



Full Stack Application Development

BITS Pilani



Module 8: Accessibility and Performance

Agenda



- ☐ Inclusive Design
- ☐ Web content accessibility Guidelines
- ☐ Optimizing Websites for Accessibility
- ☐ Testing Accessibility
- ☐ Tools and Metrics for Measuring Performance
- ☐ Options for Optimization
- ☐ Caching- Client side, Server side
- ☐ Minifying Code, Compressing files Lazy Loading

Inclusive Design



Inclusive Design



Inclusive design, also known as universal design, is a design process that makes products, services, and environments usable for as many people as possible, especially groups that are typically excluded.

It involves making informed decisions based on user diversity, which includes different needs, capabilities, and aspirations.

Inclusive design may address topics such as **age, culture, gender, geographic location, language, race, economic situation, and education.**

Principles of Inclusive Design



- Increase the usability of products for everyone
- Make products more marketable to a wider audience
- Reduce the need for assistive technologies
- Create a more positive user experience for everyone

Inclusive Design



User-Centric Design

- Inclusive design starts with understanding the needs of the user. This means involving users in the design process from the beginning.

Accessibility

- Inclusive design ensures that products are accessible to people with disabilities. This includes things like providing alternative ways to access information and interact with products.

Usability

- Inclusive design makes products easy to use for everyone. This means using clear and simple language, designing interfaces that are easy to understand, and providing appropriate instructions.

Inclusive Design



Flexibility

- Inclusive design allows for customization and adaptation. This means that products can be used by people with a variety of needs.

Affordability

- Inclusive design considers the cost of products and services. This means that products should be affordable for everyone.

Inclusive Design



Text Legibility and Dark Mode for Older Users



During one [research with seniors](#), one participant remarked, “One thing I don’t like about computers is they make the type too small on the screen.”

To ensure [legibility](#), designers must use reasonably **large font sizes**, have [high contrast](#) between characters in the foreground and background, and use a **clean typeface**.

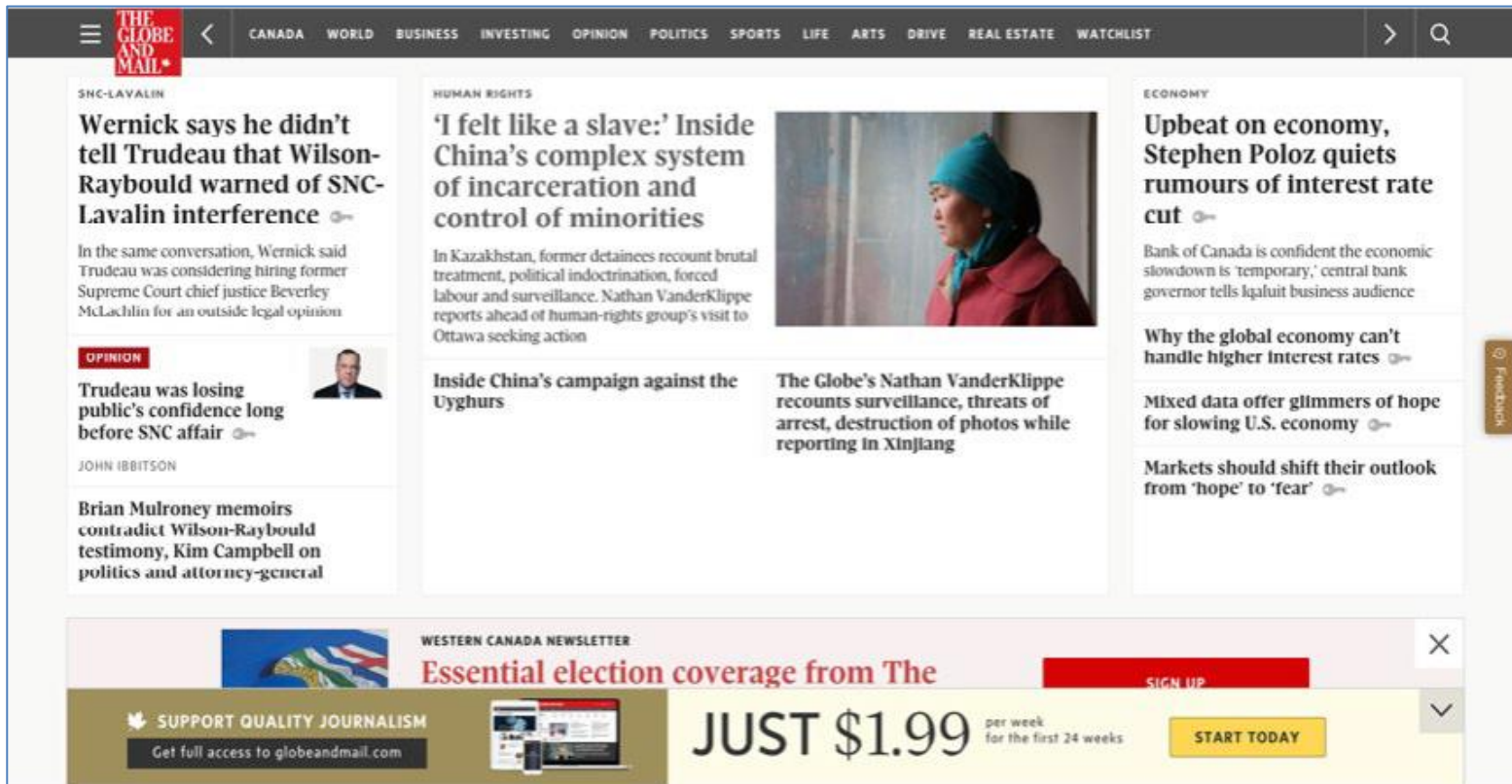
These characteristics are important to all users; however, older users are particularly challenged by interfaces with poor legibility because presbyopia, a form of farsightedness, is common after middle age.

Text Legibility

innovate

achieve

lead



An 81-year-old participant browsed recent articles on the Globe and Mail website but struggled to read the text on the screen. While reading headlines, he lost his place multiple times (due to the densely organized text) and leaned closer to the laptop screen to see the text better.

