UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Colección de componentes web mediante tecnología Web Components

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Díaz-Jorge García, Pablo

Tutor/a: Micó Tormos, Pau

CURSO ACADÉMICO: 2022/2023

Resumen y palabras claves

Pablo Díaz-Jorge García

El proyecto se trata de la creación de una librería de Web Components y una aplicación de presentación y documentación de cada componente desarrollado con ayuda del framework StencilJS.

Se hace uso de la herramienta Figma para crear un Design System. En base a estos diseños, se crean los Web Components necesarios para el desarrollo de la aplicación web y posteriormente se empaquetan en una librería llamada Palaze.

Está librería se publica haciendo uso de NPM permitiendo así el uso a nivel global de la librería.

A su vez, se despliega online la aplicación web de presentación y documentación de los componentes haciendo uso de la plataforma Netlify.

Palabras clave:

- Web Components
- StencilJS
- NPM
- Design System
- Biblioteca

Summary and keywords

Pablo Díaz-Jorge García

The Project is about the creation of a Web Components library and a presentation and documentation web application for each component developed with the help of StencilJS framework.

The Figma tool is used to create a Design System. Based on these designs, the Web Components necessary for the development of the web application are created and later packaged in a library called Palaze.

This library is published using NPM thus allowing the global use of the library.

In turn, the web application for the presentation and documentation of the components is deployed online using the Netlify platform.

Keywords:

- Web Components
- StencilJS
- NPM
- Design System
- Library

Índice

Índice de ilustraciones

Índice de tablas

1 Introducción

1.1 Motivación

La tecnología Web Components es muy prometedora para el mundo del desarrollo web y esto implica un gran potencial para el futuro desarrollo frontend.

En el entorno profesional, es una práctica común diseñar componentes personalizados para los proyectos de clientes. A su vez, se suelen utilizar librerías de componentes creadas por terceros para recortar costes en el proyecto debido al tiempo que necesitas emplear para desarrollar esta suite tecnológica.

Los Web Components pueden ser utilizados con frameworks o sin frameworks, son independientes de cualquier framework o biblioteca. A partir de esta y muchas más características surgió la motivación de crear una librería de componentes junto con una aplicación web de documentación donde se explique y se muestre ejemplos de los componentes creados, permitiendo así que cualquier desarrollador pueda hacer uso de la librería para sus proyectos.

1.2 Objetivos

El objetivo principal de este proyecto es la creación de una librería de web components y una aplicación web de documentación que los respalde.

Este objetivo principal está formado por una serie de objetivos más pequeños:

- Diseño de la página de presentación de los componentes.
- Diseño de ejemplo de la página de documentación de los componentes.
- Diseño de cada componente a partir de los dos puntos anteriores.
- Creación de tokens de diseño.
- Desarrollo incremental de los componentes aplicando best practices y análisis mediante la herramienta Lighthouse.
- Creación de la aplicación web con los componentes web implementados.
- Control de versiones y gestión de repositorio.
- Despliegue de la aplicación de documentación online.
- Publicación del paquete de librería.
- Prueba de implementación de los componentes fuera del proyecto.

Otro de los objetivos de este proyecto es profundizar en la tecnología web component aprendiendo lo máximo posible. El factor del desarrollo incremental nos permitirá aprender de forma más eficiente durante el proyecto.

1.3 Estructura del documento

Este proyecto consta de dos documentaciones. Una parte de la documentación se verá reflejada en la aplicación web donde se explicarán todas las características y funcionalidades de cada uno de los componentes con ejemplos gráficos y con ejemplos con código. También habrá una sección de introducción acerca de cómo se usan y se implementa la librería en otros proyectos. Por último, habrá una sección de Design System donde se comentará de forma breve algunas características acerca de este.

La segunda parte de la documentación, la memoria, no se centra únicamente en la biblioteca de componentes, sino en el proceso hasta llegar al resultado final. Consta de las siguientes secciones:

- **Sección 2:** Se analizará toda la información relevante de las tecnologías y herramientas a utilizar, haciendo una segmentación entre las propuestas de desarrollo y una justificación final de la elección para este proyecto.
- **Sección 3:** Se explicará la estructura del proyecto y cómo se ha hecho la gestión de control de versiones del repositorio y también cómo se distribuirá la biblioteca de componentes para uso público.
- **Sección 4:** Explicación y muestra del diseño de la web y los componentes junto al proceso de desarrollo a partir de estos diseños.
- **Sección 5:** Se comentarán las conclusiones a las que se ha llegado después de todo el desarrollo del proyecto y se aportarán ideas para futuras líneas de desarrollo.
- **Sección 6:** Se comentarán las fuentes de referencia de información usadas para la realización del proyecto, tanto la memoria como el código fuente y las herramientas y tecnologías utilizadas.

2 Anteproyecto

2.0 Definición de conceptos relevantes

En esta sección se presentarán algunos de los conceptos que se deberían de conocer para poder entender correctamente la memoria.

- **Design system:** Es un conjunto de guías y patrones para que los productos sean consistentes independientemente de cómo se interactúen con ellos. Facilita el trabajo de los diseñadores debido a que solo se crea una vez el componente. Busca que los productos sean escalables y repetibles.

- **Design tokens:** Abarca desde propiedades de tipografía, estilos de componentes hasta iconos, colores, espaciados… Son independientes del lenguaje y se pueden utilizar en cualquier tipo de proyecto.

- **DOM:** Document Object Model es una representación estructurada y orientada a objetos de un documento HTML o XML. Consiste en una interfaz de programación que permite manipular y acceder dinámicamente a los elementos de un documento web.

