IOP Conference Series: Materials
Science and Engineering

PAPER • **OPEN ACCESS**

Research and Application of Micro Frontends

To cite this article: Caifang Yang *et al* 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* **490** 062082

View the article online for updates and enhancements.

You may also like

- Designing Online Healthcare Using DDD in Microservices Architecture
M Rizki, A N Fajar and A Retnowardhani
- Design of decision support system service in the Space Science Center using microservices approach
Elyyani, Y. Andrian, A.Z. Utama et al.
- Design of Information System Architecture of Garment Enterprises Based on Microservices
Weilun Tang, Li Wang and Guangtao Xue

This content was downloaded from IP address 85.216.234.43 on 18/03/2024 at 20:18

IOP Conf. Series: Materials Science and Engineering **490** (2019) 062082 doi:10.1088/1757-899X/490/6/062082

Research and Application of Micro Frontends

Caifang Yang[1], Chuanchang Liu[1,*] and Zhiyuan Su[2]

[1]State Key laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, Beijing, China
[2]School of Automation,
Beijing University of Posts and Telecommunications, Beijing, China

*Corresponding author e-mail: 295459877@qq.com

Abstract. The content management system is an indispensable part of enterprise information construction. With the rise of the concept of front and rear separation and the maturity of related technologies, it is a better choice to create a multi-functional front-end web application in complex large-sized or medium-sized projects, which sits on top of a microservices-based architecture. As the business grows, the single front end becomes more and more bloated, resulting in a single page application that is not well scaled and deployed, and difficult to maintain. Based on the analysis of above problems, combined with the corresponding front-end technology, this paper applies the microservice concept to the development of front-end and proposes a design scheme of content management system based on micro frontends. Besides, the key issues in practice process, such as micro frontends design concept and implementation methods are described in detail.

1. Introduction

With the development of Internet technology, information technology has become an indispensable part of various fields. The content management system(CMS) provides strong support for informatization and becomes an important part of enterprise information construction and e-government. In the past, a large number of traditional application systems have adopted a traditional single-architecture mode due to their small size, simple structure, small user population, and low real-time communication requirements. The so-called single architecture means that the functional modules and operational data of the entire software system are treated as a whole, and are uniformly set up, developed, packaged, and deployed [1]. With the refinement of requirements, complex systems need to have high real-time performance, high reliability, high scalability, and need to deal with the complexity of business logic as well as the huge amount of data. The single architecture has many shortcomings in these aspects.

Along with the rise of the concept of front and rear separation and the maturity of related technologies, front and rear separation architecture enables system development to achieve decoupling. A single-page application is an implementation of front and rear separation. Based on single-page application technology, a front-end team often creates and maintains a web application that uses the REST API to retrieve data from back-end services. This approach can provide a great user experience, but result in a single page application not being well scaled and deployed. As time goes by and business development, the front end will become more and more bloated, front-end projects will become more and more difficult to maintain, and a single front-end team may become a development

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

bottleneck. Especially when a feature-rich, powerful front-end web application sits on top of the back-end microservices architecture. This situation will intensify in the context of the Serverless architecture. Based on the above analysis, this paper proposes a micro-front-end based content management system design, which has been applied to a practical system.

2. Idea of micro frontends

In recent years, the microservices has received great attention from academia, and has become one of the key research objects in the field of information science.

In traditional single application, the system based on single architecture contains a large number of modules, the module's dependencies are not clear, and the boundaries between the modules are blurred. Moreover, each modification of the project must redeploy the entire project. This full deployment method takes a long time and has a large impact range. As a strongly coupled whole, single application cannot be targeted for the characteristics of different business modules, and can only be expanded as a whole, resulting in waste of resources.

Microservices are a variant of the service-oriented architecture architectural style that builds applications as a collection of loosely coupled services. It combines complex large applications in a modular way based on small functional blocks that communicate through a collection of language-independent APIs. Each functional block focuses on a single responsibility and function, and can be independently developed, tested, and deployed [2]. This makes the application easier to develop in parallel [3]. It also enables continuous delivery and deployment [2].

In the back-end, the micro-service architecture has a relatively mature implementation solution. Combined with the front and rear separation mode, the web development mode has changed.

