Web Rendering Strategies

Software System Design

Spring 2024 @ Ali Madooei

Learning Outcomes

By the end of this lecture, you should be able to:

- Define web rendering strategies and their importance in web development.
- Identify key aspects of web rendering strategies, such as delivery methods, content generation, and user experience.
- Discuss the pros and cons of different web rendering strategies, including Client-Side Rendering (CSR), Server-Side Rendering (SSR), and Static Site Generation (SSG).
- Analyze the suitability of different web rendering strategies for specific web applications and use cases.

A **web rendering strategy** defines the approach used to generate and deliver the content and functionality of a website or web application to a user’s browser for best performance and user experience.

Web Rendering Strategies: Key Aspects

- **Delivery method:** How the content is prepared and sent from the server to the client (browser). This could involve server-side processing, client-side execution, or a combination of both.
- **Content generation:** Where and how the content is created. This can involve pre-generating static content, generating content on the server for each request, or dynamically generating content on the client-side using JavaScript.
- **User experience (UX):** How the website interacts with the user. This includes factors like initial load time, interactivity, and responsiveness to user actions.

Web Rendering Strategies: Importance

Choosing the best web rendering strategy depends on various factors like:

- **Project goals:** Prioritizing SEO, performance, user experience, or content management complexity.
- **Content type:** Whether content is static, dynamic, or requires frequent updates.
- **Target audience:** Optimizing for specific devices, network conditions, or accessibility requirements.

Understanding and selecting the appropriate web rendering strategy plays a crucial role in creating efficient, performant, and user-friendly web experiences.

So many acronyms: CSR, SSR, SSG, ISR, DRP, ESR, SPA, PWA, MPA, ... 🤯

Early Days (1990s) of Web

- **Static Web Pages:** Simple HTML files with no dynamic content.
- **Simple Rendering:** Browser requests a page, server sends HTML, browser renders it.
- **Limited Interactivity:** No real-time updates or dynamic content.
- **Enhanced CSS Use:** Enabled better styling and layout control.
- **Introduction of JavaScript:** Enabled client-side interactivity and dynamic content.

Early 2000s to Early 2010s

- **Dynamic Content:** Introduction of server-side scripting languages like PHP, ASP, JSP.
- **Server-Side Rendering (SSR):** Server processes requests and sends fully rendered HTML (along with CSS and JavaScript).
- **Multi-Page Applications (MPAs):** Each interaction loads a new page from the server.
- **AJAX:** Asynchronous JavaScript and XML for dynamic content updates.
- **Client-Side Rendering (CSR):** Browser renders content using JavaScript.
- **Single Page Applications (SPAs):** Entire app runs in the browser, no full page reloads.

Late 2010s to Present

- **Modern JavaScript Frameworks:** React, Angular, Vue.js.
- **Mobile-First Approach:** Responsive design for mobile devices.
- **Progressive Web Apps (PWAs):** Web apps with native app-like features.
- **JAMstack:** JavaScript, APIs, and Markup for modern web development.
- **Static Site Generation (SSG):** Pre-rendered pages for faster load times.
- **Server-Side Rendering (SSR):** Revival of SSR with modern frameworks.
- **Hybrid Approaches:** Incremental Static Regeneration (ISR), Distributed Persistent Rendering (DPR), Edge Side Rendering (ESR).

Case Study: The "Posts" App

You are developing a social media app where users can post, like, and comment on content. The app should offer real-time updates and interactions, and users expect a smooth and responsive experience.

Case Study: E-commerce Product Catalog

You are building an e-commerce platform where users can browse and search for products. The platform should offer features like product filtering, sorting, and pagination. It’s crucial for the site to be SEO-friendly to attract more customers through search engines.