  Cuando se carga una web, automáticamente se genera un DOM. Los desarrolladores acceden y modifican el contenido y la estructura del documento posibilitando la creación de aplicaciones web interactivas.

- **Web Components:** De forma genérica, son un conjunto de elementos con lenguaje HTML, estilos con lenguaje CSS y lógica en lenguaje JavaScript que se construyen como un componente siendo personalizables y reutilizables, pudiéndose utilizar de forma independiente en cualquier aplicación web.

- **CSS properties:** Cascading Style Sheets son un conjunto de reglas que definen el diseño de elementos HTML en una página o aplicación web.

- **Custom properties:** Son conocidas como variables personalizadas y es una característica de CSS que nos permite definir variables en un lugar en concreto del documento para que puedan usarse con esa misma referencia a lo largo de este. Siempre se deben definir con dos guiones antes de la o las palabras descriptivas. Este tipo de variables proporciona una amplia gama de posibilidades para desarrollos interactivos.

  Ejemplo: --primary-color:#FFFFFF;

- **Encapsulación:** Es la capacidad de ocultar detalles estableciendo una barrera entorno a los datos permitiendo así solo mostrar una interfaz pública con la que el usuario pueda interactuar con ella. Así se evita el riesgo a comprometer la integridad de los datos y también permite un acceso controlado a la información.

- **Render:** En el desarrollo web, se asocia a la interpretación de código HTML, CSS y JavaScript para construir su representación gráfica encapsulando los elementos en un DOM.

- **W3C:** World Wide Web Consortium tiene como objetivo principal desarrollar y promover estándares web. Abarca muchas áreas y tecnologías, pero en este proyecto se enfocará en el área asociada a los web components. Este consorcio permite que exista una coherencia entre las distintas aplicaciones y navegadores web, evitando problemas de incompatibilidad dependiendo de dispositivos o navegadores.

- **API:** Application Programming Interface es un conjunto de reglas o protocolos para la comunicación e interacción entre distintos software.

  En el caso concreto de este proyecto, cuando se refiera a API en la memoria, se estará refiriendo a API biblioteca, que son un conjunto de reglas, funciones y métodos para facilitar el desarrollo web en un lenguaje específico.

  Ejemplo: API de StencilJS (Documentación acerca del uso e implementación de las herramientas y funcionalidades del framework).

2.1 Estado del arte

En el mundo del desarrollo web el uso de los Web Component Framework se ha vuelto muy común para la creación de aplicaciones web. Algunos de estos frameworks te permiten estructurar, crear y modular componentes reutilizables.

Es relevante echar un vistazo a la adopción en los navegadores de los web components para comprobar cuáles de los estándares aplicados por el W3C están siendo apoyados y soportados.

Por otra parte, los Design System están siendo una parte fundamental en el desarrollo web, tanto de aplicaciones, interacciones de usuario o de componentes.

Teniendo en cuenta todos estos conceptos, a continuación, se investigará cada uno de ellos.

2.1.1 Web Components Framework

Los Web Components son una tecnología dentro del desarrollo web en estado emergente que permite la creación de componentes personalizables y reutilizables para las aplicaciones. De forma general, un web component es un conjunto de elementos en lenguaje HTML, con lógica en lenguaje JavaScript y con estilos en lenguaje CSS que se pueden utilizar de manera independiente en cualquier página o aplicación web.

Los Web Components están formados por cuatro **estándares** propuestos por **W3C**: Custom Elements, Shadow DOM, HTML Templates, y ES Modules. Estos cuatro estándares proporcionan las pautas necesarias para la creación de web components.

- **Custom Elements** proporciona una API estándar que es la base de los componentes web. Permite crear o extender etiquetas HTML con comportamiento propio, que modifica o reemplaza al comportamiento original de las etiquetas HTML existentes. Esto genera una gran libertad para los desarrolladores a la hora de implementar nuevas funcionalidades a partir de etiquetas HTML ya existentes.

- **Shadow DOM** permite encapsular y aislar un árbol DOM de un elemento web, incluyendo toda la lógica JavaScript y sus estilos CSS. De esta manera se crean componentes autocontenidos con estilos y comportamientos propios con su árbol DOM aislado y conectado a elementos individuales.

- **HTML Templates** permite crear plantillas y facilita la reutilización de código HTML. Se suele usar para que los desarrolladores definan bloques de HTML para clonar y utilizar en cualquier parte de la página o aplicación.

- **ES Modules** permite la modularización de código JavaScript y la reutilización en partes de la misma aplicación o página. Estos módulos son partes de código independiente que se pueden exportar e importar cuando sea necesario.

Aunque los Web Components sean una tecnología emergente, ya existen varios frameworks que permiten su uso. Entre estos frameworks, algunos de los más conocidos son StencilJS, Polymer, LitElement y Bosonic. Cada uno de ellos proporciona una manera sencilla y eficiente para crear componentes web personalizados basándose en los estándares propuestos por la W3C.

- **Stencil** es un framework de código abierto que se centra en su eficiencia y en la sencillez de uso. Proporciona una API sencilla para crear componentes personalizados y un compilador propio que optimiza el rendimiento de los componentes.

- **Polymer** es uno de los primeros frameworks de Web Componentes. Proporciona una API sencilla para crear componentes personalizados. También proporciona gran cantidad de elementos predefinidos listos para usar.

- **LitElement** también es un framework orientado a Web Components que se orienta a la creación de componentes web personalizados, pero destacando en intentar reducir la cantidad de código usado lo máximo posible. Como los otros frameworks, proporciona una API sencilla, destacando una mayor cantidad de elementos predefinidos listos para usar que sus competidores.