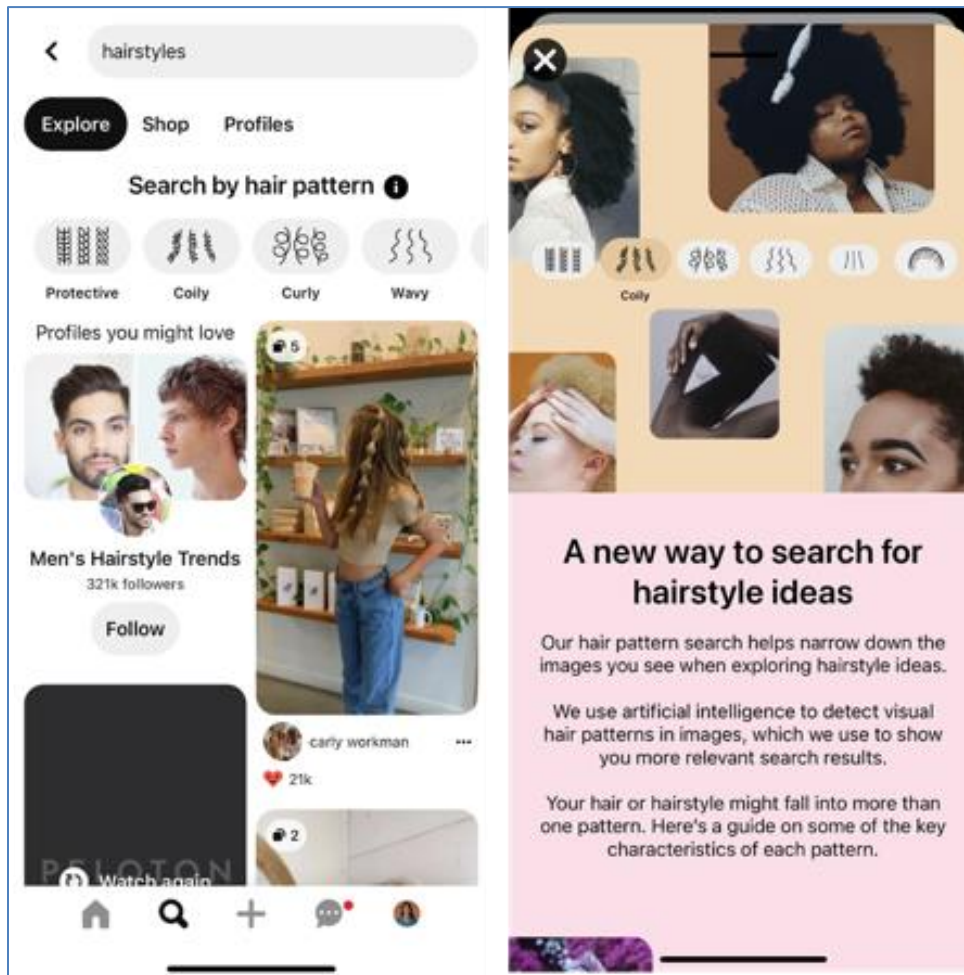
Diverse Illustrations

AirBnb



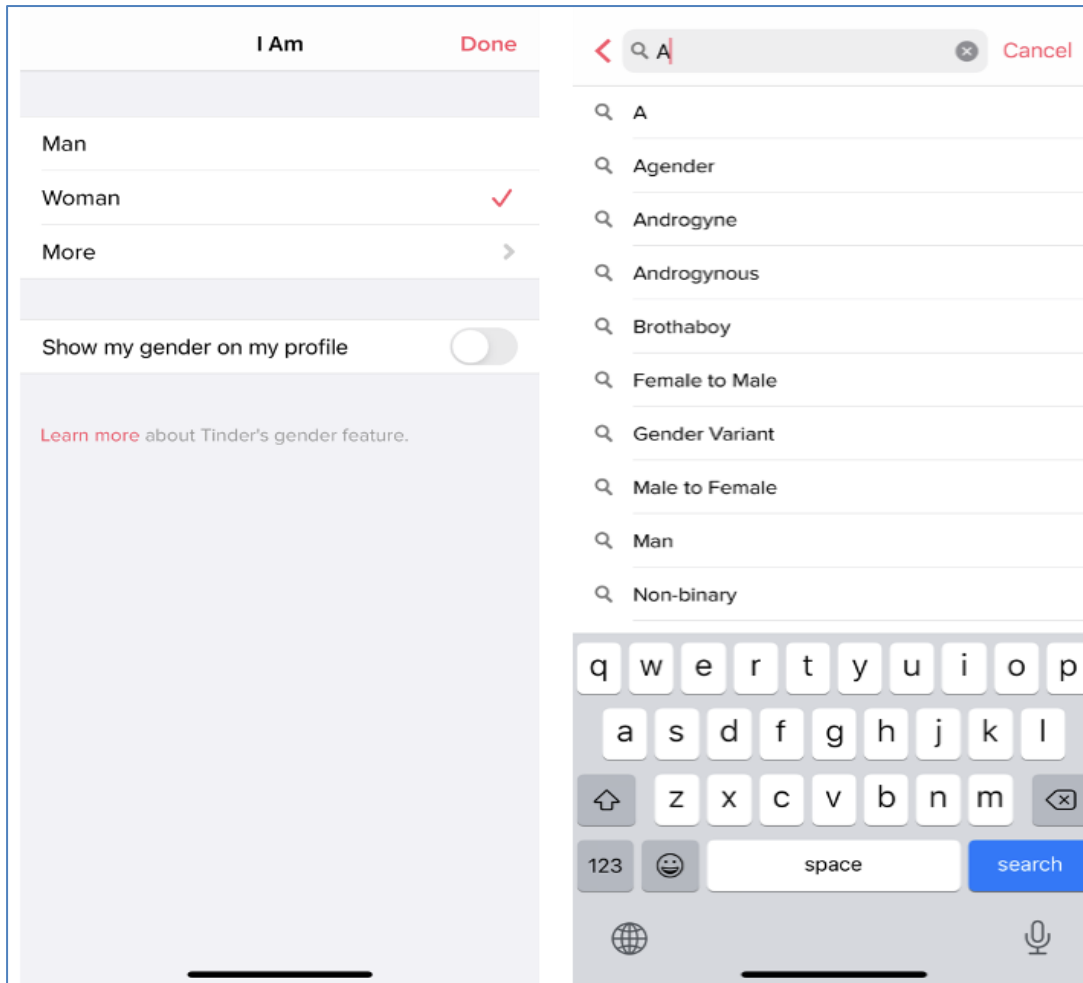
A before (left) and after (right) of Airbnb's illustration evolution, which now prioritizes inclusion and diversity. (Sourced from Airbnb <https://airbnb.design/your-face-here/>)

Inclusive Facets



Pinterest, an image-sharing app, released a new feature which allowed users to [filter](#) based on some nontraditional options — for example, by hair pattern (e.g., *protective*, *curly*, *straight*). Though this pattern may seem like a minor addition, it is empathetic to Pinterest's diverse user base and created an inclusive browsing experience.

Demographic Identifiers



Tinder, a mobile-dating app, allowed users to select from a variety of gender identities. The interface also enabled users to share their gender on their profile, even though this setting was off by default.

7 principles of inclusive design

innovate

achieve

lead



1. Recognise
exclusion is often
subconscious



2. Build inclusive
design thinking in
from the start



3. Engage and
research widely



4. Learn from
diversity



5. Make services
feel familiar



6. Provide easy
access to human
assistance



7. Keep
everything
simple

Case Study



Imagine you're tasked with redesigning a workplace kitchen to be more inclusive for all employees.

Consider factors like:

- **Physical limitations** (mobility issues, reaching high shelves)
- **Sensory sensitivities** (bright lights, loud noises)
- **Dietary restrictions** (allergies, cultural preferences)
- **Lack of clear instructions** (confusing labels, unclear microwave settings)
- **Socio Economic factors** (limited access to storage space, ability to bring lunch)

Ideas : Reimagine the space



- ❑ Lower shelf space and install pull-out drawers for easier access.
 - ❑ Implement adjustable lighting and designated quiet areas.
 - ❑ Stock the pantry with a variety of dietary options and clear labels.
 - ❑ Use universal symbols and pictograms on appliances instead of text-based instructions.
 - ❑ Provide mini fridges or lockers for employees who bring lunch.
-
- ❖ Employees who use wheelchairs (ensure accessible countertops and appliances)
 - ❖ Employees who are neurodivergent (provide designated quiet spaces)
 - ❖ Employees with visual impairments (use contrasting colors, Braille labels)

Web content Accessibility Guideline



<https://www.w3.org/TR/WCAG21/>

Optimizing for Accessibility SEO



Optimize page titles, use headings, include alt text, and improve readability. SEO can help a website be fast, easy to navigate, and relevant to its target audience.



1: How does accessibility impact SEO?

While accessibility is not an SEO ranking factor, they work together. Accessibility helps search engine algorithms understand website content better to increase the chances of the site getting found.

2: How important is an accessible web design for accessibility and SEO?

Web accessibility is essential for SEO. It makes content more accessible for users and crawlers. Accessibility ensures everyone, including those with disabilities, can find and understand your content.

3: Does accessibility have no impact on SEO?

Accessibility does affect SEO, making the site easier for Google to rank and improve your SEO rank report.

Optimizing for Accessibility

Page Speed Insights



This tool can provide insights on how to improve a website's accessibility, performance, and user experience. Improving page speed can also help a website rank higher on Google.

<https://pagespeed.web.dev/>

Optimizing for Accessibility

Others



☐ **Automated tools**

These tools can help prevent issues that affect a website's performance, such as broken links, slow page speed, and accessibility issues.

☐ **Testing**

A combination of automated and manual testing can help identify accessibility issues and ensure that a website is accessible to all users.

☐ **Content audits**

These audits can help organizations identify issues that prevent content from being accessible to users with disabilities.

☐ **Alt text**

Descriptive alt text for images can improve accessibility for visually impaired users and help search engines index images correctly.

☐ **Color**

Colors can help users with visual impairments or color blindness navigate and use a website.

Accessibility Testing Tools



WAVE Web Accessibility Evaluation Tool (<https://wave.webaim.org/>)

<https://www.totalvalidator.com/index.html>

<https://www.w3.org/WAI/test-evaluate/tools/list/>



BITS Pilani
Pilani Campus



Module 8: API Performance

Response Time and Latency:

- This is a fundamental metric that gauges how long it takes for your API to respond to a request. Ideally, you want response times to be as low as possible for a seamless user experience.
- Consider using percentiles (e.g., 95th percentile) to get a better understanding of response time distribution. This way, you can identify potential bottlenecks that might be affecting a small but significant portion of users.

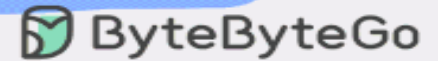
Throughput:

- Throughput measures your API's capacity to handle incoming requests and deliver responses within a specific timeframe. It's often measured in requests per minute (RPM) or transactions per second (TPS).
- Monitoring throughput helps you identify potential issues like system overload due to spikes in traffic or unexpected usage patterns. Setting up alerts on throughput outliers can help you proactively address these situations.

Errors and Error Rates:

- Track the number and types of errors your API encounters. This includes status codes, exceptions, and other irregularities.
- By analyzing error rates, you can pinpoint areas for improvement and ensure your API is functioning reliably.

How to Improve API Performance ?