Case Study: Software Documentation Website

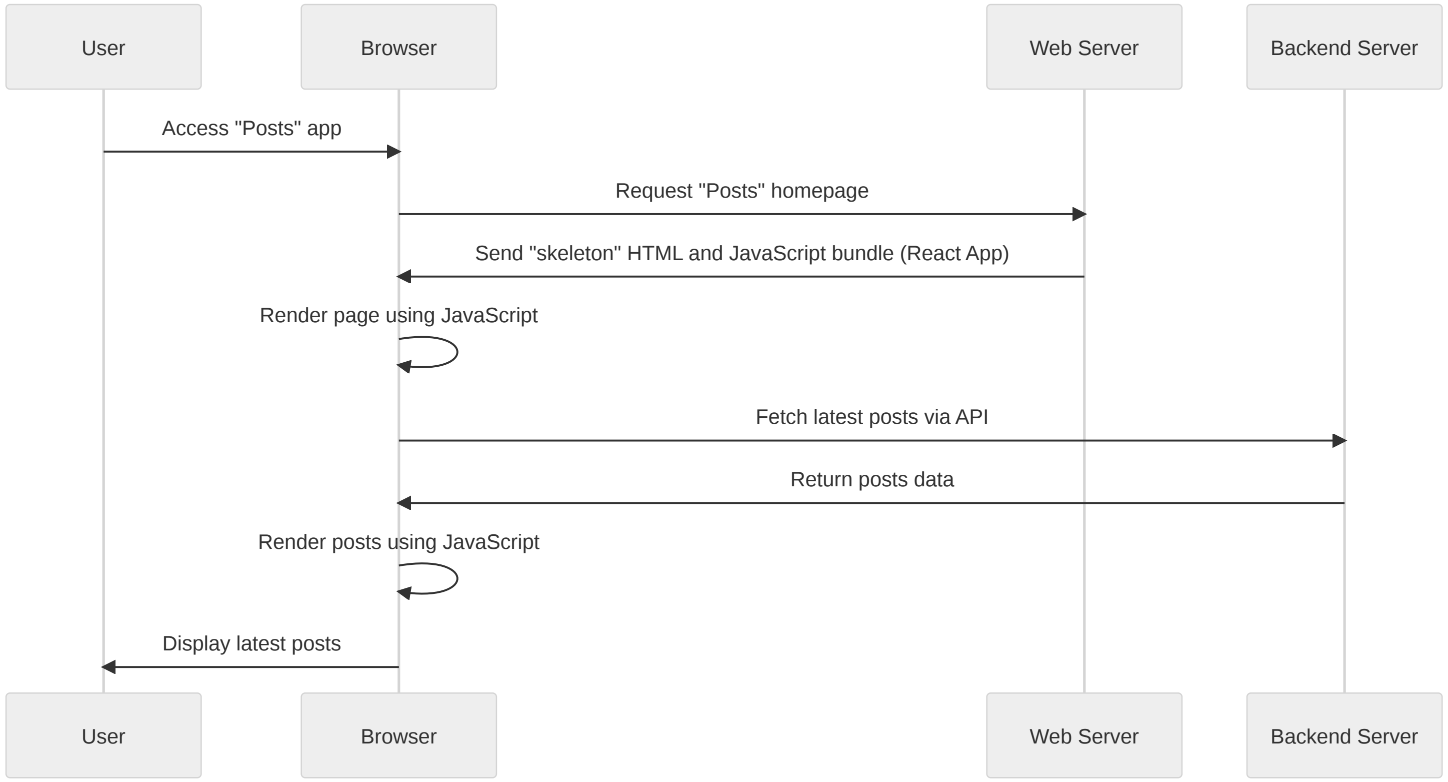
You are developing a documentation website for a software product. The site should offer a comprehensive set of documentation, including guides, tutorials, and API references. It’s essential for the site to be fast, easily navigable, and SEO-friendly.

Client-Side Rendering (CSR)

Client-Side Rendering (CSR) is a technique where the browser renders the web page content using JavaScript.

- Initial load fetches a "skeleton" HTML and CSS, along with a JavaScript bundle.
- JS code in the browser fetches data and dynamically updates the DOM (Document Object Model), allowing for more interactive elements and user actions.
- CSR is great for dynamic web apps with frequent content changes.
- Many modern web apps use CSR, especially those built using React and Vue.js.