Por otra parte, se ha podido encontrar más frameworks menos populares pero con enfoques y características diferentes como pueden ser SkateJS, Fast Element y Atómico. Cada uno de estos frameworks, al igual que los anteriores, permiten crear componentes web basados en los estándares propuestos por W3C.

Algunas de las cualidades más importantes de la tecnología de Web Componentes son la escalabilidad, la modularidad, la reutilización, la personalización, la interoperabilidad, la independencia y su estandarización.

Debido a que un mismo componente es usado en diferentes partes de una misma o distintas aplicaciones web, la escalabilidad de los componentes permite realizar cambios sobre el componente y que estos cambios se apliquen en todos esos lugares donde el componente se usa. Para aplicaciones en la que la cantidad de datos aumenta y se debe de estar en continua evolución, utilizar este tipo de tecnología facilita mucho el trabajo a lo largo del proyecto.

Hay que tener en cuenta que los cambios mencionados anteriormente no se podrían realizar si no fuera por otra de las cualidades de los Web Components, la modularidad, que permite al componente dividirse en partes independientes y reutilizables. Esto reduce el código de la aplicación web en gran medida, ya que no es necesario estar escribiendo de nuevo el código en cada parte que se utilice el componente, todo el código se encontrará en una misma ubicación, lo que resulta en reducción de costos de desarrollo debido a un código más fácil de mantener y mejora la productividad a la hora de crear nuevas aplicaciones web. También, debido al uso del mismo componente en distintas partes dentro de una aplicación web se fomenta la estandarización de las interfaces de usuario, creando cohesión a lo largo de las interacciones del usuario.

Otro factor muy importante de la modularidad es la mejora en la seguridad de las aplicaciones web, ya que, al aislar y proteger los componentes, se puede evitar amenazas y vulnerabilidades más fácilmente.

La gran personalización de los Web Components permite diseñar y desarrollar de manera modular, haciendo así que cada componente puede ser adaptado a distintas necesidades para cada contexto. Pudiendo así para un mismo componente disponer de distintas variantes con cambios ligeros o significativos dependiendo de las necesidades de la aplicación web.

La interoperabilidad permite a los Web Components ser usados en diferentes bibliotecas de JavaScript y en distintos frameworks, haciendo así que el desarrollador no tenga que perder tiempo adaptando un mismo componente a las distintas tecnologías cada vez que se quiera implementar en otro proyecto. Se mejora la eficiencia y permite un desarrollo mucho más limpio y sencillo de mantener.

Por último, los Web Components están basados en estándares web abiertos y ampliamente aceptados, permitiendo que sean compatibles con la gran mayoría de los navegadores y dispositivos.

Estas cualidades y beneficios hacen que los desarrolladores creen aplicaciones web más escalables, flexibles y fáciles de mantener, reduciendo costos de proyecto y aumentando la productividad y sostenibilidad.

Cabe recalcar que conforme pasa el tiempo, esta tecnología se va abriendo cada vez más paso en el ecosistema web creando cada vez más comunidad, aunque actualmente no puede competir a la cantidad de información y comunidades de otros framework y tecnologías web más populares.

2.1.2 Adopción de Web Components en los navegadores

Los Web Componentes son una tecnología relativamente nueva que ha ganado mucha popularidad estos últimos años.

La adopción por parte de navegadores ha sido un proceso lento y complicado, ya que, aunque se han hecho grandes avances, no ha sido hasta hace 2 años que se ha adoptado a la tecnología Web Components en todos los navegadores conjuntamente con sus estándares respaldados por W3C.

El desafío más grande actualmente es la falta de documentación que se puede encontrar acerca de cómo desarrollar Web Components. La mayoría de los ejemplos que se encuentran en artículos y foros de programación son o muy complejos o muy sencillos, por lo que la curva de aprendizaje para hacer uso de esta tecnología varía mucho dependiendo de los conocimientos previos en desarrollo web del programador. Aunque los estándares de Web Components son relativamente sencillos de entender, es complicado aprender a utilizarlos correctamente.

Por otra parte, si se hace uso de frameworks como los que se ha visto anteriormente (StencilJS, Polymer, LitElement), se reduce la curva inicial de aprendizaje, ya que no se tiene que empezar a desarrollar de cero y tienes unas herramientas para hacer un poco más sencillo el hecho de empezar en esta nueva tecnología. A parte de las herramientas, también existen las guías o APIS de cada uno de estos frameworks, aunque acaba pasando lo mismo que con documentación que se pueden encontrar en foros o artículos, las APIS tienen una curva exponencial de aprendizaje por lo que se hace sencillo entenderlas al principio, pero se vuelve complicado encontrar información para casos concretos. Conforme la adopción en las comunidades de desarrolladores aumente, a su vez, aumentarán la cantidad de ejemplos disponibles para los diferentes casos en los que vendría bien tener experiencias de otros desarrolladores para hacer más ameno el trayecto en esta nueva vertiente que son los Web Components.

El estado actual de adopción en los navegadores es todo el conjunto de estándares respaldados por W3C en la tecnología Web Components. A lo largo de los últimos años se añadieron y retiraron estándares para llegar a un acuerdo conjunto con los grandes navegadores. Una de las últimas adopciones de tecnología Web Component en los navegadores fue la especificación Template Instantiation, que permite la creación de instancias múltiples de un componente sin tener que renderizar de nuevo todo el árbol de componentes. Esto permite mejorar el rendimiento en las aplicaciones web y poder crear así componentes personalizados más complejos sin comprometer el tiempo de carga de la aplicación.