Pagination



- on ordinal numbering of pages
- handles alarge number of results



Async Logging



- send logs to a lock-free ring buffer and return
- flush to the disk periodically
- higher throughput and lower latency



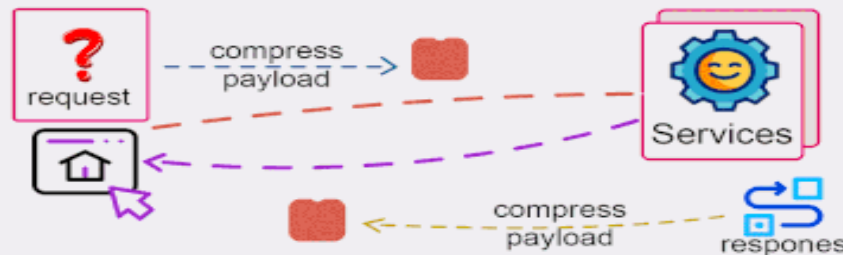
Caching



- store frequently used daya in the cache instead of database
- query the database when there is a cache miss



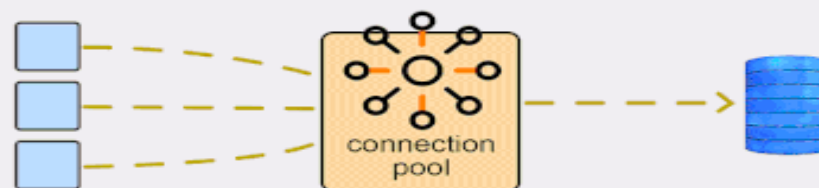
Payload Compression



- reduce the data size to speed up the download and upload



Connection Pool



- opening and closing DB connections add significant overhead
- a connection pool maintains a number of open connections for applications to reuse

API Performance



Pagination:

Handling large result sets efficiently is a common challenge in API development. Result pagination is a technique that addresses this issue by streaming data back to the client in manageable chunks. By breaking down large datasets into smaller portions, this method enhances service responsiveness and user experience. Developers can implement pagination by including parameters in API requests to specify the size and offset of data chunks.

API Performance



Asynchronous Logging:

Traditional logging mechanisms can introduce significant I/O overhead, impacting API performance. Asynchronous logging is a strategy that involves sending logs to a **lock-free buffer and returning** immediately, without waiting for the logs to be written to disk. Periodically, the buffered logs are flushed to the disk, reducing the impact on response times. This approach ensures efficient handling of logs without compromising the performance of API calls.

API Performance



Data Caching:

Caching frequently accessed data is a powerful technique to accelerate data retrieval and minimize database queries. By storing data in a cache, such as Redis, developers can avoid redundant database calls for commonly requested information. Clients can first check the cache before querying the database, resulting in faster response times and reduced load on the backend. Data caching is particularly effective for scenarios where certain datasets remain relatively static over short periods.

API Performance



Payload Compression:

Optimizing data transmission is essential for reducing latency in API calls. Payload compression, achieved through techniques like gzip, is a method to minimize the size of requests and responses. By compressing data before transmission, the upload and download processes become quicker, leading to improved API performance. This strategy is particularly beneficial when dealing with large payloads, such as images or extensive JSON responses.

API Performance



Connection Pooling:

Managing database interactions efficiently is vital for optimizing API performance. Connection pooling involves maintaining a pool of open database connections that can be reused, eliminating the overhead of opening and closing connections for each data query. This technique ensures a more efficient use of resources and contributes to faster response times. Connection pooling is especially valuable in scenarios where frequent database interactions are necessary.

Comparative View



Topic	Description	Use Cases	Other Considerations
API Performance	Optimizing speed and efficiency of API requests	High traffic applications, real-time systems	Consider factors like latency, throughput, and response time. Techniques include caching, load balancing, and optimizing database queries.
Pagination	Breaking large datasets into manageable chunks	Displaying search results, list views	Consider usability, flexibility, and performance. Techniques include offset-based pagination, cursor-based pagination, and keyset pagination.
Async Logging	Writing logs asynchronously to improve speed	High-throughput applications, microservices	Avoid blocking operations and potential performance bottlenecks. Implement strategies for error handling, log rotation, and log aggregation.

Comparison (cont.)



Data Caching	Storing frequently accessed data in memory	Improving response time, reducing database load	Consider cache invalidation strategies, cache coherence, and data consistency. Use appropriate cache eviction policies and monitor cache utilization.
Payload Compression	Compressing data to reduce network bandwidth	Large payloads, high network traffic	Consider CPU overhead and compatibility with client-side decompression. Use efficient compression algorithms like GZIP or Brotli.
Connection Pooling	Reusing database connections to minimize overhead	Database-intensive applications, web servers	Configure pool size and timeout settings based on application requirements. Implement connection validation and monitoring to ensure connection health and availability.

Tools for Performance



1. **Postman:** Postman is a popular API testing and development tool that allows you to create and run API tests, monitor API performance, and generate performance reports.
2. **Apache JMeter:** Apache JMeter is an open-source performance testing tool that can be used to measure the performance of APIs by simulating a large number of users making requests to the API.
3. **Gatling:** Gatling is another open-source load testing tool that is often used for testing the performance of APIs. It provides a high-performance, scalable, and easy-to-use framework for load testing.
4. **New Relic:** New Relic offers performance monitoring and analytics tools that can be used to monitor API performance in real-time, track response times, error rates, and other key performance indicators.
5. **Datadog:** Datadog provides monitoring and analytics solutions that can be used to monitor API performance, track latency, errors, and throughput, and identify performance bottlenecks.

Metrics



Metric	Description	Importance	Tools for Measurement
Response Time	Time taken by the API to respond to a request	Core metric for user experience and service reliability	Postman, Apache JMeter, New Relic, Datadog, custom scripts
Throughput	Number of requests processed per unit time	Indicates API capacity and scalability	Apache JMeter, Gatling, New Relic, Datadog, custom scripts
Error Rate	Percentage of requests resulting in errors	Measures service reliability and error handling	New Relic, Datadog, custom scripts
Latency	Time taken for a request to travel round trip	Captures network and server processing delays	Postman, Apache JMeter, New Relic, Datadog, custom scripts

Metrics (cont.)



Concurrency	Number of simultaneous connections or requests being processed	Measures system load and performance scalability	Apache JMeter, Gatling, New Relic, Datadog, custom scripts
CPU Usage	Percentage of CPU utilized by the API	Identifies resource bottlenecks and inefficiencies	System monitoring tools, New Relic, Datadog
Memory Usage	Amount of memory utilized by the API	Indicates memory leaks, inefficient memory usage	System monitoring tools, New Relic, Datadog
Network Usage	Amount of network bandwidth utilized by the API	Identifies network-related performance issues	System monitoring tools, New Relic, Datadog

How to optimize for API Performance?



Caching: Implement caching mechanisms to store frequently accessed data and avoid redundant processing. This can reduce database load and improve response times for repetitive requests.

Caching at Different Layers: Implement caching at multiple layers, including client-side caching, server-side caching, and CDN caching. This reduces the load on backend servers and improves response times for clients.

Compression: Compress API responses to reduce the size of data transferred over the network. This can significantly decrease latency and improve overall performance, especially for clients with limited bandwidth.

Asynchronous Processing: Utilize asynchronous processing for long-running or non-blocking operations. This frees up resources to handle other requests and improves concurrency and responsiveness.

Database Optimization: Optimize database queries and indexing to improve query performance. Use techniques like query optimization, indexing, and database sharding to distribute load and improve scalability.

Aspect	Server-Side Caching	Client-Side Caching	CDN Caching
Location	Caching happens on the server hosting the API	Caching happens on the client-side, typically in the browser	Caching happens on distributed CDN edge servers
Access Control	Easier to control access and permissions	Limited control over client-side cache	Limited control over CDN cache
Cache Management	More control over cache expiration and invalidation	Limited control; relies on browser cache settings	Limited control; relies on CDN cache settings and TTL
Cache Hit Ratio	Can achieve high cache hit ratio with well-configured cache policies	Highly dependent on individual client behavior and cache size	Can achieve high cache hit ratio with CDN edge caching
Network Overhead	Reduced network overhead for subsequent requests	Subsequent requests bypass the network, reducing latency	Subsequent requests served from CDN edge, reducing latency
Customization	Can implement custom cache logic tailored to API requirements	Limited customization options; relies on browser cache controls	Limited customization options; relies on CDN cache policies
Cache Invalidation	Can trigger cache invalidation based on API events or changes	Relies on cache expiration or manual cache clearing by users	Relies on cache expiration or purging by CDN administrators
Scalability	Cache scalability depends on server infrastructure capacity	Scalability depends on client-side device capabilities	CDN caching scales automatically with CDN infrastructure

Summary



- ❑ **Server-Side Caching**: Provides more control over cache management, access control, and cache policies. Ideal for APIs with frequently accessed data and predictable caching requirements. However, scalability depends on server infrastructure capacity.
- ❑ **Client-Side Caching**: Relies on the client's browser to store and manage cached responses. Offers reduced network overhead and faster subsequent requests, but limited control and customization options. Scalability depends on client-side device capabilities.
- ❑ **CDN Caching**: Leverages distributed CDN edge servers to cache API responses closer to end-users. Offers low-latency access to cached content and high cache hit ratios. Limited control over cache policies and cache invalidation. Scalability is automatic and depends on CDN infrastructure.

Choosing the appropriate caching strategy depends on factors such as the nature of the API, access patterns, performance requirements, and scalability considerations. In many cases, a combination of server-side, client-side, and CDN caching may be employed to maximize performance and reliability.

Improve Performance



Minifying Code

Purpose: Minifying code involves removing unnecessary characters such as whitespace, comments, and formatting from source code files without changing their functionality. This technique aims to reduce the size of files transmitted over the network, resulting in faster load times and reduced bandwidth consumption.

