American Research Journal of Computer Science and Information Technology

Volume 7, Issue 1, 11-15 Pages
Research Article | Open Access
ISSN (Online)- 2572-2921
DOI : 10.21694/2572-2921.24003

AMERICAN RESEARCH JOURNALS
An Academic Publishing House

Effectiveness of Different Approaches to Organizing Frontend Development

Nikhil Badwaik
Software Engineer at NIKE INC, Portland OR, USA.

ABSTRACT

The research is devoted to the analysis of various approaches to the organization of Front-end development with an emphasis on optimizing the architecture of web applications. The main focus is on the impact of structural solutions on performance, scalability, and ease of maintenance. The work examines various architectural strategies, such as modular division, and the use of micro-frontends and hybrid approaches, and analyzes their effectiveness in the context of current trends in technology development. By comparing theoretical foundations and practical examples, the key factors influencing the choice of architecture for specific projects are identified. The study shows how adaptation to changing requirements and pre-planning contribute to the creation of stable and aesthetically pleasing web applications.

KEYWORDS: *front-end development, interface, organization of front-end development, IT, software, websites.*

INTRODUCTION

Front-end architecture represents a fundamental element of web application design, determining its structure, performance, and user interaction. Let us review the main aspects of this key concept.

Web application front-end design is a comprehensive process of structuring the user interface. It includes the design of the hierarchy of code, files, and folders, as well as the selection of appropriate platforms and libraries, which contributes to a structured and functional system.

The importance of a stable front-end architecture to a successful web application cannot be understated. It has a direct impact on how users interact with the website and how they perceive it.

Creating an effective interface involves selecting appropriate frameworks and libraries, managing the state to keep information up-to-date, and structuring components for maintainability and scalability, and optimizing performance to improve the speed and functionality of the website.

In general, designing a front-end architecture requires thoughtful decisions and the use of effective techniques to create an aesthetically pleasing and consistently performing web application [1].

For developers whose goal is to continuously influence the industry, it is important to stay up-to-date with the latest programming languages, interface platforms, and technological innovations to maintain a competitive edge in a dynamic market [2].

COMMON INTERFACE ARCHITECTURE

The simplest and most common way to partition a frontend application nowadays can be roughly as represented in Figure 1.

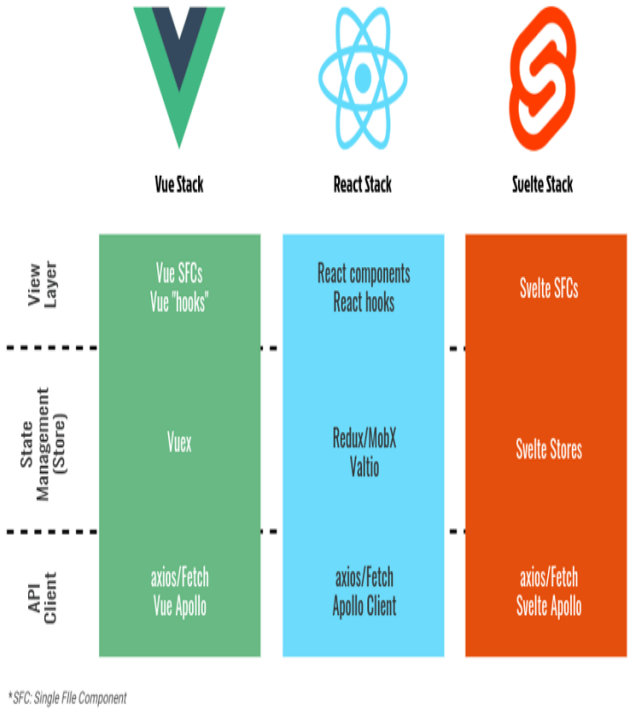

Fig.1. The method of dividing a frontend application [3].

Initially, it may seem that the described architecture has no drawbacks. However, it reveals a typical problem: some elements of the architecture are too tightly integrated.

