International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307

Available Online at : www.ijsrcseit.com

doi : https://doi.org/10.32628/CSEIT23564521

Design Patterns and Performance Strategies for Scalable Frontend Applications

Sree Harsha Palli

College of Arts and Sciences, Lamar University, Beaumont, Texas, 77705, USA

ARTICLEINFO

Article History:

Accepted: 10 July 2023

Published: 24 July 2023

Publication Issue

Volume 9, Issue 4

July-August-2023

Page Number

676-684

ABSTRACT

The current alarming increase in scale and complexity of web applications ought to be addressed significantly using various viable strategies. Also, there currently exists a vast gap in research on design patterns and performance for scalable frontend applications. This study attempts to increase the volume of research on this important theme, while innovatively bridging a laid-bare knowledge gap. The study is aimed at exploring the nexus between design patterns and performance strategies for optimization and sustainable development of scalable frontend applications. It focuses state management paradigm, component-based architecture, and container-separation. These are analyzed to determine their impact on system performance, reusability and maintainability of code. The analysis shows that using built tools, splitting of code, effective rendering techniques, and lazy loading are some of the viable strategies for design patterns and performance optimization. Further illustrations of the efficacy of these approaches in real-world applications are demonstrated with performance benchmarks and several case studies. The novelty of the study lies majorly in its intersection of design patterns and performance strategies for the realization of best practices and enhanced performance. The proposed framework displays practicability, built-in responsiveness, maintainability, and frontend systems having high performance, which can scale the expansion of features and improved user growth.

Keywords : Design patterns, Performance strategies, Scalable frontend applications, Optimization

Introduction

As the scale and complexity of web applications increase worrisomely these days (Paladugu, 2023; Perera & Jayasuriya, 2023), there is need to devise and sustainably apply viable design patterns and performance strategies for scalable frontend applications. Freeman and Freeman (2014) observe that the misapplication of design patterns

can increase complexity, eventually reducing the effectiveness of the web development procedure. To that end, this study rises to examine the design patterns and performance strategies that can yield the desired results in addressing the issues of scale and complexity of web applications. The advent of frameworks has eased the difficulties in web programming to some extent (Akinloye & Shawana, 2023; Mwiinga, 2023; Oguntona, 2023; Shawana, 2022; Alkharabsheh et al., 2019; Recupero et al., 2019). Consequently, full-stack web design is now possible.

Optimizing and managing websites and online applications now entail the full-stack developers' reliance on frameworks and the requisite digital tools (Wijaya et al., 2021). Although it is difficult to decide on which tools to employ, developers can surmount the challenges by learning and mastering how to tackle the challenges using effective, well-designed and performance-driven full-stack web development roadmap (Prasad, 2022; Vemuri, 2020). The following diagram shows categories of web developers:

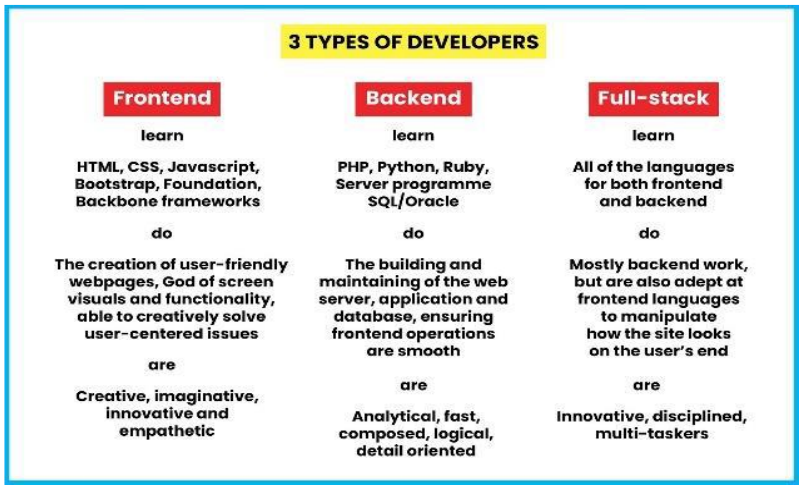

Fig. 1: Kinds of web developers

Source: Prasad (2022)

As Mohammed and Faraj (2021) rightly note, server-side technologies, including PHP, Python, Java, and Ruby, among others, are utilized by developers for connection of the server to application and the database. The implication of the foregoing is that different criteria are required and met in designing, implementing and operating scalable frontend applications. Additional to the aforementioned requisitions are design patterns and performance strategies for the realization of effective and suitable scalable frontend applications. That is why this study seeks to explore design patterns and performance strategies for scalable frontend applications.

