Jurnal Galaksi (Global Knowledge, Artificial Inteligent and Information System) E-ISSN: 3048-2399
Vol 1, No 2, August 2024, pp. 92-100
https://doi.org/10.70103/galaksi.v1i2.19

Analysis and Comparison of Micro Frontend and Monolithic Architecture for Web Applications

Dwiky Chandra Hidayat[1*], Ketut Jaya Atmaja[2], Ida Bagus Gde Sarasvananda[3]

[1*,2,3] *Informatika, Institut Bisnis dan Teknologi Indonesia, Denpasar, Indonesia*
[1*] *dwikychandra21@gmail.com* , [2]*ketutjayaatmaja@instiki.ac.id* , [3]*sarasvananda@instiki.ac.id*
**Corresponding Author*

ARTICLE INFO

Article history:
Received 11 August 2024
Revised 30 August 2024
Accepted 30 August 2024
Available Online 31 August 2024

Keywords:
Micro Frontend;
Module Federation Plugin;
Monolithic Architecture;
Run Time Integration

ABSTRACT

A robust system to support the learning aspect is a major concern at PT Bina Taruna Wiratama, a technology-based education company. To achieve this goal, the technology team is required to be able to develop the system quickly and with minimal bugs. Various aspects affect the programmer's ability to produce a high-quality system that is easy to develop further. One of the key aspects is system architecture, which serves as the initial foundation of a system. Currently, the frontend application is still developed as one large application, which causes the build time to be quite long, which is around 30-50 minutes. To overcome this problem, the author explores techniques that can be applied in frontend application development, one of which is the use of micro frontend architecture with run-time integration techniques. This study aims to evaluate how the implementation of micro frontend architecture can be applied to the Smart BTW application and to understand the advantages and disadvantages of the architecture. The implementation results show that the sub-project application can use different JavaScript framework versions, with a faster build time, on average 1-2 minutes, although the initial load is slightly longer, which is 2-3 seconds compared to the monolithic architecture.

1. Introduction

In building a system, there are many aspects that need to be considered in order to create a system that not only can be used and runs well, but must also be easy for future development. Building a system certainly involves a development team and a strategic business management team, therefore the challenges faced in building a system are very varied, depending on the scope of the system you want to build, the processing time, and the process flow within it (Musiolik et al., 2020).

The development team, especially programmers, in the process of building a system are often in a dilemma regarding the technology and architecture to be used because there are many considerations that need to be taken, one of which is the relatively short time limit for developing the system. The most common system architecture applied in the initial stages of system development is a monolithic architecture. Monolithic architecture is an architecture that describes an application that runs all the logic on one application server (Mufrizal & Indarti, 2019; Saransig & Tapia, 2019). And in the process of developing a system, the solution of monolithic architecture which has also been widely applied to modern applications is to divide the application into two different projects, namely the frontend to

manage the logic related to the interface display and the backend to manage all process logic from the server side. And the architecture that has been widely implemented is the microservice architecture, especially in backend applications. Meanwhile, in the frontend application, even though it stands alone and is separate from the backend application, as development continues to occur in the frontend application, it can cause the program code to enlarge and similar problems will be encountered again in the application monolith. Several companies such as Amazon, Facebook, Zalando, SAP, Dazn, AirBNB, HelloFresh, Allegra, Klarna, and others have found and implemented unique solutions to address their company's specific frontend problems. Most of them use a Server-Side Rendering (SSR) approach that requires multiple servers and a framework to manage component composition on the server side. The technique is also known as Micro Frontend architecture (Bühler et al., 2022). Micro frontends are not a technology. They are alternative organizational and architectural approaches. The most significant difference between micro frontend and other architectures is the team structure.

There are several related studies that discuss micro frontend architecture. The ERP application that is used as the object of research still uses a monolith architecture (Blinowski et al., 2022). The author of this research considers that applications with a monolith architecture are not isolated so they can cause several problems from both the technical and business sides. The results of this research include micro frontend architecture that can be implemented in ERP applications and make it easier for companies to increase their development team and make it easier for the team to be more productive in the future development process. The application of micro frontend architecture with the use of different frameworks to find out whether micro frontend can be applied in a consultancy environment and how micro frontend can help companies reduce pressure in frontend application development and minimize development costs (Li et al., 2020; Wanjala, 2022). The micro frontend implementation method in this research uses Web Components and the LitElement library. For the frontend framework used is React.js and Angular. The results of this research are that the implementation of micro frontends in the consultancy environment has been successful, but what is still lacking from this research is determining whether the implementation of micro frontends is able to reduce pressure in developing frontend applications and minimize development costs (Benavente et al., 2022).