CSR Example: The "Posts" App



Single Page Applications (SPA)

Single Page Applications (SPAs) are web applications that load a single HTML page and dynamically update that page as the user interacts with the app.

- Uses Client-Side Rendering (CSR)
- Can reduce the need for full page reloads, improving user experience
- Modern UI libraries like React were designed for SPAs
- Many modern web apps are SPAs: Google Docs, Facebook, Twitter.

CSR & SPA: Pros and Cons

- **Potential Benefit:** Smooth user interactions after initial load, especially for Single Page Applications (SPAs).
- **Potential Challenge:** Slower initial page load and SEO complications without proper configurations.

Search Engine Optimization (SEO)

Search Engine Optimization (SEO) ensures websites rank well on search engines. With CSR, content is rendered dynamically using JavaScript, which can pose challenges.

- Search engines primarily index HTML content.
- Dynamic CSR content may not be fully indexed, so it can reduce search engine visibility.
- **Potential Solution:** Use server-side rendering (SSR) or prerendering for critical pages.

Server-Side Rendering (SSR)

Server-Side Rendering (SSR) is a technique where the server renders the web page content before sending it to the browser.

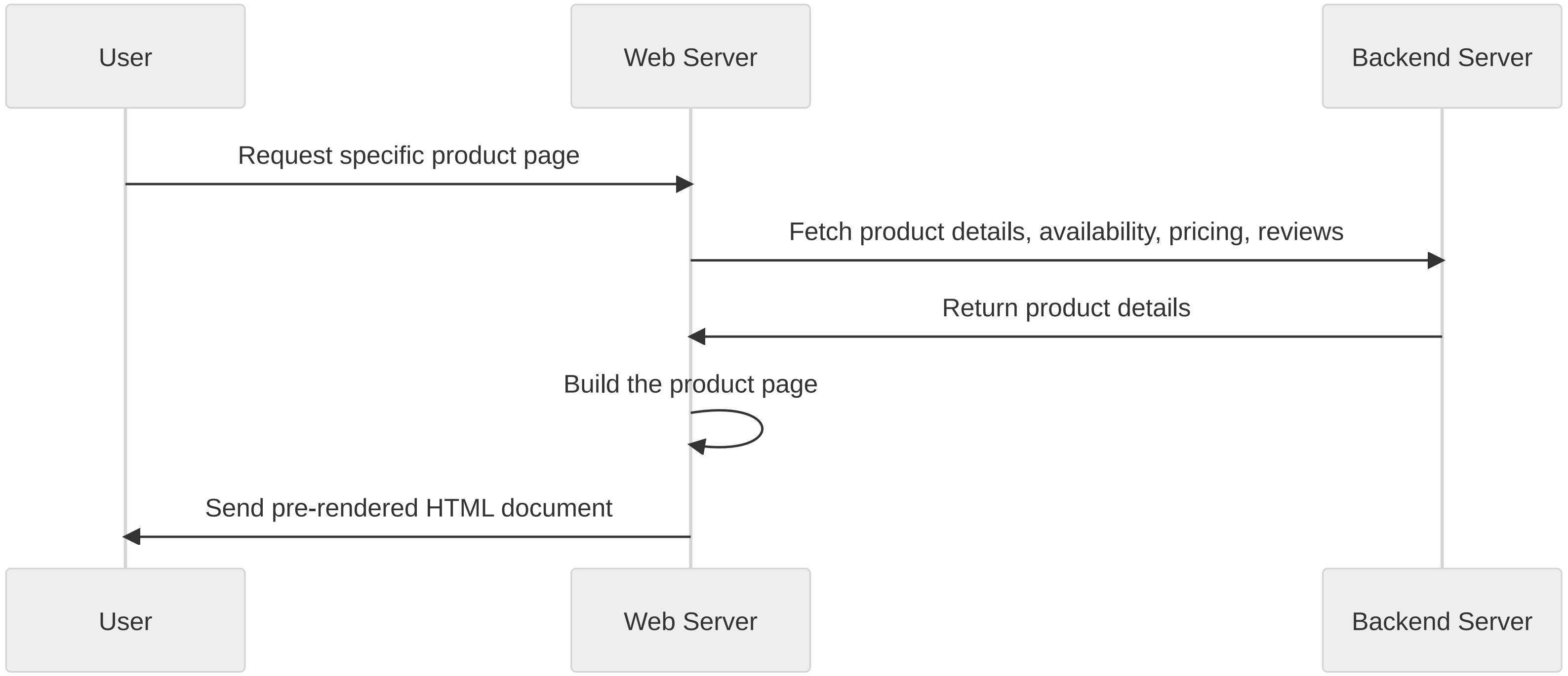
- Allows for immediate page view, improving perceived performance.
- Enhances SEO as content is pre-rendered, aiding search engines.
- Older web technologies like PHP and ASP used SSR by default.
- Modern frameworks like Next.js and Nuxt.js facilitate SSR with React and Vue.js.