Technique: Minification tools strip out unnecessary characters and optimize code structure to minimize file size while preserving its functionality. JavaScript minifiers like **UglifyJS**, CSS minifiers like **cssnano**, and HTML minifiers are commonly used for this purpose.

Impact on Size: Minification can significantly reduce file size, especially for large codebases. It helps optimize network performance by reducing the amount of data that needs to be transferred between the server and the client.

Improve Performance



Compressing Files

Purpose: Compressing files involves reducing the size of static assets like CSS, JavaScript, images, and fonts by applying compression algorithms like GZIP or Brotli. This technique aims to minimize file size and improve load times by reducing network latency and bandwidth consumption.

Technique: Compression algorithms like GZIP and Brotli analyze file content and remove redundancy to achieve compression. GZIP is widely supported and can be configured at the web server level, while Brotli offers better compression ratios but requires modern browser support.

Impact on Size: Compressing files reduces file size by a certain percentage, depending on the content and complexity of the file. The level of compression achieved depends on the compression algorithm used and the configuration settings.

Improve Performance



Lazy Loading

Purpose: Lazy loading defers the loading of non-essential resources until they are needed, reducing initial page load times and improving perceived performance. This technique is especially useful for web pages with large or numerous assets that are not immediately visible to the user.

Technique: Lazy loading is implemented by delaying the loading of certain resources, such as images, scripts, or iframes, until they enter the viewport or are explicitly requested by the user. JavaScript libraries like Intersection Observer or custom lazy loading implementations can be used to achieve this.

Impact on Size: Lazy loading reduces the initial page load time by deferring the loading of non-critical resources. It minimizes the amount of data transferred over the network and improves the perceived performance of the web page.



Module 9:

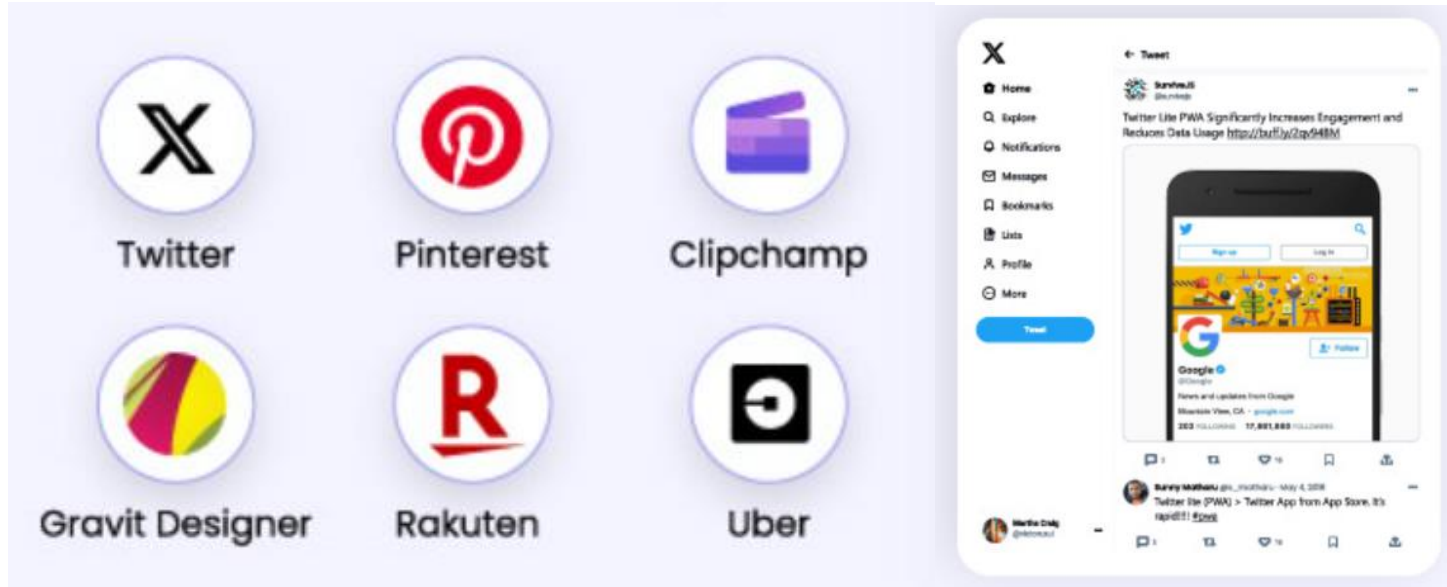
Latest Advancements - PWA

Progressive Web Application

innovate







achieve

lead



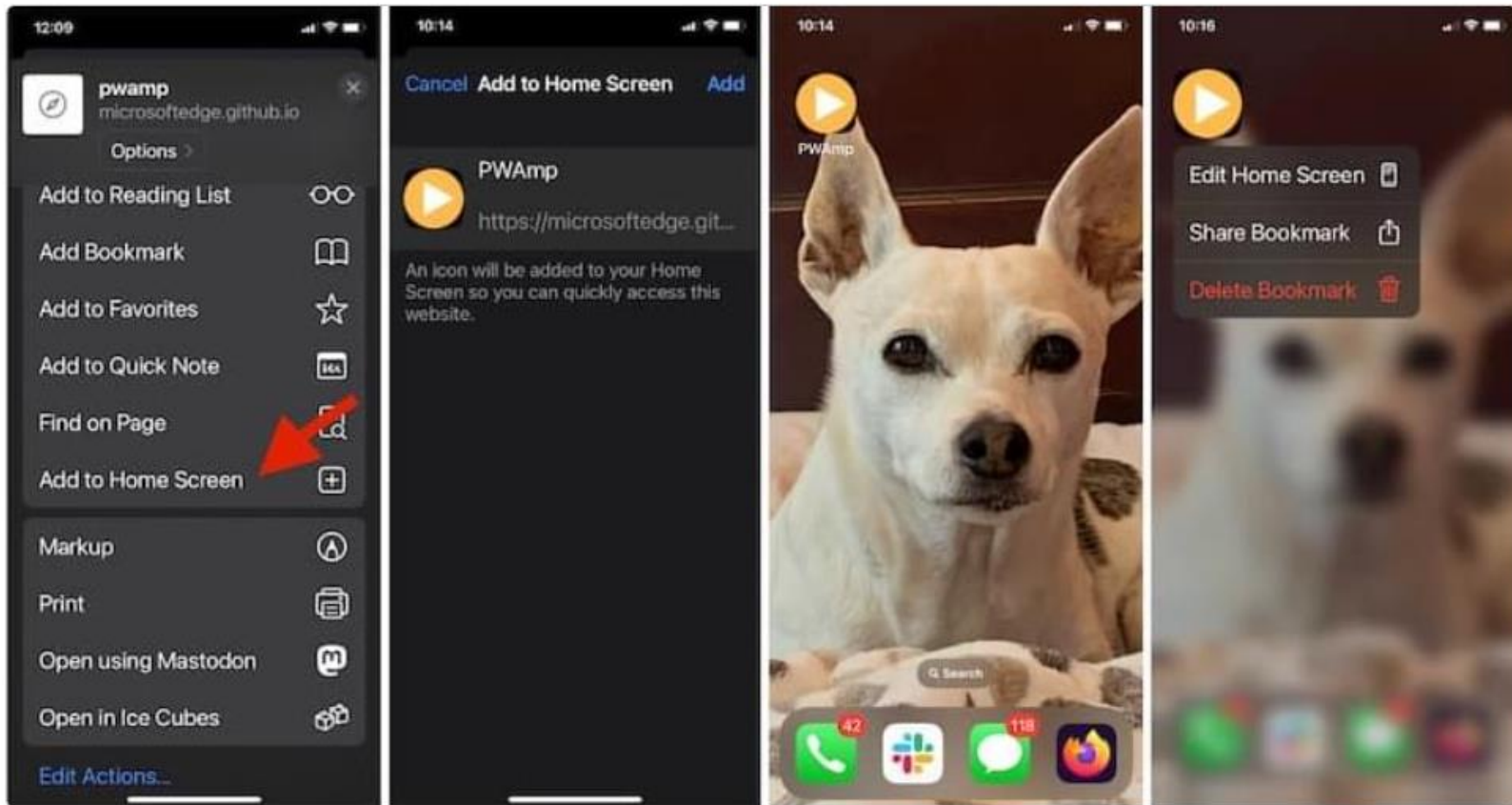
A progressive web application (PWA) is an app that uses web technologies to provide a user experience similar to a platform-specific app. PWAs can run on multiple devices and platforms from a single codebase. They are built using HTML, CSS, JavaScript, and WebAssembly, and are intended to work on any platform with a standards-compliant browser, including desktop and mobile devices.