Based on the explanation above, the precise aim of this research is to design and implement a micro frontend architecture on the Smart BTW monolith web application. Micro frontends are intended to bring the benefits of microservices to the user interface. With this micro frontend concept, large projects can be broken down into smaller sub-projects, making it easier to develop and maintain the system. Apart from that, the micro frontend also carries the concept of an application that can stand independently and not affect other projects. The research contribution is that applications built with micro frontend architecture can run and be well integrated from the development process to the application deployment process to the production environment.

2. Literature Review

The architectural landscape for web applications has evolved significantly, with two prominent paradigms emerging: monolithic architecture and micro frontend architecture. Each of these approaches presents distinct advantages and challenges that influence their adoption in modern web development.

Monolithic architecture has traditionally been the standard approach for web applications, where all components are tightly integrated into a single codebase. This model simplifies deployment and can lead to better performance in certain scenarios, particularly in terms of throughput under concurrency testing, where monolithic systems have shown a 6% advantage over microservices in some cases (Peltonen et al., 2020). However, as applications grow in complexity, monolithic architectures can become cumbersome and inflexible, leading to bottlenecks in development and deployment processes (Emmanni, 2024). The tightly coupled nature of monolithic systems makes it difficult to scale individual components independently, which can hinder innovation and responsiveness to market changes (Nunes et al., 2019).

In contrast, micro frontend architecture offers a more modular approach, allowing teams to develop and deploy independent frontend applications that can be integrated into a larger system.

This architectural style draws inspiration from microservices, promoting flexibility and scalability by enabling teams to work on different parts of an application concurrently without interfering with one another (Taibi & Mezzalira, 2022). Research indicates that applications built using micro frontend architecture can achieve higher performance scores compared to their monolithic counterparts, with one study reporting a score of 99 for a micro frontend application versus 86 for a monolithic application (Wanjala, 2021). This modularity not only enhances performance but also facilitates the adoption of diverse technologies and frameworks across different teams, thus fostering innovation (Peltonen et al., 2020).

Despite the advantages of micro frontends, challenges remain. Integrating technology-independent applications, managing data communication between micro applications, and ensuring resource management are significant hurdles that developers must navigate (Tilak et al., 2020). Furthermore, the complexity of managing multiple micro applications can lead to difficulties in maintaining a coherent user experience and can increase the overhead associated with deployment and monitoring (Gashi, 2024).

In summary, while monolithic architecture may offer simplicity and better performance in specific scenarios, it often lacks the flexibility required for modern web applications. Micro frontend architecture, on the other hand, provides a scalable and innovative approach that aligns well with contemporary development practices, despite its inherent complexities. The choice between these architectures ultimately depends on the specific needs of the application, the development team's structure, and the desired speed of innovation.

3. Research Methods

Architectural Design Analysis

The architectural design analysis stage is the stage used to analyze the needs in architectural implementation and determine the rules that will be used as a reference in the architectural implementation process. These needs are:

1. Code editor, using Visual Studio Code as a place to write program code.
2. Code repository, using GitHub which is useful for storing and distributing code originating from Visual Studio Code to cloud services (Wurster et al., 2020).
3. Cloud services, using Amazon Web Services (AWS), a cloud computing service that is widely used by large companies today. One service from AWS that is useful for storing data online is Amazon Simple Storage Service (Amazon S3)(Engström et al., 2023).
4. Frontend framework, using react.js which is useful for facilitating the creation of interactive, stateful & reusable UI components (Thabit et al., 2023).
5. Webpack, a module bundler for modern JavaScript applications, facilitates code to be designed, written, and tested in a structured manner, using the latest editions of the ECMAScript (ES) language, such as ES6 (Paltoglou et al., 2021).
6. Styling library, using TailwindCSS for quick customization of interface styling. TailwindCSS does not offer predefined styling components. Instead TailwindCSS provides utility classes that can be customized and combined to create varied styling (Attardi, 2020).
7. Twin.macro, is an open source library that can convert TailwindCSS classes into CSS objects in JavaScript at build time. Twin.macro combines the convenience of TailwindCSS and the benefits of CSS-in-JS.

