Web Development

Beyond the Basics

André Restivo

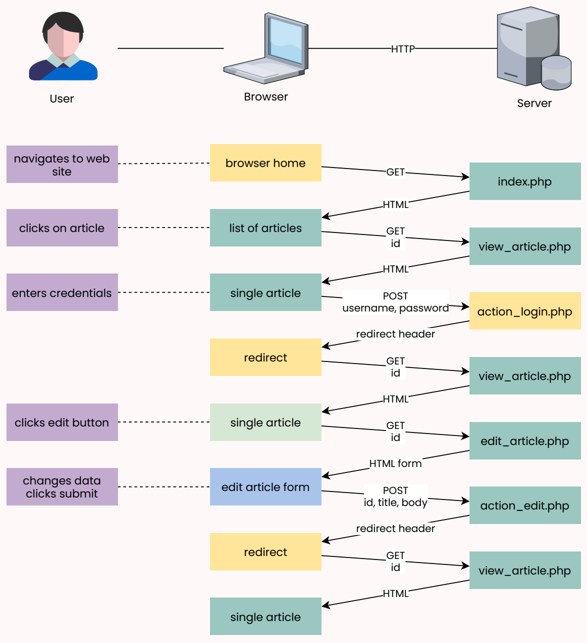
Index

MPA vs. SPA

Multiple Page Application:

Classic Web

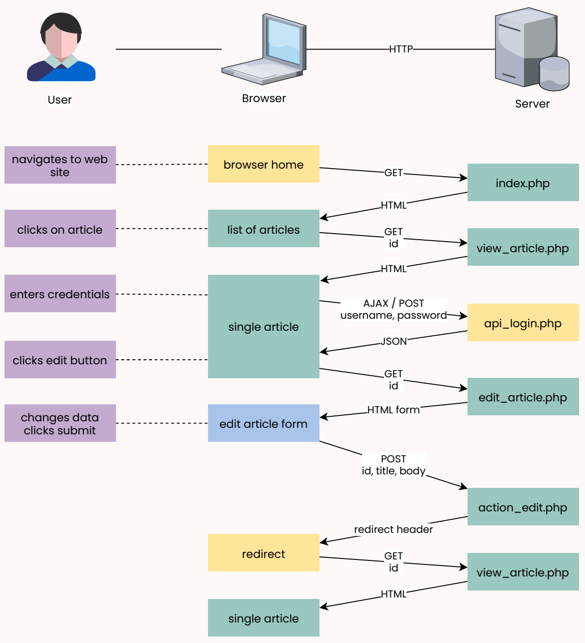
- Each interaction renders a **different page**; a different HTTP request.
- The HTML can mention other **resources** prompting more HTTP requests (*e.g.*, images, CSS, scripts).
- Actions change the server state and **tell the browser** where to go next.



- **Not easy** to **reuse the backend** for different purposes (*e.g.*, as an API).
- **Easy** to understand but can be **slow**; pages are heavy and contain **repeated code**.

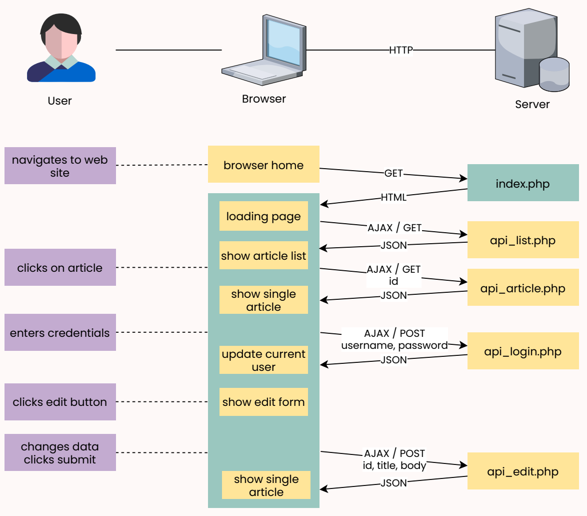
Enter AJAX: The Hybrid Model

- Pages can request **more information** from the server.
- Some interactions **do not reload** the entire page (*e.g.*, login and logout, load more items).
- API calls can return **different types** of data:
  - JSON / XML with **client-side** rendering.
  - HTML with **server-side** rendering.
- Faster, as we do not need to transfer so much data, but might require **duplicated rendering code** (client/server).



The Single Page Application (SPA)

- The **first interaction** returns an HTML page, and the application **never leaves** that page during its **entire lifecycle**.
- All the remaining interactions are the result of AJAX calls to **fetch more data** and **client-side rendering**.



