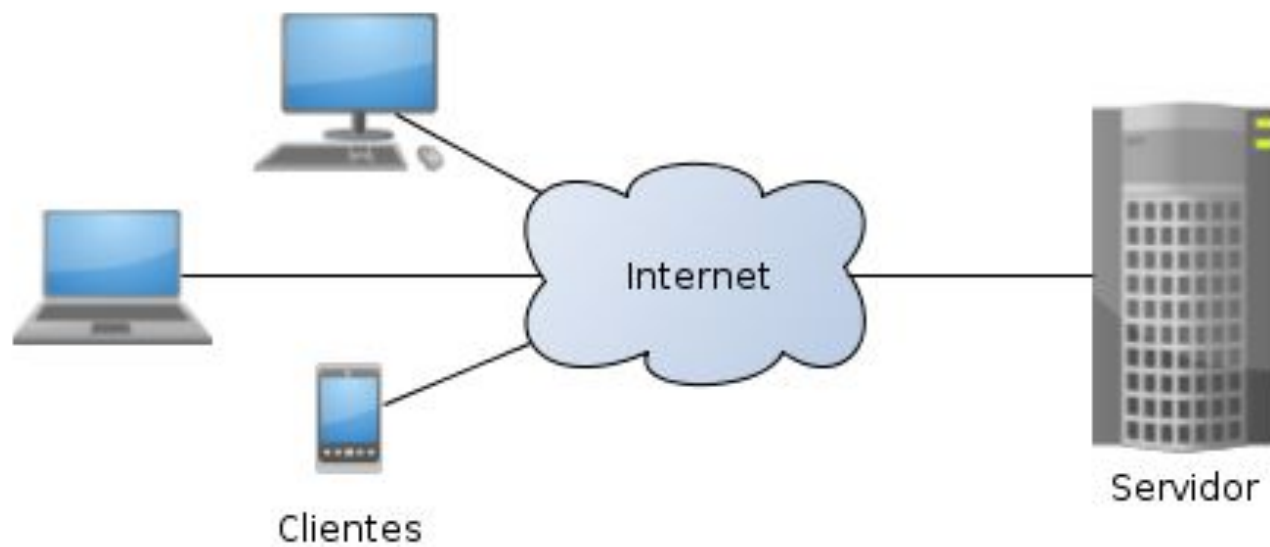


# Arquitectura Web Frontend

website design  
is my passion





# Cientes HTTP

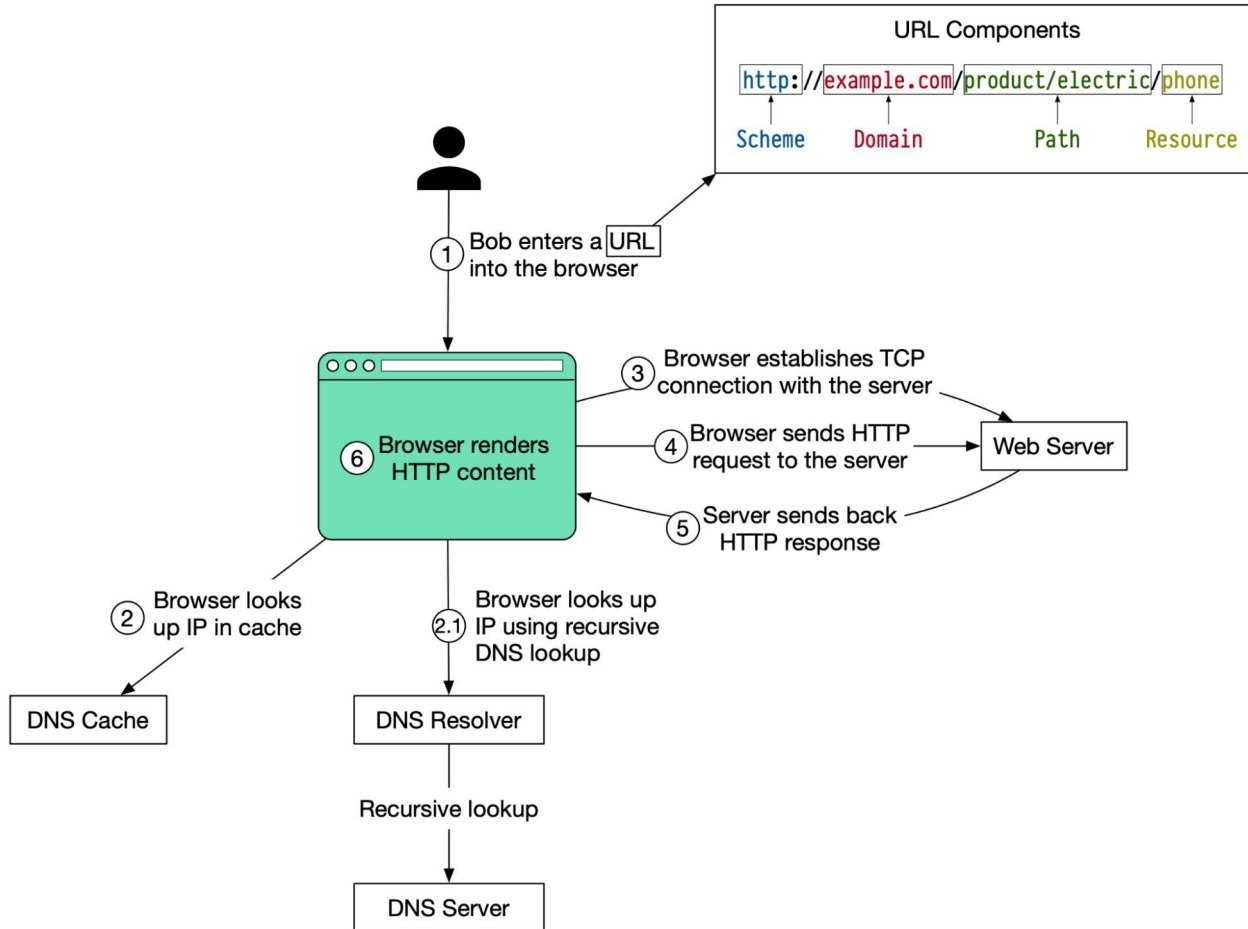


curl://

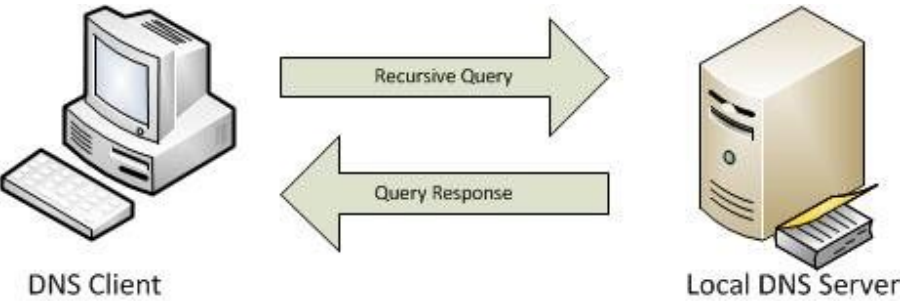


¿Qué pasa cuando  
entramos una URL en el browser?

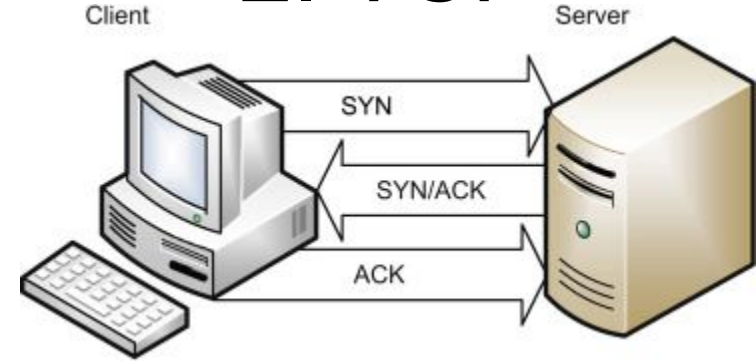
## What happens when you type a URL into your browser?