Consider a situation where Redux was used to manage state in the application, but then the team decides that Redux is too complex for current needs and wants to switch to another tool. This would require rewriting all the repositories and associated logic of the React components, which is a time-consuming process due to the strong dependencies between layers.

This problem is not unique and affects all aspects of the application architecture. As a result, replacing even one part may not be possible without completely redesigning the entire system. It may be tempting to leave things as they are or even rewrite the application from scratch.

However, this is not the only possible approach. A properly designed modular architecture makes it possible to replace elements of the system, for example, to move from React + Redux to React + MobX, or even React + Vuex, or Vue + Redux without interfering with other parts of the application.

The obvious question that arises is "How do we achieve this flexibility?". The main question is how to isolate the components of the application from each other so that changes in one part do not affect the functionality of the others. We present another approach (Fig.2.).

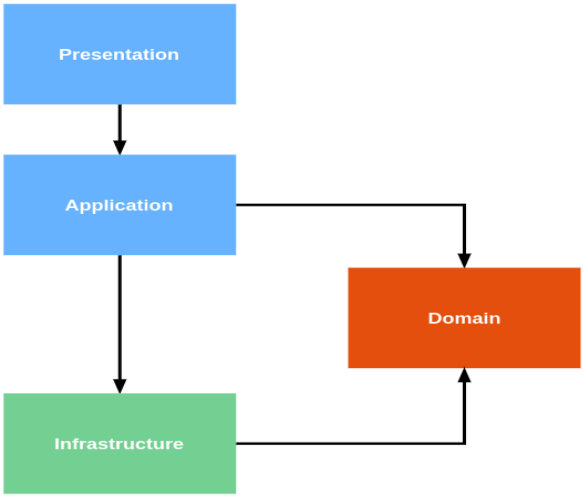

Fig.2. An example of a different approach to frontend development [3].

Layers in software architecture have the following characteristics:

Presentation layer: This layer primarily includes the user interface elements. Depending on the framework used, these could be Vue Single File Components (SFCs) for Vue, React components for React, or Svelte SFCs for Svelte. This layer interacts closely with the application layer.

Application Layer: This is where the application logic is centered. This layer interacts with the domain and infrastructure layers and is implemented, for example, using React Hooks in React or similar constructs in Vue 3.

Domain Layer: This layer is designed to implement domain or business logic. It contains solely business logic and is implemented in pure JavaScript or TypeScript without involving additional frameworks or libraries.

Infrastructure layer: This layer provides interaction with the outside world, including sending requests and receiving responses, as well as managing local data. Various libraries can be used to implement these functions, such as Axios or Fetch API for HTTP requests, and Vuex, Redux, MobX, or Valtio for application state management.

If this architecture is applied to an application, it will look like Figure 3.

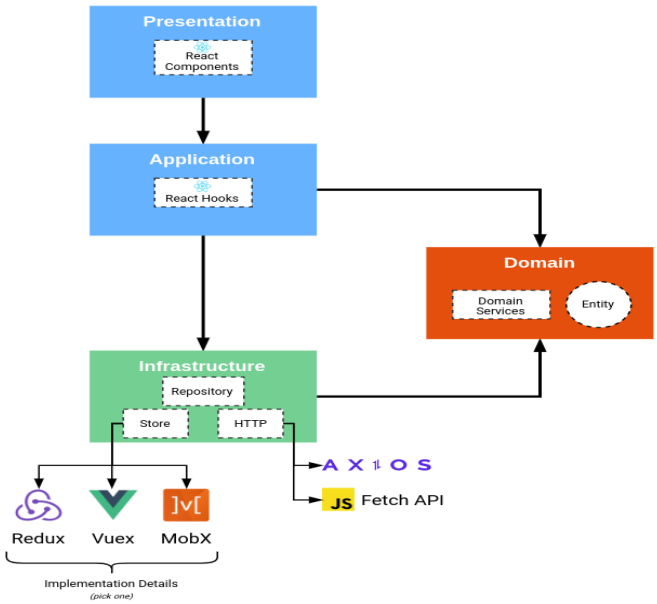

Fig.3. An example of the application architecture [3].