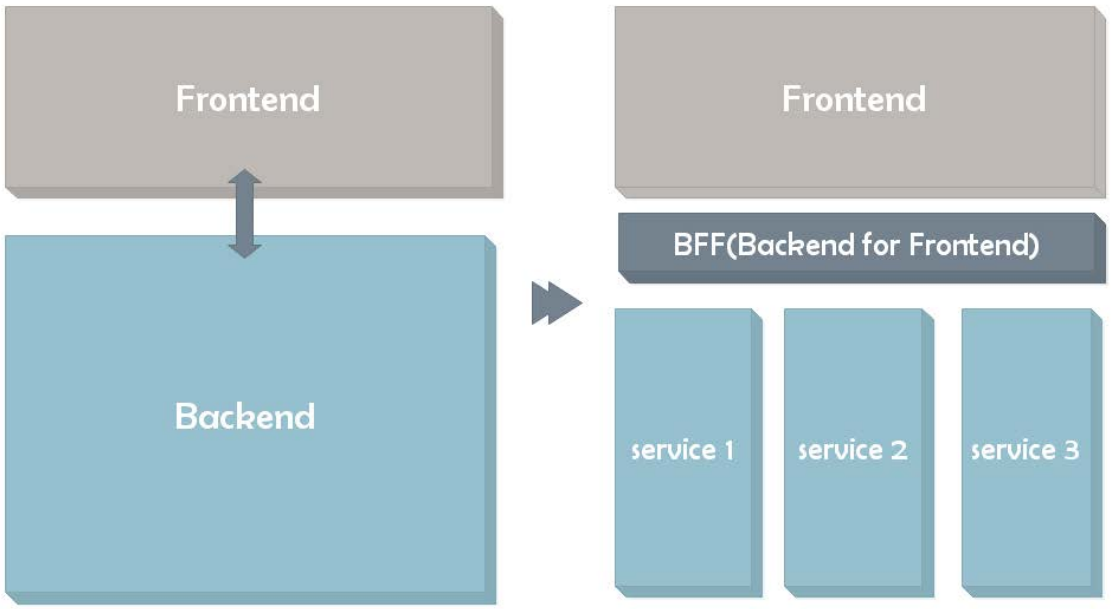

Figure 1. Web development mode.

The benefits of microservices to the backend are obvious, so it's a good idea to apply the microservices concept to the front end to solve the front-end monolith problem. Micro frontends extends the concepts of microservices to the frontend world. It applies the concept of microservices to the browser side, transforming web applications from a single application to an application that combines multiple small front-end applications. Each front-end application can run, developed, and deployed independently.

The idea behind micro frontends is to treat a web application as a combination of features owned by different teams. Each team has an independent business or function that they focus on. A team is cross-functional and develops its features end-to-end, from backend to frontend [4].

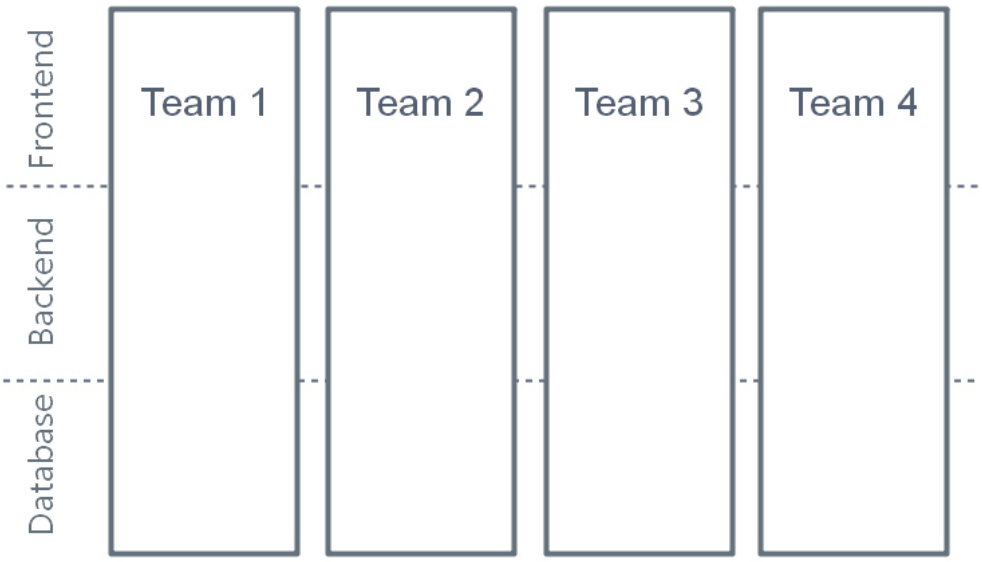

Figure 2. Micro frontends architecture.

