Design of WeChat Service System Based on Microservice Architecture

Sikai Wu, Xiao Wang and Quanyin Zhu*
Faculty of Computer & Software Engineering, Huaiyi Institute of Technology, Huaian, China
*Corresponding author’s email: hyitzqy@126.com

Abstract. In order to design a stable WeChat service system which provides one-stop service for Huaiyi Institute of Technology students, a method for building system based on microservice architecture is proposed. Different from monolithic application, application based on microservice consists of a mount of service modules. Users request web service module which invokes via service registry. Service modules communicate with RESTful API interface, a mechanism based on Http protocol. The results show that the proposed WeChat service system can satisfy the requirements for the users.

1. Introduction
The rapid development of the Internet has recently given rise to a great change of the system architecture. Traditional applications, mainly monolithic applications, appear gradually various shortcomings like huge amount of code, difficulty of expansion and singleness of technology stack [1]. How to solve these problems in WeChat service system has become the focus of our efforts. In order to solve these problems, a method of designing application based on microservice is proposed [2].

Reference [3] explains the reason why most of major companies are moving to a new paradigm is that monolithic application makes it difficult for different development teams to coordinate with each other. Reference [4] describes how the properties of microservice architectures facilitate scalability. Reference [5] proposes key architectural components and building blocks for implementing.

Microservice is a paradigm that divides system into independent service modules. Microservice modules expose RESTful API interfaces which can be requested by other modules for communication [6-7]. Service modules communicate with others through JSON, a lightweight data format, to promote loose coupling. Come up in 2014, microservice has developed some skills like Docker, Dubbo and Spring Cloud [8-9].

Based on our past work [10-12], we build an efficient microservice-based WeChat service system. Microservice architecture componentizes and modularizes different parts of system [13-14]. This way can improve maintainability and scalability of the system [15-17]. To provide service to each other, every service must register to service registry. Service registry manages all service modules.

2. System Architecture
The system is divided into three parts: service providers, service consumers and service registries. Service providers provide API for other microservice modules in the system architecture. They can be independently deployed and maintained. Service provider can run multiple service instances to ensure high availability of the system. The service consumer obtains the data provided by the service provider through the HTTP request and presents it to users. The registries accept sign-ups from service providers
and service consumers and update the state of the microservices through the heartbeat mechanism. Figure 1 shows the full system architecture.

![Diagram of system architecture](image)

**Figure 1. System architecture**

2.1. *Presentation Layer*

Presentation layer consists of some web pages. This layer runs on the WeChat client and is rendered to the user through the Thymeleaft templating engine. This layer interacts with user information through the WeChat OpenID and Token Access mechanisms to render the corresponding page to the specified user.

2.2. *Service Layer*

Service layer includes WeChat server, service registries, service providers and service consumers. The WeChat server receives a request from a WeChat client to obtain a corresponding OpenID and initiates a request to the service consumer. The service consumer obtains the service provided by the service provider through the service registries, sends the request to the service provider and obtains the data through the open API and returns it to the user.

2.3. *Data Layer*

The data layer includes a number of database management systems (DBMS), such as MySQL, Oracle, and SQL Server. MySQL mainly stores the users' personal information, log information and function module information. Oracle mainly stores students' grades and timetable information. SQL Server mainly stores data on morning exercises.

3. *Service Registry*

The registry is essentially decoupled from service providers and service consumers. For any microservice, in principle, there should be multiple providers, which are determined by the distributed property of microservices. Still further, the number and distribution of providers of microservices tend to be dynamic and unpredictable in order to support elastic scaling capabilities. Therefore, the static
load balancing mechanism, which was originally used in the monolithic application, is no longer applicable. It is necessary to introduce additional components to manage the registration and discovery of the microservice providers, and this component is the service registry. Figure 2 shows the service registry and the registered microservices.

![Figure 2. Service registry](image)

Each service instance registers its own information to the registry, usually containing information such as addresses, ports, protocols, versions, and so on. Multiple instances of each service are registered to the registry, and the registry maintains a list of multiple instances of each service. At the same time, the registry will check the availability of each service instance by some mechanism, and if an instance has failed, it will be eliminated. A service instance is automatically written off to the registry when a service instance is closed.