Debido al auge en popularidad de esta tecnología, ya se han desarrollado y se están desarrollando nuevas herramientas y recursos como pueden ser plantillas de componentes con características predefinidas, código y bibliotecas ya predefinidas y listas para uso. Cada vez más se facilita la creación de componentes de alta calidad a medida que la comunidad aumenta.

En la actualidad, es complicado encontrar distintas librerías de componentes predefinidos, siempre acabas encontrando las más conocidas y a no ser que emplees mucho tiempo buscando en foros es difícil ver algo distinto a lo más popular.

Cabe recalcar que grandes conglomerados dentro del desarrollo web están trabajando ya en sus propias librerías con tecnología Web Components, entre ellas la más conocida es Bootstrap.

En conclusión, la adopción a lo largo de los años de la tecnología Web Components ha sido lenta y complicada pero actualmente se han adoptado los estándares W3C en todos los navegadores. A medida que se vayan haciendo cambios en esta tecnología, se deberán actualizar los estándares y se entrará en otro proceso de negociación con los grandes navegadores. Este hecho no preocupa a las comunidades del ecosistema, ya que esto mismo sucede con todas las

tecnologías web. Conforme vaya aumentando la popularidad de la tecnología, la cantidad de ejemplos, discusiones y experiencias a cerca del desarrollo de componentes y sus casos concretos aumentarán y cada vez será más sencillo aprender e iniciarse con los componentes web sin la necesidad de ser un desarrollador web intermedio o avanzado.

2.1.3 Design System

La organización a la hora de lanzar un producto es clave, se debe de definir, diseñar y desarrollar a base de unas normas que tiene que marcar el equipo en cuestión. A partir de esto suelen surgir problemas:

- **Exceso de líneas en archivos CSS** donde hay diferentes estilos personalizados para cada componente.
- Se encuentran **componentes duplicados** y esto supone distintos estilos para cada uno de ellos que, pese a ser parecidos no acaban siendo iguales, ya sean botones, tipografía, cabeceras, etc.
- Se hace complejo realizar un **cambio en el estilo del producto**, ya que cada elemento web tiene su propio estilo y esto provoca que en aplicaciones con más carga se convierta en un proceso largo de llevar a cabo, lo que conlleva en un gasto excesivo y aumenta el tiempo en el que los miembros del equipo estén ocupados.
- En ocasiones, los **diseños no** están **disponibles** a la velocidad que el desarrollador los necesita y se avanza sin el diseño, creando así inconsistencias en las vistas.

Estos problemas implican que se produzca inconsistencia en el producto, que se vuelva complicada la escalabilidad de este y que aumente el tiempo de desarrollo de los productos. Para solucionar esto se necesitan normas y estas vienen dadas por un Design system.

Dentro del desarrollo web, un Design System establece un conjunto de patrones de diseño, directrices y reglas de manera coherente que permite poder reutilizarlo en el mismo proyecto o proyectos para desarrollar aplicaciones web consistentes.

El objetivo de estas directrices es crear una mejor experiencia de usuario durante toda las páginas de la aplicación web, evitando así cambios en estilo que puedan parecer menores pero que inducen a errores en las interacciones de los usuarios.

Toda esta vertiente empezó en las empresas de diseño de software para mantener la coherencia visual mejorando así la experiencia del usuario en todos los productos. Actualmente, los Design System son una herramienta muy popular en el desarrollo web y prácticamente

indispensable. Es especialmente necesario para empresas de gran tamaño ya que facilita la creación y manutención de los sitios web.

Hay dos tipos de Design System, está el **Design System privado**, que se utiliza en una organización determinada para sus propios productos o servicios y luego existe el Design System público, que permite compartir la información con la comunidad para utilizarlo en distintos proyectos. También existen otros tipos de **Design System** que son los **comerciales**, estos son vendidos por empresas que se dedican a la creación de estos sistemas.

Normalmente en los Design System se incluyen diseños de páginas divididos en componentes, como pueden ser botones, tarjetas, separadores, logos y muchos más. Por otra parte, se suelen definir tipografías, paletas de colores, espacio entre letras, espacio entre líneas, curvatura de las esquinas de los elementos de la web y muchas más cualidades que se verán representadas mediante tokens de diseño con un lenguaje definido que suele ser CSS (generalmente css properties). Los tokens de diseño permiten definir todas las características relacionadas a estilos, tipografía, iconos, colores, bordes, espacios de componentes, etc. Se pueden crear características y utilizarlas para crear un estilo para secciones o componentes e incluso para el sitio web completo. Son independientes del lenguaje y se pueden utilizar en cualquier tipo de proyecto por lo que pueden usarse también en distintas plataformas. Se debe determinar una nomenclatura a la hora de nombrar las características que formarán los tokens de diseño.

La implementación de un Design System suele ser un proceso complicado, se debe de hacer una planificación detenida y debe haber también una comunicación fluida en el desarrollo de este entre los diseñadores y los desarrolladores. De esta manera se puede conocer más profundamente los requisitos y necesidades específicos para ese proyecto y si se diera el caso para otros proyectos.

Es importante que al definir las directrices de componentes se tenga en cuenta, dentro de lo posible, la accesibilidad para todos los usuarios, incluidos los usuarios con discapacidades. Para ello se pueden seguir las sugerencias de accesibilidad web WCAG (Web Content Accessibility Guidelines). De esta forma se incluye a un mayor número de usuarios permitiendo que los componentes se utilicen de una forma sencilla, ya se tengan discapacidades auditivas, visuales o físicas (este paso suele realizarse parcialmente, ya que adaptarlo para todo el público puede suponer un costo más elevado del proyecto debido a emplear más tiempo tanto en el diseño como en el desarrollo de los componentes).

