UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Divide et impera: desarrollo modular de aplicaciones web

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Isidro Torregrosa Torralba
Tutor: Patricio Orlando Letelier Torres
Curso 2019 - 2020

Resumen

Resumen en español.
Palabras clave: a, b, c, d.

Resum

Resumen en valencià.
Paraules clau: a, b, c, d.

Abstract

Abstract in English.
Keywords : a, b, c, d.

Tabla de contenidos

1. INTRODUCCIÓN 9

1.1. MOTIVACIÓN 9
1.2. OBJETIVOS 10
1.3. ESTRUCTURA 11

2. ESTADO DEL ARTE DEL DESARROLLO DE APLICACIONES WEB 13

2.1. ARQUITECTURAS DE APLICACIONES WEB 13
2.2. TECNOLOGÍA PARA EL DESARROLLO DE APLICACIONES WEB 14
2.2.1. TECNOLOGÍAS POPULARES 14
2.2.2. WEB COMPONENTS V1 16
2.3. MÓDULOS DE JAVASCRIPT 18
2.3.1. FORMATOS DE MÓDULOS ESTANDARIZADOS POR LA INDUSTRIA 18
2.3.2. FORMATO DE MÓDULOS ESTANDARIZADO POR EL LENGUAJE 18
2.3.3. EMPAQUETADORES 19
2.3.4. NUEVAS PROPUESTAS PARA EL ESTÁNDAR: IMPORT MAPS Y CSS MODULES 19
2.4. DESPLIEGUE DE APLICACIONES WEB 22
2.4.1. CALIDAD DEL CÓDIGO 22
2.4.2. PRUEBAS UNITARIAS 22
2.4.3. PRUEBAS DE INTEGRACIÓN 22
2.4.4. PROCESO DE CONSTRUCCIÓN 22
2.4.5. CACHÉ 22
2.5. APLICACIONES WEB MONOLÍTICAS 22
2.6. MICRO FRONTENDS 22
2.6.1. QUÉ SON LOS MICRO FRONTENDS 22
2.6.2. SOLUCIONES EXISTENTES 22
2.7. GESTIÓN DE LA BASE DE CÓDIGO Y EL CONTROL DE VERSIONES 23
2.7.1. GIT 23
2.7.2. MULTI REPOSITORIO 23
2.7.3. SUBMÓDULOS DE GIT 23
2.7.4. MONO REPOSITORIO 23
2.7.5. VERSIONADO DE LA APLICACIÓN 23

3. ANÁLISIS DEL PROBLEMA 23

3.1. PROBLEMAS DEL DESARROLLO WEB EN LA ACTUALIDAD 23
3.2. POSIBLES SOLUCIONES 23
3.3. SOLUCIÓN PROPUESTA 23
3.4. PLAN DE TRABAJO 23

4. DISEÑO DE LA SOLUCIÓN 25

4.1. ARQUITECTURA DEL SISTEMA 25
4.2. DISEÑO DETALLADO 25
4.3. TECNOLOGÍAS UTILIZADAS 25

5. DESARROLLO DE LA SOLUCIÓN PROPUESTA 27
6. EVALUACIÓN DE LA SOLUCIÓN 29
7. CONCLUSIONES 33

7.1. RELACIÓN DEL TRABAJO DESARROLLADO CON LOS ESTUDIOS CURSADOS 33

8. TRABAJOS FUTUROS 35
9. REFERENCIAS 36

10. ANEXOS 43
11. GLOSARIO 45

Tabla de figuras

Figura 1.1: serie temporal de la descarga de Kilobytes por página [2] 9
Figura 1.2: número de módulos por año por ecosistema [3]..... 10
Figura 2.1.1: ciclo de vida de una MPA [9]..... 13
Figura 2.1.2: ciclo de vida de una SPA [9] 14

1. Introducción

1.1. **Motivación**

El desarrollo web ha experimentado un cambio de tendencia con respecto a sus objetivos durante la última década. La finalidad principal del desarrollo web ha sido, históricamente, la creación de páginas: documentos que aportan contenido gráfico y textual al usuario sin que este necesite interactuar con el mismo. En la actualidad son cada vez más las empresas u organizaciones que optan por el desarrollo de aplicaciones, las cuales no sólo aportan contenido e información, si no que tienen como principal objetivo ofrecer funcionalidad por medio de la interacción con el usuario. Este cambio de tendencia se debe en parte a que este tipo de aplicaciones están ocupando el lugar de las aplicaciones nativas, tanto de plataformas móviles como de escritorio, pues, entre otras ventajas son más baratas de desarrollar y mejora en varios aspectos la experiencia del usuario final [1].

El aumento de requisitos funcionales a los que las aplicaciones web deben hacer frente supone un incremento en el tamaño de las mismas. Como se aprecia en la figura 1.1, la transferencia total de recursos pedidos por una página creció de media, desde 2010, un 334,4% en dispositivos de escritorio y un 1187,4% en dispositivos móviles.

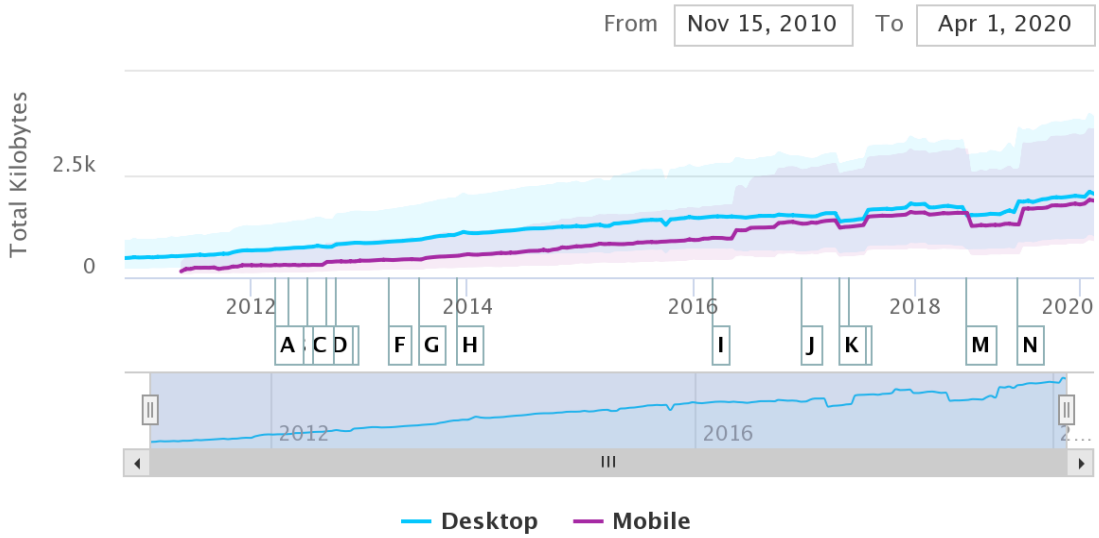

Figura 1.1: serie temporal de la descarga de Kilobytes por página [2]