Aim and Objectives

The aim of this study is to explore the nexus between design patterns and performance strategies for optimization and sustainable development of scalable frontend applications. Its objectives are to:

(i) Explore the impact of design patterns on scalable frontend applications.

(ii) Determine performance strategies for scalable frontend applications.

(iii) Demonstrate the essential role of design patterns in fostering the performance of scalable frontend applications.

Gap and Novelty

The need for this research is premised on the current vast gap in research on design patterns and performance for scalable frontend applications. In other words, following the current vast gap in research on design patterns and performance strategies for scalable frontend applications, this study rises to engage with the thematic concerns,

bridge a laid-bare knowledge gap, contribute to the growing volume of research on the thematic concerns, and demonstrate how design patterns enhance performance for scalable frontend applications. By so doing, the study makes significant contributions to knowledge and practices.

More so, the study's intersection of design patterns and performance strategies for the realization of best practices and enhanced performance makes it undoubtedly novel. Also, it draws attention to the thematic concerns, the associated issues and the impact of design patterns and performance strategies on scalable frontend applications. Besides, by presenting the innovations offered by design patterns to scalable frontend applications, the study underscores the importance of technological innovations, the dire need for significant technology integration, and the role of design patterns in enhancing the performance of scalable frontend applications.

Design Patterns

It is noted that design patterns provide comprehensive and repeatable approaches to tackling common issues of design; thereby assisting developers build adaptable and easy-to-maintain code (Paladugu, 2023; Ruponen, 2019; Ampatzoglou et al. 2012; Rohnert, 1996). The study by Prasad (2022) shows that frontends applications are valuable in e-businesses, trade and commerce, public administration, library and information management, informatics, and many other essential sectors. It points out that frontend programming using frontend development frameworks and libraries optimizes web services. It also shows that the advantages of web development frameworks and libraries outweigh the disadvantages. Thus, there is need to address the challenges to lessen the disadvantages while exploiting the advantages. The following figure itemizes some major elements of design patterns:

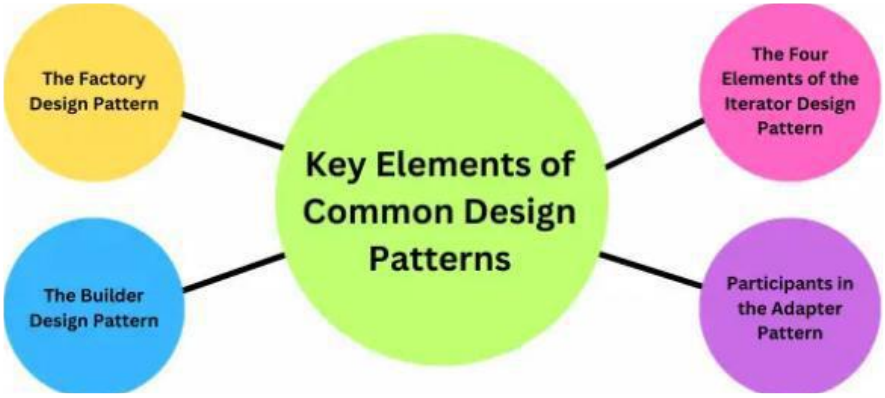

Fig. 2: Major components of common design patterns

Source: Ampatzoglou et al. (2012)

The creation of any web applications requires the use of tools like MySQL and ASP.NET, Oracle or SQL Server alongside other backend databases. The use of frameworks is also required, which a developer ought to be knowledgeable in. For example, the use of PHP frameworks, such as Symfony, Zend and CakePHP, is imperative. A developer also ought to be experienced and knowledgeable in utilizing version control tools, such as SVN or CVS, and Linux. These are additional to utilizing a development and an implementation platform respectively (Amoretti, 2021; Pratomo, et al. 2021).