SSR Example: E-commerce Product Catalog

E-commerce websites are prime candidates for Server-Side Rendering.

- **Dynamic Content:** Up-to-date information on product availability, pricing, and user reviews.
- **Interactive Elements:** Sections like "Customers also bought" change based on user behavior.
- **SEO:** Crucial for product pages to drive sales and attract potential customers.

SSR Example: E-commerce Product Catalog


User
Web Server
Backend Server
Request specific product page
Fetch product details, availability, pricing, reviews
Return product details
Build the product page
Send pre-rendered HTML document
User
Web Server
Backend Server

SSR Activity: The "Posts" App

Consider the "Posts" app using Server-Side Rendering:

- How would the Posts app work with SSR?
- What would be the potential benefits and challenges of using SSR for the app?

SSR Activity Solution

The user would request a specific post, and the server would fetch the post details from the database, build the page, and send the pre-rendered HTML document to the user.

Considering the "Posts" app’s highly dynamic content, SSR might not be the optimal choice.

- **Server Load:** Increased server processing for each user request.
- **User Experience:** Delay in content visibility as updates wait for server processing.
- **SEO Implications:** While SSR improves SEO, the dynamic nature of "Posts" and frequent server requests might neutralize the benefits.

Multi Page Applications (MPA)

Multi Page Applications (MPAs) load a new HTML page from the server every time a user navigates or interacts with the application.

- Uses Server-Side Rendering (SSR)
- Designed for traditional navigation and user flow
- Suitable for large-scale projects with diverse content
- E-commerce sites, online banking platforms, and enterprise applications often use MPAs.

SSR & MPA: Pros and Cons

- **Potential Benefit:** Better initial load time and SEO benefits compared to CSR.
- **Potential Challenge:** Slower navigation and more server requests.

Static Site Generation (SSG)

Static Site Generation (SSG) is a method where web pages are generated at build time, not during run-time.

