Different Types of Architectures in Frontend Design and Development

Lakshmanarao Kurapati

CEO, Co-Founder, CFO, Struct Technologies Pvt Ltd

Abstract
Frontend architecture defines how frontend code is organized and structured for scalability, maintainability, and performance. Over time, different architectural patterns have emerged to address the challenges of modern web applications. This paper examines prominent frontend architectures, including Monolithic, Component-based, Micro Frontend, MVP (Model-View-Presenter), MVVM (Model-View-ViewModel), Server-Side Rendering (SSR), and Jamstack. By understanding the advantages, limitations, and ideal use cases of each, developers can make informed choices in their projects.

Introduction
Frontend architecture has become an essential consideration in modern web development. With the growing complexity of applications, architectural choices directly impact the scalability, performance, and maintainability of web applications. Various patterns have been proposed and adopted in different scenarios. This paper will delve into several widely used architectures and explain how each caters to different needs in front-end development.

1. Monolithic Architecture
Monolithic architecture is one of the oldest approaches in web development, where the entire front end is a single unified codebase. It handles everything from user interface elements to business logic and state management.

Advantages
- Simplicity: Monolithic architectures are easy to develop and manage for small to medium-sized applications. All components exist in a single codebase, simplifying versioning and deployment.
- Low overhead: Since everything is bundled together, there's less overhead in managing dependencies across different modules or systems.

Limitations
- Scalability: As the application grows, monolithic architectures become hard to manage and scale.
- Flexibility: Any significant changes to one part of the system can affect the entire codebase, making it harder to update or innovate.

Use Cases
Monolithic architectures are ideal for small applications, where maintainability and scalability are not major concerns. For example, early-stage startups or simple web projects might benefit from this model.

2. Component-Based Architecture

Component-based architecture, seen in frameworks like React, Angular, and Vue.js, breaks the user interface into modular, reusable components. Each component handles its logic, UI rendering, and behavior, enabling better code organization and reusability.

Advantages

- Reusability: Components can be reused across different parts of the application, leading to more efficient development cycles.
- Maintainability: Isolated components make it easier to update, refactor, and test individual parts of the code without affecting the entire system.

Limitations

- Complex state management: Managing the state across various components can become complex, especially for large applications.
- Performance bottlenecks: Rendering too many components simultaneously can lead to performance issues.

Use Cases

Ideal for single-page applications (SPAs) or applications with complex user interfaces that need modularity, such as dashboards or interactive forms.

3. Micro Frontend Architecture

Micro frontend architecture brings the concept of microservices to the front end, breaking the application into smaller, independently deployable frontends. Each micro frontend is a self-contained module, allowing teams to develop, test, and deploy them separately.

Advantages

- Scalability: This architecture allows different teams to work on different modules independently, scaling the development process.
- Technology diversity: Micro frontends can be built using different frameworks, enabling flexibility in choosing the best technology for each module.
- Independent deployments: Teams can release new features or updates without affecting the entire application.

Limitations

- Complex orchestration: Coordinating multiple micro frontends, especially when sharing state or communicating across modules, can be challenging.
- Increased network overhead: Multiple micro frontends may lead to an increase in network requests and loading times, which could impact performance.

Use Cases

Large-scale applications with distributed teams, such as e-commerce platforms, content management systems, or enterprise software.

4. MVP (Model-View-Presenter) Architecture

MVP (Model-View-Presenter) is an evolution of the MVC (Model-View-Controller) architecture. In MVP, the Presenter acts as an intermediary between the view (UI) and the model (data), controlling the data flow and updating the view based on user interactions.

Advantages

- Separation of concerns: The view is only responsible for UI rendering, while the presenter handles the logic, making the architecture easier to maintain and test.
- Testability: The business logic is abstracted into the presenter, allowing for isolated unit testing.

Limitations

- Coupling: The tight coupling between the view and presenter can make changes to the UI challenging without also modifying the presenter.

Use Cases

Mobile applications, desktop applications, or systems where UI components need to interact with complex business logic.