- Users **do not need to wait** for interactions to finish before performing the next.
- Slower first load, **fast** afterward; feels like a **desktop application**.
- But...

Do Not Break the Web

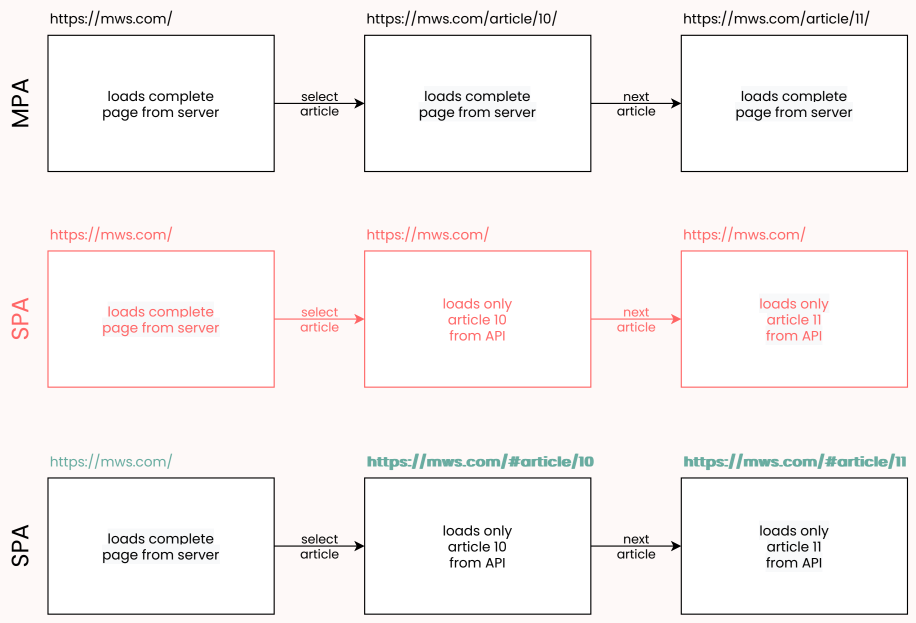
Users have some **expectations** regarding how web pages work:

- The **back button** will take them to the previous page.
- **Copying/Bookmarking/Sending** the current URL allows them to **resume** their session later, **save** a specific page location, or **send** someone a link to a particular web page.

Break these, and **users won't be happy**!

Fragment

Using the **URL fragment** to store the **current state**:



MPA
https://mws.com/
loads complete page from server
select article
https://mws.com/article/10/
loads complete page from server
next article
https://mws.com/article/11/
loads complete page from server
SPA
https://mws.com/
loads complete page from server
select article
https://mws.com/
loads only article 10 from API
next article
https://mws.com/
loads only article 11 from API
SPA
https://mws.com/
loads complete page from server
select article
https://mws.com/#article/10
loads only article 10 from API
next article
https://mws.com/#article/11
loads only article 11 from API

Fragment Example

Using the **URL fragment** and **regular expressions** to call functions that **load the current state**:

```
function parse_fragment() {
  const hash = window.location.hash

  if (hash) {
    const category = /#category\/(\d+)/.exec(hash
    if (category) return load_category(category[1

    const article = /#article\/(\d+)/.exec(hash)
    if (article) return load_article(article[1])

  }

  load_articles()
}
```

SPA vs. MPA

Single Page Applications (SPA) have several advantages:

- **Speed**: Most pages load faster.
- **Network**: Less network intensive.
- **Decoupling**: The backend and Frontend are decoupled.
- **UX**: Better user experience.

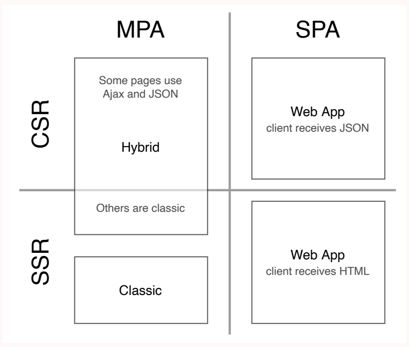
But Multiple Page Applications (MPA) also have some strong points:

- **SEO**: Easier search engine optimization.
- **First Load**: The speed of the first load is usually better.
- **JavaScript Dependency**: Works without JS.
- **Navigation**: Simpler and standard navigation (*e.g.*, link, back button).