The following characteristics are taken from the above architecture diagram:

When we replace the UI library/framework, only the presentation and application layers are affected.

At the infrastructure layer, we have a facade, so when replacing storage implementation details (e.g. replacing Redux with Vuex) it only affects the storage itself. The same goes for replacing Axios with Fetch API or vice versa. The application layer does not know the implementation details of the repository or the HTTP client. In other words, we have separated React from Redux/Vuex/MobX. The repository logic is also fairly universal, so it can be used not only with React but also with Vue or Svelte.

If the business logic changes, the subject matter layer will have to change accordingly, and this will affect other parts of the architecture.

What's even more interesting about this architecture is that it can be further modularized:

However, even though the architecture can separate parts of the application from each other, there is a price to pay: increased complexity. So if working on a small application, this is not recommended [3].

CRITERIA FOR A GOOD FRONTEND ARCHITECTURE

In the initial development phase of an application, it is usually a compact and cleanly structured system without much complexity. However, over time, as the application evolves and expands, both the amount of code and the number of interrelationships between modules increases. If these processes are not controlled, the interrelationships can become so rigid and inflexible that even minor changes to the code become problematic. An example from practice, though not my personal one, shows a situation when due to rigid architecture developers had to rewrite a large part of the program just to change the organization of data in a table, which took more than a week with all the approvals taken into account.

To avoid such chaos in the code, pay attention to three key properties of a good architecture:

Usability. Developers should easily navigate the project structure and understand the interrelationships of its components. This will ensure comfortable and productive work, especially for front-enddevelopers.

Time-to-market (Time-to-market). It is ideal when this time is minimized, allowing updates to be released quickly. The reduction of development time depends on the organization of business processes in the company.

Scalability. As the company grows, attracts new customers, and adds functionality, the load on the application increases. It is important to lay down scalability from the very beginning to support sustainable development.

However, the first two criteria are affected not only by internal but also by external factors. For example, if business processes in the company are organized inefficiently, it will hinder the comfortable work of developers, even if there is a high-quality architecture, and will not allow to speed up the release of updates.

As an example from my experience, during the development of a large application, we used Gitflow, dividing tasks into two-week sprints. This allowed us to make changes to the release branch steadily and without rushing. However, the business wanted to speed up the process, which led to the need to implement local releases, which significantly worsened the comfort and convenience of the developers' work due to the need to rework already established processes. This example shows that the success of a project depends not only on its architecture but also on many external circumstances [4].

STEPS IN THE DEVELOPMENT PROCESS

The first step starts with communication with the potential client, where the sales manager finds out the needs and specifications of the project. If the client's needs are beyond the company's competence, for example, a request for PWA development while specializing in mobile applications, the company may decide not to accept the order.

Next comes the research and planning of the project. In this phase, the team conducts research on the client's business requirements and selects a suitable technology stack, keeping in mind scalability and future project needs. This research helps determine if the proposed solution fits both the client's vision and budget.

The third step is the creation of wireframes and prototypes. Wireframes, or structural drawings of a web page, are created to visualize the layout of elements and functionality of a future product. They are a key tool in evaluating and detailing the design.

After the weirframes are approved, the design process begins, including the application of the client's corporate style, color scheme and adaptive layout, which makes the site convenient for viewing on different devices.

The fifth stage is the development of the server part. At this stage the server part is installed and configured, for example, on the Drupal platform. Then the developers customize the necessary modules and perform the initial integration with the frontend. After that, frontend developers realize the visual aspects and interactivity of the site, ensuring its correct operation in different browsers and on different devices.