There are rules for implementing this architecture, namely:

1. Sub-project communication with sub-projects, there must be no relationship between fellow sub-projects.
2. Communication of the main application with sub-projects, the main application should not assume that the sub-project uses a certain framework.
3. CSS scope, the scope of using CSS from one project must not affect other projects.
4. Authentication, centered on the main application.

5. Each sub-project has its own route.
6. Each sub-project has its own automation deployment.

Micro Frontend Architecture Design

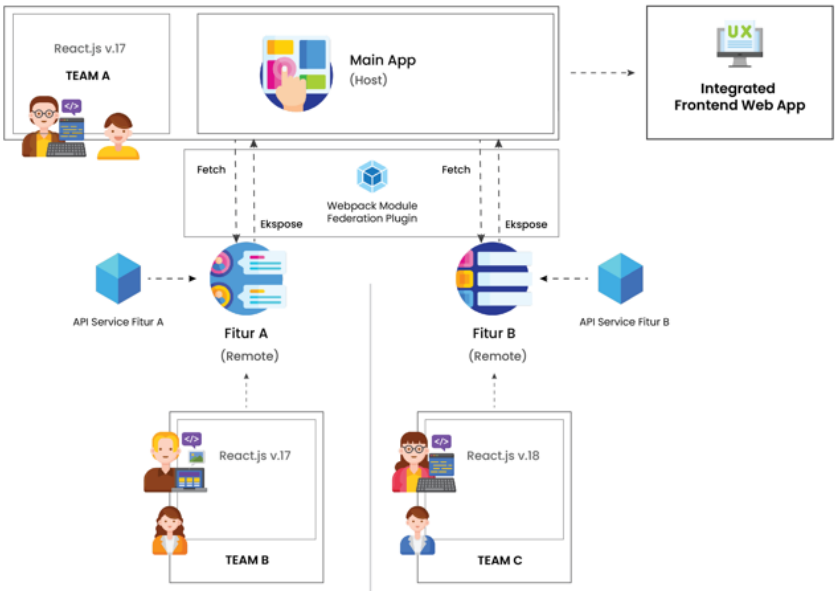

Fig. 1. Micro Frontend Architecture Design

Based on Figure 1, it can be explained that the design of the micro frontend architecture takes two example features which will be separated into stand-alone web application sub-projects. These two sub-projects will later work as remote applications, said to be remote applications because this remote application will expose remote files to be used by the main application. Each sub-project application is designed to have its own API service according to the features of that sub-project. Then there is the main application which acts as a host. This host application will later be tasked with calling remote files from sub-projects which will then be used in the main application itself so that it becomes a unified frontend web application. For host and remote applications to communicate with each other, the help of the webpack module federation plugin tool is needed. The react.js library is used to build the main application sub-project, but the feature B sub-project will use a different version from the other projects. By implementing this micro frontend architecture, there will be each team that can focus on working on specific features in the sub-project application, and a team that will combine them in the main application.

4. Results and Discussions

Implementation Stages

Implementation is a stage carried out based on the design and micro frontend architecture rules that have been created and determined previously. Implementation of the sub-project application starts from initing the project and installing the required dependencies. After completing the init process, continue with writing program code according to the required features. After the program code has been created, you can continue by configuring the sub-project so that it can be exposed using the webpack module federation plugin, and the configuration is carried out in the webpack config file by specifying the project name, the name of the file to be exposed, and the location of the file to be exposed. The webpack module federation plugin project name configuration must be the same as the file name in package.json. For the Webpack module federation configuration for sub-projects, see Figure 2.

```
{
  name: 'feed',
  filename: 'remoteEntry.js',
  exposes: {
    './FeedApp': './src/bootstrap',
  },
}
```

Fig.2. Configure Webpack Module Federation Sub Project Application Plugin

The implementation of the webpack module federation plugin in the main application is not much different from the sub-project application, but in the webpack configuration file, the main application does not expose the file, but calls the file address from the remote application. An example of calling a file address from a remote application can be seen in Figure 3.

```
remotes: {
  feed: `feed@${domain}/feed-app/latest/remoteEntry.js`,
  ranking: `ranking@${domain}/ranking-app/latest/remoteEntry.js`,
}
```

Fig.3. Configure The Webpack Module Federation Plugin in The Main Application

Implementation of architectural rules that do not allow the main application to assume that sub-project applications use a certain framework is carried out by implementing communication in the most general way possible, namely by implementing callback functions. By implementing this callback function, the sub-project application will export a mount function which will then be used by the main application to render the sub-project application. The mount function is a simple function that takes a reference to an HTML element. And the mount function will create an instance and will be rendered on the div element. With this concept, we can use the same method if our sub-project application uses another framework, as long as the framework can render the application to an HTML element. When a sub-project replaces the entire program code with a different framework, in this way the main application will not change drastically.