CSR vs. SSR

A second dichotomy is **where to render** the **HTML** code:

- Client-side Rendering (**CSR**): The browser receives data in a different format (*e.g.*, JSON or XML) and renders that data as HTML.
- Server-side Rendering (**SSR**): The server already sends data as HTML.



We can use the **HTTP Accept header** to allow using a single server-side script to generate both JSON and HTML.

Progressive Web Apps (PWA)

PWA: Apps that have the capabilities of **native** apps with the **reach** of **web** apps:

- **Installable**: Just like native apps.
- **Cached Content**: Using web workers and local storage.
- **Web APIs**: Uses web APIs to do more.
- **Responsive**: Works on several devices.

Progressive comes from **progressive enhancement**, starting with a basic level of user experience and using more advanced functionality that will automatically be available to browsers that can use it.

As opposed to **graceful degradation**: starting with all bells and whistles, and degrading to a lower level of user experience on older browsers.

Web APIs

Web APIs

Modern browsers can take advantage of several Web APIs.

These allow the creation of **web apps** with capabilities similar to **native apps** (PWAs).

Web Workers

Web Workers make it possible to run a script in a **background thread**:

- **Dedicated Workers**: When used by a single script.
- **Shared Workers**: Can be used by multiple scripts, possibly running on different windows, and communicating using an active port.
- **Service Workers**: Act as a proxy that sits between web applications and the network.

Web workers **can't** directly **manipulate the DOM**.

Service Workers

- Intercepts resource requests acting as a **network proxy**.
- Typically used to cache resources and provide an **offline experience**.
- An **enhancement** to existing websites. **No** baseline **functionality is broken** if the browser does not support them.
- After a service worker is installed and activated, it controls the page to offer **improved reliability** and **speed**.

```
navigator.serviceWorker.register("/serviceworker.
```

On the service worker:

```
self.addEventListener("install", event => {
   console.log("Service worker installed");
})
self.addEventListener("activate", event => {
   console.log("Service worker activated");
})
```

Cache

To manage the cache, **service workers** interact with the Cache Storage API.

The Cache API is a storage mechanism for Request / Response pairs cached in long-lived memory.

```
const urls = ["/", "style.css", "script.js"]
// self is a service worker
self.addEventListener("install", event => {
  event.waitUntil(
    // assets is the name of the cache
    caches.open("assets").then(cache =>
        cache.addAll(urlsToCache)
    )
  )
})
```

Web Storage

Allows browsers to store **key/value pairs** much more intuitively than with cookies.

- sessionStorage: separate storage for each origin.
- localStorage: the same but persists browser restarts.

```
localStorage.setItem('color', 'blue')
const color = localStorage.getItem('color') // bl
```

IndexedDB

A low-level API for **client-side** storage of **significant amounts** of structured data:

- **Object store** paradigm: data as objects.
- **Primary Keys** and **Indexes**.
- **CRUD** operations: create, read, update, and delete.
- **Versioning**: dealing with different database versions.

Other APIs

- Contact Picker: access to contact lists.
- Image Capture: taking pictures.
- Canvas: drawing pictures in canvas elements.
- Clipboard: read and write from the system clipboard.
- Geolocation: get the current location.
- Websockets: open two-way communication channels with a server.
- History: change the browser history.
- Many More.

Frameworks

Frameworks

Until now, we have been dealing with **low-level(-ish)** web development.

But most **modern web development** is done with the support of several **frameworks**.

Full-stack Frameworks

A framework that supports development of front-end user interfaces, back-end logic, and database communication:

- Laravel (PHP).
- Django (Python).
- CakePHP (PHP).
- Meteor (JavaScript / NodeJS).

Typically, they include several services:

- **Authentication** and permission management.
- **Routing**: Mapping URLs to resources.
- **Object Relational Mapping** (ORM): Almost no need to write SQL.
- **Templating**: Easy HTML rendering.

They can also be used to create just the backend.

Client-side Frameworks

Several reactive client-side frameworks that can be easily coupled with an API:

- React.
- Vue.
- Angular.
- Svelte.

Svelte example:

```
<script>
    let count = 0

    function handleClick() {
        count += 1
    }
</script>

<button on:click={handleClick}>
    Clicked {count} {count === 1 ? 'time' : 'time
</button>
```

CSS Frameworks

CSS is complicated; CSS frameworks can be a **good start** for a great design:

- Bootstrap.
- Foundation.

More **semantically** (*class-less*) friendly ones:

- Pico CSS.
- Milligram.