También es notable el crecimiento del ecosistema con el fin de hacer frente a los nuevos problemas y a las expectativas cada vez más exigentes de los usuarios. La cantidad de nuevas herramientas de JavaScript, el lenguaje de programación predominante en el desarrollo web, ha aumentado de forma drástica comparada con cómo lo han hecho las herramientas de otros ecosistemas como el de Java o .NET,

plataformas muy populares en el desarrollo de aplicaciones de servidor. La figura 1.2 muestra la cantidad de módulos publicados en los repositorios centrales de los distintos ecosistemas desde el 2012 hasta la actualidad.

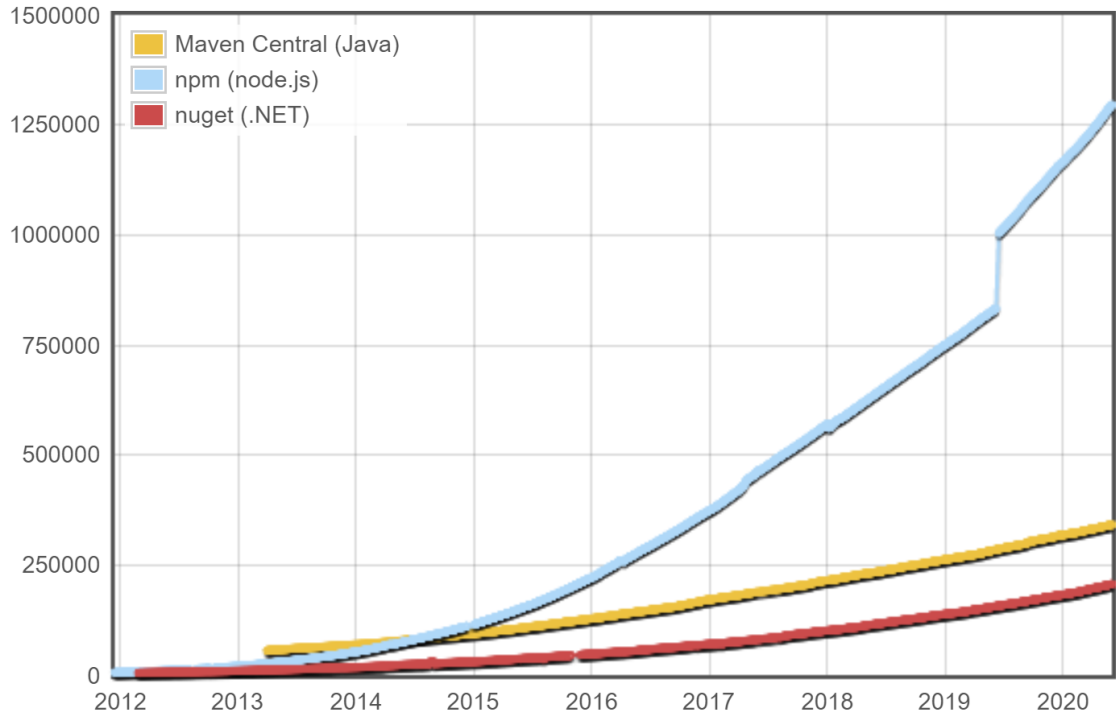

Figura 1.2: número de módulos por año por ecosistema [3]

Si bien la introducción de nuevas herramientas en el desarrollo y su sofisticación hacen que muchas de las tareas sean más simples no por ello el trabajo del desarrollador resulta más sencillo pues "*Cuando tu campo cambia tan rápido que las buenas prácticas quedan obsoletas en un plazo de dos años, tienes que correr para mantenerte al día*" [4].

Así pues, el aumento de los requisitos funcionales, que necesariamente conlleva un aumento en el tamaño del código fuente, sumado al cambio constante del ecosistema, tiene un impacto directo en la complejidad de las aplicaciones, repercutiendo de forma negativa en los procesos de desarrollo y mantenimiento [5]. Esto a su vez tiene una correlación con el aumento del tiempo transcurrido entre la implementación de una funcionalidad y su puesta en producción [6], es decir, el plazo de lanzamiento, y con el aumento de la posibilidad de introducir errores a la hora de realizar cambios en la funcionalidad existente.

Esta memoria pretende proponer un marco de trabajo en el que el incremento en la complejidad de la aplicación no suponga un incremento en la complejidad de su desarrollo, paliando así los efectos negativos que de ella derivan, mejorando la experiencia del usuario final y del equipo de desarrollo en el proceso.

1.2. **Objetivos**

El objetivo de este proyecto, a alto nivel, es la implementación de un marco de trabajo que permita desarrollar una aplicación web de forma sencilla y mantenible, independientemente de la envergadura de la misma. Los objetivos específicos son:

- Desarrollar un marco de trabajo en el que la complejidad sea estable a medida que aumenta la cantidad de funcionalidad ofrecida por la aplicación.
- Estudiar diferentes alternativas tecnológicas a la hora de implementar este marco de trabajo.
- Evaluar los resultados obtenidos desarrollando una aplicación de banca electrónica utilizando este marco de trabajo.

Cabe mencionar que el alcance de este trabajo se limita a la parte de cliente de las aplicaciones web y por tanto no abarca la parte de servidor.

1.3. **Estructura**

Repaso breve sobre lo que sigue y en qué partes de alto nivel se divide el proyecto.