App	Industry/Sector	Key Features
 <u>Twitter Lite</u>	Social Media	Data-efficient, lower storage requirements
 <u>Clipchamp</u>	Video Editing	Collaborative capabilities, high-quality video exports
 <u>Gravit Designer</u>	Graphic Design	Extensive design tools, cross-platform compatibility
 <u>Rakuten 24</u>	eCommerce	User-friendly navigation, 24/7 accessibility
 <u>Alibaba</u>	eCommerce	Multilingual support, detailed product catalogs
	On-demand	Real-time tracking, multiple drop-off



Installing PWA

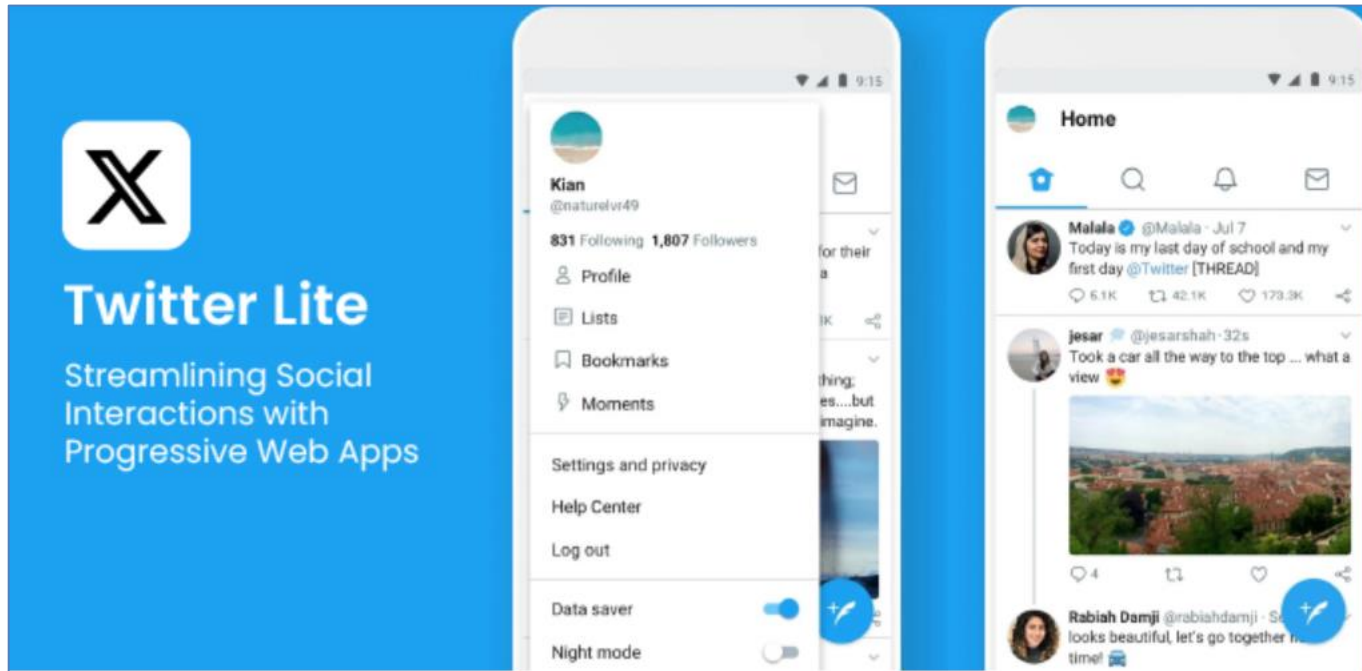


PWA

innovate

achieve

lead



Twitter Lite is a progressive web app (PWA) developed by Twitter (social media platform) to enhance its mobile web experience. It was designed to be faster, more reliable, and more engaging than the traditional mobile web experience. Twitter Lite combines the best of the modern web and native mobile app features, and it became the default mobile web experience for all users globally in April 2017.

PWAs are different from web apps because they can be **installed on a device**, while web apps are designed to run inside of a web browser.

PWAs also **function like websites but offer an app-like experience**, while Native Apps are specifically designed for platforms like iOS or Android.

PWAs are meant to eliminate a range of issues from slow networks to data limitation or complete lack of connectivity. They satisfy advanced user experience requirements such as offline work or push notifications.

PWA Applications



The Remarkable Result from Twitter Lite's PWA Adoption

- 65% increase in pages per session, 75% increase in Tweets sent, and 20% decrease in bounce rate.
- Uses less than 3% of the device storage space compared to Twitter for Android. The data saver mode gives users control over when Twitter Lite downloads media assets.
- First loads clock in at under 5 seconds over 3G networks on most devices, and subsequent loads are nearly instant.
- Users experience a 50% reduction in 99th percentile time-to-interactive latency, and logged-in users have a 30% reduction in average load time.

Why PWA for ex-Twitter (X)



- ❑ Twitter's native app consumed a large amount of data, which was a deterrent for users with limited data plans.
- ❑ Performance issues arose in regions with limited or low-quality internet connection. These issues led to slow load times and reduced user interaction with the platform.

How have they overcome challenges?



- ❑ Through the implementation of an 'Add to Homescreen' prompt and web push notifications, Twitter Lite **effectively re-engaged users on the mobile web**.
- ❑ Twitter Lite reduced data usage by implementing optimized defaults, **utilizing smaller media resources**, and leveraging **cached data**. This resulted in a remarkable **70% data consumption reduction** through optimized images.
- ❑ Twitter Lite's design caters to **low-bandwidth conditions**, thus making Twitter **faster, easier to use**, and more accessible to **users in emerging markets**.

Why use PWA?