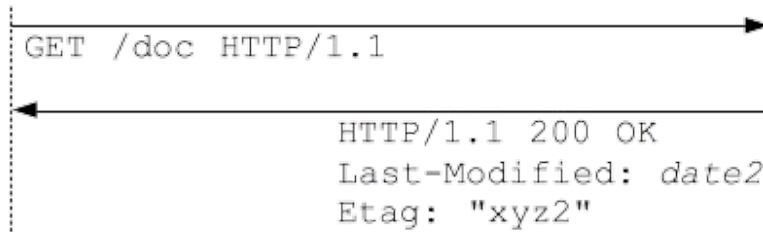
# 1. DNS



# 2. TCP

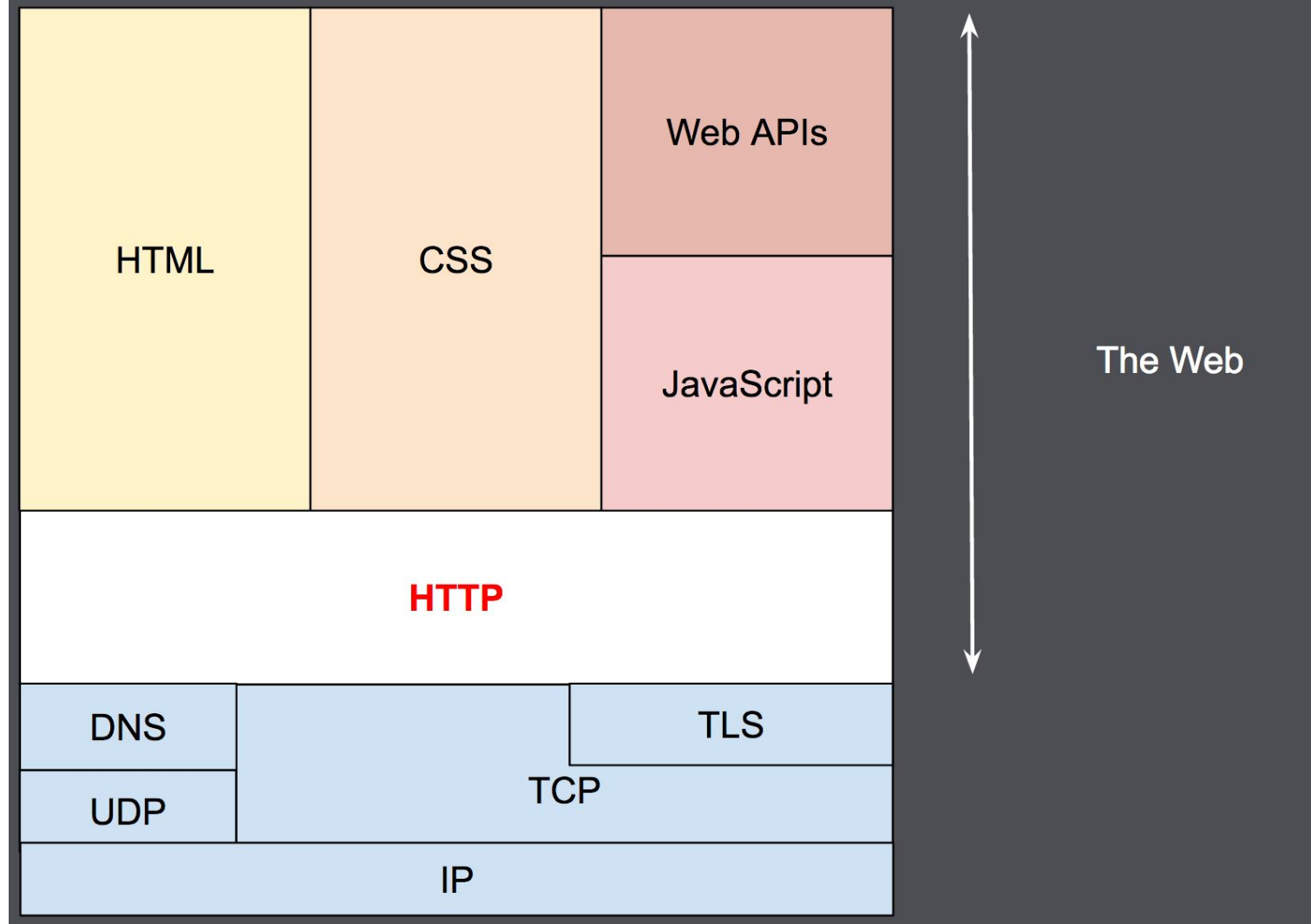


# 3. HTTP



# 4. HTML

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Example</title>
5     <link rel="stylesheet" href="st
6   </head>
7   <body>
8     <h1>
9       <a href="/">Header</a>
10    </h1>
11    <nav>
12      <a href="one/">One</a>
13      <a href="two/">Two</a>
14      <a href="three/">Three</a>
15    </nav>
```



Ejemplo con Google 

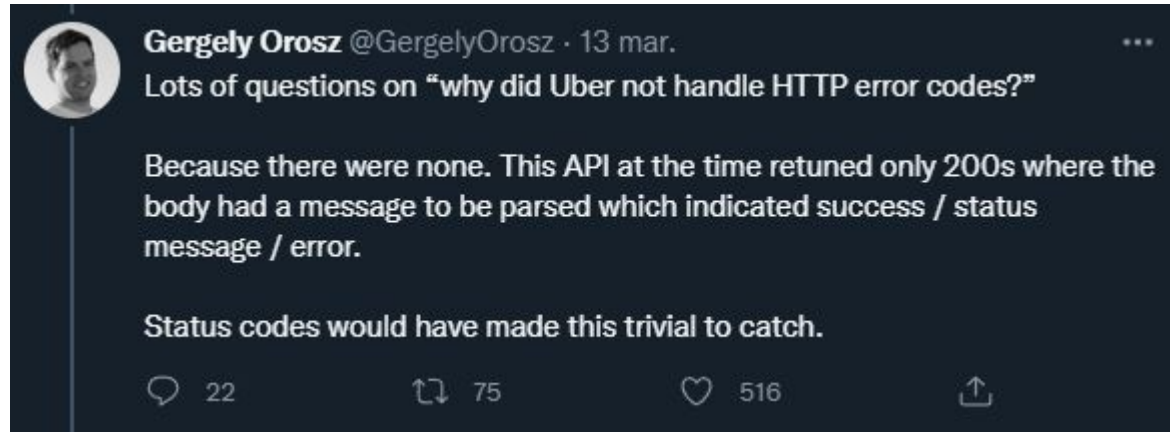




# ¿Y si elijo no seguir el estándar?



# La falla de Uber Eats que permitió pedir comida gratis durante un fin de semana en India



<https://twitter.com/GergelyOrosz/status/1502947315279187979>

# Un booking sin idempotencia?



Please do not refresh this page as this may result in a duplicate booking.

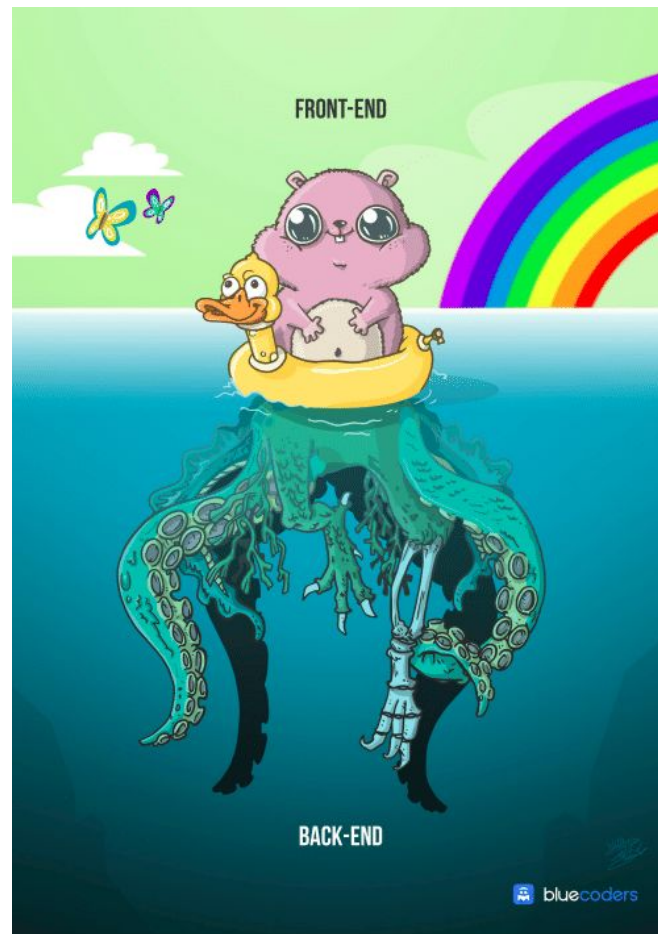
# ¿Qué cosa es frontend?

En la industria usamos “frontend” para referirnos a los sistemas de cara al usuario.

El frontend puede ser una aplicación web servida por HTTP, pero también puede ser una aplicación desktop (Electron, WPF), mobile (React Native, iOS, Android), etc.

En resumen, pueden ser tanto aplicaciones nativas como plataformas web.

En esta clase vamos a profundizar en las tecnologías que hacen a la web.



FrontEnd VS BackEnd

¿Por qué ocuparse del  
frontend?



# ¿Cómo se hace una página web?

El browser interpreta archivos en formato HTML, que a su vez pueden solicitar la carga e interpretación de archivos CSS y Javascript

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet"
      href="helloworld.css">
    <script
      src="./helloworld.js"></script>
  </head>
  <body>HOLA</body>
</html>
```

# HTML

Es un lenguaje de programación de markup, primo del XML, con el que se define la estructura de nuestra página, y la semántica del contenido.



- Main page
- Contents
- Featured content
- Current events
- Random article
- Donate to Wikipedia
- Wikipedia store

```
▼ <ul>
  ▼ <li id="n-mainpage-description">
    <a href="/wiki/Main_Page" title="Visit
      [Alt+Shift+z]" accesskey="z">Main page
    </li>
  ▼ <li id="n-contents">
    <a href="/wiki/Portal:Contents" title=
      browsing Wikipedia">Contents</a>
    </li>
  ▼ <li id="n-featuredcontent">
    <a href="/wiki/Portal:Featured_content
      content – the best of Wikipedia">Featu
    </li>
  ▼ <li id="n-currentevents">
    <a href="/wiki/Portal:Current_events"
      background information on current even
      Current events</a>
```



# CSS

Es un lenguaje con el que definimos la apariencia de nuestros elementos; permite aplicar estilos de manera selectiva a elementos en documentos HTML.

Colores, fuentes, tamaños y posiciones, todo puede ser definido mediante este lenguaje

```
<link rel="stylesheet" href="helloworld.css">
```

Main page  
Contents  
Featured content  
Current events  
Random article  
Donate to Wikipedia  
Wikipedia store


```
#mw-panel .portal .body li a {  
  color: #0645ad;  
}  
  
a {  
  text-decoration: none;  
  color: #0645ad;  
  background: none;  
}
```

# JavaScript

JavaScript es un lenguaje de programación que maneja el browser. Si bien inicialmente se pensó para “agregar comportamiento” a las páginas, hoy en día se puede manipular el DOM íntegramente con JS.

Se usa para darle **interactividad** a las páginas web

```
<script src="._/helloworld.js"></script>
```



A vertical arrow points from the `src` attribute of the script tag in the top box to the JavaScript code block in the bottom box, indicating that the code is loaded from the specified file.

```
const miTitulo = document.querySelector('h1');  
miTitulo.textContent = '¡Hola mundo!';
```