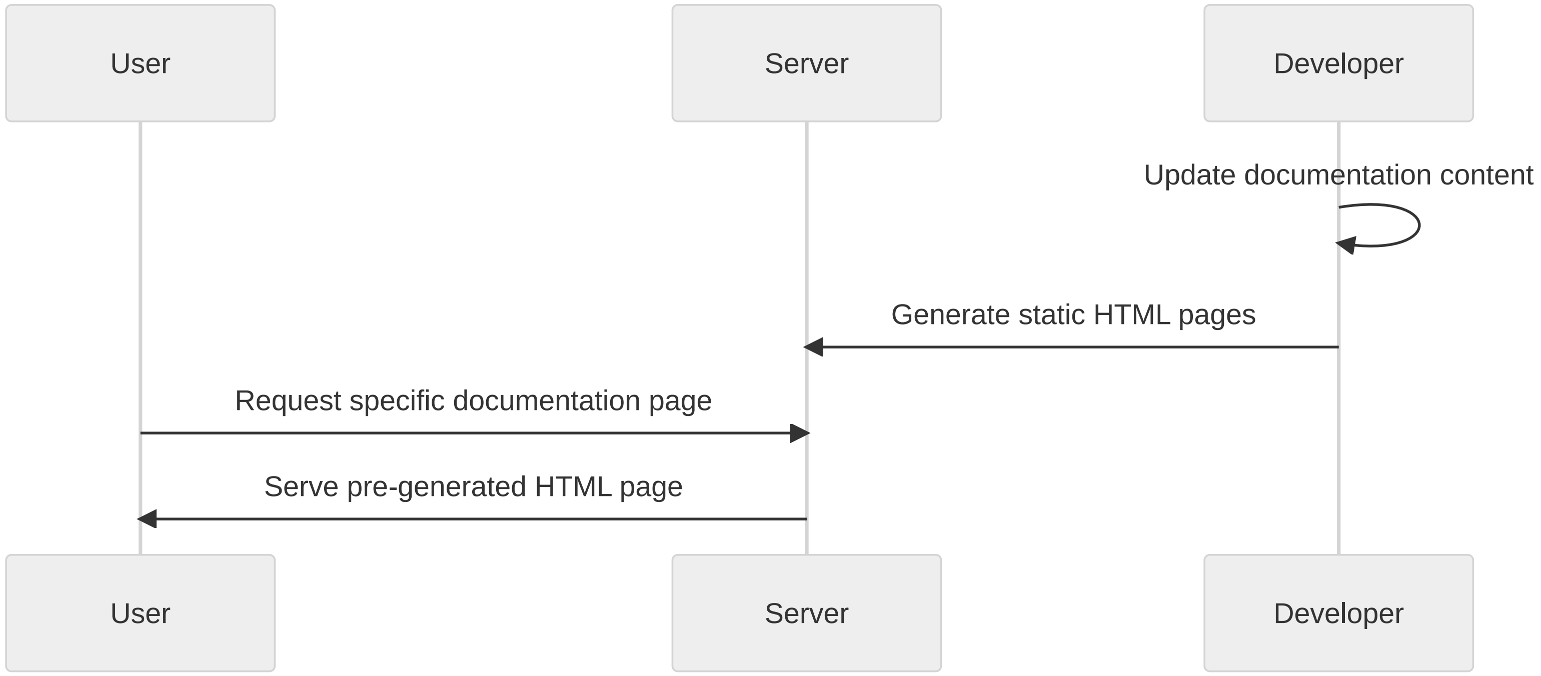
- Web pages are pre-built and ready to serve.
- Best suited for sites where content changes infrequently.
- In a way the early mimics the early days of the web, but with modern tools and frameworks.
- Blogs, documentation websites, and marketing pages are ideal candidates for SSG.

SSG Example: Documentation Website

Documentation websites are an excellent fit for Static Site Generation.

- **Stable Content:** Documentation remains consistent until product updates.
- **Fast Load Times:** Pre-generated pages ensure quick access without server processing.
- **SEO:** Important for tech products to be discoverable and indexable by search engines.

SSG Example: Documentation Website


User
Server
Developer
Update documentation content
Generate static HTML pages
Request specific documentation page
Serve pre-generated HTML page
User
Server
Developer

SSG Activity: The "Posts" App

Consider the "Posts" app using Static Site Generation:

- How would the Posts app work with SSG?
- What would be the potential benefits and challenges of using SSG for the app?

SSG Solution

The user would request a specific post, and the server would serve the pre-generated HTML page.

Considering the "Posts" app’s highly dynamic content, SSG might not be the optimal choice.

- **Content Updates:** Frequent rebuilds required for new posts and interactions.
- **User Experience:** Delay in content visibility as updates wait for rebuilds.
- **SEO Implications:** While SSG offers SEO benefits, the dynamic nature of "Posts" and frequent rebuilds might neutralize the benefits.

JAMStack: Modern Web Development

JAMStack stands for JavaScript, APIs, and Markup.

- **JavaScript:** Any dynamic functionalities are handled by JavaScript, running entirely on the client.
- **APIs:** Server-side operations are abstracted into reusable APIs, accessed over HTTPS with JavaScript.
- **Markup:** Templated markup should be pre-built at deploy time, usually using a site generator.