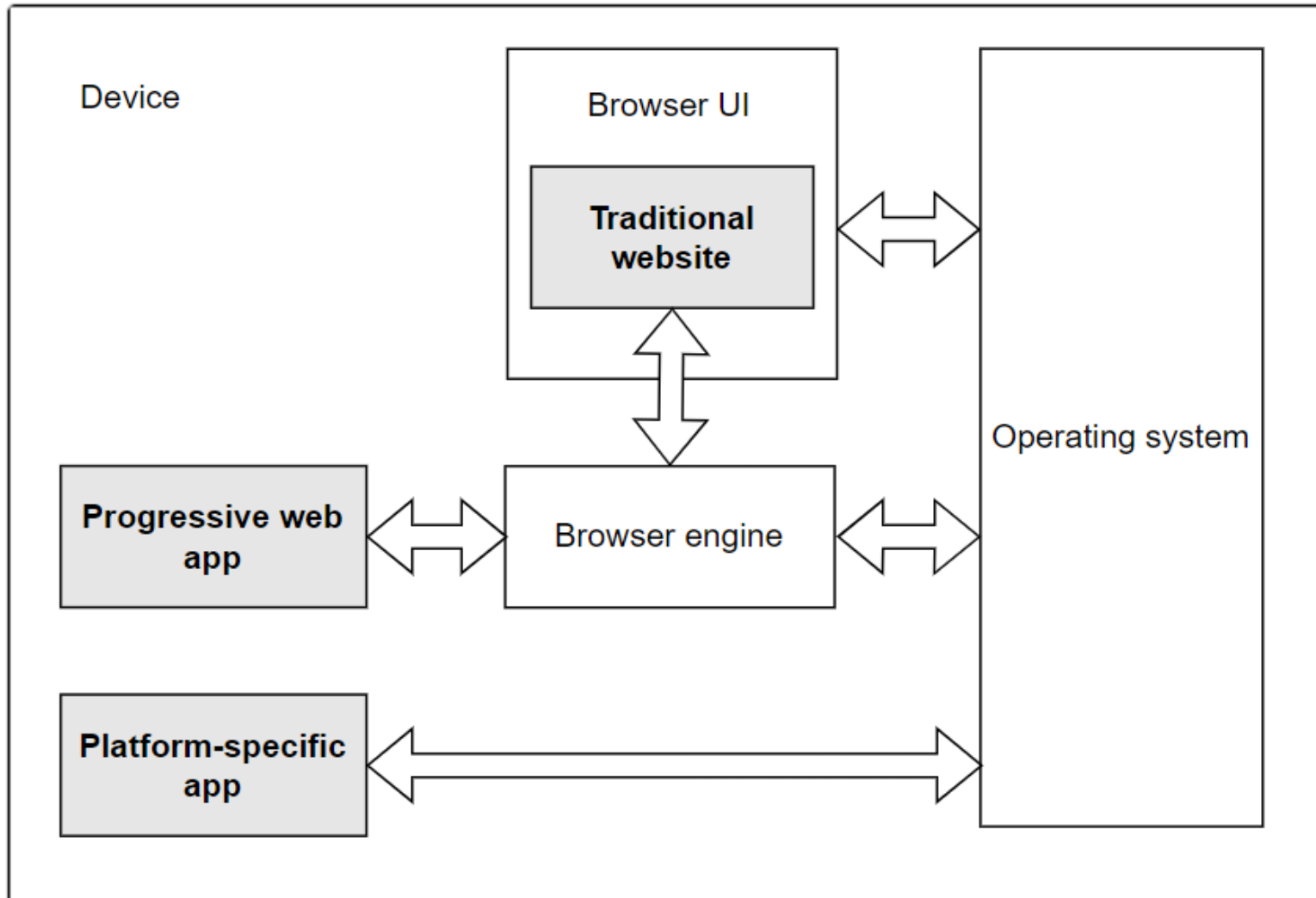
innovate

achieve

lead



PWA - Concepts

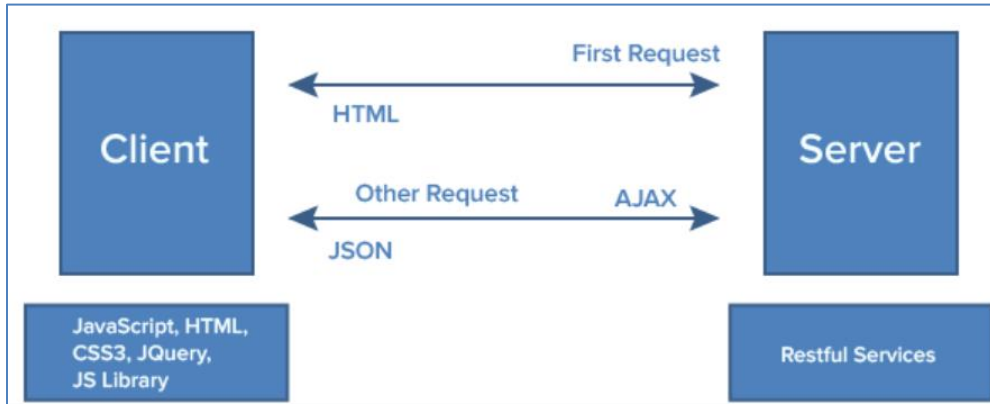


Code Structure



- ❑ A [web app manifest](#) file, which, at a minimum, provides information that the browser needs to install the PWA, such as the app name and icon.
- ❑ A [service worker](#), which, at a minimum, provides a basic offline experience.

PWA vs SPA

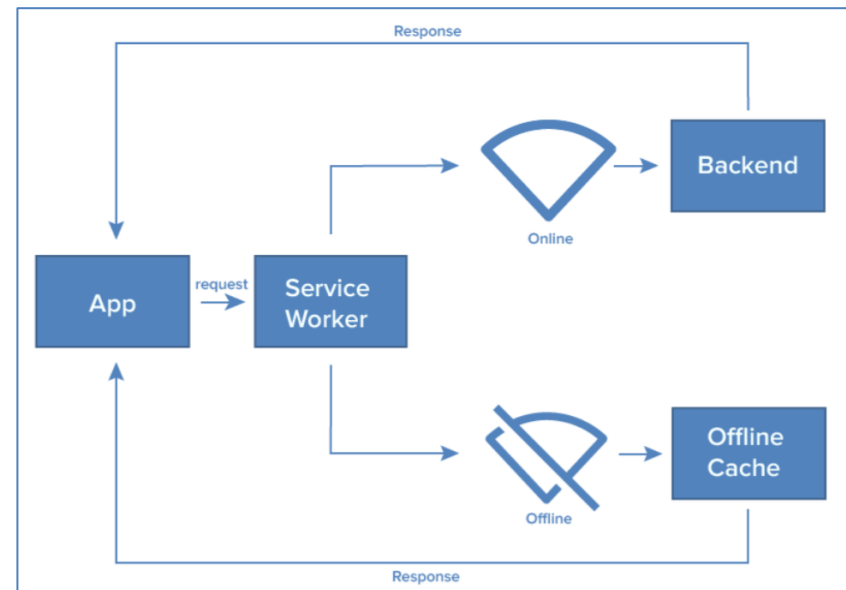


SPA Used By

- Facebook
- Gmail
- LinkedIn
- Netflix
- Google Maps



PWA



PWA vs SPA



Feature	PWA	SPA
Technology	Web technologies (HTML, CSS, JS)	Web technologies (HTML, CSS, JS)
Installation	Installed on device (like a native app)	Accessed through web browser
Offline functionality	Can work to some extent offline	Limited or no offline functionality
App-like features	Push notifications, home screen icon, background sync	May have some app-like features but not all
Discoverability	Not directly discoverable in app stores (especially iOS)	Discoverable through web searches and links
Development complexity	Can be more complex to achieve perfect app-like experience	Generally, less complex to develop than PWA
Performance	May have better performance due to caching and offline capabilities	Reliant on internet speed for optimal performance

Limitations



- ❖ **Limited offline functionality:** While PWAs can work to some extent offline, they typically rely on an internet connection for full functionality. This can be a drawback for users who need an app to function without a signal.
- ❖ **Restricted hardware access:** PWAs may not have the same level of access to a device's hardware as native apps. For instance, some PWAs might not be able to use the camera or microphone as seamlessly.
- ❖ **App store limitations:** While PWAs can be bookmarked like any website, they generally don't appear in app store searches. This can make them less discoverable for some users, particularly on iOS where home screen installation isn't as straightforward.
- ❖ **Development considerations:** While PWAs can be efficient to develop, achieving a perfect app-like experience might require more development effort compared to a simpler website.



Module 9:

Latest Advancements - Web Assembly



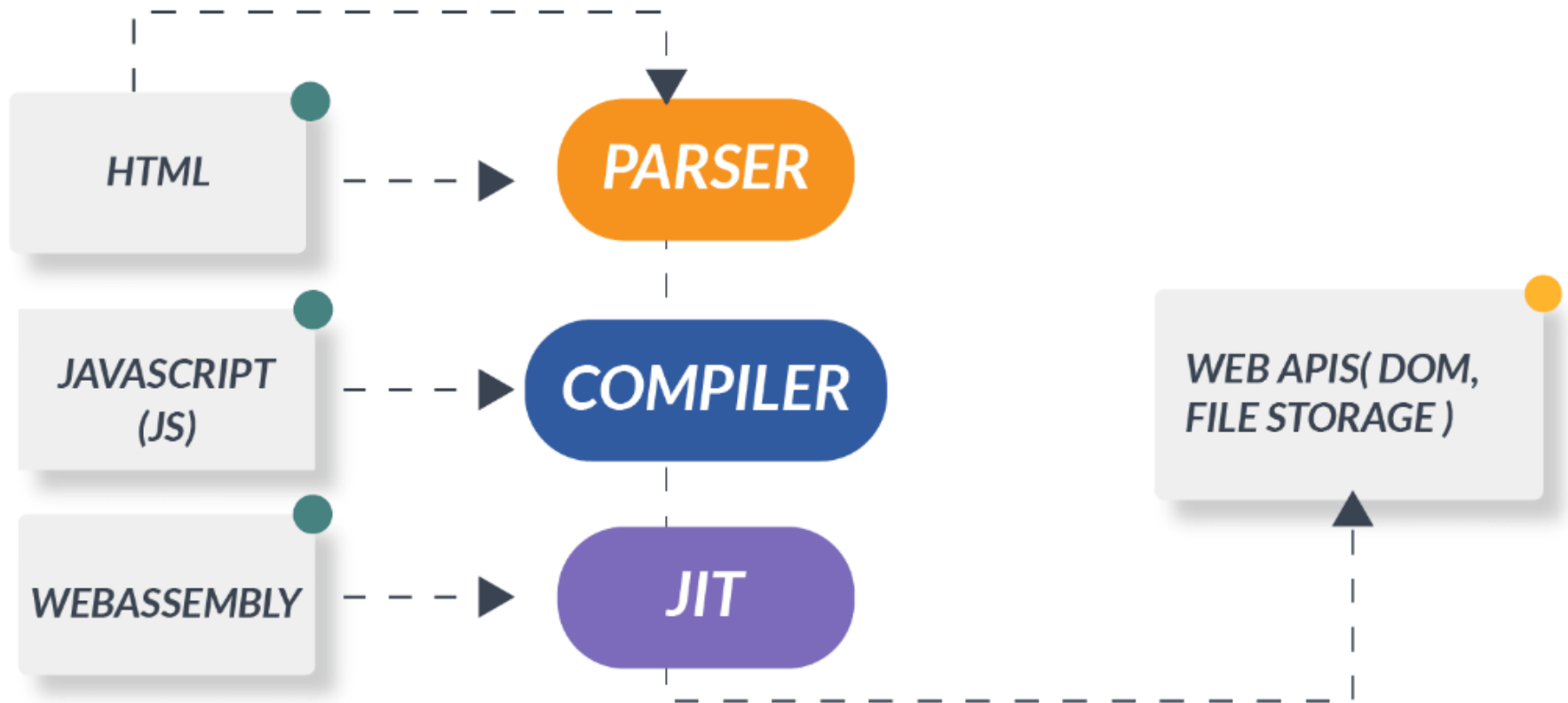
WebAssembly, often abbreviated as Wasm, is a game-changer for web development. It lets you run code written in various languages at near-native speeds within your web browser.

Portable binary format: WebAssembly code is compiled into a compact binary format (.wasm) that can be understood by modern web browsers.