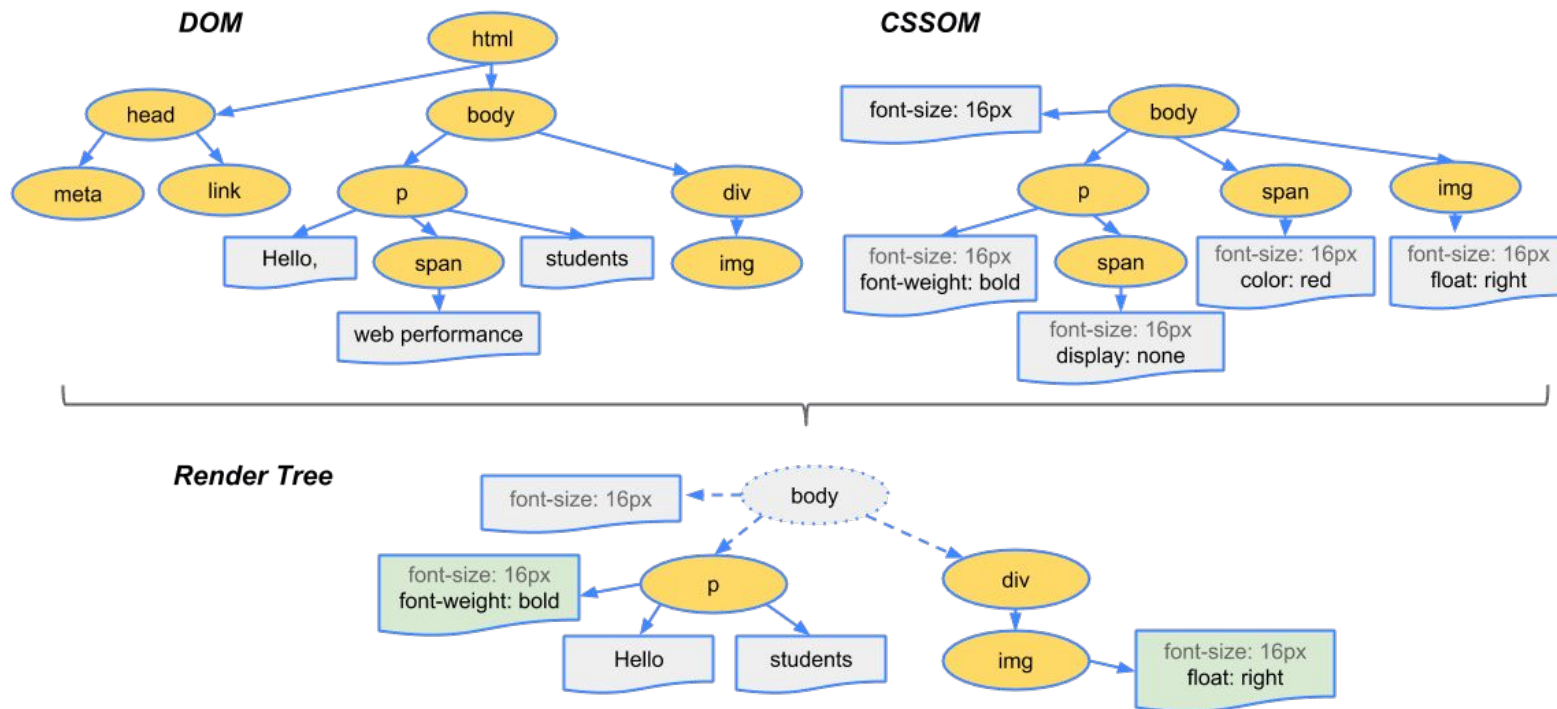
# DOM + CSSOM

Characters

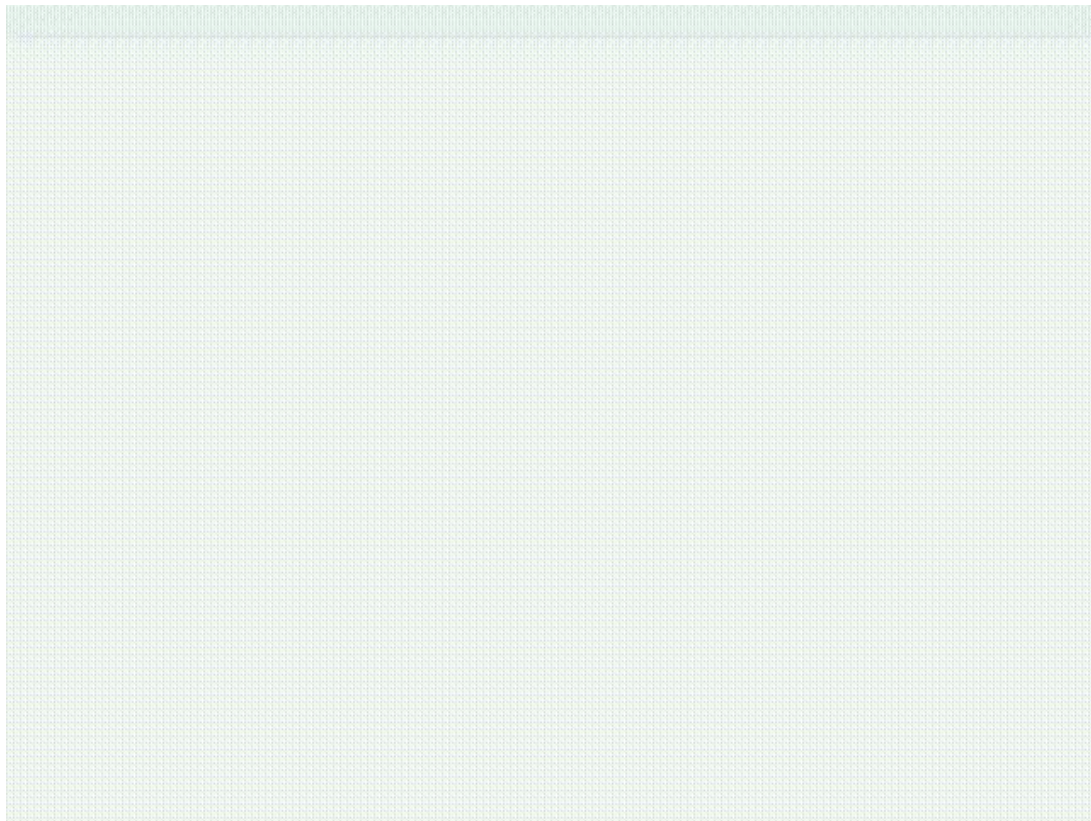
Tokens

Nodes

DOM



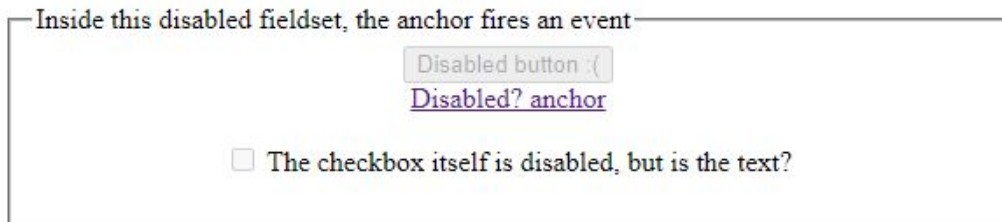
# DOM + CSSOM



# Entonces... ¿cada browser decide?

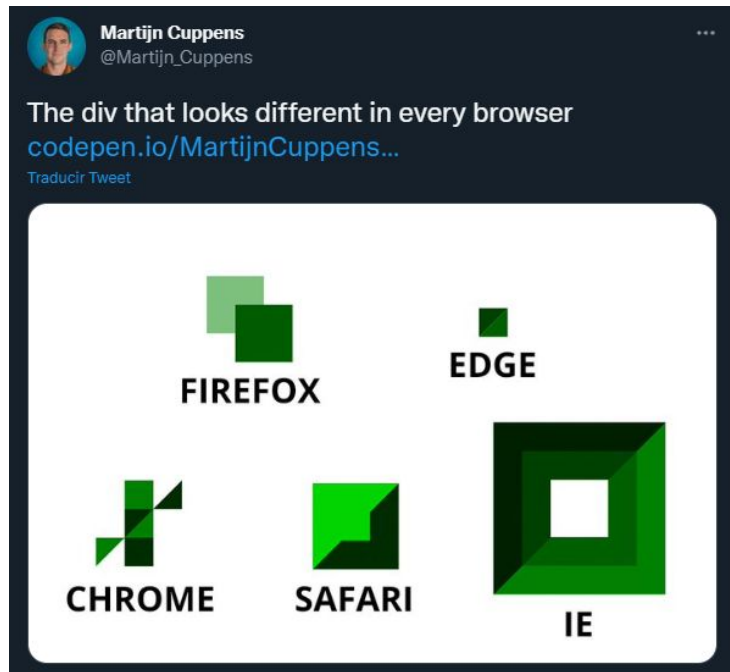
Sí, y es mucho muy importante que si quiero hacer una plataforma web de uso masivo, verifique que mi aplicación funcione correctamente en los browsers más populares.

## Ejemplo HTML



<https://github.com/francobatta/browser-dependent>

## Ejemplo CSS



[https://twitter.com/Martijn\\_Cuppens/status/1015169981368225793](https://twitter.com/Martijn_Cuppens/status/1015169981368225793)

# Organismos contra el caos

Distintas organizaciones, como W3C, WHATWG y ECMA, se dedican a revisar propuestas y definir estándares para que los desarrolladores no tengamos que hacer una página para cada navegador.

Por ejemplo: WHATWG hace estándares (DOM, HTML, Web APIs) y W3C revisa y aprueba los cambios (WHATWG nace como un “fork” de W3C).

ECMA describe en ECMAScript un lenguaje que se comporta de cierta manera. JavaScript es ECMAScript-compliant.



JavaScript siendo JavaScript?



¿Cómo sé que mi página genera valor (\$), y en todos los browsers?



User Analytics



Cross-browser testing



User Analytics



# Ejemplo Vanilla JS 🍦



Powered by: <https://www.youtube.com/watch?v=IhmSidOJSeE>

¿Le ven algún defecto a este  
approach?

# Performance

¿Cuánto tiempo es aceptable?

# Performance

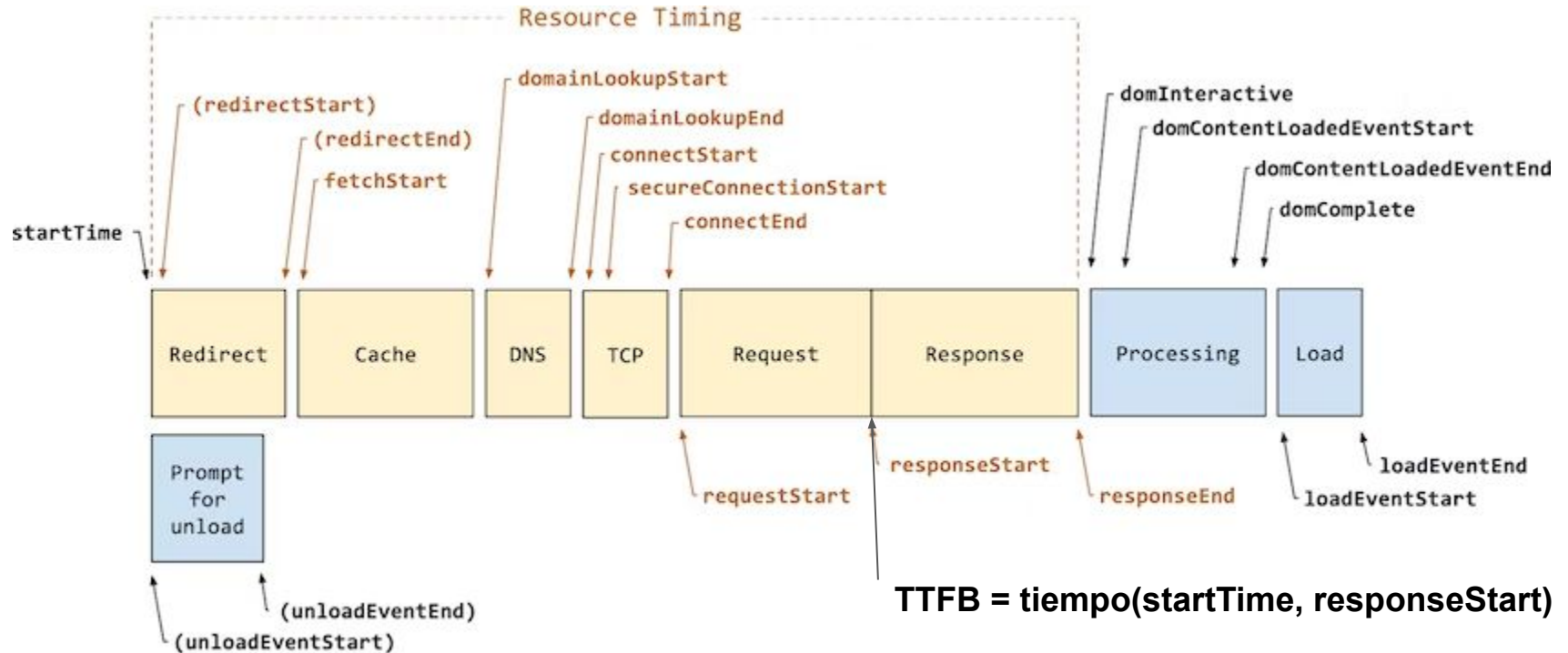
Los tiempos de bounce rate y conversión están directamente relacionados con los tiempos de carga en la web. Algunas formas de medirlo (del chrome [web-vitals](#)):

- **Time to First Byte (TTFB)**: tiempo desde que comienza la request hasta recibir el primer byte (ideal: P75 < 0.8 segundos)
- ~~**First Meaningful Paint (FMP)**: cuando cargó el layout, contenido principal y fuentes. Cuando la respuesta a "¿Es útil?" se convierte en "sí", se completó la FMP → deprecado!~~
- **Largest Contentful Paint (LCP)**: tiempo para renderizar una imagen o el bloque de texto más grande visible dentro de la ventana de visualización (ideal = P75 < 2.5 segundos)

¿Cómo hacemos para encontrar oportunidades de mejoras en la performance de nuestro sitio?