Se han desarrollado herramientas para hacer posible la creación de estos sistemas facilitando su proceso. Algunas de estas herramientas que ayudan a definir estos estilos y crear

componentes visuales de forma más rápida y eficiente son **Figma, Sketch, AdobeXD** y **Zepelin** entre otras muchas.

- **Figma** es la herramienta más popular dentro de las usadas en el desarrollo de Design System. Consiste en una herramienta de diseño online que también dispone de aplicación de escritorio enfocada a crear interfaces de usuario, prototipos web y las colaboraciones entre diseñadores.

  La colaboración en tiempo real junto con la opción de trabajar en distintos proyectos a la vez ha aumentado mucho su popularidad en el ecosistema web. Figma ofrece gran cantidad de características orientadas a crear componentes reutilizables, también ofrece bibliotecas compartidas y componentes maestros, teniendo una integración con servicios de terceros como pueden ser Jira, Trello, Slack, Github y muchas más tecnologías que permiten controlar el flujo de trabajo y tener una mejor planificación sobre tu proyecto.

- **Sketch** es una herramienta de diseño gráfico vectorial para dispositivos Mac que permite crear diseños de interfaces de usuario. Debido a su gran cantidad de plugins y bibliotecas y como facilitan estas a facilitar la experiencia de una creación más rápida de componentes y de estilos, se ha convertido en una herramienta popular dentro del sector de diseñadores.

- **Adobe XD** es una herramienta de diseño de interfaces de usuario que igual que las competidoras nombradas en este estado del arte permite crear diseños y prototipos interactivos. Ha sido diseñado específicamente para crear Design Systems y ofrece opciones para colaborar online, esto proporciona a los diseñadores crear sistemas escalables y consistentes. También incluye una biblioteca de componentes para reutilizarla en distintos proyectos.

- **Zeplin** es una herramienta de diseño enfocada a Design Systems. Una de las cualidades más importantes de esta herramienta es que los diseñadores pueden exportar diseños desde Sketch, Adobe XD y Figma, entre otras herramientas, para generar pautas de estilos y obtener recursos de diseño para desarrolladores.

Zeplin es el principal competidor junto con Figma a nivel de popularidad en la actualidad, ya que, al igual que Figma, tienen una comunidad muy grande y características que permiten que los desarrolladores y diseñadores reduzcan errores en comunicación siendo así más eficientes.

Aparte de este tipo de herramientas, también existen marcos de trabajo que proporcionan sus Design System al público con directrices para la escritura de código,

accesibilidad y UX (Experiencia de usuario). Además, disponen de un amplio abanico de componentes y estilos. Algunos de los más conocidos son Material Design de Google y Bootstrap de Twitter.

A la hora de crear un Design System se puede seguir diferentes metodologías, las dos metodologías más conocidas son Atomic Design y Design System First.

La primera opción, **Atomic Design** es una metodología de diseño que se centra en sistemas escalables y sostenibles. Parte de la idea de que todo diseño está formado por componentes más pequeños (átomos y moléculas), combinándolos en componentes grandes (organismos y plantillas). Se puede definir el proceso en cinco etapas:

- Las **páginas finales** que formarán el diseño final.
- Las **plantillas** que son estructura de diseño para las páginas. Combinan organismos y componentes grandes.
- Los **organismos** que consisten en combinaciones de moléculas que juntos forman componentes grandes como headers y footers.
- Las **moléculas** que consisten en combinaciones de átomos para formar elementos complejos, en este caso botones, tarjetas o spinners entre otros.
- Los **átomos** que consisten en elementos pequeños y básicos como campos de texto, iconos…

Seguir esta metodología permite crear un sistema de diseño escalable, ya que debido a la segmentación de elementos de más grandes a más pequeños y viceversa, adaptarse a los proyectos y necesidades se vuelve más sencillo.

A la hora de aplicar esta tecnología, siempre se recomienda empezar con la etapa de átomos e ir aumentando en tamaño las etapas siguiendo moléculas, organismos, plantillas y acabar en páginas finales, pero se acaba utilizando una mezcla de etapas dependiendo de los diseñadores y su forma de visualizar los componentes, por ejemplo, un diseñador puede empezar con la forma superficial de un componente tarjeta que correspondería a la etapa de molécula y luego a partir de una visión general definir los elementos del interior del componente que correspondería a los átomos del interior de la molécula.

La segunda opción, **Design System First** es una metodología de diseño que se centra en crear un Design System antes de empezar con un proyecto en específico. Establece una base sólida de diseño sistemático y modular, no se empieza diseñando componentes para crear un Design System a partir de ellos. Se puede definir el proceso en tres etapas:

- Se crea el sistema de diseño empezando por definir los componentes y directrices de diseño para todos los componentes.
- Se integra el sistema de diseño a todos los flujos de trabajo, es decir se utilizan las herramientas correspondientes para integrar el diseño para poder trabajar de una forma colaborativa y más efectiva.
- Se crean los proyectos específicos partiendo de los componentes y directrices definidos en el diseño creado e integrado en un flujo de trabajo.

La gran ventaja de seguir esta metodología es la capacidad de empezar un proyecto teniendo definido todo lo necesario para el desarrollo de tus proyectos. La gran desventaja es la pérdida de posibilidades de modificaciones y escalabilidad, se requiere de mucho más tiempo de planificación antes de empezar un proyecto.