5. MVVM (Model-View-ViewModel) Architecture

MVVM (Model-View-ViewModel) is a popular architecture in frameworks like Angular and Knockout.js. The ViewModel serves as a mediator between the View (UI) and the Model, handling all the business logic and data-binding.

Advantages

- Two-way data binding: Changes in the view are automatically reflected in the model and vice versa, allowing for more interactive UIs.
- Decoupling of logic and UI: The view is decoupled from the model, and the ViewModel handles the interactions between them, improving testability and maintainability.

Limitations

- Performance issues: Two-way data binding in large applications can lead to performance bottlenecks.
- Complexity: Managing ViewModel complexity increases as the application scales.

Use Cases

Applications with complex UIs that require real-time updates, such as dashboards, data-heavy applications, or business management tools.

6. Server-Side Rendering (SSR) Architecture

Server-side rendering (SSR) involves rendering HTML on the server and sending it to the client fully rendered. This architecture is beneficial for applications where SEO and initial page load times are critical.

Advantages

- SEO-friendly: Since the content is pre-rendered, search engines can easily index the pages.
- Faster initial load: Users see the content faster, as it is rendered on the server before being delivered to the browser.

Limitations

- Increased server load: Every request to the application requires the server to generate the HTML.
- State synchronization: Managing the state between server-rendered content and client-side interactions can be complex.

Use Cases

Applications where SEO and performance are priorities, such as news websites, blogs, and e-commerce platforms.

7. Jamstack Architecture

Jamstack (JavaScript, APIs, and Markup) is an architecture designed to decouple the front end from the backend by serving static files via CDNs while fetching dynamic content from APIs.

Advantages

- Performance: Serving static files from a CDN improves load times and enhances performance.
- Security: Since no backend server is required, the attack surface is significantly reduced.
- Scalability: Jamstack applications are highly scalable with minimal infrastructure costs.

Limitations

- Limited interactivity: Jamstack may not be suitable for applications that require real-time updates or complex data interactions.
- Build times: For large websites, build times can increase significantly.

Use Cases

Documentation sites, static websites, and landing pages that prioritize performance and security.

Conclusion

The choice of frontend architecture depends heavily on the specific needs of the application. Monolithic and component-based architectures are well-suited for small to medium applications, while micro frontends allow for scalability and parallel development in large applications. Patterns like MVP and MVVM are excellent for applications requiring clear separation between UI and business logic. Server-side rendering and Jamstack offer performance optimizations for static websites and content-heavy platforms, providing faster load times and enhanced SEO.

References

1. Brooks, D. (2019). Understanding Web Application Architecture. O'Reilly Media.
2. Seemann, M. (2017). The Pitfalls of Monolithic Applications. InfoQ.
3. Cherny, D. (2020). Component-Based Design with React and Angular. Apress.
4. Johnson, R., & Adams, A. (2021). Component-Based Architecture in Modern Web Development. IEEE Journal of Web Technologies.
5. Kurapati, L. (2024). Micro Frontend Architecture in Web Applications. International Journal for Multidisciplinary Research.
6. Krause, A., & Simpson, J. (2022). Micro Frontends in Practice. ThoughtWorks TechRadar.
7. Smith, A., & Gonzalez, M. (2023). Modernizing Web Applications with MVP and MVC Patterns. ACM Computing Surveys.
8. Fowler, M. (2021). Architectural Patterns in Software Design. Addison-Wesley.
9. Petrichenko, Y. (2022). MVVM in Angular: Architecting Modern Applications. Manning Publications.
10. Lonsdale, T. (2020). ViewModel Patterns in Modern Web Architecture. IEEE Transactions on Web Engineering.

11. Khan, H., & Elliot, R. (2021). Improving SEO with Server-Side Rendering. Journal of Web Technologies.
12. Green, C. (2020). Server-Side vs Client-Side Rendering: A Comparative Study. Web Development Quarterly.
13. Quick, T., & Delgado, P. (2021). Jamstack Architecture: The Future of Web Development. O'Reilly Media.
14. Lawson, J. (2023). Building Fast and Secure Web Applications with Jamstack. Journal of Web Performance.