Page Load Time (seconds)	Bounce Rate (%)
1	7
2	6
3	11
4	24
5	38
6	46

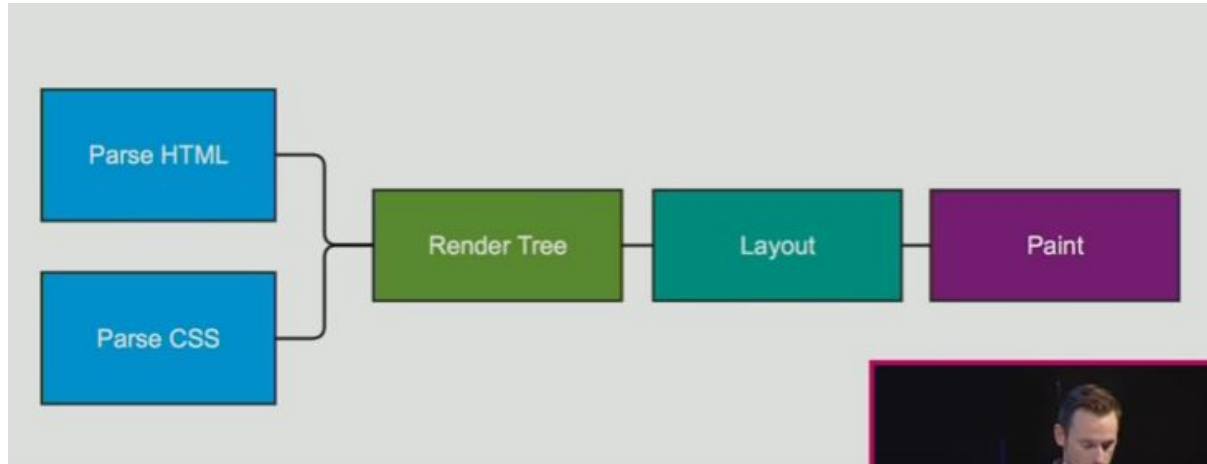
# TTFB en el ciclo de vida de carga de una página



# 1- Entender lo que hace el browser (Critical Resource Path)

El render inicial se produce tras completarse la carga y procesamiento del HTML + CSS. Para que nuestro sitio cargue rápidamente, queremos que los tamaños de estos dos archivos sean mínimos, y que servirlos esté **tuneadísimo**.

Siempre hay un archivo inicial a renderizar de HTML y un CSS que lo estila



Robado de una charla

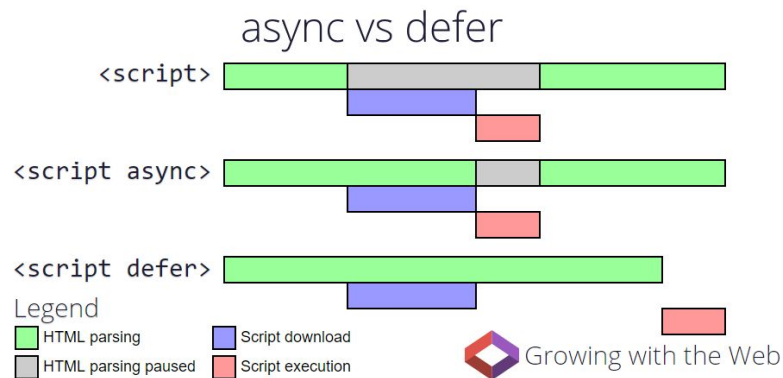
[https://developer.mozilla.org/en-US/docs/Web/Performance/How\\_browsers\\_work](https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work)

## 2- Entender lo que hace el browser (con nuestra página)

- Cargar los scripts JS de forma async - no bloquear al preload scanner. (defer = paralelo y ejecutar luego, async = paralelo y ejecuto ni bien descargo)
- Minificar y comprimir el HTML y CSS
- Inline CSS (dentro del HTML) o archivos separados?

→ Inline headers + lo que se vea al entrar, async el resto

- Fonts, assets: async
- TCP Slow Start (14 KB/ trip) → necesitamos cargar el HTML y CSS críticos en 14 KB o menos, *idealmente*
  - En la práctica, hasta 100 KB se considera ideal debido a los pocos roundtrips TCP (exponenciales).
- Declarar el charset (UTF8) en el primer KB de HTML. Si no lo hacemos, enlentecemos el parser + es un error de Lighthouse.



# 3- Usar lighthouse

Lighthouse es una herramienta incluida en las tools de chrome que nos da un reporte exhaustivo sobre distintos aspectos de un sitio en particular.



## Performance

### Metrics

First Contentful Paint	2.6 s	First Meaningful Paint	3.1 s
Speed Index	6.4 s	First CPU Idle	8.5 s
Time to Interactive	10.1 s	Estimated Input Latency	120 ms

[View Trace](#)

Values are estimated and may vary.



### Opportunities

These optimizations can speed up your page load.

Opportunity	Estimated Savings
1 Defer offscreen images	2.85 s



¿Y si quiero mandar un update?  
¿Me traigo todo de nuevo?

# AJAX (JavaScript Asíncrono + XML) y formas más felices

A lo largo del tiempo, fueron surgiendo distintas formas de manejar asincronismo:

Antes: cualquier update → full-page refresh

- XMLHttpRequest (AJAX) →
- jQuery (basado en callbacks)
- Promises
- async-await (syntactic sugar para Promises)

↓

```
async function handleSubmit () {
  try {
    const user = await getUser("tylermcginnis");
    const weather = await getWeather(user.location);

    handleSuccess({ user, weather });
  } catch (e) {
    handleFailure(e);
  }
}
```

```
var xhr = new XMLHttpRequest();
xhr.open("get", "/ajax?name=value", true);

xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == 200 || xhr.status == 304) {
      var statusNode = xhr.responseXML.getElementsByTagName("status")[0],
          dataNode = xhr.responseXML.getElementsByTagName("data")[0];

      if (statusNode.firstChild.nodeValue == "ok") {
        handleSuccess(processData(dataNode))
      } else {
        handleFailure()
      }
    } else {
      handleFailure()
    }
  }
};

xhr.send(null);
```

<https://www.youtube.com/watch?v=rivBfgaEyWQ>

# Los frameworks JS

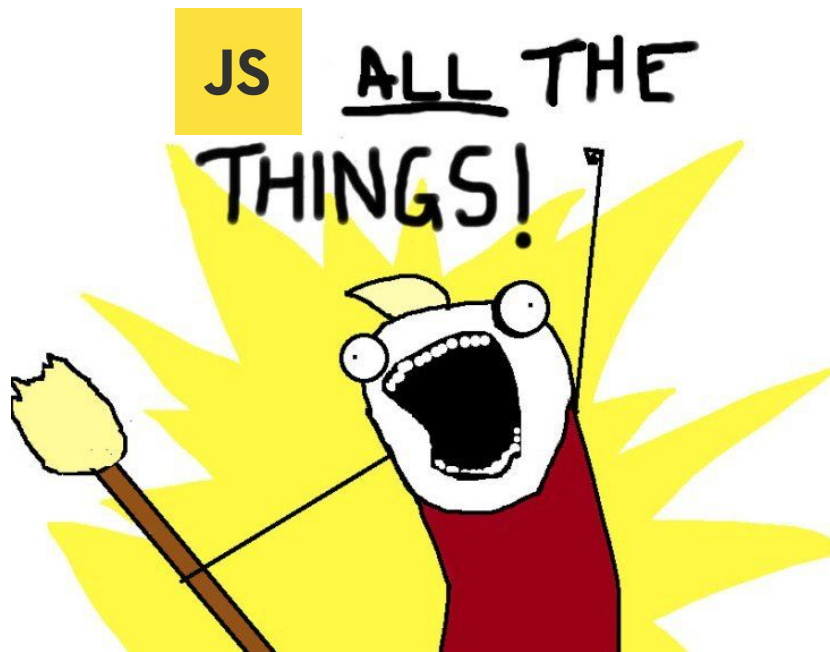
(y sus problemas)

# Los frameworks JS

La idea es mejorar la experiencia de usuario haciendo transiciones entre pantallas con javascript, evitando la necesidad de volver a cargar otro documento.

Prometen facilitar el desarrollo, reduciendo los tiempos de llegar al mercado y mantenimiento

Nuevamente, \$\$\$



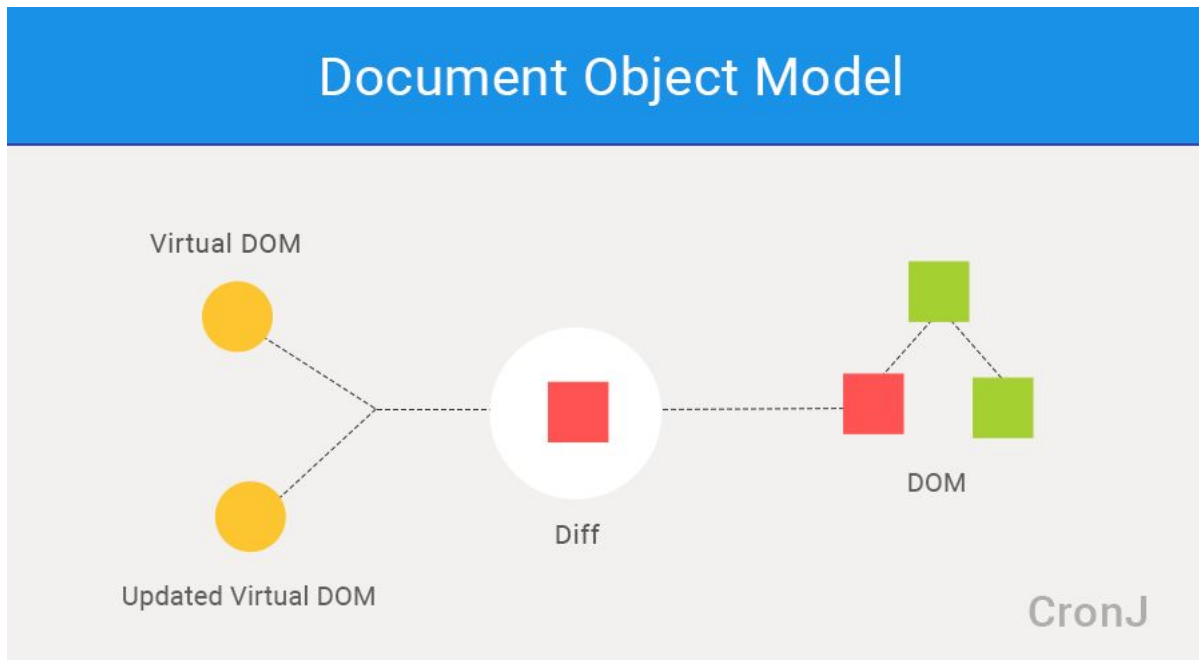
# ReactJS

La historia de facebook es que tenían problemas para mantener las UIs, demasiadas interacciones y era todo un desastre.

```
class HelloMessage extends React.Component {  
  render() {  
    return (  
      <div>  
        Hello {this.props.name}  
      </div>  
    );  
  }  
}
```

# ¿Cómo funciona ReactJS?

ReactJS genera una estructura interna, Virtual DOM, que se genera rápidamente a partir del estado de los componentes. Cuando se dispara un cambio en algún componente, se vuelve a generar un nuevo virtual dom, se realiza un diff con el virtual dom actual, y se actualizan únicamente los componentes necesarios.