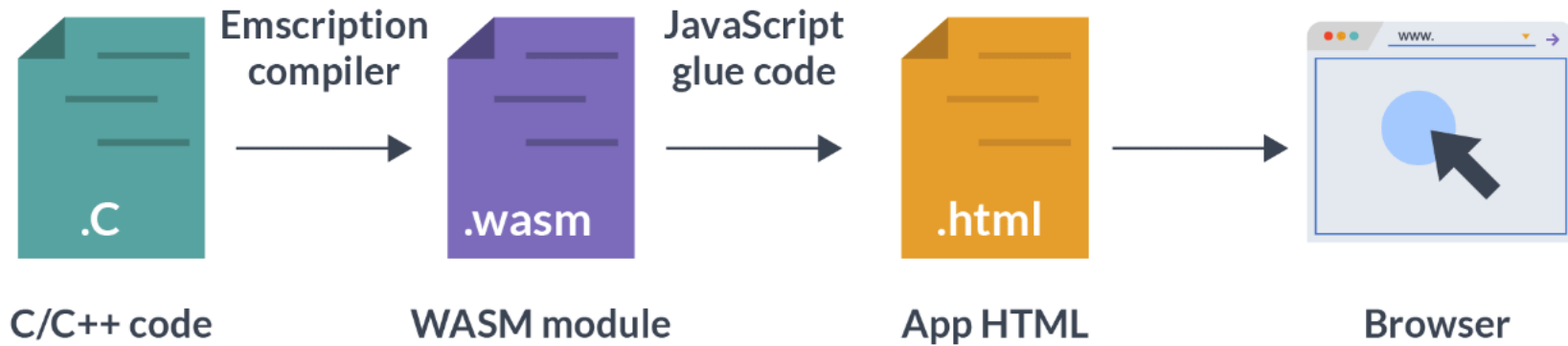
Compilation target: Many languages like C/C++, Rust, and even AssemblyScript can be compiled to WebAssembly, allowing their code to run on the web.

Complements JavaScript: WebAssembly is designed to work alongside JavaScript, not replace it. You can leverage WebAssembly for performance-critical parts of your web app and JavaScript for logic and interaction.

Architecture



WA Code Structure



Goals of the WA



The main objective of WebAssembly is to enable high-performance applications on web content, but the format is meant to be executed and integrated into other environments further.

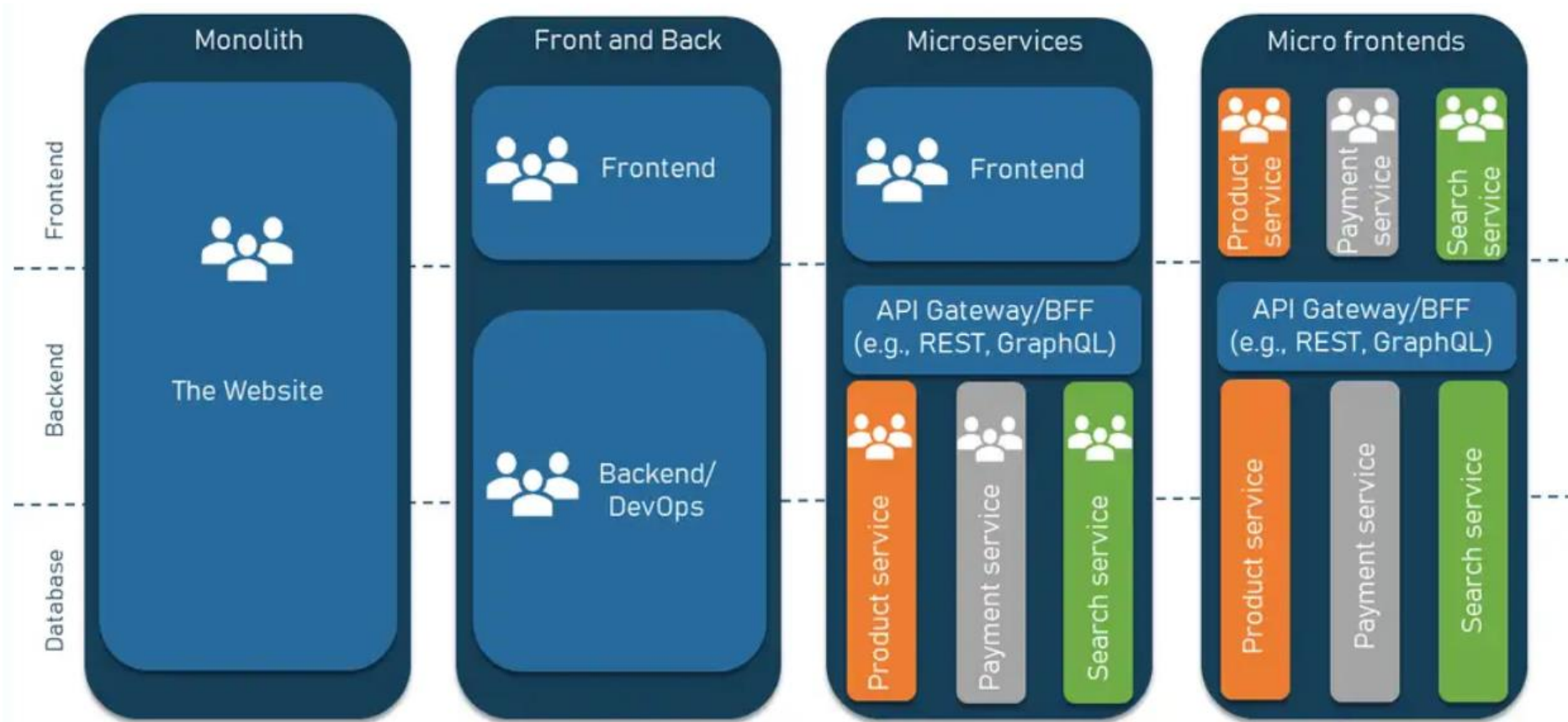
- ❑ **Be portable, fast and efficient** — WebAssembly code is often executed at near-native speed across different platforms by taking advantage of general hardware capabilities.
- ❑ **Be accessible to readable and debuggable** — WebAssembly could be a low-level programming language. Still, it does have a human-readable text format (the specification that continues to be being finalized) that enables code to be reviewed, written and debugged by hand.
- ❑ **Keep secure** — Actually, WebAssembly is specified to be run during a safe and sandboxed execution environment. Like other web code, it'll enforce the browser's same-origin and permissions policies.
- ❑ **Don't break the Internet** — WebAssembly is intended, so it plays nicely with other web technologies and maintains backwards compatibility.



Module 9:

Latest Advancements – micro frontends

Transition



Micro-frontend



Micro frontends are a web development pattern that splits an application into smaller, independent modules, called micro frontends.

Each micro frontend can have its own source code repository, dependencies, test suite, and delivery pipeline.

The components can be developed, tested, and deployed independently, allowing teams to work on specific features or functions within the application.

Micro-frontend



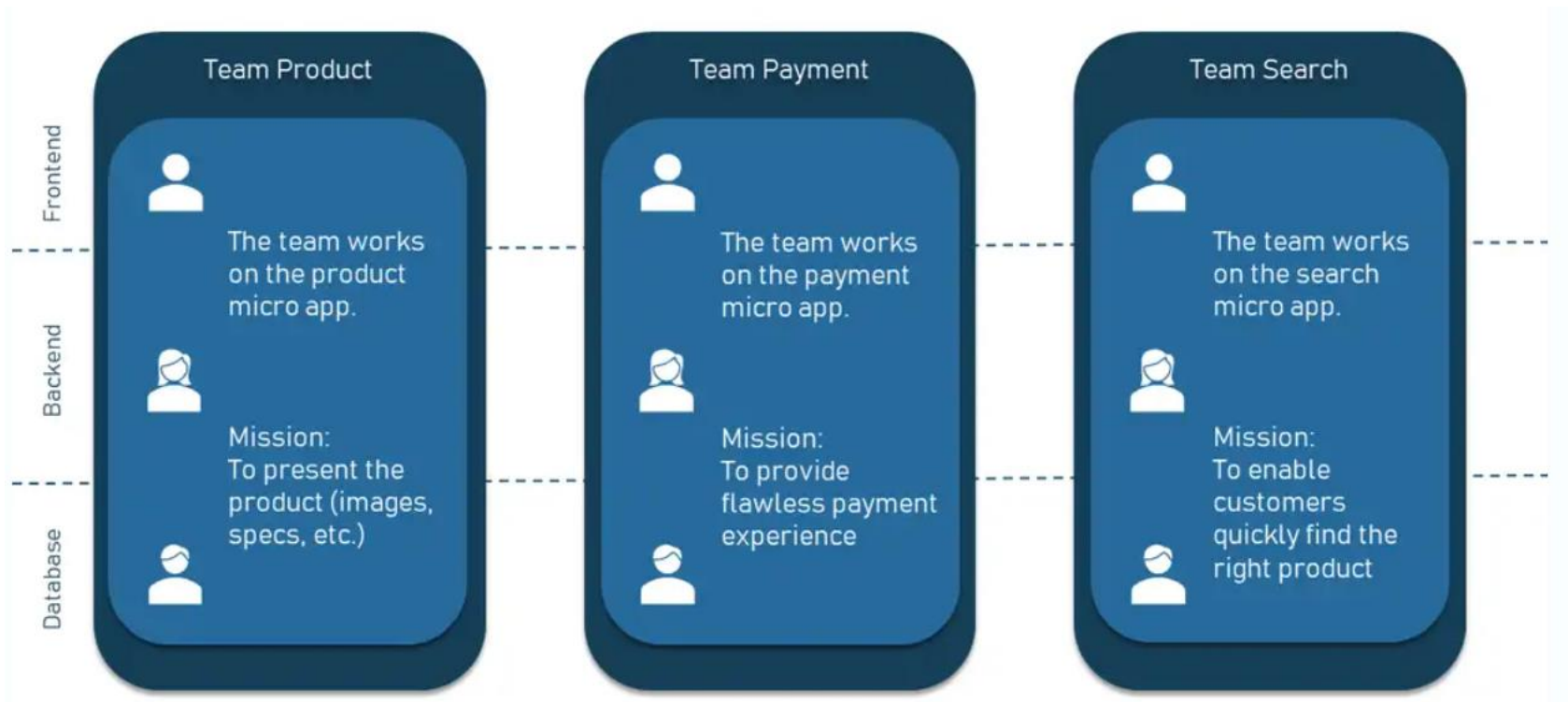
Core Idea:

- **Modular front-end architecture:** A single application is composed of multiple, independently developed and deployable micro-frontends.
- **Focus on ownership:** Each micro-frontend is owned by a dedicated team with freedom in technology stack and deployment cycles.
- **Loose coupling:** Micro-frontends interact with each other minimally, promoting isolation and reducing complexity.