The core idea of micro frontends is reflected in the following aspects.

- Each team can complete the project independently by choosing their own technology stack.
- Isolate the team's code to avoid sharing runtime and global variables.
- Use prefixes to identify ownership of variables to avoid conflicts.
- Use browser events whenever possible to communicate.

3. Implementation comparison

After each independent team develops its own App module, the Web application can be thought of as a combination of functions of the various modules, that is, sub-applications. The front end will only be responsible for selecting and deciding which modules to import by the router to provide a consistent user experience for the end user. Three typical technical practices for implementing a micro frontends architecture are shown as follows.

3.1. Route distribution

A route-distributed micro frontends that distribute different services to different, independent front-end applications through routing. It is a very simple and efficient way to slice modules. It can usually be implemented by a reverse proxy of the HTTP server, or by the routing that comes with the application framework. But this approach looks more like a collection of front-end applications, that is, these different front-end applications are grouped together and look like a complete whole. In the process of routing jumps, it is often necessary to refresh the page and there will be a white screen process. In this process, the application before the jump and the application to be jumped lose control of the page. If there is a problem with this application, the user experience is poor.

3.2. <iframe> embedding

The HTML inline frame element <iframe> represents a nested context being browsed, effectively embedding another HTML page into the current page. IFrame can create a completely new, stand-alone hosting environment, which means our front-end applications can run independently of each other [5]. Each sub-application can be embedded in its own <iframe>, which allows each module to use any framework they need without having to coordinate with other teams, but can still use some libraries or Window.postMessageAPI to interact.

The biggest advantage of using <iframe> is that it isolates the runtime environment of the component and the application. So each module can be developed independently and can be independent of other parts of the technology. So system development can take a completely different front-end framework, develop a part in React, develop a part in Angular, and then develop other parts

using native JavaScript or any other technology. As long as each <iframe> comes from the same source, messaging between them is fairly straightforward and powerful. The disadvantage is that the size of the bundle is very obvious, so the same library may eventually be sent multiple times, and since the applications are separate, the public dependencies cannot be extracted at build time.

3.3. Web components technology

Web components is a suite of different technologies consisting of Custom elements, Shadow DOM, HTML templates, and HTML Imports. It allows you to create reusable custom elements with encapsulated functionality and use them in your web applications.

Web components create pieces of front end which can be imported into web applications in a very elegant way. Those pieces can be packaged into microservices together with back end [6]. But it has several shortcomings. Web components are still not fully supported in all browsers. JavaScript bundle has to load first and register the components in order for the DOM to load, which means that to gain the advantages of server-side rendering you'll probably need to be more clever.

4. Implementation and application

Based on the analysis of the advantages and disadvantages of the above implementation schemes, this paper uses the micro frontends framework Mooa to develop applications. Mooa is a micro frontends framework that services for Angular, a micro frontends solution optimized for IE 10 and IFRAME based on Single-SPA. Mooa framework uses the Master-Slave architecture [7]. For a web page, it can have two or more Angular applications at the same time. One of them exists as the main project, which is responsible for loading other applications and the core control functions such as user access control management, and the rest are sub-applications, which is responsible for the specific business code of different modules.

4.1. System architecture

The content management system is split into several modules by function. Firstly, multiple independent angular-based front-end projects are created as sub-applications, each of which is responsible for different business modules. Then, a project based on the mooa framework is created as the main project, responsible for loading other applications, and core control functions such as user access control management. When the main project is running, it will go to the server to get the latest application configuration. After the main project gets the configuration, it will create the application one by one and bind the lifecycle to the application. When the main project detects the route change, it will find out if there is a corresponding route matching to the application. When matching to the corresponding application, the corresponding application is loaded. When the main project changes the secondary route of the sub-application, the sub-application is notified by the event, and the sub-application also needs to monitor whether it is the route of the current application. The system architecture diagram is shown in Figure 3.

