers can be turned on and off, they are ideal for dynamic scaling up and down.

(3) Forming a microservice architecture. With multiple containers, one machine can run multiple services, so a microservice architecture can be simulated locally.

III. DATABASE DESIGN

Since the backend of Rose dating platform adopts a microservice architecture, its database modeling cannot be modeled by the traditional E-R diagram. In traditional software development, the data model is considered the core of the whole system, and the business logic is just a CRUD of the data plus a simple computational presentation. Some project teams' architecture reviews spend a lot of time discussing the huge ER diagram (Entity-Relationship Diagram) design of the system [4]. However, ER diagrams are not the best choice when designing microservice architectures, especially when modeling with ER diagrams for service partitioning can lead to different business logics with similar entities being coupled together at design time.

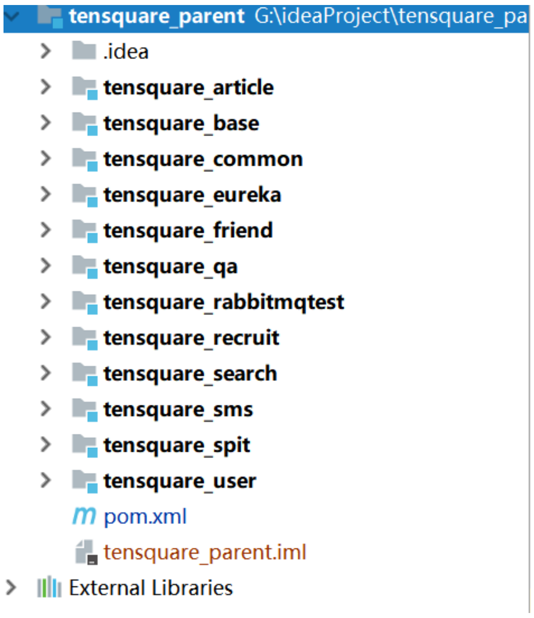
Therefore I used a microservice corresponding to a database. This ensures that the services are loosely coupled and that changes to the database of one service do not affect other services. Each service can use a more suitable database type. For example, a service that does full-text search can use ElasticSearch.

A total of 7 libraries are designed: tensquare_article article, tensquare_base base

tensquare_friend Dating, tensquare_gathering Events, tensquare_qa Q&A, tensquare_recruit Recruitment, tensquare_user User.

IV. Back-end Microservices Implementation

The entire project directory of back-end microservices is shown in Figure 1 below.



```
tensquare_parent G:\ideaProject\tensquare_pa
  > .idea
  > tensquare_article
  > tensquare_base
  > tensquare_common
  > tensquare_eureka
  > tensquare_friend
  > tensquare_qa
  > tensquare_rabbitmqtest
  > tensquare_recruit
  > tensquare_search
  > tensquare_sms
  > tensquare_spit
  > tensquare_user
    pom.xml
    tensquare_parent.iml
> External Libraries
```

Fig. 1. Back-end project directory structure

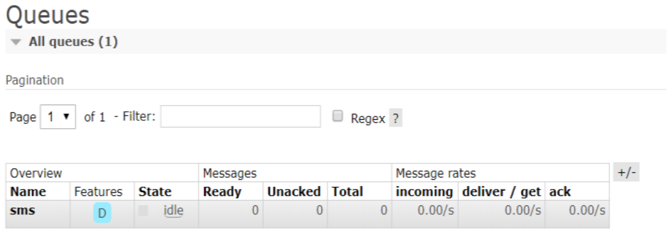
Tensquare_parent is the overall parent project, which involves 10 subprojects, each of which represents a microservice. Among them, tensquare_common is the public sub-module, which contains some public entity classes and some tool classes. Next, a few key modules are selected for a brief introduction.

Implementation idea: write API (receive mailbox address) in user microservice, generate verification code, store verification code in Redis database (for registration verification) and send verification code to specified queue in RabbitMQ message queue with mailbox in a map. In the send mail microservice, it listens to the specified queue in RabbiMQ, resolves the mailbox address and the verification code, and sends the verification code to the user's registered mailbox through the mailbox tool class written according to javax.mail.