There are different design patterns. These include state management patterns, component-based architecture, container-separation pattern, render props, observer pattern, micro-frontends, and higher-order components. State management patterns (SMPs) consist of use patterns, such as Redux, Flux and Context API, which are used for managing the state of applications (Paladugu, 2023; Geary, 2016). The SMPs are beneficial in various ways. These include easing debug and maintenance of scalable frontend applications, and centralizing state management.

The component-based architecture is the design pattern that breaks down the UI into reusable components. Each of the components manages its own state and lifecycle. It comprises react, vue.js, and angular (Sharma, 2022).
Luo and Wang (2019) note that web designers can create efficient frontend applications and other web app frameworks using Angular. The performance impact of Angular made Google to acquire it in 2010 and advanced it to Angular 2 in 2016 for more results. Its initial design was targeted at creating more effective consistent online form. Angular is proven to be an outstanding framework, used for building web and mobile platforms, such as Windows, Linux and Mac. It is also efficient for hardware design of nodes (Node Hardware), as in NFC labels and tags, A PIC16F1947 nanoWatt XLP microcontroller, RN4020 Communication module, and DRF2619C ZigBee module (Angular Team, 2022). The following table highlights Angular in data binding:

Table 1: Data binding in frontend frameworks

	Angular 1	Angular 2	React	vue
Data binding	Two way	One way & two way	One way	One way two way

Source: Prasad (2022)

The component-based architecture is beneficial in several regards. These include promoting reusability of components, making testing easier and better, and improving the separation of concerns (Aviv et al., 2023; Degawa et al., 2023; Kothapalli, 2021; Gunnulfsen, 2013). React.JS & React Native are two of the most popular front-end frameworks (Jonathan & Suprihadi, 2023; Wieruch, 2018). Facebook and Instagram websites rely heavily on these frameworks and get huge positive results. The innovative and influential ability of React.JS made Facebook to release React as an open-source JavaScript ES6-based library to developers and enterprises across the globe. In 2015, Facebook launched React Native, a mobile app that can run on both Android and iOS (Prasad, 2022). Consider the following diagram for best JavaScript framework:

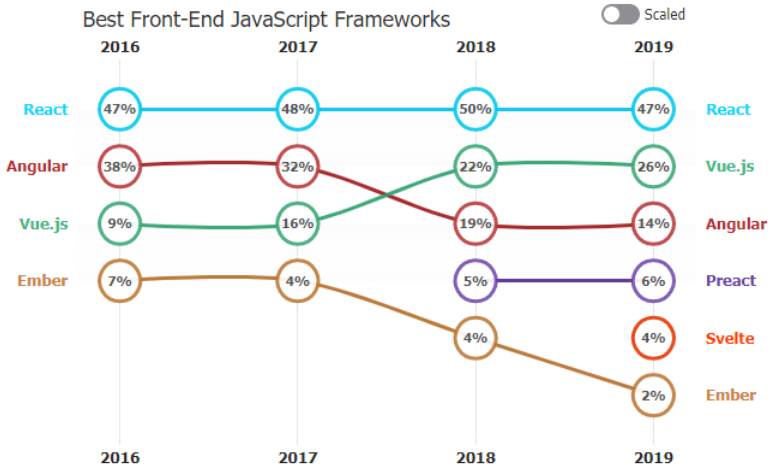

Fig. 3: Frontend JavaScript Frameworks

Source: Prasad (2022)

React stores the different components constituting website's content in the Document Object Model (DOM); whereas the content is displayed in browsers like JavaScript (Mawer, 2016). The JavaScript rendering performs faster than any typical dynamic webpages having Chrome V8 technology. As Mawer (2016) notes, React builds an updated virtual DOM in the first place, before comparing the virtual DOM to the shown DOM in order to

determine any differences. Afterwards, re-rendering follows with only the actual transform part being done after each component's differences are summarized.