2. Estado del arte del desarrollo de aplicaciones web

2.1. Arquitecturas de aplicaciones web

Actualmente existen diferentes alternativas a la hora de construir una aplicación web en función de dónde reside la lógica de la interfaz de usuario, de cómo el navegador obtiene los recursos cuando el usuario visita una página y de qué ocurre en las subsiguientes navegaciones [7].

2.1.1. *Multi Page Application (MPA)*

MPA es el nombre que recibe la arquitectura utilizada tradicionalmente en el desarrollo de aplicaciones web, basado en confeccionar una aplicación a partir de múltiples páginas. Se caracteriza por ser el servidor de la aplicación el que realiza la mayor parte de la lógica y el que ante cada navegación genera los contenidos de la página y los envía al cliente.

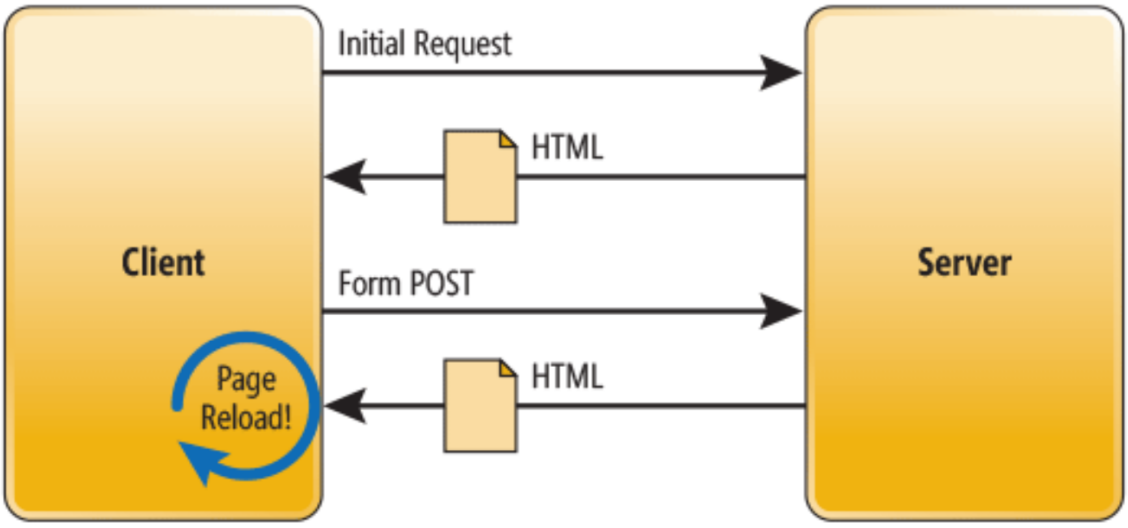

Figura 2.1: ciclo de vida de una MPA [8]

Como el estado de la aplicación se pierde con cada navegación al reemplazarse la página por una diferente, esta arquitectura suele ser preferible cuando los requisitos funcionales en la parte de cliente son simples o de solo lectura [9].

2.1.2. *Single Page Application (SPA)*

En las aplicaciones de una sola página el servidor provisiona al navegador de los recursos necesarios para presentar el contenido, bien en la primera carga de la página o de forma dinámica y bajo demanda. El navegador, que contiene la mayor parte, si no

toda, la lógica de la interfaz gráfica, se encarga de reemplazar el contenido de la página con cada navegación.

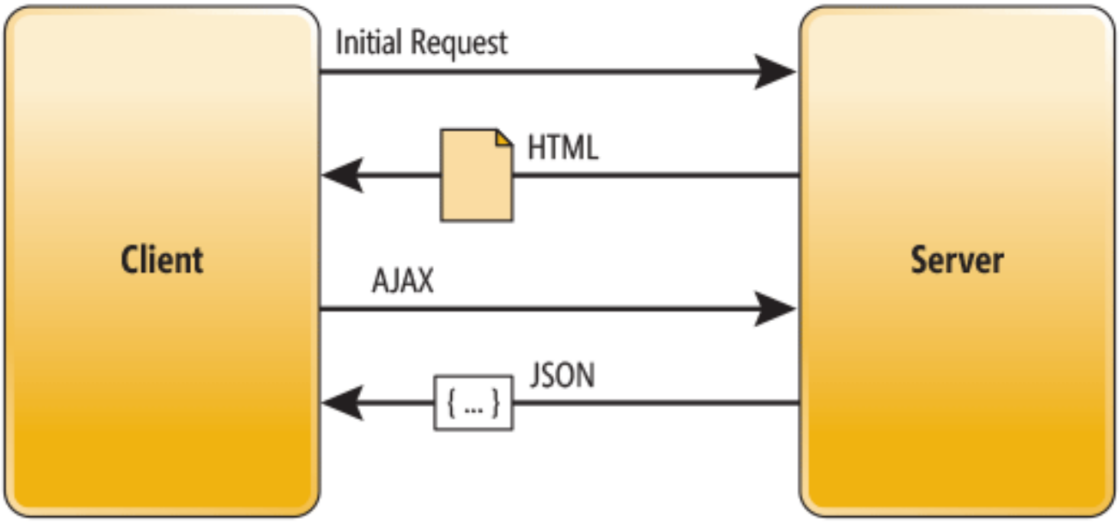

Figura 2.2: ciclo de vida de una SPA [8]

Al no recargarse la página, el estado se mantiene en el cliente lo cual permite ofrecer una mejor experiencia a los usuarios y soportar funcionalidad más compleja [9], haciendo que la página web se asemeje a una aplicación nativa.

Este modelo de aplicaciones trae mejoras sustanciales para la experiencia de usuario, pero con un coste. Mover la lógica de la interfaz del servidor al cliente no elimina la complejidad, simplemente la desplaza, introduciendo a su vez nuevos retos en cuanto a la arquitectura y el despliegue de la aplicación web.

Esta será la arquitectura a emplear durante el desarrollo del proyecto debido a las oportunidades que ofrece.

2.2. Tecnología para el desarrollo de aplicaciones web

Con el auge de la popularidad de las aplicaciones web de una página han surgido una gran cantidad de librerías y *frameworks* con el objetivo de facilitar el desarrollo de este tipo de aplicaciones. Si bien cada una de estas herramientas tienen características que las hacen únicas, las más populares y modernas coinciden en emplear un modelo de desarrollo baso en componentes.