The implementation of handling css scope rules should not affect the rest of the project is done using css-in-js techniques. With this technique, the scope of the styling created will be limited to the component that uses it, and with this technique you can also obtain a unique class name, so it will not affect other projects. And in this research the author uses tailwindcss which by default does not support the css-in-js concept. Therefore, an additional library called twin.macro is needed to support the implementation of CSS-in-JS in the TailWindcss library.

Implementation of authentication rules is carried out by implementing centralized authentication on the main application and distributing user data to each sub-project that needs it. This method was chosen so that the logic for dealing with authentication is not duplicated in each sub-project.

For application route configuration, the main application and sub-projects have their own routes. So that application routing in sub-projects can run well when used in the main application, navigation communication is needed between the main application and the sub-project application. In implementing navigation communication, it is carried out using event callbacks. The callback event can be seen in Figure 4. The onParentNavigate callback event is a callback event that is obtained from the sub-project application and will be used by the main application, then the main application will send a callback event called onNavigate which will later be used by the sub-project.

```
const { onParentNavigate } = mount(ref.current, {
  authUser: user,
  initialPath: history.location.pathname,
  onNavigate: ({ pathname: nextPathname }) => {
    const { pathname } = history.location;
    if (pathname !== nextPathname) {
      history.push(nextPathname);
    }
  },
});
history.listen(onParentNavigate);
```

Fig. 4. Callback Event Main Application Navigation with Sub Projects

Implementation of automated deployment on each sub-project is carried out using github actions. To be able to use github actions, you need to configure github workflows for each sub-project. This github workflows configuration is where the auto deployment process that we want will be carried out. The general description of the automation deployment process in this research is as follows:

1. Determines the event that will be listened to. In this research, the author arranges that when there is a push on the sub-project repository, the automated deployment will be executed.
2. Specifies the directory to run. In this research, the author chose the root project as the directory to be run.
3. Then, the application contained in the repository needs to be built and the build results are uploaded to an Amazon S3 bucket and later the build results can be accessed by the main application via Amazon Cloudfront.
4. And finally, automate the invalidation of the remoteEntry.js file so that there are no cache problems.

The GitHub workflows sub-project configuration can be seen in Figure 5.

```
name: feed-app

on:
  push:
    branches:
      - main
    paths:
      - '**'

defaults:
  run:
    working-directory: '.'

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - run: npm install
      - run: npm run build
        env:
          REACT_APP_API_URL: ${{ secrets.REACT_APP_API_URL }}
          REACT_APP_API_URL_V2: ${{ secrets.REACT_APP_API_URL_V2 }}
          REACT_APP_API_URL_V3: ${{ secrets.REACT_APP_API_URL_V3 }}

      - uses: shinyinc/action-aws-cli@v1.2
      - run: aws s3 sync dist s3://${{ secrets.AWS_S3_BUCKET_NAME }}/feed-app/latest
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          AWS_DEFAULT_REGION: ap-southeast-1

      - run: >
          aws cloudfront create-invalidation --distribution-id
          ${{ secrets.AWS_DISTRIBUTION_ID }} --paths "/feed-app/latest/remoteEntry.js"

        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          AWS_DEFAULT_REGION: ap-southeast-1
```

Fig. 5. Github Workflows Sub-Project Configuration

By implementing auto deployment on each sub-project, if changes occur to the sub-project, the main application does not need to rebuild, just build on each sub-project, then the main application and users will still be able to get the latest changes.

Testing Stage

At this stage we discuss the testing environment and framework for the micro frontend architecture. The testing framework includes:

Testing the Use of Sub Project Applications in the Main Application

This test is carried out to find out whether the sub-project application can be called on the main application during the run-time process and is in accordance with the Webpack module federation plugin configuration. Testing is carried out by checking network requests on the browser via developer tools.