Container-separation pattern, also regarded as container-presentational pattern, is the type that separates components into container components and presentational components (Abramov & Clark, 2015). While the former components comprise handlNe state and logic, the presentational components focus on the rendering of UI. This design pattern is reputed for enhancing code organization and reusability. Higher-order components (HOCs) describe a function that obtains a component and thereafter returns a component with additional props and/or behavior. The benefits of HOCs include separation of concerns and allowing for the reuse of codes (Jonathan & Suprihadi, 2023).

Render props are deign patterns deployed for sharing of code between/among React components, using a functional prop (Paladugu, 2023). These design patterns generate props in different contexts and present suppleness in how components render. Micro-frontends are the design patterns that break down a monolithic frontend into smaller autonomous applications, which can be developed, arranged and used separately (Nikulina & Khatsko, 2023; Titova et al., 2021). They enable teams to work autonomously, pave way for technology diversity, and reduce operation risks. And, observer pattern is one in which components can subscribe or adjust to changes in state and re-render wherever necessary (Macenski, 2020). The benefits of this pattern include improving performance, reducing manual updates, and automating frontend applications and their operations.

Performance Strategies

Performance strategies for scalable frontend applications include code splitting, lazy loading, memoization, virtualization, debouncing and throttling, service workers and caching, optimizing images and assets, minification and bundling, and performance monitoring (Petcu et al.,2023; Ain et al., 2019). Accordingly, code splitting is the strategy that splits code into smaller chunks, which can be loaded on demand. As noted by Dawson and Szakonyi (2020), frontend developers have the task of developing code that visitors see and interact with on a website. Code splitting techniques are React.lazy, Webpack and dynamic imports. This performance strategy improves performance and reduces initial load time. The strategy only loads load images or load components when they are needed, as in when they get into the viewpoint. Its benefits include improving perceived performance and reducing initial load time (Nambisan et al., 2017).

Debouncing and throttling are dual performance strategies that limit the rate in which a function is being executed. For example, during resize or scroll events, debouncing and throttling obtain. The role played by debouncing and throttling helps to reduce the numbers of function calls and enhances performance. These are the top benefits of debouncing and throttling. The virtualization strategy renders only portions of a large table or list that is visible. Its main tools or techniques are React Window and React Virtualized (Dimitrov & Ostrovska, 2020). Virtualization is important because it reduces the number of rendered DOM nodes. By so doing, it helps improves the performance of scalable frontend applications.

Memoization is the strategy for cache management, as it caches the results of expensive function calls and returns the cached results when the same inputs take place again (Degawa et al., 2023). Its techniques are useCallback, React.memo, and useMemo. The actions carried out by memoization lead to better performance and the reduction of unnecessary re-renders. Optimizing assets and images employ suitable formats and sizes (Jonathan & Suprihadi, 2023). Additionally, they compress images and other assets using appropriate formats and sizes. The techniques for optimizing images and assets are WebP format, SVGs, and image optimization tools. Load times are reduced and performance enhanced. These make them beneficial. Service workers and caching utilize service workers to

cache API responses and assets for offline access and faster load times. By doing so, performance is improved and user experience is made better and satisfactory. Improved performance and user experience particularly obtain with repeat visits.

The minification and bundling performance strategies minify CSS, HTML files and JavaScript, and bundle them afterwards to reduce the number of requests (Santoso, 2021; Crockford, 2008). The techniques of minification and bundling are UglifyJS, Webpack and Terser. Additional to reducing file sizes, these strategies enhance the performance of scalable frontend systems. Prasad (2022) observes that frontend developers must familiarize themselves with the three major languages of the computer: CSS, HTML, and JavaScript, and must be knowledgeable in ReactJS, EmberJS, AngularJS, Bootstrap Foundation, and Bootstrap, among other frameworks and libraries (Santoso, 2021). These are needed to be able to produce visually appealing content across different devices.