2.2.1. Desarrollo orientado a componentes

El desarrollo orientado a componentes propone una metodología para construir aplicaciones de forma modular, basada en la composición de piezas bien definidas e independientes a las que denomina componentes. Estos componentes pueden, a su vez, estar compuestos a partir de otros componentes más pequeños o que cumplen con una función más específica [10].

```
1   import React from 'react';
2   import {AppLayout, Navigation, Header, Content} from './components';
3
4   export class App extends React.Component {
5       render(){
6           return (
7               <AppLayout>
8                   <Navigation />
9                   <Header />
10                  <Content />
11              </AppLayout>
12          );
13      }
14  }
```

Figura 2.3: ejemplo de componente implementado en React

Desarrollar una aplicación aplicando esta metodología, es decir, separando los distintos bloques funcionales en componentes reutilizables, independientes y diseñados para interoperar entre sí aporta una serie de ventajas notables:

- **Interfaces pequeñas:** al reducir la funcionalidad que implementa cada una de las piezas que forman la aplicación se consigue reducir el tamaño de sus interfaces, lo que hace que sean más fáciles de utilizar.
- **Reusabilidad:** cuando la funcionalidad de la aplicación está separada en módulos bien definidos diseñados para interoperar entre sí se favorece la reutilización de los mismos, evitando así la necesidad de volver a implementar funcionalidad ya existente.
- **Capacidad de composición:** la posibilidad de crear nuevos componentes a partir de los ya existentes supone una gran ventaja en cuanto a la reutilización de código.
- **Mantenibilidad:** el hecho de que cada componente pueda ser dividido en otros componentes que realicen una función específica favorece el testo y la documentación de los mismos.
- **Separación de responsabilidades:** si cada componente tiene la responsabilidad de resolver un problema en concreto su implementación y evolución serán más sencillas.

Los puntos anteriores pueden resumirse en *"El secreto para construir grandes cosas de forma eficiente es, por lo general, evitar construirlas. En su lugar, compón esa gran cosa a partir de piezas más pequeñas y concentradas"* [11].

2.2.2. Tecnologías populares

Actualmente existen tres herramientas que destacan por su gran acogida en la industria, estas son React, Angular y Vue.

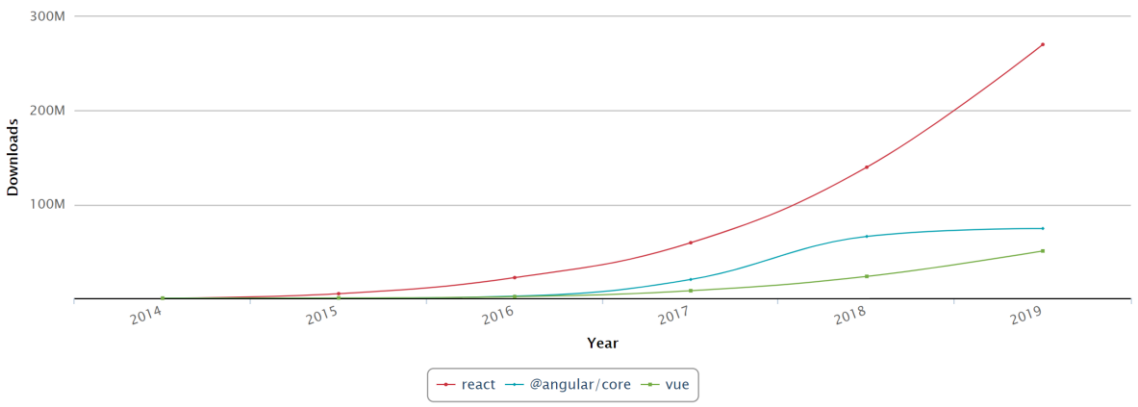

Figura 2.4: descargas anuales por librería durante los últimos cinco años [12]

Estas herramientas tienen grandes ecosistemas y documentación abundante lo que hace que aprender a utilizarlas sea relativamente sencillo. De la misma forma, cada herramienta cuenta con una serie de ventajas e inconvenientes derivados de las decisiones técnicas o los compromisos tomados por sus respectivos equipos de desarrollo [13] pero las tres presentan un gran problema que resulta especialmente relevante a la hora de desarrollar aplicaciones web de cierta envergadura, y es que, debido a que cada herramienta propone un modelo de componente diferente al del resto, los componentes desarrollados en cada una de las distintas tecnologías **no son interoperables** [14].

Esto implica, por un lado, que migrar un proyecto a otra tecnología, pese a suponer esta una ventaja competitiva, puede no ser viable debido al coste derivado de la necesidad de reescribir cada uno de los componentes y, por otro lado, la imposibilidad de compartir componentes entre distintos proyectos desarrollados en diferentes tecnologías.

2.2.3. *Web Components v1*

La web utilizada documentos escritos en *HyperText Markup Language* (HTML), para definir la estructura y el contenido de la página, también referida como documento. Este lenguaje de marcado proporciona una gran cantidad de etiquetas o elementos que constituyen los componentes básicos a partir de los cuales construir un documento HTML [15].

La especificación *Web Components v1* consiste en una serie de características web que permiten crear elementos HTML personalizados y ofrecen mecanismos para su encapsulación [16].

Los dos grandes bloques que conforman la especificación son los ***Custom Elements v1***, que definen un modelo de componentes de bajo nivel basado en estándares web, y el ***Shadow DOM v1,*** un mecanismo por el cual los componentes pueden encapsular su apariencia y comportamiento. Un elemento que utilice *Shadow DOM* ocultará los detalles de su implementación en un fragmento de documento [17] denominado *ShadowRoot* [18]. Los contenidos de cada fragmento no estarán conectados directamente con el resto de la página, lo cual implica que las reglas de estilo definidas

por ese componente no tendrán efecto fuera de él de la misma forma que no se verá afectado por las definidas a nivel global y que la única forma de interactuar con estos componentes desde JavaScript sea a través de la interfaz de programación (API) definida por su desarrollador.