4. Service Provider and Service Consumer
Each microservice has multiple instances, and each microservice instance is isolated in a single Docker container individually. The service provider submits the request to the service registry. The registry checks the health status of the service and forwarding the request to the healthy instance to achieve load balancing. The scheme solves the single point problem of the server load, and each client implements its own load function. If the load balancing process is down, the client is also down, but only affects itself, does not affect other client. Figure 3 shows this schema.

![Figure 3. Client load balancing scheme](image)
The above content is based on the implementation of the load balancing on the client side, and the client load balancing has the following characteristics:

It is implemented by the client internal program without the additional load balancer’s hardware and software.

The program needs to solve the problem of the overload of the business server.

Server failures are less transparent to applications.

The system ensures high availability and flexibility of services through the health mechanism of microservices. When the service is started, it registers and sends a heartbeat request to the service registry for the first time, and detects whether the service is down through a heartbeat mechanism. The default interval of a heartbeat connection for a microservice is 30 seconds, and if the service connection fails over three times, the service center will be eliminated, and the other application will not be invoked.

5. Process of Wechat Service System
With the rapid development of mobile Internet, we have developed a service system based on the WeChat platform in order to provide users with efficient and cross-platform services. The system takes advantage of the features of WeChat's cross-platform and deploys the entrance of the system on the WeChat Public Platform. Users can use the WeChat Official Account and use the system from the entrance of the WeChat Official Account. Figure 4 shows a flow chart of the system.

![Service registry diagram](image-url)
As the figure 4 shows, the whole process can be divided into the following steps:

1) Enter the main interface. Users should subscribe WeChat Official Account to get the entrance of the system. When users enter the main interface, they can see some functions. Go to Step 2.
2) Choose function. User should choose the function they want. Go to Step 3.
3) Check whether user has bound. If user has bound, go to Step 7, otherwise go to Step 4.
4) Fill in the form. User need to fill in the account and password. Go to Step 5.
5) Request account and password to user information microservice server. Go to Step 6.
6) Check whether user exists. If user exists, go to Step 7, otherwise go to Step 4.
7) Request to the specific microservice server. If the user needs to query curriculum, go to Step 8. If the user needs to query score, go to Step 9. If the user needs to set up the CET score reminding, go to Step 11. If the user needs to query attendance, go to Step 13.
8) Display curriculum. Go to Step 14.
9) Choose term. User needs to choose the term of the score you want to query. Go to Step 10.
10) Display the score of specific term. Go to Step 14.
11) Fill in the information of CET. Go to Step 12.
12) Display the information of CET. When the result comes out, the system sends messages to the user. Go to Step 14.
13) Display record of attendance. Go to Step 14.
14) Check whether user uses other function. If user uses other function, go to Step 1, otherwise go to Step 15.
15) Complete system process.

Figure 5 is the main interface mentioned in Step 1. There are multiple functional modules on the main interface, and different functions correspond to different microservices. The corresponding module can be selected to enter the corresponding function.
Selecting the attendance checking on the main interface can jump to the interface in figure 6. Users can check the number of attendance and the detail of attendance through the function. This function is to facilitate the user to understand their own attendance situation so as to complete the requirements of the attendance in time.

![Attendance interface](image)

**Figure 6. Attendance interface**

The other functions are similar to the attendance checking, if the system has bound the user’s account, it does not need to enter the binding interface again. The corresponding module can be selected according to the function required by the users.

6. **Conclusions**

The microservice architecture and the development of WeChat provide a convenient technical support for this system. We build a service system for students based on microservice and WeChat, and realize the integrated operation of multiple platforms. At the same time, we can lay a solid foundation for the better development of the system. Through the analysis of this paper, it can be seen that the application of microservice in the construction of information is practical, and has a great prospects for development. Therefore, in the process of information construction, we need to strengthen the system design while we use the microservice architecture, only in this way can we make the microservice structure play the biggest role in the information system construction.