Finally, performance monitoring strategy employs techniques, such as WebPageTest and Lighthouse, to monitor and analyze performance metrics (Petcu et al., 2023; Wedyan & Abufakher, 2020). As it does that, performance rate is determined and areas of improvement are identified alongside the needed actions being taken. The overall importance of the performance monitoring strategy rests on its detection of failures and prompting of success-driven actions to make up for lapses in areas needing improvement (Degawa et al., 2023; Aladib & Lee, 2019). The following figure offers valuable insights into performance metrics in scalable frontend applications:

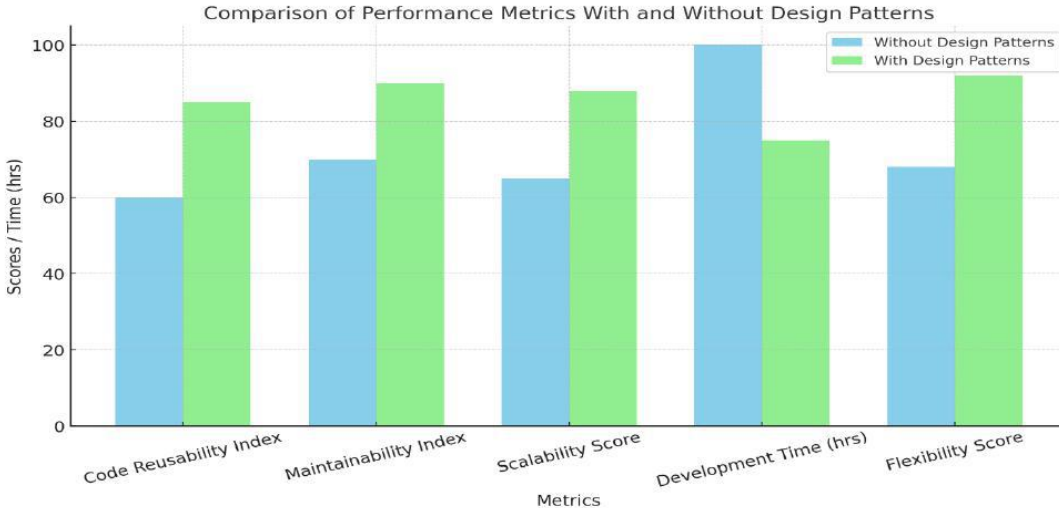

Fig. 4: Comparative performance metrics

Source: Petcu et al. (2023); Macenski (2020)

Summary of Findings

The following findings are established:

(i) Management paradigm, component-based architecture, and container-separation are some of the major performance strategies applied to scalable frontend applications.

(ii) The strategies have the capacity to significantly optimize the performance of scalable frontend applications.

(iii) They particularly exert huge impact on system performance, reusability and maintainability of code.

(iv) Through effective management paradigm, component-based architecture, and container-separation, maximum scalability and performance, usability and maintenance of frontend applications are realized.

(v) In leveraging management paradigm, component-based architecture and container-separation for design patterns and performance optimization, built tools, splitting of code, effective rendering techniques, and lazy loading are used.

(vi) The explored strategies and approaches have practical realizations in real-world applications.

(vii) The overall impact of design patterns on scalable frontend applications manifests in the performance optimization of the applications, allowing for effectiveness, innovative design, functionality and operations.

Conclusion

The study has been concerned with design patterns and performance strategies for scalable frontend applications. It has explored several categories of design patterns and performance strategies that can be leveraged for scalable frontend applications. The strategies are needed to allow for optimization, automation and scalability of the applications. As evident in its analysis and illustrations of real-world applications, the study has shown that design patterns and performance optimization of scalable frontend applications can be attained by using effective rendering techniques, built tools, splitting of code and lazy loading, among other strategies. The illustrations and several case studies made in the study typify performance benchmarks. The proposed framework shows practicability, maintainability, built-in responsiveness and scalability of frontend systems.

In all, by showing how high performance can scale the expansion of features and improved user growth, the study underscores the importance of leveraging design patterns and performance strategies for the attainment of high performance in scalable frontend applications. From all indications, it is quite understandable that by combining the presented design patterns and performance strategies, effective, optimized, practicable, and maintainable scalable frontend applications can be created and sustained. Such applications are highly characterized by satisfactory user experience. As such, scalable applications should be tactfully designed, patterned, and made flexible for continuous improvement. Besides, stakeholders in the industry should imbibe the proposed framework in scalable frontend systems in order to attain high performance and significant, operational efficiency, digital system security, web innovations and organizational productivity.