Next, testing and quality assurance needs to be done. And after testing is completed and all detected bugs are eliminated, the team makes final customizations and launches the product. The final stage includes training the client to manage the site, add content and other necessary operations. The team continues to maintain the site by resolving issues that arise and updating it [5].

The web application development process involves several stages, each of which is an important element in creating a successful product.

EVOLUTION OF WEB APPLICATION RENDERING TECHNIQUES

Traditional websites displayed on the server side represent one of the earliest types of web applications. They predominantly served to display static text and images with minimal interactivity. In such sites, HTML code, along with the necessary data, was generated by the server and sent back to the browser for display.

Each time the page was refreshed or navigated to another page, the server sent a new HTML code. This was repeated every time, except when the page was cached in the browser. This approach could slow down the loading of the site because the server was re-generating the HTML each time, even with minimal changes to the content.

Factors such as internet connection speed, server location, and the volume of simultaneous requests from users also affected site speed. While this wasn't as critical for small sites, large modern sites with thousands of lines of code and complex logic suffered from latency.

However, the main advantage of server-side rendering is its compatibility with SEO, as content is indexed by search engines before the user gets it.

Modern single-page applications (SPAs) use client-side rendering. In a SPA, the browser loads the initial page along with all the necessary scripts, styles, and resources once. Transitions between pages occur without reloading, thanks to the HTML5 History API mechanism that updates the URL in the browser's address bar.

The client side of SPA is developed independently, receiving data through the API. This separation allows the development of separate clients for different platforms, using different technology stacks, without changing the server side. This reduces the number of HTTP requests and reduces the size of the data transferred, speeding up processing.

Disadvantages of SPA include difficulties with SEO, as dynamically generated content is difficult for search engines to index. Large initial downloads can also slow down application response times, especially on devices with limited processing power.

In addition, when SPAs are developed by large teams, the diversity of approaches and solutions can complicate work consistency and code understanding. This requires additional effort to organize and maintain a clean codebase.

Isomorphic applications, or hybrid approaches, combine the benefits of server-side and client-side rendering to create web applications that load quickly and are well-optimized for search engines. These applications can run on both the client and server using common code, making them easy to develop and maintain.

On the first visit, the application is generated on the server using server-side rendering technologies such as Node.js and then passed to the browser. Then, as the user navigates through the application, subsequent pages are rendered on the client side using single-page application (SPA)-style JavaScript. This provides instant content updates via the API without the need for a full page reload.

One of the challenges of the isomorphic approach is managing the state between the server and the client. An effective solution is to create a state on the server, and then pass it to the browser, which uses that state to initially load the SPA. This reduces user wait time as the server page is displayed instantly and the dynamic client logic continues to run without delay.

To optimize, minimize the amount of content transferred on the first load by including only necessary elements such as inline CSS and minimal HTML. This speeds up the initial load, allowing a faster transition to client-side dynamic rendering.

The hybrid approach also solves routing issues by allowing server-side rendering for initial loading and client-side routing for internal navigation control. In this way, flexibility in loading application components can be achieved, which is important for maintaining high application performance when scaling.

However, isomorphic applications may face scalability issues when the number of users is large. Efficient caching on the server can minimize these issues, speeding up page display and improving overall performance.

Micro-frontend architecture, similar to microservices, aims to break monolithic applications into smaller, independently developed, and deployable components. This approach allows teams to develop and update different parts of the application independently, improving flexibility and speeding up development and deployment processes.

Application partitioning can be done either horizontally or vertically, depending on functional requirements and team structure. Each micro frontend can be designed for a specific business logic or user process, simplifying code management and maintenance.

Key aspects of micro-frontend architecture include component definition, micro-frontend composition, routing, and communication between them. This requires careful planning and coordination, as each component must integrate with the rest of the system while maintaining its independence.

In the context of horizontal architecture partitioning, web pages are decomposed into multiple micro-interfaces, each developed by different teams. This strategy improves flexibility, as individual micro-interfaces can be reused in different parts of the application. However, it also entails the need for tighter governance and coordination between teams to prevent unnecessary interface fragmentation.