Gracias a estas dos especificaciones, es posible desarrollar componentes basados en estándares y que, por tanto, deberían poder ser utilizados por cualquier herramienta enfocada al desarrollo web, sin hacer distinciones entre estos elementos y cualquier otro elemento HTML definido en el estándar [19].

Es importante tener en cuenta que los *Web Components* no son un *framework* y, por tanto, no cubren todos los aspectos del desarrollo web, su función es la de proveer los bloques básicos para construir aplicaciones, a partir de los cuales se pueda implementar la funcionalidad que se desee, como el manejo del estado de los componentes o un sistema de renderizado basado en plantillas declarativas [20].

Así pues, han surgido diversas librerías basadas en este estándar para definir su modelo de componentes, incorporando las ventajas que de ello derivan y aportando funcionalidad adicional útil para el desarrollador.

Durante el desarrollo de este proyecto se hará uso de ***LitElement*** [21], anteriormente conocida como ***Polymer.*** Existen otras alternativas como ***HyperHTML*** [22] o ***SetencilJS*** [23]. La elección de esta herramienta está motivada, en primer lugar, por tratarse de un complemento sencillo y eficaz que cubre los casos de uso no abordados por el estándar sin introducir cambios sustanciales a la hora de desarrollar y, en segundo lugar, por contar con el respaldo de Google [24].

```
1  import { html, LitElement } from 'lit-element';
2  import './components';
3  class MyApp extends LitElement {
4      render() {
5          return html`
6              <my-app-layout>
7                  <my-header></my-header>
8                  <my-navigation></my-navigation>
9                  <my-content></my-content>
10             </my-app-layout>
11         `;
12     }
13 }
14 customElements.define('my-app', MyApp);
```

Figura 2.5: ejemplo de componente implementado con LitElement

```
1    <body>
2        <my-app></my-app>
3        <script src="my-app.js"></script>
4    </body>
```

Figura 2.6: ejemplo de uso del componente definido en la figura 2.5

2.3. Módulos de JavaScript

Inicialmente los programas de JavaScript eran relativamente pequeños, consistían de unas pocas instrucciones para dotar a la página web de cierta interactividad. A medida que los requisitos funcionales fueron aumentando y, principalmente, como consecuencia de la introducción de este lenguaje en entornos de servidor y la proliferación de aplicaciones web basadas en el modelo de *Single Page Application*, se da la necesidad de contar con un mecanismo capaz de dividir el programa en módulos separados que puedan interactuar entre sí [25].

2.3.1. Formatos de módulos estandarizados por la industria

Dada esta nueva necesidad y a falta de una solución estándar implementada en el propio lenguaje surgen varios formatos de módulos que pueden ser consumidos por librerías de JavaScript o empleados en ciertos entornos de ejecución de JavaScript, como Node.js [26].

Tal fue el grado de adopción de estos formatos de módulos que se convirtieron en estándares de facto, contando, algunos de ellos, incluso con propia especificación, como es el caso de CommonJS [27], el formato de módulos empleado por Node.js o AMD (*Asynchronous Module Definition)* [28], un formato de módulos que ha sido altamente relevante en el desarrollo de aplicaciones web y que continúa utilizándose a día de hoy.

Esto ha supuesto nuevas oportunidades y ha motivado la especificación de un formato de módulos estándar para el lenguaje, pero también ha introducido una fuente de complejidad adicional en el desarrollo de aplicaciones web: la necesidad de consumir módulos definidos empleando distintos formatos.

2.3.2. Formato de módulos estandarizado por el lenguaje

Los módulos de ECMAScript, la especificación del lenguaje JavaScript, fueron introducidos en la versión del 2015, también conocida como ECMAScript6, ES6 o ES2015 [29]. Es común ver referencias a este formato de módulos por el nombre de *ES Modules* o ESM.

Esta especificación define la sintaxis tanto como para exportar variables declaradas en un módulo como para importar las mismas desde otro, así como las reglas que deben cumplir los distintos implementadores a la hora de dar soporte a este formato de módulos en sus respectivas plataformas.

A día de hoy este formato es soportado por las últimas versiones de los navegadores web más populares, lo que supone un 91.23% de los usuarios globales de la web como muestra la figura 2.7:

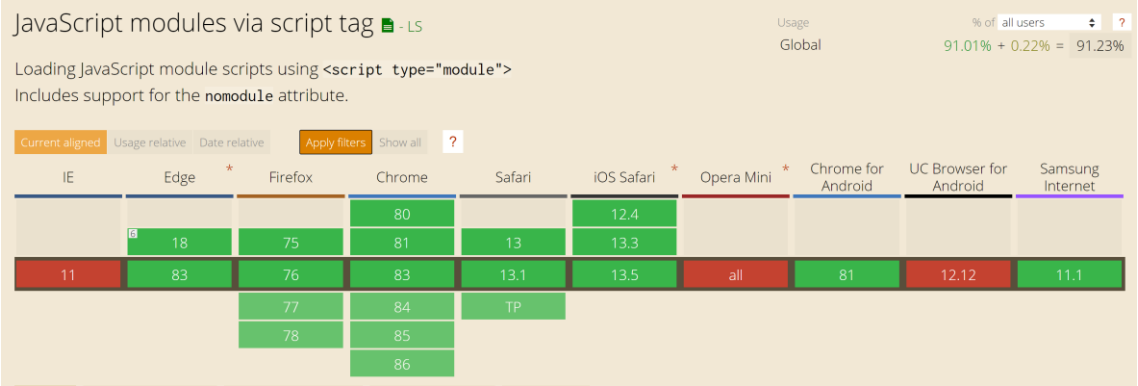

Figura 2.7: tabla de soporte de navegadores para módulos ECMAScript [30]

La especificación inicial contempla un modelo de módulos estático, es decir, cada módulo debe especificar sus dependencias al comienzo del mismo y su ejecución no podrá comenzar hasta que estas hayan sido descargadas y ejecutadas.