References

1. Abramov, D., & Clark, A. (2015). Redux: a predictable state container for JavaScript Apps. redux.js.org.
2. Ain, Q. U., Butt, W. H., Anwar, M. W., Azam, F., & Maqbool, B. (2019). A systematic review on code clone detection. IEEE access, 7, 86121-86144.
3. Akinloye, A., & Shawana, T. A. (2023). Gas caps detection, hydrocarbon recovery and environmental sustainability. Journal of Applied Ecology and Environmental Design (JAEED), 2(2), 118-130.
4. Aladib, L., & Lee, S. P. (2019). Pattern detection and design rationale traceability: an integrated approach to software design quality. IET Software, 13(4), 249-259.
5. Alkharabsheh, K., Crespo, Y., Manso, E., & Taboada, J. A. (2019). Software design smell detection: a systematic mapping study. Software Quality Journal, 27, 1069-1148.
6. Amoretti, M. (2021). An effective framework for full-stack benchmarking of quantum computers. Quantum Views, 5. 10.22331/qv-2021-04-26-52.
7. Ampatzoglou, A., Frantzeskou, G., & Stamelos, I. (2012). A methodology to assess the impact of design patterns on software quality. Information and Software Technology, 54(4), 331- 346.

8. Angular Team (2022). Angular documentation. angular.io.
9. Crockford, D. (2008). JavaScript: the good parts. O'Reilly Media.
10. Dawson, M., & Szakonyi, A. (2020). Cybersecurity education to create awareness in artificial intelligence applications for developers and end users. Scientific Bulletin, 25(2), 85-92. 10.2478/bsaft-2020-0012.
11. Degawa, Y., Kolzumi, T., Nakamura, T., Shioya, R., Kadomoto, J., Irie, H., & Sakai, S. (2023). A principal factor of performance in decoupled front-end. IEICE Trans. Inf. Syst, 106(12), 1960-1968.
12. Dimitrov, W., & Ostrovska, T. (2020). Overview of virtual and augmented reality application development tools. Online, available: https://doi.org/10.13140/RG.2.2.29293.82407/1
13. Freeman, E., & Freeman, E. (2014). Head first design patterns: a brain-friendly guide. O'Reilly Media.
14. Geary, D. (2016). Introducing redux. IBM DEVELOPER. https://developer.ibm.com/tutorials/wa-manage-state-with-redux-p1-david-geary/
15. Gunnulfsen, M. K. (2013). Scalable and efficient web application architectures. Master thesis, Department of Informatics, University of Oslo.
16. Jonathan, R., & Suprihadi (2023). Development of front-end web applications utilizing single page application framework and react.js library. International Journal Software Engineering and Computer Science (IJSECS) 3(3), 529-536. DOI: https://doi.org/10.35870/ijsecs.v3i3.1943
17. Kothapalli, M. (2021). The evolution of component-based architecture in front-end development. Journal of Scientific and Engineering Research, 8(7), 261-264.
18. Luo, G., & Wang, Q. (2019). Developers competition behaviour of land reserve in assembled buildings development: Based on game analysis framework. E3S Web of Conferences, 136, 040-44. 10.1051/e3sconf/201913604044.
19. Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics, 7(66), eabm6074.
20. Mawer, C. (2016). Something must be done but chemotherapy isn't the only option. BMJ, i6455. 10.1136/BMJ.i6455.
21. Mohammed, H., & Faraj, K. (2021). A Python-WSGI and PHP-Apache web server performance analysis by search page generator (SPG). UKH Journal of Science and Engineering, 5(1), 132-138, 2021. 10.25079/use.v5n1y2021.pp132-138.
22. Mwiinga, P. (2023). Privacy-preserving technologies: Balancing security and user privacy in the digital age. Int. J. Sci. Res. Publ, [Online]. https://zenodo.org/records/10406538
23. Nambisan, S., Lyytinen, K., Majchrzak, A., & Song, M. (2017). Digital innovation management. MIS Quarterly, 41(1), 223-238.
24. Nikulina, O., & Khatsko, K. (2023). Method of converting the monolithic architecture of a front- end application to microfrontends. Bull. Natl. Tech. Univ. "KhPI" Ser. Syst. Anal. Manag. Inf. Technol, 10(2), 79-84.
25. Oguntona, T. I. (2023, March). Mitigating the impact of high energy prices on UK households using machine learning techniques. International Journal of Modelling & Applied Science Research, 27(9), 215-230.
26. Paladugu, S. (2023). Design patterns for effective front-end development in modern web applications. International Journal of Leading Research Publication (IJLRP), 3(12), 1-5.