The result of this test is that the sub-project application can be used in the main application during the run-time process. This means that the configuration of the sub-project application that exposes the file and the main application that uses the remote file is successful.

Initial Load Testing

This test is carried out to find out and compare the time needed for the application to be used by users interactively or what is commonly known as TTI (Time to Interactive). Testing will be carried out 20 times on each application. Testing is carried out on feature pages that have architectural changes made. Testing will be carried out using the Lighthouse tool with simulations using the Motorola Moto G4 mobile device, and using a slow 4G connection.

The results of initial load testing were successfully carried out on applications with micro frontend architecture and also applications with monolith architecture. The results of initial load performance testing on the two applications show that the average time required by applications with monolith architecture is faster.±3 seconds compared to micro frontend architecture. The results of the time comparison obtained by both can be seen in Table 1.

Table 1. Comparison of Initial Load Performance Tests

	Time to Interactive	
No	Monolith architecture	Micro frontend architecture
1.	11 seconds	13.8 seconds
2.	9.8 seconds	12.9 seconds
3.	10 seconds	13.5 seconds
4.	10.3 seconds	14.3 seconds
5.	10.4 seconds	12.5 seconds
6.	10.4 seconds	12.4 seconds
7.	10.2 seconds	14 seconds
8.	10.2 seconds	11.7 seconds
9.	9.6 seconds	11.9 seconds
10.	10.2 seconds	11.7 seconds
11.	10.7 seconds	13 seconds
12.	10.4 seconds	12.3 seconds
13.	11 seconds	12.5 seconds
14.	10.4 seconds	12 seconds
15.	10.4 seconds	12.1 seconds
16.	10.2 seconds	13.5 seconds
17.	10.3 seconds	13.7 seconds
18.	11 seconds	13.7 seconds
19.	10.1 seconds	11.7 seconds
20.	9.9 seconds	12.2 seconds
Average	10.33 seconds	12.77 ethics

Testing Build Time

This test is carried out to determine the build time of the sub-project application after it is separated into a stand-alone project and has its own auto deployment as well. The sub-project build time will be seen on the GitHub repository page when code is pushed to the repository.

The results of testing the build time of the sub-project application with micro frontend architecture showed that the build time was 1-2 minutes. This time is faster compared to the build time when the sub-project application was still being created on a monolith application. The results of the sub-project build time can be seen in Figure 6. And the results of the build time of the monolith frontend application can be seen in Figure 7.

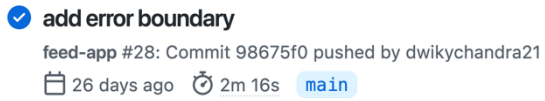

Fig. 6. Sub-Project Build Time

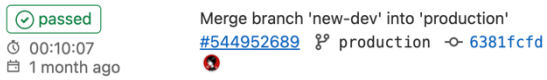

Fig. 7. Build Time of Monolith Frontend Application

Implementing auto deployment on sub-projects results in faster build times because the system scope has become smaller. Another advantage is that if changes occur to a sub-project, the main application does not need to be rebuilt, it is enough to build each sub-project and main application and users will still be able to get the latest changes.

5. Conclusion

Based on the research that has been done by analyzing, designing and implementing, the following conclusions can be drawn 1) Implementation of micro frontend architecture with run-time integration techniques in the Smart BTW monolith application at PT Bina Taruna Wiratama can be implemented on the feed page and ranking page. The micro frontend implementation uses the webpack module federation plugin. The feed application and ranking application can run individually or when used in the main application. Application sub-projects already have auto deployment independently in each project repository, so making changes to one of the sub-projects does not need to make changes and in the main application. 2) From the implementation results, with the application of micro frontend architecture, the scope and resources of the application built are smaller, so that the application deployment process does not take a long time, which is 1-2 minutes. 3) Initial load on applications with micro frontend architecture is slightly slower than with monolithic architecture.