En una aplicación web donde el contenido a presentar y, por tanto, los módulos a consumir, dependen de la interacción del usuario, es necesario un sistema de módulos que permita la carga dinámica y bajo demanda de los mismos, con tal de no afectar negativamente al rendimiento de la aplicación empleando recursos en descargar y ejecutar módulos que no van a ser inmediatamente utilizados. Este problema se aborda en la propuesta para añadir la sintaxis *import(specifier)* al lenguaje [31], la cual da la posibilidad de cargar y consumir un módulo de forma dinámica. Esta propuesta ha sido aprobada y se incluirá en la especificación del lenguaje ES2020 [32].

A pesar de que esa versión de la especificación todavía no ha entrado en vigor, actualmente la carga dinámica de módulos ES, está soportada por los navegadores que suponen el 88.77% del uso global de la web [33].

2.3.3. Empaquetadores

Con la introducción de formatos de módulo no estándar surgen herramientas llamadas empaquetadores o *bundlers.* Estas herramientas, permitían, inicialmente, empaquetar los módulos consumidos por una aplicación, independientemente del formato en el que estuviesen definidos, en uno o más archivos denominados *bundles*, los cuales, en función del formato de salida especificado, podrían ser consumidos por otras aplicaciones o entornos de ejecución de JavaScript. Un ejemplo de estos primeros empaquetadores es Browserify [34], el cual permitía y permite ejecutar módulos CommonJS en el navegador.

La necesidad de utilizar estas librerías impone un proceso de construcción en el flujo de trabajo del desarrollo de aplicaciones web, ya no basta con escribir código en archivos JavaScript e incluirlos en la página, estos archivos deberán ser analizados por un empaquetador que genere el código necesario para su ejecución. A medida que este proceso de construcción se hace prácticamente obligatorio si se desea tener la capacidad de consumir librerías de terceros, surgen nuevos empaquetadores que no sólo permiten la interoperabilidad entre distintos formatos de módulos, sino que, además, aplican transformaciones en el código fuente con tal de optimizarlo o soportar casos de usos que de otra forma no hubieran sido posibles, como, por ejemplo, incluir hojas de estilo en el paquete final.

Los dos empaquetadores más populares a día de hoy son Webpack [35] y Rollup.js [36].

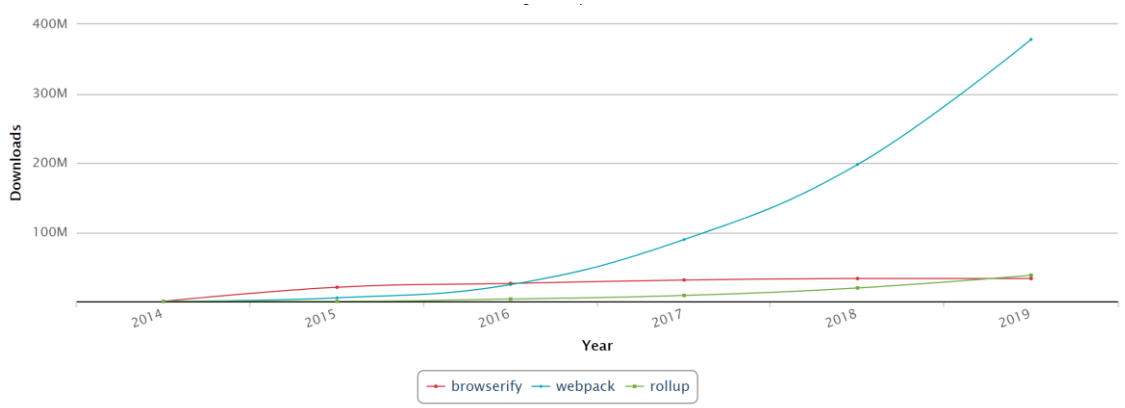

Figura 2.8: descargas anuales por empaquetador durante los últimos cinco años [37]

Ambas herramientas vienen a aportar la misma funcionalidad, extensible vía el uso de complementos, aunque con unas diferencias notables:

- **Compatibilidad**: los dos empaquetadores permiten consumir módulos definidos en cualquier formato y generar paquetes en el formato de nuestra elección. Webpack no soporta la emisión de módulos ES.
- ***Code-splitting:*** esta característica permite la división del código en distintos paquetes que podrán ser cargados bajo demanda y en paralelo, mejorando así la experiencia del usuario final de la aplicación. Rollup únicamente permite aplicar esta técnica al generar paquetes en formatos con soporte para la carga dinámica de módulos mientras que Webpack utiliza un entorno de ejecución personalizado en el que carga módulos definidos en su propio formato no estándar.
- ***Tree-shaking*:** este término, acuñado y popularizado por Rollup e introducido posteriormente en Webpack consiste en la eliminación de código muerto, esto es, el código no utilizado por la aplicación no será incluido en el paquete generado.
- **Resolución de dependencias:** debido a la falta de un formato de módulo estándar la mayoría de librerías están distribuidas para ser consumidas

empleando el algoritmo de resolución de dependencias de Node [38]. Estas herramientas aplican el mismo algoritmo para localizar y empaquetar librerías de terceros.

- **Inclusión de archivos estáticos:** si bien la finalidad principal de estos sistemas es la de empaquetar módulos JavaScript escritos en distintos formatos, las posibilidades que brinda disponer de un proceso de construcción para la aplicación han hecho que este se utilice para incluir todo tipo de archivos estáticos necesarios para el funcionamiento de una aplicación web en el paquete final, como pueden ser hojas de estilo, fuentes o imágenes.

La gran popularidad de Webpack y su enorme ecosistema hacen que sea el empaquetador por defecto en para la mayoría de aplicaciones web, no obstante, el hecho de que no soporte la emisión de módulos en formato estándar y que introduzca su propio entorno de ejecución para dar soporte a la carga dinámica de módulos hace que Rollup sea la herramienta preferida a la hora de empaquetar código para ser compartido [39].