# SPA

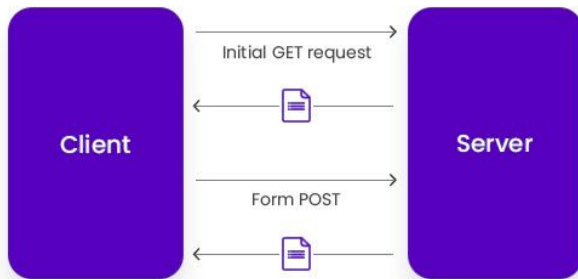
## Ventajas

- Transiciones fluidas
- Luce como una app
- Requests dentro de la app son más livianas
- Bien aplicado y en el escenario correcto mejora la UX

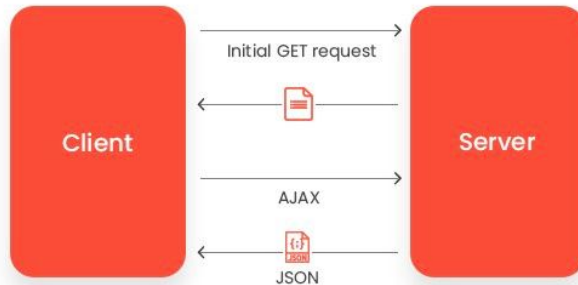
## Desventajas

- Se lleva pésimo con SEO
- Manejo de History / Routing
- Alta complejidad del lado cliente
- Initial Request carga gran parte de la app.

### Traditional Page Lifecycle



### SPA Lifecycle



# Reactivity

```
var duel = $("#Duel");  
  
duel.css("color", "red");  
  
duel.html("<h1>Dark side</h1>");
```



```
var duel = new Vue({  
  
  el: '#Duel',  
  
  data: {  
  
    color: 'green'  
  
    title: 'Light side'  
  
  },  
  
  methods: {  
  
  }  
  
})
```



# ¿Qué pasa con la navegación?



Una facilidad que ofrecen los browsers es la capacidad de navegar hacia adelante y atrás en la historia. Esto es algo que no funciona out-of-the box cuando hacemos navegación client side en JS.

Algunas librerías resuelven este problema haciendo uso de la API de history (React Router con useHistory).

```
<Router>
  <div>
    <Header />

    <Route exact path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/topics" component={Topics} />
  </div>
</Router>
```