7. **Acknowledgment**

The work described in this paper was supported by National Undergraduate Innovation and entrepreneurship training program (201711049017Y). We would like to express our thanks to the anonymous reviewer for their constructive comments.
8. References

[1] M. Villamizar et al., "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," 2015 10th Computing Colombian Conference (10CCC), Bogota, 2015, pp. 583-590. doi: 10.1109/Columbian CC. 2015.7333476

[2] C. Y. Fan and S. P. Ma, "Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report," 2017 IEEE International Conference on AI & Mobile Services (AIMS), Honolulu, HI, 2017, pp. 109-112. doi: 10.1109/AIMS.2017.23

[3] V. Singh and S. K. Peddouj, "Container-based microservice architecture for cloud applications," 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, 2017, pp. 847-852. doi: 10.1109/CCAA.2017.8229914

[4] W. Hasselbring and G. Steinacker, "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce," 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, 2017, pp. 243-246. doi: 10.1109/ICSASAW.2017.11

[5] Yale Yu, H. Silveira and M. Sundaram, "A microservice based reference architecture model in the context of enterprise architecture," 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Xi'an, 2016, pp. 1856-1860. doi: 10.1109/IMCEC.2016.7867539

[6] A. Balalaie, A. Heydarnoori and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," in IEEE Software, vol. 33, no. 3, pp. 42-52, May-June 2016. doi: 10.1109/MS.2016.64

[7] D. Guo, W. Wang, G. Zeng and Z. Wei, "Microservices Architecture Based Cloudware Deployment Platform for Service Computing," 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), Oxford, 2016, pp. 358-363. doi: 10.1109/SOSE.2016.22

[8] H. T. Ciptaningtyas, B. J. Santoso and M. F. Razi, "Resource elasticity controller for Docker-based web applications," 2017 11th International Conference on Information & Communication Technology and System (ICTS), Surabaya, 2017, pp. 193-196. doi: 10.1109/ICTS.2017.8265669

[9] Y. Jin-Gang, Z. Ya-Rong, Y. Bo and L. Shu, "Research and Application of Auto-Scaling Unified Communication Server Based on Docker," 2017 10th International Conference on Intelligent Computation Technology and Automation (ICICTA), Changsha, 2017, pp. 152-156. doi: 10.1109/ICICTA.2017.41

[10] K. Xu, S. Yu, J. Ding, R. Hu and Q. Zhu, "Design and Implementation of Check-In System Based on Workflow Technology," 2017 International Conference on Smart Grid and Electrical Automation (ICSGEA), Changsha, 2017, pp. 60-63. doi: 10.1109/ICSGEA.2017.69

[11] M. Yang, T. Liu, X. Wang, Y. Yan, R. Hu and Q. Zhu, "Design of WebGIS System Based on Javascript and ArcGIS Server," 2017 International Conference on Smart Grid and Electrical Automation (ICSGEA), Changsha, 2017, pp. 709-712. doi: 10.1109/ICSGEA.2017.68

[12] Q. Zhu and P. Zhou, "The System Architecture for the Basic Information of Science and Technology Experts Based on Distributed Storage and Web Mining," 2012 International Conference on Computer Science and Service System, Nanjing, 2012, pp. 527-530. doi: 10.1109/CSSSS.2012.138

[13] D. Richter, M. Konrad, K. Utecht and A. Polze, "Highly-Available Applications on Unreliable Infrastructure: Microservice Architectures in Practice," 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Prague, 2017, pp. 130-137. doi: 10.1109/QRS-C.2017.28

[14] T. Soenen, W. Tavernier, D. Colle and M. Pickavet, "Optimising microservice-based reliable NFV management & orchestration architectures," 2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM), Alghero, 2017, pp. 1-7. doi: 10.1109/RNDM.2017.8093034

[15] B. Butzin, F. Golatowski and D. Timmermann, "Microservices approach for the internet of things," 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, 2016, pp. 1-6. doi: 10.1109/ETFA.2016.7733707
[16] M. Ahmadvand and A. Ibrahim, "Requirements Reconciliation for Scalable and Secure Microservice (De)composition," 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW), Beijing, 2016, pp. 68-73. doi: 10.1109/REW.2016.026

[17] H. Kang, M. Le and S. Tao, "Container and Microservice Driven Design for Cloud Infrastructure DevOps," 2016 IEEE International Conference on Cloud Engineering (IC2E), Berlin, 2016, pp. 202-211. doi: 10.1109/IC2E.2016.26