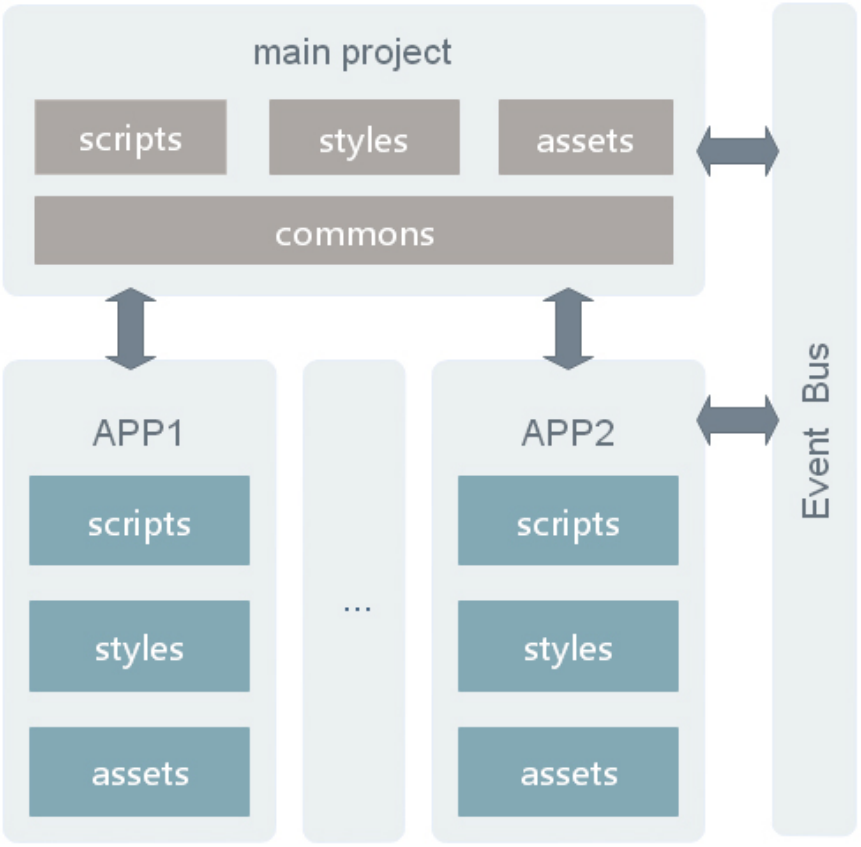

Figure 3. System architecture diagram.

4.2. Route registration and deployment

After adding the mooa dependency in the main project, add a configuration file apps.json in the assets directory, and register the relevant subproject names in the file. Next, read the configuration file in the app.component.ts file using the registerApplication method for application registration. Create a route for the sub-application in the app-routing.module.ts file. Since the main project also needs to be responsible for the user access control management function, the routing configuration of each sub-application needs to be loaded into the navbar component of the left navigation bar.

Each sub-application is built when the project needs to be merged. Copy the files in the built dist directory to the src/assets/ directory of the main project. Each sub-application has a separate name. Every time you deploy, we only need to point apps.json to the latest configuration file. Then start the main project to get the configuration.

5. Conclusion

The micro-front-end-based content management system design enables teams to develop independently, quickly deploy and test individually, helping with continuous integration, continuous deployment, and continuous delivery. To some extent solved the front-end monolith problem. However, the micro frontends also brings some shortcomings. First of all, this paper based on the mooa framework system can not implement a variety of different technology stack development. Secondly, the integration of multiple sub-projects becomes complicated. It is necessary to consider segregating JS, avoiding CSS conflicts, and considering loading resources on demand. The dependency redundancy between sub-projects after integration increases the complexity of management. At present, micro frontends is still in the exploration stage, and it is not mature enough. It is necessary to continue to explore more mature practical applications.

Acknowledgments
The work was supported by the National Natural Science Foundation of China (Grant No.U1536112) and Special Funds for Public Industry Research Projects of State Administration of Grain of China (Grant No.201513002).

References
[1] Dragoni N, Giallorenzo S, Lafuente A L, et al. Microservices: yesterday, today, and tomorrow[M].Present and Ulterior Software Engineering. Springer, Cham, 2017: 195-216.
[2] Chen, Lianping. "Microservices: Architecting for Continuous Delivery and DevOps." IEEE International Conference on Software Architecture (ICSA). 2018.
[3] Richardson, Chris. "Microservice architecture pattern". microservices.io. Retrieved 2017-03-19.
[4] Micro Frontends on https://micro-frontends.org
[5] Extend the microservices concept to the front end on http://insights.thoughtworkers.org/micro-frontends-2
[6] Including Front-End Web Components Into Microservices on https://technologyconversations.com/2015/08/09/including-front-end-web-components-into-microservices
[7] A micro-frontend Framework for Angular from single-spa on https://github.com/phodal/mooa