# ¿Y si no usamos JavaScript?

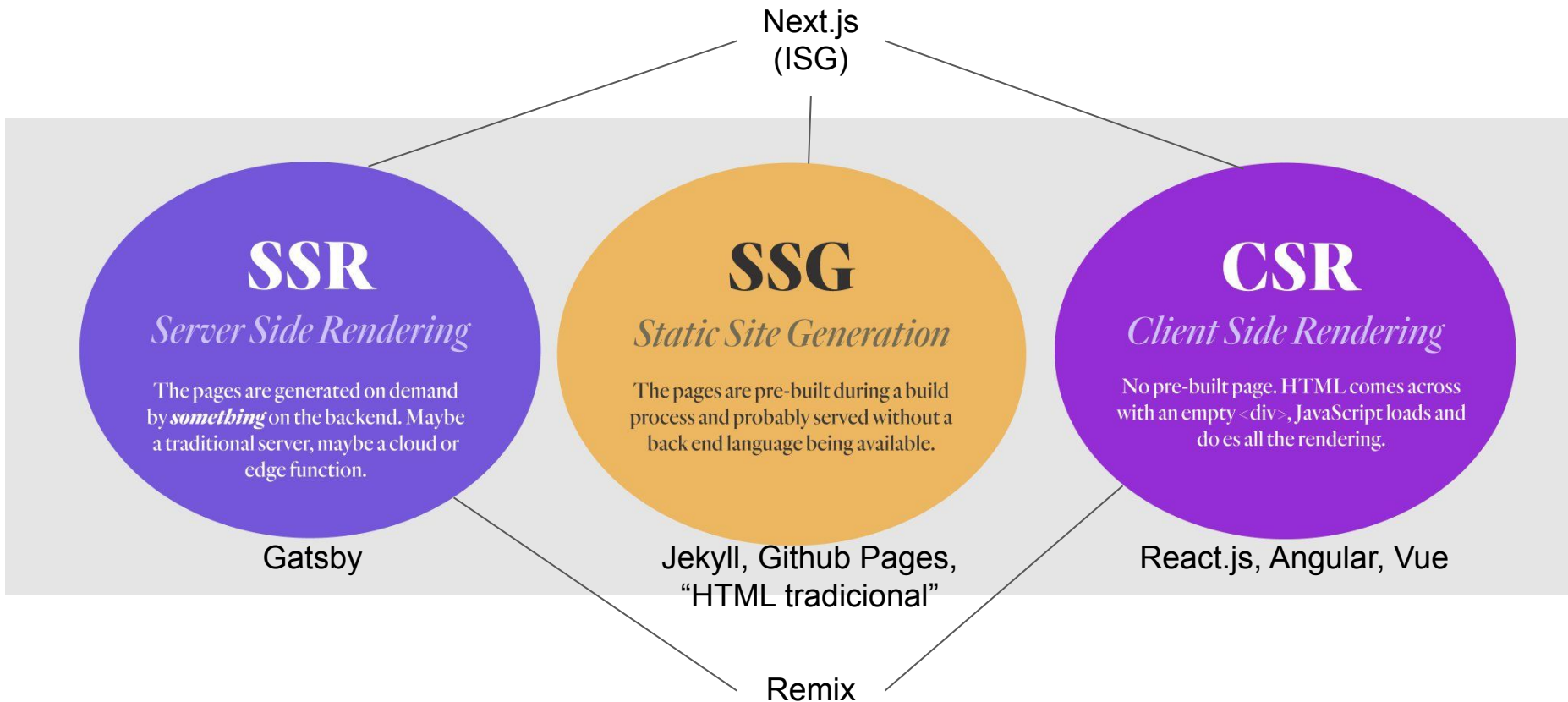
Hoy en día todos los browsers modernos incluyen un motor de JS, y lo tienen habilitado por defecto. Pero es interesante considerar el caso de clientes de mail, y darle soporte a algunas herramientas diseñadas para interpretar el contenido de las páginas (El motor de indexing de google)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <div id="app"></div>
    <script src="app.js"></script>
  </body>
</html>
```

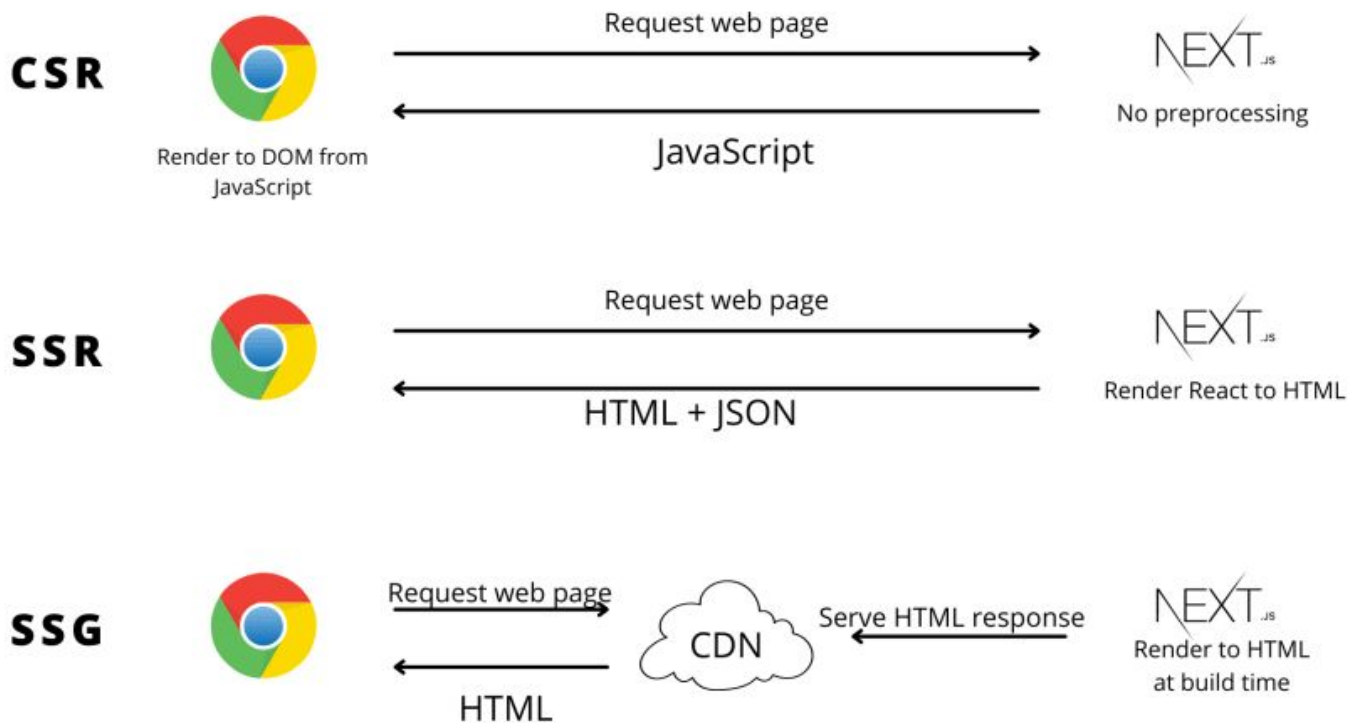
Bot de google



# Distintas formas de servir una app front

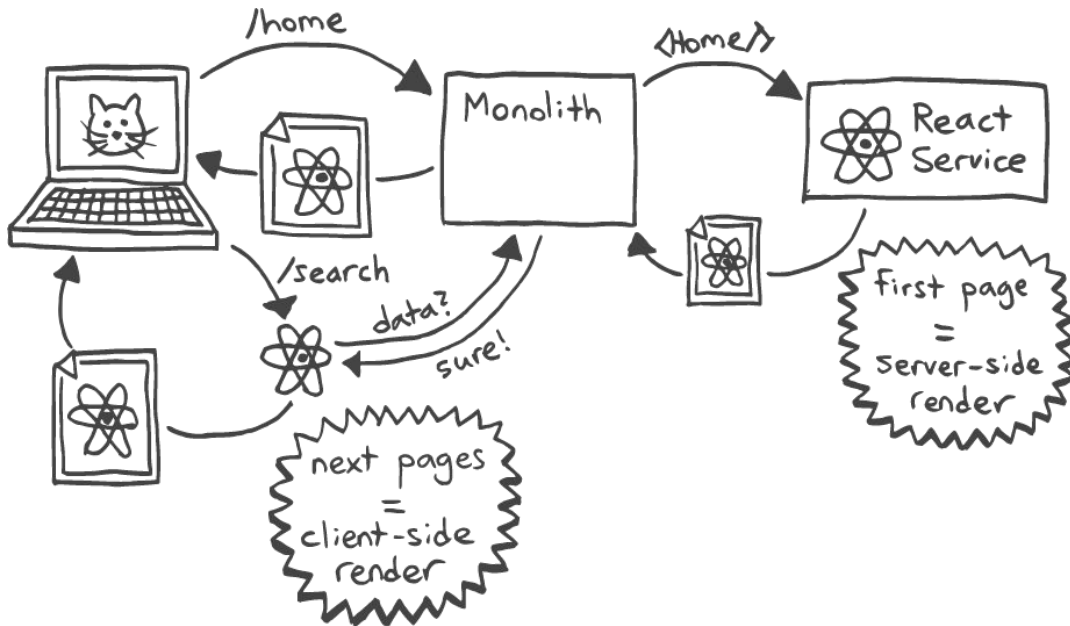


# Distintas formas de servir una app front



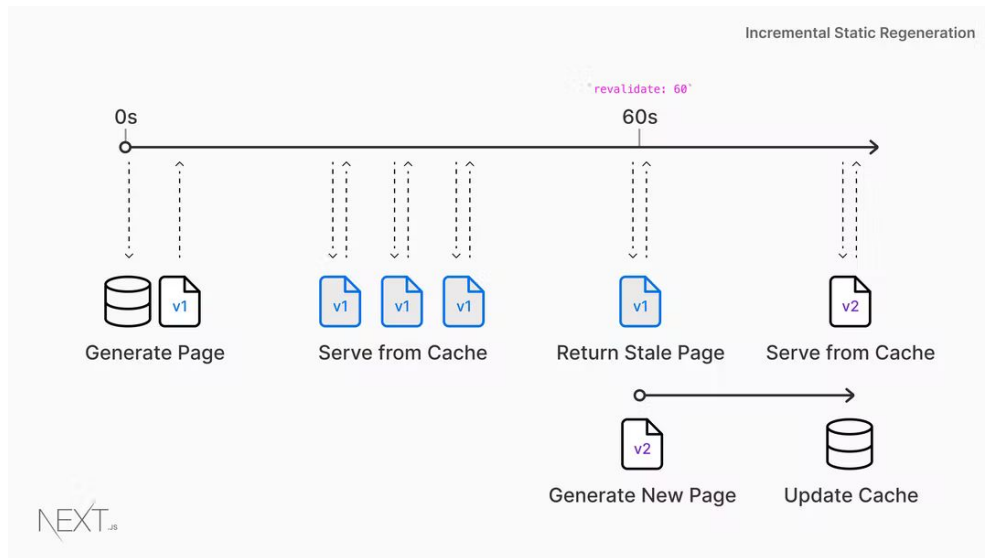
# Server-Side Rendering (SSR)

Se suele poner un servicio que interpreta el javascript, y genera como resultado HTML tradicional, y además sirve la aplicación, de manera que las navegaciones siguientes se realicen desde el cliente.



# Incremental Static (Re)Generation (SSR + SSG) con Next.js

- SSG nos ahorra procesamiento, llamadas a la DB, red. Pero solo muestra contenido estático (no sabe fetchear contenido dinámico a un backend)
- SSR es bastante performante, tiene buen SEO pero requiere un server generando HTML por cada request
- ISG nos permite elegir **por página** si queremos generarla en build-time (SSG) o en runtime (SSR)
  - Si elegimos build-time, permite servir la primera página buildeada e ir cacheando en el tiempo. Podemos hacer esto para, por ejemplo, tener en SSG los 10k productos más mirados
  - Si elegimos runtime buildeamos más rápido (no generamos HTML de antemano) y tenemos SSR



<https://vercel.com/docs/concepts/next.js/incremental-static-regeneration>

# Y volvemos full-circle... server components con Remix

- SSR out of the box. No SSG, no ISR/ISG. Los server components son inmutables por default
- Se puede pensar como un framework VC (de MVC), donde C es el server y V lo renderizado en el browser
- El framework deja elegir el M en base a cualquier DB, ORM o modelado del lado del server
- Actions: los updates (por ejemplo, POSTear un <form>, PUT, PATCH, DELETE) son llamados actions e implican ir de la vista al server (controller) directamente, y Remix se encarga de abstraer el proceso

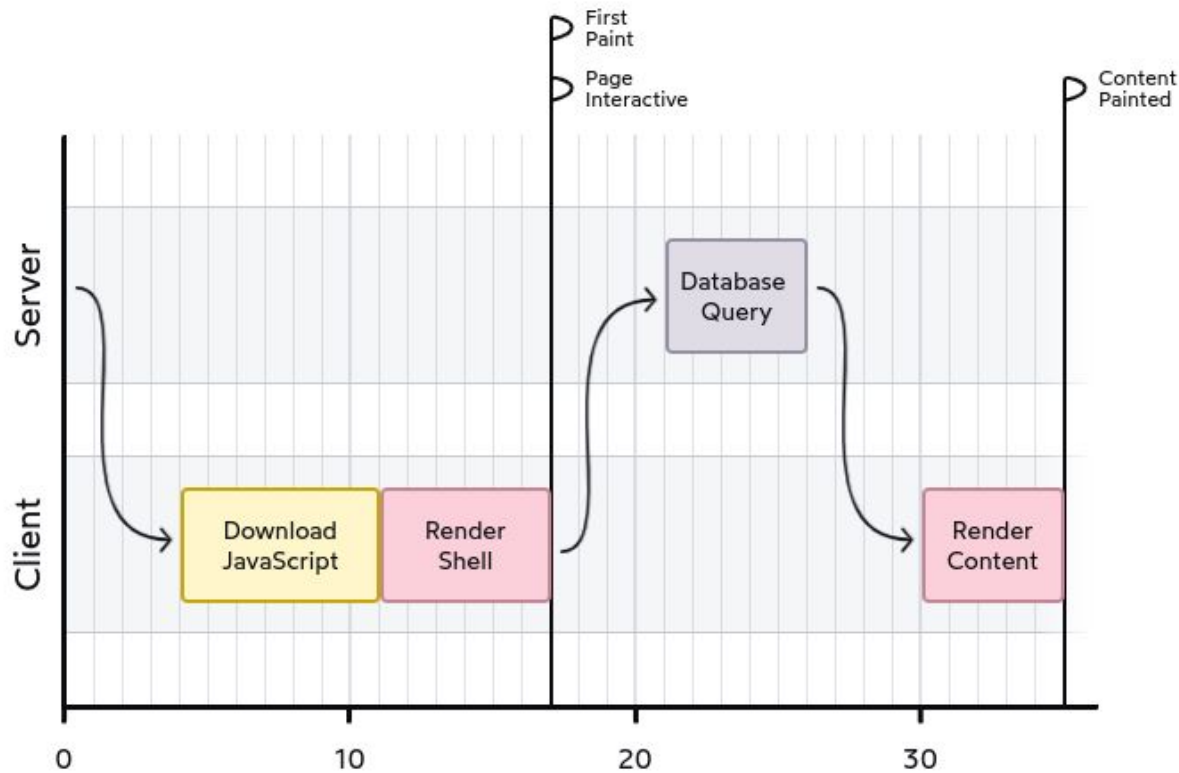
```
export async function getPosts(): Promise<Array<Post>> {  
  return [  
    {  
      slug: "my-first-post",  
      title: "My First Post",  
    },  
    {  
      slug: "90s-mixtape",  
      title: "A Mixtape I Made Just For You",  
    },  
  ],  
};
```

Server (mockeado, podría ser llamado a DB)