Las dos metodologías tienen sus ventajas y desventajas aunque para las comunidades de desarrolladores y diseñadores la metodología más popular es la de Atomic Design, ya que el caso de que todos los integrantes de un equipo incluyendo el cliente sepa desde un inicio cómo se va a realizar cada parte del proyecto de una forma detallada no suele darse. Lo más típico en un proyecto es que sucedan cambios durante su desarrollo y se deban de hacer cambios en el Design System, para ello un modelo escalable es la opción más sostenible.

Por último, cabe destacar el movimiento tecnológico que está teniendo la inteligencia artificial en el sector del diseño. Muchas herramientas entre ellas Figma, que es en la que basaré este ejemplo, están incorporando inteligencias artificiales para fines productivos y de mejoras de rendimiento.

Se está utilizando para el diseño generativo que consiste en que los diseñadores puedan generar múltiples diseños a partir de unas líneas de código o conjuntos de parámetros. De esta forma puedes obtener ideas nuevas y adaptarlas a tus propios diseños.

En cuanto al análisis de datos, Figma utiliza una IA para analizar grandes cantidades de datos de usuario y de diseño, conociendo cada vez mejor las necesidades de usuario y dando mejores sugerencias durante el desarrollo de los diseños.

También se pueden vincular inteligencias externas mediante plugins como podrías ser Midjourny que es una IA centrada en arte y generación de imágenes a partir de comandos. Esta inteligencia artificial se está haciendo muy popular y cada vez es mejor en las imágenes generadas, llegando a ser difícil diferenciar si algunas imágenes han sido creadas por un diseñador o por una máquina.

Cabe destacar, que este movimiento tecnológico que son las inteligencias artificiales enfocadas a herramientas de diseño está en continuo desarrollo y todavía se encuentra en una fase temprana. Pese a ser una tecnología con mucho potencial, todavía no es posible hacer uso de ella de una manera en la que pueda sustituir a los diseñadores, se prevé que las siguientes actualizaciones de estas tecnologías permitirán aumentar la productividad en el sector del diseño web, por lo que se requerirá de menos personal para hacer trabajos que ahora necesitan más personal.

En resumen, un Design System está en continua evolución, es tan amplio y complejo como el proyecto lo requiera. Han evolucionado durante los años hasta convertirse en una parte inseparable del desarrollo web permitiendo así mejorar la consistencia de las aplicaciones web y las experiencias de los usuarios a lo largo de estas.

Existen muchas herramientas y marcos de trabajo que se utilizan actualmente para mejorar la eficiencia y productividad en la creación de los Design System. Implementarlo en un proyecto tecnológico presenta numerosos beneficios tanto para los diseñadores, para los desarrolladores y como para el proyecto final, por esta razón se recomendará siempre hacer uso de ellos en cualquier proyecto, ya sea con un Design System complejo o con uno más sencillo, ya que es importante mantenerlo actualizado, no tiene que ser una solución estática, se suelen hacer modificaciones en el a lo largo de un mismo proyecto, pudiendo así ir añadiendo directrices cuando sea necesario.

2.2 Estudio de propuestas

Después de analizar esta suite tecnológica surgieron 4 propuestas a la hora de implementar componentes web:

- Creación de Web Components independientes.
- Usar Web Components FrameWork.
- Extender/crear Web Component para un framework existente.
- Implementar Web Component Framework.

(Antes de analizar las propuestas, recalcar que este apartado ha surgido de la duda en cuanto a qué tecnología utilizar para el desarrollo de los Web Components. El suite completo de tecnología y herramientas que se utilizaran se encuentra en el punto 3.2 Tecnologías utilizadas).

2.2.1 Creación de Web Components independientes

La propuesta de creación de Web Components independientes permite tener una gran flexibilidad, reutilización, a su vez permite utilizar los estándares respaldados por la W3C y facilidad para mantener los componentes web una vez construidos, ya que, al ser independientes, podrás actualizarlos individualmente y no se requiere una actualización global de librería, se puede ir actualizando cada uno de ellos gradualmente.

Esta propuesta implica la creación de componentes web personalizados con su posible utilización en distintas aplicaciones y tecnologías web. Se construyen utilizando lenguajes HTML, CSS y JavaScript, pudiéndose utilizar sin necesitar ningún marco de trabajo en especial. Hay que recalcar que estos componentes permiten más control en aspecto y funcionalidad de este.

Para crear Web Components independientes se deben definir utilizando los lenguajes HTML, CSS y pese a no ser necesario en algunos componentes, se utiliza JavaScript cuando es necesario. Este último lenguaje suele usarse para temas interactivos, acciones y funcionalidad para los componentes.

Una vez se define el Web Component, hay que encapsularlo dentro de una clase JavaScript que debe extender de la clase base de Web Components (HTMLElement) y utilizar los ciclos de vida de los componentes connectedCallback() para poder inicializar el componente al insertarse en el DOM. Cuando se necesite retirar del DOM el componente, se utilizará el método disconnectedCallback(), limpiando así cualquier recurso relacionado.

Por último, hay que terminar de definir el componente registrándolo con el método define() también perteneciente a la clase personalizada Web Component. Así se define el nombre del componente registrándolo, siempre con una nomenclatura que incluye al menos un guión separando dos palabras, por ejemplo “plz-button”.

Habiendo acabado el proceso, este componente se puede utilizar en cualquier aplicación web. Para poder utilizar el componente, se debe de importar la clase JavaScript de este y utilizarla en el fichero HTML donde se desee hacer uso del componente.