27. Perera, Y., & Jayasuriya, D. (2023). "Enhancing the front-end web applications performance using design patterns and microservices based architecture." Online. https://doi.org/10.13140/RG.2.2.36067.53286
28. Petcu, A., Frunzete, M., & Stoichescu, Dan A. (2023). Benefits, challenges, and performance analysis of a scalable web architecture based on micro-frontends. U.P.B. Sci. Bull., Series C, 85(3), 319-334.
29. Prasad, A. (2022). Research and analysis of the frontend development frameworks and libraries with voice recognition in remote areas for developing ecommerce business for people of remote areas where people have less or no knowledge of technical devices. International Journal of Engineering Applied Sciences and Technology, 6(11), 47-52.
30. Pratomo, A., Schriek, E., & Veen, T. (2020). Test-driven development in OWOW's full-stack web development. International Journal of Industrial Research and Applied Engineering, 4(2), 10.9744/jirae.4.2.46-50.
31. Recupero, D., Dessì, D., Concas, E. (2019). A flexible and scalable architecture for human-robot interaction. Proc. 15th Eur. Conf. Ambient Intell., Cham, Switzerland, 311-317.
32. Rohnert, H. (1996). {Pattern-oriented} Software architecture. In 2nd USENIX Conference on Object-Oriented Technologies (COOTS 96).
33. Ruponen, E. (2019). The front-end architectural design and implementation of a modularized web portal. Master's Degree Programme in Information Technology, Faculty of Information Technology and Communication, Tempere University.
34. Santoso, M. F. (2021). Teknik single page application (Spa) layout web Dengan Menggunakan react js dan bootstrap. Jurnal Khatulistiwa Informatika, 9(2), 107–114. DOI: https://doi.org/10.31294/jki.v9i2.11357
35. Sharma, I. (2022). Benefits of react js for your front-end development process. TatvaSoft. https://www.tatvasoft.com/outsourcing/2022/02/benefits-of-react-js.html
36. Shawana, T. A. (2022). Carbon emissions as threats to environmental sustainability: Exploring conventional and technology-based remedies. African Journal of Environmental Sciences and Renewable Energy, 8 (1). https://publications.afropolitanjournals.com/index.php/ajesre/article/view/736
37. Titova, O., Luzan, P., Sosnytska, N., Kulieshov, S., & Suprun, O. (2021). Information and communication technology tools for enhancing engineering students' creativity. Lect. notes Mech. Engin, 332-340. https://doi.org/10.1007/978-3-030-77719-7_33
38. Vemuri, V. P. K. (2020). Addressing critical challenges in front-end performance and scalability. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 156- 161. DOI: https://doi.org/10.54660/.IJMRGE.2020.1.5.156-161
39. Wedyan, F., & Abufakher, S. (2020). Impact of design patterns on software quality: a systematic literature review. IET Software, 14(1), 1-17.
40. Wieruch, R. (2018). The road to react: Your journey to master react.js. An independent blog post.
41. Wijaya, E, Kosasi, S., & David, D. (2021). Implementasi Aplikasi web full stack Pendataan Cloversy.id. Jurnal Sisfokom (Sistem Informasi dan Komputer), 10(3), 320-327. 10.32736/sisfokom.v10i3.1293.