```
export default function Posts() {  
  const { posts } = useLoaderData();  
  return (  
    <main>  
      <h1>Posts</h1>  
      <ul>  
        {posts.map((post) => (  
          <li key={post.slug}>  
            <Link  
              to={post.slug}  
              className="text-blue-600 underline">  
              <br>  
              {post.title}<br>  
            </Link>  
          </li>  
        ))}<br>  
      </ul>  
    </main>  
  );  
}
```

Vista del cliente

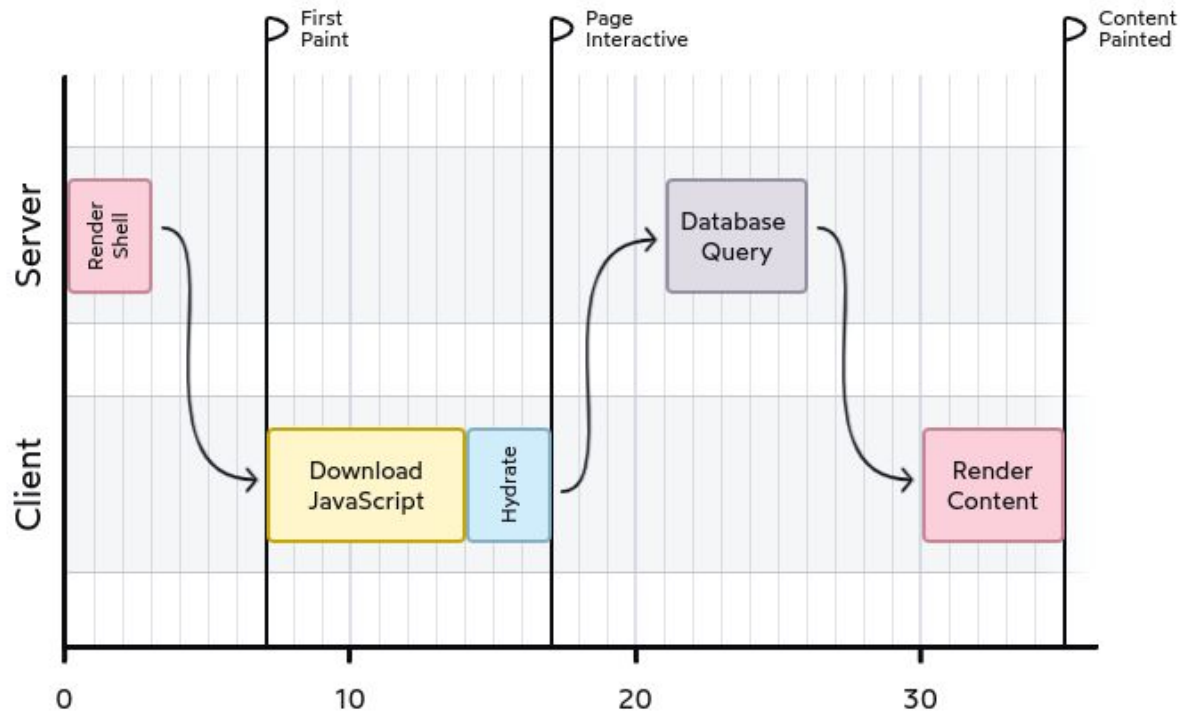
# CSR vs SSR vs RSC (React Server Components)



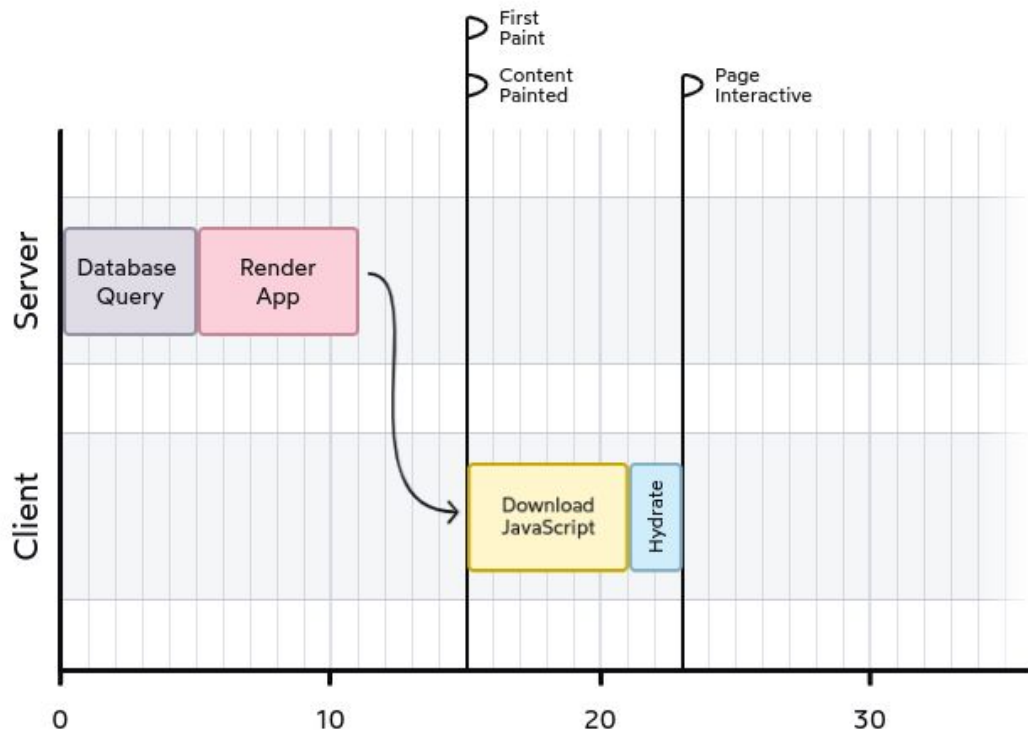
<https://www.joshwcomeau.com/react/server-components/>



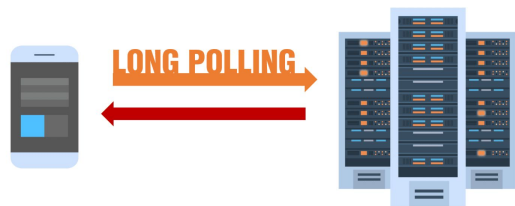
# CSR vs SSR vs RSC (React Server Components)



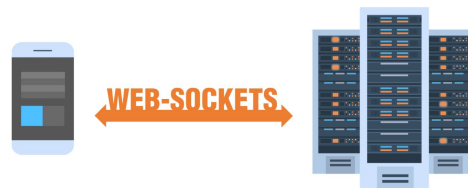
# CSR vs SSR vs RSC (React Server Components)



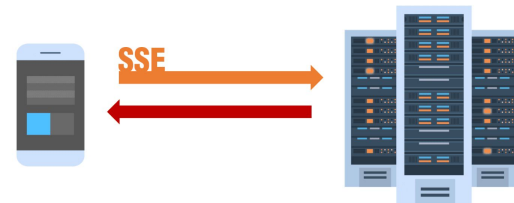
# ¿Y si quiero enviar actualizaciones a mis clientes?



- Unidireccional
- Se mantiene la conexión y se devuelve una respuesta cuando haya novedades o timeout
- Encontrar un timeout correcto
- Exceso de requests



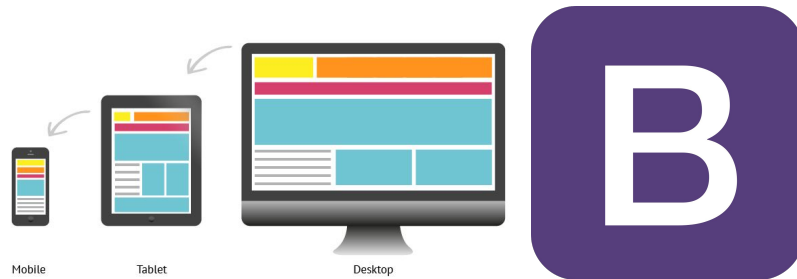
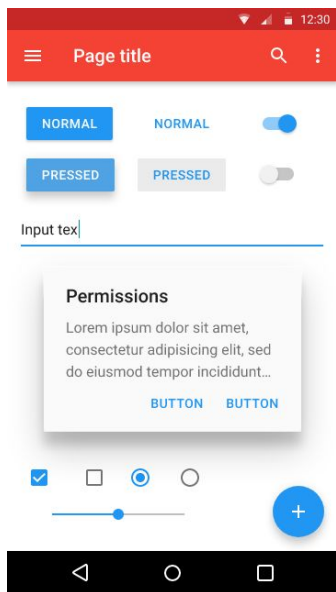
- Bidireccional
- Se mantiene la conexión punto a punto
- Reduce requests



- Inicia el cliente
- Unidireccional del lado server
- Cliente no envia data

# UX - Responsive

- Mobile first
- Bootstrap
- Material Design



```
1
2  /*=====
3  =           Bootstrap 3 Media Queries           =
4  =====*/
5
6  /*===== Mobile First Method =====*/
7
8  /* Custom, iPhone Retina */
9  @media only screen and (min-width : 320px) {}
10
11 /* Extra Small Devices, Phones */
12 @media only screen and (min-width : 480px) {}
13
14 /* Small Devices, Tablets */
15 @media only screen and (min-width : 768px) {}
16
17 /* Medium Devices, Desktops */
18 @media only screen and (min-width : 992px) {}
19
20 /* Large Devices, Wide Screens */
21 @media only screen and (min-width : 1200px) {}
22
23 /*===== Non-Mobile First Method =====*/
24
25 /* Large Devices, Wide Screens */
26 @media only screen and (max-width : 1200px) {}
27
```

# Styling methods

Styled components (~ CSS-in-JS)

```
const Button = styled.button`
  background: transparent;
  border-radius: 3px;
  border: 2px solid palevioletred;
  color: palevioletred;
  margin: 0 1em;
  padding: 0.25em 1em;
`
```

Tailwind (utility-first)

```
<div class="p-6 max-w-sm mx-auto bg-white rounded-xl shadow-md flex items-center space-x-4">
  <div class="flex-shrink-0">
```

SCSS (preprocessors)

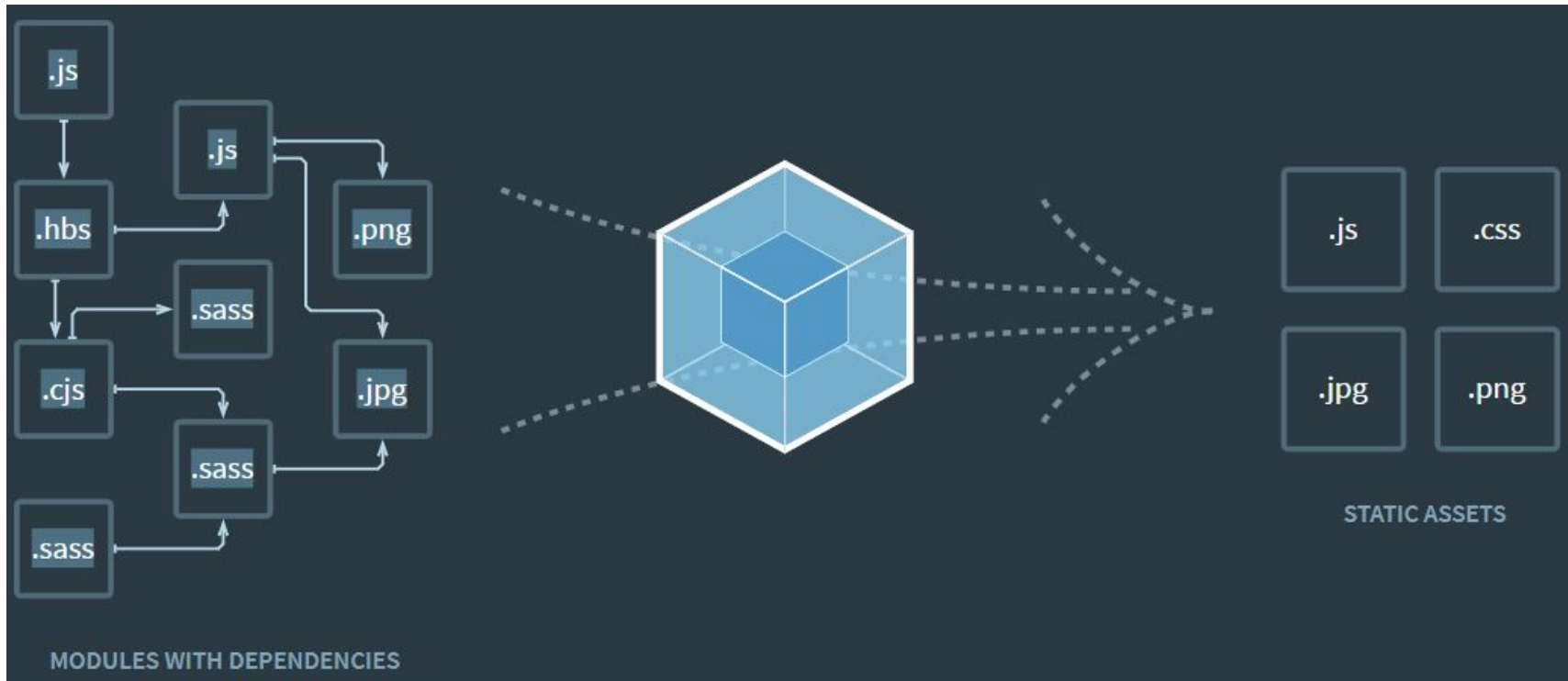
```
@mixin reset-list {
  margin: 0;
  padding: 0;
  list-style: none;
}