In the requirement of user registration and sending CAPTCHA, I used RabbitMQ technology to achieve decoupling and asynchronous operation of two microservice modules, user registration and sending CAPTCHA, and then introduce this message queue middleware. Message queue middleware is an important component in distributed systems, mainly to solve the problems of application coupling, asynchronous messages, traffic clipping, etc. to achieve high performance, high availability, scalability and eventual consistency [architecture] .

RabbitMQ, which I use, is an open source implementation of AMQP developed in Erlang, originally originated in financial systems for storing and forwarding messages in distributed systems, and is notable for its ease of use, scalability, and high availability.

The queue (sms) created in the Rose dating platform for sending CAPTCHAs is shown in Figure 2.



Queues

All queues (1)

Pagination

Page 1 of 1 - Filter: Regex ?

Overview			Messages			Message rates		
Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
sms	D	idle	0	0	0	0.00/s	0.00/s	0.00/s

Fig. 2. SMS queue diagram

V. Frontend Page Implementation

Server Side Rendering also known as SSR (Server Side Render) is to complete the content of the page on the server side instead of fetching the data on the client side through AJAX. The main advantage of Server Side Rendering (SSR) over traditional SPA (Single-Page Application) is: better SEO, as search engine crawlers can directly view the fully rendered page.

As of today, Google and Bing do a good job of indexing synchronized JavaScript applications. Here, synchronization is key. If your application initially displays a loading daisy-chart and then fetches the content via Ajax, the crawler tool will not wait for the asynchronous completion before crawling the page content. That said, if SEO is critical to your site and your pages are fetching content asynchronously, you may need server-side rendering (SSR) to solve this problem. Faster time-to-content, especially for slow network conditions or slow-running devices. No need to wait for all JavaScript to finish downloading and executing before displaying server-rendered markup, so your users will see the fully rendered page more quickly. This generally results in a better user experience and is critical for applications where "time-to-content is directly related to conversion".

VI. Code Testing

In order to check if the system is working properly and if all functional modules can be implemented, we must test the target system and measure the value of the system software application by the actual test results. System testing is the routine checking of the target application in a specific operating environment to see if the target software can meet the user's requirements. In designing and developing system software, there is no absolute way to eliminate errors. Even with the best technology, errors can occur. How do we minimize the occurrence of errors? This is done through testing. In each phase of system software design and development, it can be said that testing is quite necessary. The testing effort is generally a large part of the total system effort, perhaps half or more, and the cost of testing can be more than half of the total system development cost. The reason testing takes a lot of time and cost is to ensure the quality of the software development. At the same time, you can see that the system program is becoming more complete and better. In the process of writing the Rose dating platform, I used easyMock for data mockup testing on the front end and unit testing and postman for testing the correctness of the interface on the back end because of the separate architecture of front and back end code.

VII. CONCLUSION

This dating platform can be said to be quite comprehensive, users can meet their various needs through different functional modules, users can browse and share articles, find jobs, or conduct some dating activities on the platform. Due to the many businesses of the dating platform, many aspects of the design still have flaws, as well as some features have not been fully implemented, Rose dating platform still needs to be perfected and improved, some ideas are not mature enough, but I will try my best to create a more perfect work.

REFERENCES

[1] Zhao Ran, Zhu Xiaoyong. Review of microservice architecture[J]. Web New Media Technology,2019,1(8): 2-4.
[2] Martin Fowler James Lewis. microservices a definition of this new architectural term [EB/OL]. https://martinfowler.com/articles/ microservices.html, 2014.
[3] Chen Qingjin,Chen Cunxiang,Zhang Yan.Analysis of Docker technology implementation[J]. Information and Communication Technology,2015,(2).
[4] Cui Yang, He Yaru. Mysql Database Applications from Beginner to Master [M]. Beijing: China Railway Publishing House, 2013.
[5] Wang Yunfei. SpringBoot in action [M]. Beijing:Electronic Industry Press,2016,03.
[6] Huang Jianhong. Redis design and implementation [M]. Beijing: Mechanical Industry Press,2014,04.