References

Attardi, J. (2020). *Modern CSS*. Springer. https://doi.org/10.1007/978-1-4842-6294-8

Benavente, V., Yantas, L., Moscol, I., Rodriguez, C., Inquilla, R., & Pomachagua, Y. (2022). Comparative analysis of microservices and monolithic architecture. *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, 177–184. https://doi.org/10.1109/CICN56167.2022.10008275

Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, *10*, 20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803

Bühler, F., Barzen, J., Harzenetter, L., Leymann, F., & Wundrack, P. (2022). Combining the Best of Two Worlds: Microservices and Micro Frontends as Basis for a New Plugin Architecture. *Symposium and Summer School on Service-Oriented Computing*, 3–23. https://doi.org/10.1007/978-3-031-18304-1_1

Emmanni, P. S. (2024). Revolutionizing Frontend Development: Embracing Micro UI Architecture With Cloud Integration. *International Journal of Computing and Engineering*, *5*(4), 1–10. https://doi.org/10.47941/ijce.1821

Engström, V., Johnson, P., Lagerström, R., Ringdahl, E., & Wällstedt, M. (2023). Automated security assessments of Amazon Web services environments. *ACM Transactions on Privacy*

and Security, *26*(2), 1–31. https://doi.org/10.1145/3570903

Gashi, E. (2024). The Advantages of Micro-Frontend Architecture for Developing Web Application. *Https://Ieeexplore.Ieee.Org/Xpl/Conhome/10577658/Proceeding*. https://doi.org/10.1109/meco62516.2024.10577836

Li, K., Ding, Y., Shen, D., Li, Q., & Zhen, Z. (2020). The design and research of front-end framework for microservice environment. *2020 International Conference on Computer Information and Big Data Applications (CIBDA)*, 124–127. https://doi.org/10.1109/CIBDA50819.2020.00036

Mufrizal, R., & Indarti, D. (2019). Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik. *Jurnal Nasional Teknologi Dan Sistem Informasi*, *5*(1), 57–68. https://doi.org/10.25077/teknosi.v5i1.2019.57-68

Musiolik, J., Markard, J., Hekkert, M., & Furrer, B. (2020). Creating innovation systems: How resource constellations affect the strategies of system builders. *Technological Forecasting and Social Change*, *153*, 119209. https://doi.org/10.1016/j.techfore.2018.02.002

Nunes, L., Santos, N., & Rito Silva, A. (2019). From a monolith to a microservices architecture: An approach based on transactional contexts. *Software Architecture: 13th European Conference, ECSA 2019, Paris, France, September 9–13, 2019, Proceedings 13*, 37–52. https://doi.org/10.1007/978-3-030-29983-5_3

Paltoglou, K., Zafeiris, V. E., Diamantidis, N. A., & Giakoumakis, E. A. (2021). Automated refactoring of legacy JavaScript code to ES6 modules. *Journal of Systems and Software*, *181*, 111049. https://doi.org/10.1016/j.jss.2021.111049

Peltonen, S., Mezzalira, L., & Taibi, D. (2020). *Motivations, Benefits, and Issues for Adopting Micro-Frontends: A Multivocal Literature Review*. https://doi.org/10.48550/arxiv.2007.00293

Saransig, A., & Tapia, F. (2019). Performance analysis of monolithic and micro service architectures–containers technology. *Trends and Applications in Software Engineering: Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018) 7*, 270–279. https://doi.org/10.1007/978-3-030-01171-0_25

Taibi, D., & Mezzalira, L. (2022). Micro-Frontends. *Acm Sigsoft Software Engineering Notes*, *47*(4), 25–29. https://doi.org/10.1145/3561846.3561853

Thabit, F., Can, O., Aljahdali, A. O., Al-Gaphari, G. H., & Alkhzaimi, H. A. (2023). Cryptography algorithms for enhancing IoT security. *Internet of Things*, *22*, 100759. https://doi.org/10.1016/j.iot.2023.100759

Tilak, P. Y., Yadav, V., Dharmendra, S. D., & Bolloju, N. (2020). A platform for enhancing application developer productivity using microservices and micro-frontends. *2020 IEEE-HYDCON*, 1–4. https://doi.org/10.1109/HYDCON48903.2020.9242913

Wanjala, S. T. (2021). A Framework for Implementing Micro Frontend Architecture. *International Journal of Computer Applications Technology and Research*, *10*(12), 267–275. https://doi.org/10.7753/ijcatr1012.1002

Wanjala, S. T. (2022). A framework for implementing micro frontend architecture. *International Journal of Web Engineering and Technology*, *17*(4), 337–352. https://doi.org/10.1504/IJWET.2022.129251

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2020). The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*, *35*, 63–75. https://doi.org/10.1007/s00450-019-00412-x