2.3.4. *Import maps* y SystemJS

Desde la estandarización y adopción de los módulos ES en la industria la mayoría de las librerías utilizan este formato para su distribución, no obstante, siguen especificando las dependencias entre módulos de forma que sólo pueden ser resueltas haciendo uso del algoritmo de resolución de Node, el cual se basa en convenciones y en la búsqueda recursiva de módulos en el sistema de archivos. La figura 2.9 ilustra dos formas diferente de especificar de dónde proviene un módulo.

```
1   import someDepenency from 'some-dependency';
2   import anotherDependency from '/node_modules/another-dependency/index.js';
```

Figura 2.9: especificadores de módulo

El especificador empleado en la primera línea, *some-dependency*, no puede ser directamente resuelto al no tratarse de una URL (Uniform Resource Locator) válida, estos especificadores reciben el nombre de *bare module specifiers.* El especificador empleado en la segunda línea, sin embargo, sí constituye una URL válida y el navegador es capaz de descargar el recurso referenciado por ella sin la necesidad de aplicar ningún algoritmo.

El mayor impedimento, por tanto, a la hora de implementar una aplicación web utilizando el formato de módulo ES, es la resolución de *bare module specifiers* desde el navegador de una forma eficiente [40].

Los ***Import maps*** son el mecanismo propuesto para controlar la resolución de especificadores de módulos desde el cliente, creando un mapa de especificadores a URLs. Este mecanismo no sólo soluciona el problema de la resolución de especificadores de módulos si no que también establece las bases para la implementación de técnicas más avanzadas, como el cacheo de larga duración de archivos estáticos [41].

```
<script type="importmap">
    {
      "imports": {
        "some-module": "/node_modules/some-module/index.js"
      }
    }
</script>
```

Figura 2.10: ejemplo de Import map

Esta propuesta todavía no ha sido aprobada, pero puede utilizarse a través del entorno de ejecución **SystemJS**, el cual permite utilizar todas las características de los módulos ES, incluyendo los *Import maps*, en cualquier navegador, siempre y cuando la aplicación sea empaquetada en formato System [41].

El hecho de emplear un formato de módulo no estándar con el fin de aprovechar las características de los módulos ES puede parecer contradictorio, la diferencia entre este formato y otros es que System no constituye una nueva propuesta, si no que funciona de forma absolutamente conforme al estándar, por lo tanto, su finalidad es la de proveer una solución temporal a la falta de soporte de algunas características por parte de algunos navegadores.

2.4. Despliegue de aplicaciones web

2.4.1. Calidad del código
2.4.2. Pruebas unitarias
2.4.3. Pruebas de integración
2.4.4. Proceso de construcción
2.4.5. Caché

2.5. Aplicaciones web monolíticas

- Descripción del enfoque hacia el desarrollo web que promueven las herramientas más utilizadas en la industria a día de hoy.

2.6. Micro Frontends

2.6.1. Qué son los Micro Frontends

- En qué consisten y qué pretenden solucionar.

2.6.2. Soluciones existentes

- Descripción de las soluciones existentes haciendo hincapié en qué problemas solucionan y cuáles no.

2.7. Gestión de la base de código y control de versiones

2.7.1. Git
2.7.2. Multi repositorio
2.7.3. Submódulos de Git
2.7.4. Mono repositorio
2.7.5. Versionado de la aplicación

3. Análisis del problema

3.1. Problemas del desarrollo web en la actualidad

- Gran cantidad de tecnologías
- Curva de aprendizaje
- Interoperabilidad de tecnologías
- Separación de responsabilidades
- Mantenimiento
- Respuesta al cambio
- Productividad

3.2. Posibles soluciones

3.3. Solución propuesta

3.4. Plan de trabajo

Dividir el proyecto entre los distintos aspectos a abordar y hacer una estimación preliminar de lo que podría tardar abordar cada uno de ellos.

4. Diseño de la solución

4.1. **Arquitectura del sistema**

Descripción de las distintas piezas y la relación entre ellas.

4.2. **Diseño detallado**

Descripción detallada de cada una de las piezas que pertenecen a cada pieza arquitectónica explicando su responsabilidad.

4.3. **Tecnologías utilizadas**

Breve repaso de las tecnologías que se han utilizado y en caso de haber tenido alternativa indicar el por qué de la decisión final.

5. Desarrollo de la solución propuesta

Recorrido llevado a cabo para implementar el sistema, reparando en los detalles más interesantes o controvertidos.

6. Evaluación de la solución

Narración de cómo es el flujo de trabajo que expone el sistema y cuáles son los resultados obtenidos.

7. Conclusiones

Conclusiones obtenidas tras la realización del trabajo y comparación entre el resultado obtenido en el apartado anterior, y los objetivos propuestos en resultados anteriores.

7.1. **Relación del trabajo desarrollado con los estudios cursados**

Identificar con cuáles de las asignaturas relación el trabajo y por qué.

8. Trabajos futuros

Posibles mejoras no abordadas y otras vías de investigación.

9. Referencias

[1] V. Collins, «The Decline Of The Native App And The Rise Of The Web App,» Forbes, 5 Abril 2019. [En línea]. Available: https://www.forbes.com/sites/victoriacollins/2019/04/05/why-you-dont-need-to-make-an-app-a-guide-for-startups-who-want-to-make-an-app/. [Último acceso: 31 Mayo 2020].

[2] HTTP Archive, «HTTP Archive,» [En línea]. Available: https://httparchive.org/reports/page-weight?start=earliest&end=latest&view=list. [Último acceso: 31 Mayo 2020].

[3] «Module Counts,» [En línea]. Available: http://www.modulecounts.com/. [Último acceso: 31 Mayo 2020].

[4] K. Ball, «The increasing nature of frontend complexity,» 30 Enero 2019. [En línea]. Available: https://blog.logrocket.com/the-increasing-nature-of-frontend-complexity-b73c784c09ae/. [Último acceso: 31 Mayo 2020].

[5] G. Kapllani, I. Khomyakov, R. Mirgalimova y A. Sillitti, «An Empirical Analysis of the Maintainability Evolution of Open Source Systems,» *IFIP Advances in Information and Communication Technology,* vol. 582, 2020.

[6] J. Wijnmaalen, C. Chen, D. Bijlsma y A. M. Oprescu, «The Relation between Software Maintainability and Issue Resolution Time: A Replication Study,» *Seminar Series on Advanced Techniques & Tools for Software Evolution,* vol. 2510, 2019.

[7] J. Posnick, «Beyond SPAs: alternative architectures for your PWA,» 14 Enero 2019. [En línea]. Available: https://developers.google.com/web/updates/2018/05/beyond-spa. [Último acceso: 6 Junio 2020].