En resumen, este proceso permite crear componentes independientes que, aunque puede suponer invertir tiempo y recursos en su creación, se ven superados significativamente por los beneficios que aportan en cuanto a reutilización y flexibilidad a largo plazo. Para cualquier proyecto en el que se busque desarrollar aplicaciones web escalables y fáciles de mantener permitiendo actualizar estos componentes de manera independiente, esta es una muy buena elección.

2.2.2 Usar Web Components Framework

En esta propuesta se sugiere el uso de un framework de Web Components, cómo podría ser StencilJS o Polymer, dos frameworks mencionados anteriormente entre otros en la sección del estado del arte acerca Web Components Framework. Esta opción parte de una base sólida que aporta herramientas para crear componentes personalizados que mejoran la eficiencia en el desarrollo.

Una ventaja de usar Web Component Framework es la capacidad para crear componentes personalizados reduciendo el tiempo de desarrollo, esto permite avanzar más en los proyectos cuando el tiempo es limitado. También proporcionan una estructura que hace más sencillo reutilizar los componentes, que sean escalables y mejora la eficiencia a la hora de mantenerlos actualizados en el tiempo.

Otra gran ventaja del uso de este tipo de frameworks es la opción de aprovechar todas las funcionalidades y características que aportan. En el caso de Polymer, existe gran cantidad de elementos preconstruidos, botones, menús y muchos más, a parte de haber gran variedad de funcionalidades comunes.

El uso de estos framework también permite mayor compatibilidad con los navegadores, ya que es una de las características en las que se ha puesto más empeño y todos los framework Web Component siguen los estándares de W3C permitiendo así aumentar la compatibilidad con la gran mayoría de dispositivos y navegadores.

Cabe recalcar que la creación de componentes personalizados hechos en este tipo de frameworks están limitados a las funcionalidades y características que ofrecen. En algunos casos no se puede alcanzar el nivel de personalización que se podría alcanzar creando web components desde cero.

Por último, se puede llegar a la conclusión de que el uso de un framework de Web Components es una gran elección para reducir la curva de aprendizaje para nuevos desarrollos, debido a la estructura con base sólida que incluyen y a las funcionalidades y características que permiten un desarrollo más eficiente en escalabilidad, reutilización y sostenibilidad. También permiten una gran compatibilidad con navegadores y dispositivos.

Por otra parte, dependiendo del tipo de componente que quieras crear, esta puede no ser la mejor opción debido a las limitaciones de las herramientas del framework. Este es un aspecto que hay que tener en cuenta antes de elegir crear componentes de esta manera.

2.2.3 Extender/crear Web Component para un framework existente

La propuesta de extender/crear Web Component para un framework existente viene de buscar la eficiencia utilizando un framework preestablecido como podría ser Bootstrap y poder personalizarlo modificando las funcionalidades ya existentes o creando nuevas haciendo uso de Web Components.

Al extender un Web Component de un framework se puede tomar uno existente y agregar funcionalidades personalizadas y nuevas características. Por otra parte, en el caso de querer crear un componente de cero, se haría utilizando las características y opciones predefinidas por el framework.

Poniendo como ejemplo Bootstrap, al utilizar este framework se debería aprovechar su arquitectura y diseño, también sus estilos y componentes preestablecidos. Bootstrap es muy conocido por sus estilos CSS principalmente y suele utilizarse por su variedad de elementos o componentes para construir interfaces que permiten usar una misma aplicación en distintos dispositivos con distintas resoluciones de pantalla (responsive).

Con esta propuesta se seguiría manteniendo las ventajas de utilizar Web Components cómo podría ser la reutilización de código pudiendo aprovechar la base de código de estilos y componentes que proporciona el framework, ahorrando mucho tiempo en desarrollo. También se mantendría la consistencia visual entre los componentes debida a hacer uso de los estilos proporcionados por el framework basados en un Design System propio.

Uno de los puntos fuertes de utilizar un framework existente para ampliar sus componentes o crear nuevos es la gran comunidad de desarrolladores que hay detrás. Puedes consultar cualquier problema en foros pudiendo así encontrar información acerca de muchas de las soluciones que puedes estar buscando. A su vez, puedes disponer de herramientas, complementos y extensiones para aumentar todavía más tu productividad.

Por último, se mantiene la modularidad y escalabilidad pudiendo cambiar tu la arquitectura del componente para ello haciéndolos independientes para no afectar a la aplicación.

Concluyendo, la propuesta de extender o crear Web Component para un framework como podría ser Bootstrap permite poder aprovechar la estructura y estilos del framework y a su vez permite personalizar y añadir componentes y funcionalidades. Por otra parte, conservas las ventajas de los Web Component cómo podría ser la reutilización de código, la cohesión entre los componentes, la modularidad y la escalabilidad de las aplicaciones web donde se usen. Todo esto permite mejorar la eficiencia y productividad de trabajo, siempre y cuando el enfoque de tu aplicación se alinee con los estilos y herramientas del framework elegido.

2.2.4 Implementar Web Component Framework

En esta propuesta se plantea la implementación de Web Component Framework haciendo uso de Angular y React entre otros como framework. Estos framework proporcionan funcionalidades y características extra que permiten mantener grandes proyectos más fácilmente.

Ofrecen una estructura de desarrollo predefinida por lo que se facilita mucho la creación de Web Components. La gran mayoría de interacciones entre distintos elementos HTML ya están pensados en estos framework y existen funcionalidades muy interesantes que permiten ahorrar una gran cantidad de tiempo en desarrollo frente a construir componentes sin framework. Esto permite acelerar el proceso de desarrollo, más productividad y consistencia entre los componentes.