JAMStack is a modern architecture that leverages SSG, APIs, and Markup for a better user experience.

SSG & JAMStack: Pros and Cons

- **Potential Benefit:** Faster load times, improved SEO, and better developer experience.
- **Potential Challenge:** Not ideal for highly dynamic content; rebuild required for updates.

Modern/Hybrid Rendering Approaches

- **Incremental Static Regeneration (ISR) by Vercel**:
  - Merges SSG speed with SSR flexibility.
  - Regenerates specific pages, not the entire site.
- **Distributed Persistent Rendering (DPR) by Netlify**:
  - Offers JAMstack benefits with dynamic updates.
  - Pre-renders only critical pages, creates others on demand.
- **Edge Side Rendering (ESR)**:
  - Utilizes CDN’s edge servers for rendering.
  - Combines benefits of SSR and SSG for faster delivery.

Progressive Web Apps (PWA)

Progressive Web Apps (PWAs) combine the best of web and mobile apps to deliver a native-app-like user experience in a web browser.

- Can work offline and provide push notifications
- Use Service Workers for background tasks and caching
- Can use any rendering method – SSR, CSR, or SSG
- **PWA Examples:** Twitter Lite, Pinterest, Uber’s web client.
- **Potential Benefit:** Engaging user experience with reduced data usage.

Service Workers

Service workers are part of modern web APIs, acting as background scripts that enable enhanced web app capabilities.

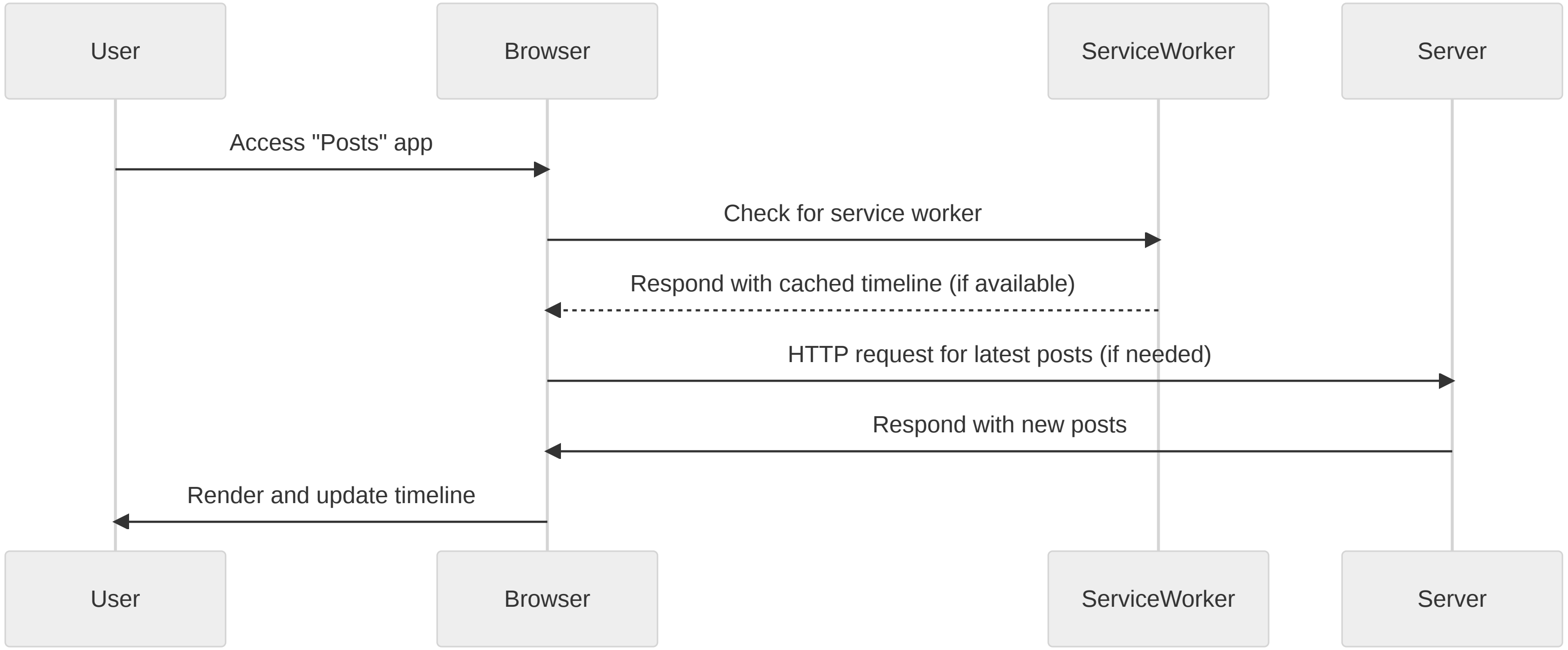
- Run separately from your web page and fully asynchronous.
- Operate in the background without needing a web page or user interaction.
- Enable caching, background sync, push notifications, and custom responses to network requests.
- Require HTTPS due to their ability to intercept and modify network requests.