@mixin horizontal-list {
  @include reset-list;

  li {
    display: inline-block;
    margin: {
      left: -2px;
      right: 2em;
    }
  }
}
```

Temas bonus

# Module bundlers



# Webpack loaders

webpack.config.js > ...

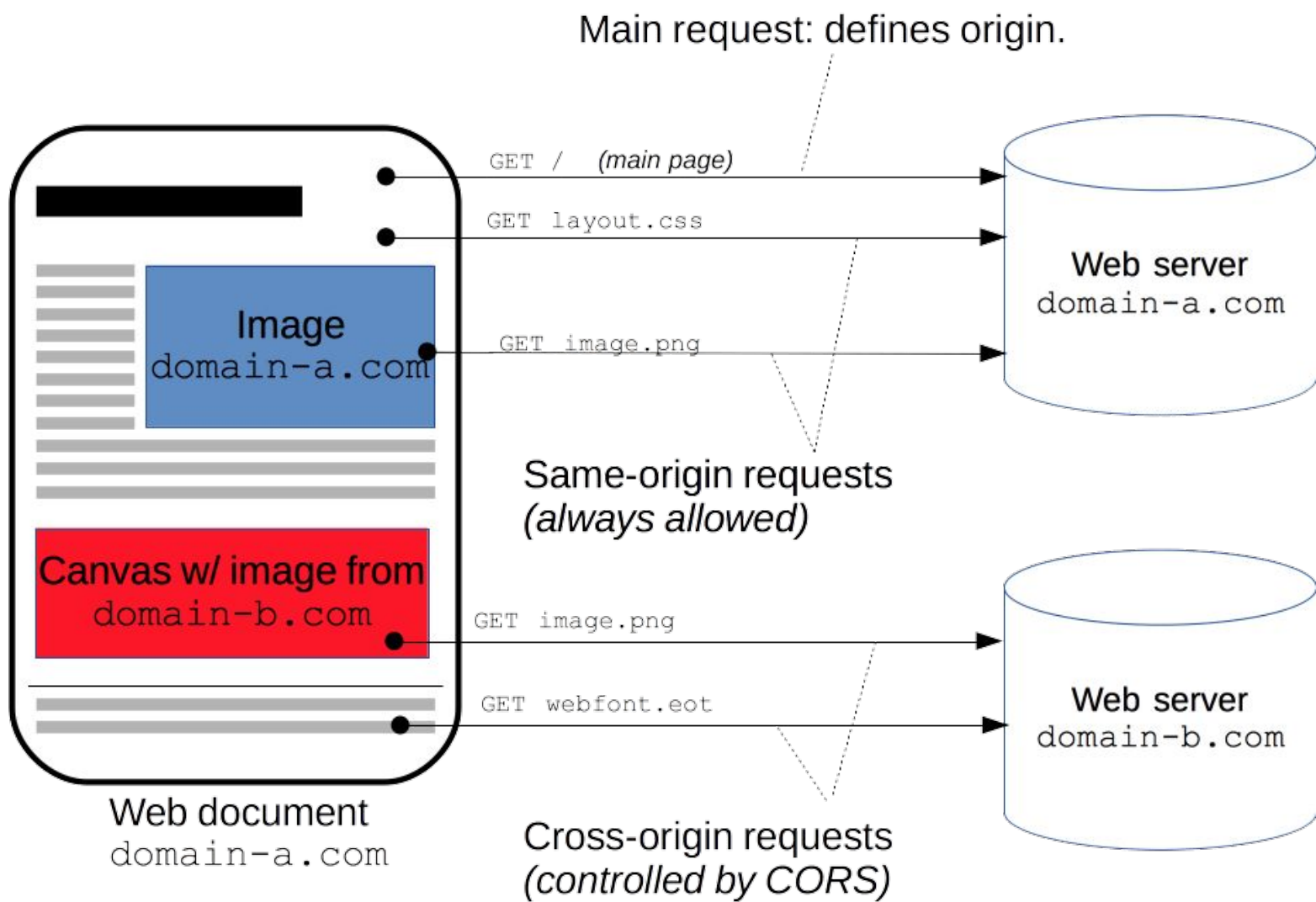
```
1  module.exports = {
2    entry: "./src/index.js", // aca podriamos usar code splitting
3    output: {
4      filename: "main.js",
5    },
6    module: {
7      // loader: objeto que contiene los "metodos" use y test. sabe procesar archivos que no sean js
8      rules: [
9        {
10         test: /\.scss$/, // que archivos puedo load con el siguiente use?
11         use: [ // loaders a aplicar
12           'style-loader', // CSS al DOM
13           'css-loader', // convierte CSS a CommonJS
14           'sass-loader' // compila Sass a CSS
15         ],
16       },
17     ],
18   },
19 };
```



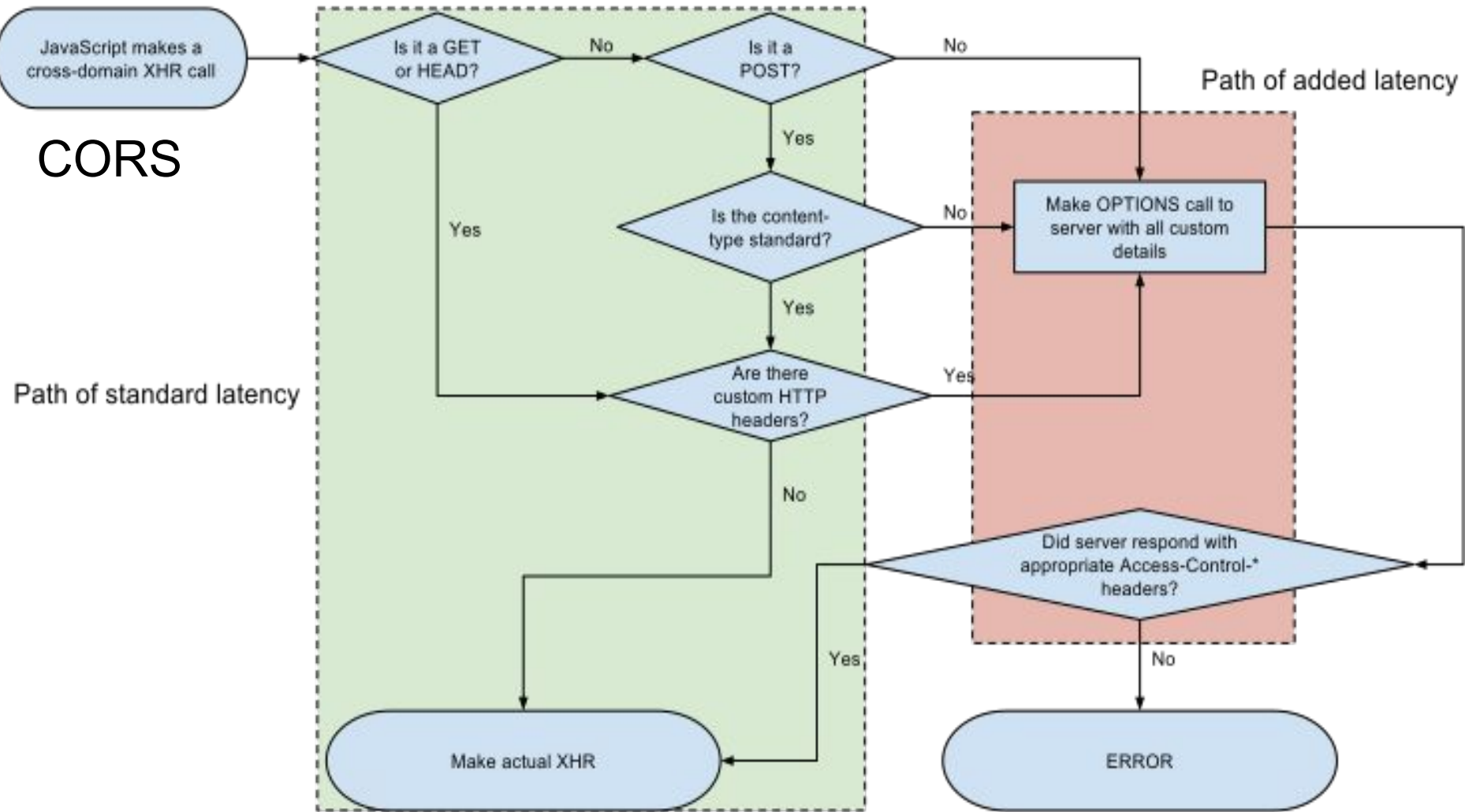
# Features de module bundlers

- Soporte para ES6 modules
- Soporte a través de loaders (CSS, JS en distintas formas)
- Compatibilidad cross-browser (target fijado)
- Polyfills
- Minifying, uglifying
- Hot reload
- Y mucho más

# CORS



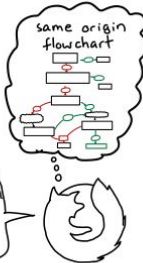
# CORS



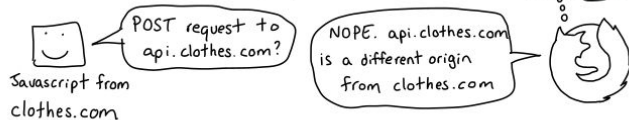
# CORS

cross-origin resource sharing

Cross-origin requests are not allowed by default:  
(because of the same origin policy!)



## CORS



If you run api.clothes.com, you can allow clothes.com to make requests to it using the Access-Control-Allow-Origin header.  
Here's what happens:



This OPTIONS request is called a "preflight" request, and it only happens for some requests, like we described in the diagram on the same-origin policy page. Most GET requests will just be sent by the browser without a preflight request first, but POST requests that send JSON need a preflight.

Preguntas?

# Bibliografía, links copados, temas extra que no entraron

- [What happens when you type a URL into your browser?](#)
- [HTTP Flow](#) - [High Performance Browser Networking](#), todos los protocolos
- [Choose a canonical URL](#) - [Googlebot y distintas representaciones de la misma URL](#)
- [Qué es el HTML?](#) - [Qué es el CSS?](#) - [Qué es JS?](#)
- [Can I use?](#) Browser compatibility
- [Measuring Frontend Performance](#)
- [Critical resources in the first 14KB???](#) - [TCP Slow Start](#) - [TCP Slow Start 2](#) - [Relevant YCombinator discussion](#)
- [The Story of Asynchronous JavaScript](#)
- [SSR, SSG, CSR](#)
- [Next.js ISG](#)
- [Svelte](#)
- [How to center in CSS](#) - [Bootstrap](#) - [Flexbox](#) y [CSS Grid](#)
- [SCSS/SASS](#), [LESS](#) - variantes de CSS y preprocesadores
- [Webpack](#) - [Charla Webpack](#)
- [Micro Frontends](#) - [Caso Micro Frontends](#) - [Module Federation](#)
- AB Testing para productos front end - [Guía de ABTasty](#)
- Distintas versiones de javascript y babel - [BabelJS](#)
- TypeScript - [Aguante TypeScript](#)
- NodeJS y javascript en el backend - [About NodeJS](#)
- Deno y por qué NodeJS es tan malo (por el creador de NodeJS) - [10 Things I Regret About Node.js](#)