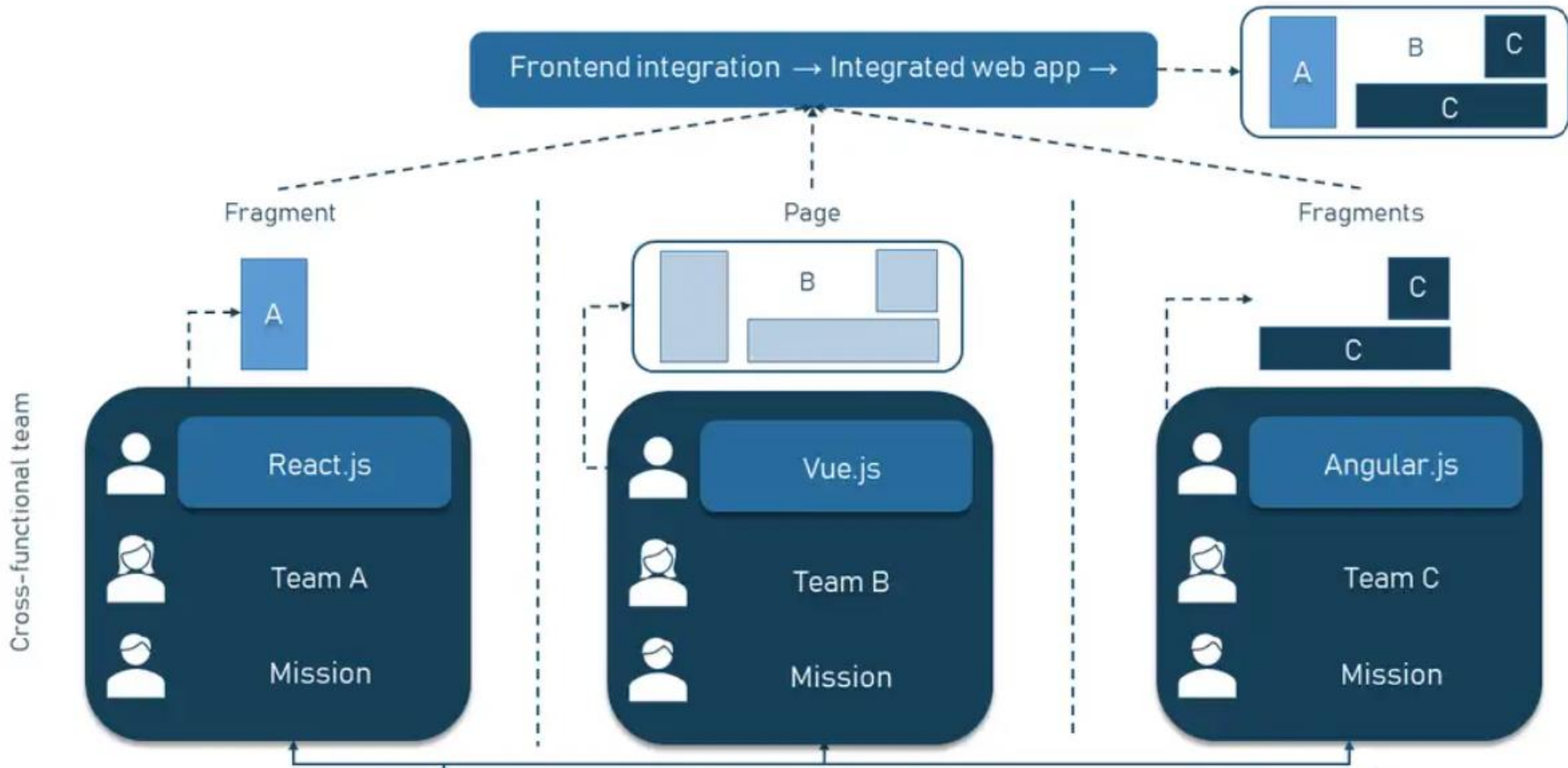
Benefits:

- **Improved maintainability:** Smaller codebases are easier to understand, debug, and update.
- **Faster development:** Independent teams can work on features concurrently, accelerating development cycles.
- **Tech stack flexibility:** Teams can choose the best tools for their specific features without impacting others.
- **Scalability:** New features can be added as micro-frontends, making the application more scalable.

Micro-frontend Team Goals



Architecture and Team Structure



Routing



Routing works for the page-level integration — when you need a service from a page owned by one team to get to a page owned by another team. **Each micro frontend is treated as a single-page app**. Routing can be approached by merely using HTML links. By clicking on hyperlinks, a user forces the browser to fetch the target markup from a server and then replace the current page with the new one.

For cases when a page must be rendered without reloads, you can opt for a shared application shell or a meta-framework like [single-spa](#).

The app shell is the minimal HTML, CSS, and JavaScript powering a UI. A user gets a static rendered page instantly even if the content data requested from the server is still pending. The central app shell acts as a parent application to the single-page applications built by the different teams.

Meta-frameworks make it possible to combine different pages, regardless of the library or framework used, into one whole. For example, the single-spa framework provides a set of solutions such as a module loader to load individual pages asynchronously;

- separate wrappers for different UI components to connect them into a whole; and
- APIs for communication between individual applications, subscription to events, etc.

Composition



Composition is what you do to the fragments to get them in the right slots within a page. The team that ships the page doesn't usually fetch the content of the fragment directly. Instead, It inserts a marker or placeholder at the spot in the markup where the fragment should go.

The final assembly is achieved by a separate composition technique. There are two main types of composition — client-side and server-side.

Client-side composition. HTML markup is created and updated directly in the web browser. Each micro frontend can both display and update its markup independently of the rest of the page. You can perform this sort of composition via Web Components, for example.

The idea is to create each fragment as a web component that can be deployed independently as a .js file, after which the applications can load and render them in the placeholders created for them in the theme layout. Web components rely on HTML and DOM API that other frontend frameworks can access and a common way of receiving and sending data using props and events.

Server-side composition. With this pattern, the UI fragments are composed on the server, which means that the client-side receives, in fact, a fully assembled page, resulting in increased loading speed. As a rule, the assembly is performed by a separate service, located between the web browser and web servers. An example of the service is CDN (content delivery network).

When micro front-end is good idea?



Medium to large projects. The style of building micro frontends is ideal for scaling development and as such makes a good fit for larger projects with multiple teams. Say, if you are working on a large eCommerce site like Zalando, you will benefit from using micro frontends.

Web projects. The concept of micro frontends isn't tied to a specific platform, but it shines the brightest on the web. Native apps for iOS and Android are monolithic in their fashion. You won't be able to compose and replace functionality on the fly.

Productivity-focused projects. Having vertically sliced teams adds up to the overall productivity. But it comes with extra costs and maintenance challenges. If you put productivity first and are ready for overhead, micro frontends can be considered.

micro-frontend not suitable



Projects with unclear vertical cuts. You need to know the business domain you're working in. If it's not clear what team is a perfect match for implementing a particular feature, you can end up with a messed-up work organization, more complexity, and fewer gains.

Small projects. You don't need to overcomplicate what can be done easily and quickly using a common monolithic approach. That's the case with small and simple websites. Just because the new style is trendy doesn't mean you have to use it everywhere.

Review Questions



- 1) Explain the concept of inclusive design and how it differs from basic accessibility compliance. Describe a real-world scenario where inclusive design would create a more positive user experience for someone with a disability.
- 2) You are building a web application. Discuss the different types of assistive technologies users might employ, and how your design choices could impact their ability to interact with your application effectively.
- 3) You are tasked with optimizing a website that has slow loading times. Explain the trade-offs between using client-side caching and server-side caching for different types of website content. In which scenario would you prioritize one over the other?
- 4) You are building a web application that requires complex 3D graphics rendering. Explain how WebAssembly can be integrated into your project to achieve better performance compared to pure JavaScript. What are some challenges you might face during implementation?
- 5) Explain how advancements like PWAs, WebAssembly, and micro-frontends are shaping the future of web development.