"Posts" App as a PWA

Would "Posts" benefit from being a PWA?

- **User Engagement**: Push notifications for new posts or interactions.
- **Offline Experience**: Access and read posts even without an internet connection.
- **Performance**: Faster load times with cached content.

"Posts" as a PWA: Sequence Diagram



Activity: Travel Blog Platform

You are developing a platform for travel enthusiasts to create and share their travel blogs. The platform should allow users to write articles, upload photos, and share their travel experiences with a community of like-minded individuals. The site should be visually appealing, easy to navigate, and optimized for search engines to attract more visitors.

List some relevant web rendering strategies for the travel blog platform.

Activity Solution

- **Static Site Generation (SSG)**: For fast loading of blog pages and improved SEO.
- **JAMStack**: For a modern architecture that leverages SSG, APIs, and Markup for a better user experience.
- **Progressive Web Apps (PWA)**: To allow users to read articles offline and enhance mobile user experience.

Activity: Real-Time Stock Market Dashboard

You are tasked with creating a dashboard that displays real-time stock market data. Users should be able to view live updates of stock prices, market trends, and news. The dashboard needs to be highly responsive and capable of handling real-time data updates without refreshing the entire page.

List some relevant web rendering strategies for the stock market dashboard.

Activity Solution

- **Single Page Applications (SPA)**: For a seamless user experience with real-time updates.
- **Client Side Rendering (CSR)**: To dynamically update the stock prices on the client side without server round trips.

Activity: Online Food Ordering System

You are developing a web application for a restaurant chain that allows customers to browse the menu, customize their orders, and make payments online. The application should be fast, user-friendly, and capable of handling high traffic during peak hours.

List some relevant web rendering strategies for the online food ordering system.

Activity Solution

- **Server Side Rendering (SSR)**: For faster initial page loads and improved SEO for the restaurant’s menu.
- **Multi-Page Applications (MPA)**: To separate different functionalities (menu browsing, order customization, payment) into distinct pages for better performance and scalability.

Activity: Personal Finance Tracker

You are creating a web application that allows users to track their expenses, set budgets, and visualize their financial data through charts and graphs. The app should be secure, private, and provide a smooth user experience.

List some relevant web rendering strategies for the personal finance tracker.

Activity Solution

- **Progressive Web Apps (PWA)**: To enable offline access to financial data and push notifications for budget alerts.
- **Client Side Rendering (CSR)**: To update the financial charts and graphs in real-time without server requests.

Conclusion

- Web rendering strategies play a crucial role in delivering efficient, performant, and user-friendly web experiences.
- Key aspects of web rendering strategies include delivery methods, content generation, and user experience.
- Different web rendering strategies, such as Client-Side Rendering (CSR), Server-Side Rendering (SSR), and Static Site Generation (SSG), have their own pros and cons.
- The choice of web rendering strategy depends on project goals, content type, target audience, and other specific requirements.