In the case of vertical separation, different aspects or modules of the application, such as authentication, streaming services, or search functions, are developed by separate teams. This allows each team to specialize and optimize its segments without cross-dependencies with other parts of the application [6].

JAMStack (JavaScript, API, Markup) is a modern architectural practice that significantly improves the process of building websites and applications. This approach combines JavaScript, the use of APIs, and static markup to create dynamic websites that can be served without direct dependence on traditional web servers.

The result of a JAMStack implementation is a static artifact containing HTML, CSS, and JavaScript, the core components of web development. These resources can be efficiently hosted and delivered via content delivery networks (CDNs), significantly speeding up page loads and reducing the load on servers.

Key benefits of JAMStack include improved performance and lower infrastructure costs due to the ability to serve over CDNs. Also, the static nature of the files provides excellent

scalability and increases security by reducing potential attack points. Integration with headless CMSs further simplifies content management and development [7].

CONCLUSION

The study emphasizes the importance of selecting the optimal architecture for front-end development, which directly impacts the success of web applications. The analyzed approaches demonstrate that there is no one-size-fits-all solution: each project necessitates an individual approach based on its specifications and objectives. Modular architecture and the use of micro-frontends offer significant advantages in terms of management and scalability, while hybrid approaches optimize performance and enhance SEO.

For large and complex applications, developers should consider employing a micro-frontend architecture to improve maintainability and scalability. This approach allows different parts of the application to be developed and deployed independently, facilitating better modularity and team collaboration. To optimize performance and SEO, adopting hybrid approaches that combine server-side and client-side rendering is recommended. This method ensures quick initial load times and a seamless user experience, which are crucial for modern web applications.

It is crucial for the chosen architecture to be adaptable to evolving requirements and technological innovations. This flexibility will help maintain the competitiveness of web applications in a rapidly changing digital landscape. Implementing micro-frontend architecture requires meticulous planning and coordination among teams. Ensuring that each component integrates seamlessly with the rest of the system while maintaining its independence is essential for successful deployment.

Effective application of architectural solutions necessitates a profound understanding of both technical aspects and the business needs of the project. Adaptability to changing requirements and the ability to innovate are paramount for sustaining the competitiveness of web applications in a dynamic digital world.

REFERENCES

1. Revolutionizing Web Experiences: 7 Powerful Frontend Architecture Strategies For Unmatched Performance. [Electronic resource] Access mode: https://bitbytesoft.com/web-experiences-with-frontend-architecture/ (accessed 8.05.2024).
2. Front-End Development Trends for 2024. [Electronic resource] Access mode: https://www.dronahq.com/front-end-development-trends / (accessed 8.05.2024).
3. A different approach to frontend architecture. [Electronic resource] Access mode: https://dev.to/itswillt/a-different-approach-to-frontend-architecture-38d4 (accessed 8.05.2024).
4. Modern Frontend Architecture: A Guide to Key Concepts. [Electronic resource] Access mode: https://www.javacodegeeks.com/2024/04/modern-frontend-architecture-a-guide-to-key-concepts.html (accessed 8.05.2024).
5. Web development methodologies and approaches. [Electronic resource] Access mode: https://www.adcisolutions.com/knowledge/web-development-methodologies-and-approaches (accessed 8.05.2024).
6. The frontend Landscape – Different Architectures. [Electronic resource] Access mode: https://learnersbucket.com/examples/web/the-frontend-landscape-different-architectures/(accessed 8.05.2024).
7. The Front End Developer/Engineer Handbook 2024. [Electronic resource] Access mode: https://frontendmasters.com/guides/front-end-handbook/2024 / (accessed 8.05.2024).

Citation: Nikhil Badwaik, "Effectiveness of Different Approaches to Organizing Frontend Development", American Research Journal of Computer Science and Information Technology, Vol 7, no. 1, 2024, pp. 11-15.