[8] M. Wasson, «ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET,» *MSDN Magazine Issues,* vol. 28, nº 11, 2013.

[9] S. Smith, G. Warren, P. Marcano y M. Wenzel, «Choose Between Traditional Web Apps and Single Page Apps (SPAs),» 12 abril 2019. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps. [Último acceso: 6 Junio 2020].

[10] J. Saring, «A Guide to Component Driven Development (CDD),» 16 Junio 2019. [En línea]. Available: https://itnext.io/a-guide-to-component-driven-development-

cdd-1516f65d8b55. [Último acceso: 6 Junio 2020].

[11] A. Osmani, «Components Should Be Focused, Independent, Reusable, Small & Testable (FIRST),» [En línea]. Available: https://addyosmani.com/first/. [Último acceso: 06 Junio 2020].

[12] P. Vorbach, «npm-stat,» [En línea]. Available: https://npm-stat.com/charts.html?package=react&package=vue&package=%40angular%2Fcore&from=2014-12-12&to=2019-12-19. [Último acceso: 6 Junio 2020].

[13] J. Hannah, «The Ultimate Guide to JavaScript Frameworks,» 16 Enero 2018. [En línea]. Available: https://jsreport.io/the-ultimate-guide-to-javascript-frameworks/. [Último acceso: 6 Junio 2020].

[14] S. Contreras, «Web Components: Seamlessly interoperable,» 15 Abril 2019. [En línea]. Available: https://medium.com/@sergicontre/web-components-seamlessly-interoperable-82efd6989ca4. [Último acceso: 6 Junio 2020].

[15] MDN Contributors, «HTML: Hypertext Markup Language,» MDN, 28 Mayo 2020. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML. [Último acceso: 7 Junio 2020].

[16] Google Developers, «Building Components,» Web Fundamentals, 12 Febrero 2019. [En línea]. Available: https://developers.google.com/web/fundamentals/web-components. [Último acceso: 6 Junio 2020].

[17] MDN Contributors, «DocumentFragment,» MDN, 13 Enero 2020. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/API/DocumentFragment. [Último acceso: 7 Junio 2020].

[18] MDN Contributors, «ShadowRoot,» MDN, 23 Abril 2019. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/API/ShadowRoot. [Último acceso: 7 Junio 2020].

[19] R. Dodson, «Custom Elements Everywhere,» [En línea]. Available: https://custom-elements-everywhere.com/. [Último acceso: 06 Junio 2020].

[20] C. Haynes, «Web Components aren't a framework replacement - they're better than that,» 18 Enero 2020. [En línea]. Available: https://lamplightdev.com/blog/2020/01/18/web-components-arent-a-framework-replacement-theyre-better-than-that/. [Último acceso: 6 Junio 2020].

[21] The Polymer Project, «Lit Element,» [En línea]. Available: https://lit-element.polymer-project.org/.

[22] A. Giammarchi, «hyper(HTML),» [En línea]. Available: https://viperhtml.js.org/hyper.html.

[23] Stencil, «Stencil,» [En línea]. Available: https://stenciljs.com/.

[24] J. Fagnani, «Lightning-fast templates & Web Components: lit-html & LitElement,» Google Developers, 12 Febrero 2019. [En línea]. Available: https://developers.google.com/web/updates/2019/02/lit-element-and-lit-html. [Último acceso: 6 Junio 2020].

[25] MDN Contributors, «JavaScript modules,» MDN, 22 Abril 2020. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules. [Último acceso: 6 Junio 2020].

[26] NodeJS Contributors, «About Node.js,» [En línea]. Available: https://nodejs.org/es/about/. [Último acceso: 6 Junio 2020].

[27] CommonJS, «CommonJS,» 9 Octubre 2011. [En línea]. Available: http://wiki.commonjs.org/wiki/CommonJS. [Último acceso: 6 Junio 2020].

[28] «AMD,» [En línea]. Available: https://requirejs.org/docs/whyamd.html#amd. [Último acceso: 6 Junio 2020].

[29] «ECMAScript® 2015 Language Specification,» Ecma International, Junio 2015. [En línea]. Available: http://www.ecma-international.org/ecma-262/6.0/#sec-modules. [Último acceso: 6 Junio 2020].

[30] Can I Use, «JavaScript modules via script tag,» Can I Use, [En línea]. Available: https://caniuse.com/#feat=es6-module. [Último acceso: 6 Junio 2020].

[31] tc39, «proposal-dynamic-import,» 20 Junio 2019. [En línea]. Available: https://github.com/tc39/proposal-dynamic-import. [Último acceso: 6 Junio 2020].

[32] B. Couriol, «ES2020's Feature Set Finalized,» Infoq, 4 Abril 2020. [En línea]. Available: https://www.infoq.com/news/2020/04/es2020-features/. [Último acceso: 6 Junio 2020].

[33] Can I Use, «JavaScript modules: dynamic import(),» Can I Use, [En línea]. Available: https://caniuse.com/#feat=es6-module-dynamic-import. [Último acceso: 6 Junio 2020].

[34] Browserify, Browserify, [En línea]. Available: http://browserify.org/. [Último acceso: 7 Junio 2020].

[35] Webpack, «Webpack,» [En línea]. Available: https://webpack.js.org/. [Último acceso: 6 Junio 2020].

[36] Rollup.js, «Rollup.js,» [En línea]. Available: https://rollupjs.org/guide/en/. [Último acceso: 7 Junio 2020].

[37] P. Vorbach, «npm-stat,» [En línea]. Available: https://npm-stat.com/charts.html?package=rollup&package=webpack&package=browserify&fro

m=2014-12-12&to=2019-12-19. [Último acceso: 7 Junio 2020].

[38] NodeJS, «Modules,» [En línea]. Available: https://nodejs.org/api/modules.html#modules_modules. [Último acceso: 7 Junio 2020].

[39] hoangbkit, «JavaScript Module Bundlers,» 20 Febrero 2020. [En línea]. Available: https://advancedweb.dev/javascript-module-bundlers. [Último acceso: 7 Junio 2020].

10. Abreviaciones

MPA:

11. Anexos

Anexos si se necesita alguno.

12. Glosario