Sin embargo, una de las mayores desventajas de usar este tipo de framework para la creación de Web Components es la curva de aprendizaje del propio framework, sus directrices, sintaxis, etc. También hay dependencias de tecnologías y herramientas específicas ofrecidas por el framework, lo que limita la flexibilidad a largo plazo.

Por otra parte, al crear componentes con un framework como React, se perdería una de las características que hacen a los Web Component ser altamente atractivos para proyectos como es la interoperabilidad, que permite a los Web Components ser usados en diferentes bibliotecas de JavaScript y en distintos frameworks, haciendo así que el desarrollador no tenga que perder tiempo adaptando un mismo componente a las distintas tecnologías cada vez que se quiera implementar en otro proyecto. Por lo tanto habrá que ser muy consciente de cuál es el marco de trabajo en el que se van a usar estos componentes, ya que en el momento en que se quieran utilizar con otras tecnologías, habrá que reconstruirlos y adaptarlos a esa tecnología en concreto, no te valdrá el trabajo realizado con este framework.

2.3 Justificación

En primer lugar, se descartan dos de las propuestas:

- Extender/crear Web Component para un framework existente.
- Implementar Web Component framework.

Debido a querer implementar en el proyecto un Design System para aplicar los estilos propios y guías a los componentes del proyecto, el hecho de usar la propuesta de extender/crear Web Component para un framework existente nos limita la creación de los mismos. Pese a que este framework (como podría ser Bootstrap) no obstaculiza el desarrollo de los componentes

personalizados, no se estaría haciendo uso de la mayoría de estructura y estilos del framework y por tanto pierde el significado de esta elección.

Por otra parte, la propuesta de implementar Web Component Framework ha sido descartada debido al factor de interoperabilidad. En este proyecto se pretende que los Web Components creados puedan ser usados en diferentes librerías y distintos frameworks, ya sea de forma individual o en conjunto.

Tanto la propuesta de creación de Web Component independientes como la propuesta de usar Web Component Framework se alinean con el objetivo del proyecto y se convierten en las propuestas más atractivas.

En la propuesta de Web Component independientes cabe destacar que la curva de aprendizaje es mayor, ya que no dispones de ningún tipo de ayuda o sugerencia de desarrollo, únicamente seguir los estándares W3C básicos y construir el componente de cero. Sin embargo, la posibilidad de tener los Web Components independientes y que puedan ser usados individualmente es muy interesante, junto con la posibilidad de que estos componentes tengan su propio estilo y sean completamente personalizables.

Por último, se encuentra la propuesta de usar Web Component Framework que nos permite reducir un poco la curva de aprendizaje teniendo una estructura básica y alguna herramienta proporcionada por el framework que nos permita tener un mejor inicio debido a la falta de conocimiento en el desarrollo web. También permitiéndonos crear componentes y funcionalidades personalizables con el estilo propio de diseño del proyecto.

2.3.1 Propuesta final

La propuesta final será hacer una combinación de dos propuestas:

- Uso de Web Component Framework.
- Creación de Web Components independientes.

Esta elección se debe al tipo de framework que se ha elegido para realizar el proyecto. Se hará uso de StencilJS, ya que permite crear Web Components nativos siguiendo los estándares W3C pudiendo funcionar así en cualquier navegador actual. Algunas de las razones del porqué de esta elección son:

Cualidades de frameworks populares: StencilJS se ha inspirado en React para su virtual DOM, que permite un renderizado asíncrono. También usa como lenguaje Typescript, JSX y CSS para la creación de componentes añadiendo el extra de decoradores de componente inspirado en Angular.

Compatibilidad: Pese a ser Web Components creados con un framework, son compatibles con todo tipo de aplicaciones web, ya sea con tecnología o frameworks distintos o con vanilla JS.

Ligero, rápido y relativamente sencillo de utilizar: los componentes creados por este framework se ejecutan rápido y existen opciones de prerenderizado junto con optimizaciones del árbol de dependencias de los componentes para mejorar el rendimiento de las aplicaciones (todo esto sin que el desarrollador tenga que meter mano). También es relativamente sencillo empezar haciendo un componente sencillo debido a la documentación y estructura de StencilJS.

Por último, este framework te permite preparar **distintas distribuciones** para la consumición de sus componentes, por lo que te da la libertad de exponerlos en forma de librería autocontenida o si así lo deseas como componentes individuales, alineándose así con la otra propuesta de creación de Web Components independientes.

3 Estructura, gestión de versiones y distribución

3.1 Arquitectura de la aplicación

En cuanto a la arquitectura de la aplicación, se dispondrá de un enfoque cliente-servidor. Los datos se mantendrán conectados entre el cliente y el servidor utilizando StencilJS como framework para el desarrollo del proyecto. Este framework proporciona herramientas de utilidad para la creación de componentes reutilizables, pudiendo crear así mismo una estructura monorepo donde se ubica tanto la librería de componentes como la aplicación web donde se mostrarán y documentarán los componentes.

El cliente, en este caso el navegador web, será el responsable de mostrar al usuario final la interfaz de usuario para que pueda haber interacción. El cliente solicitará recursos al servidor que procesará la lógica del lado de Cliente haciendo uso de HTML, CSS y JavaScript interpretando y renderizando los componentes creados con StencilJS.

El servidor, que en este caso es Node.js, será el responsable de crear un entorno en tiempo de ejecución para JavaScript en el lado del servidor y tareas de compilación de aplicación. Recibirá las solicitudes del cliente y procesará la lógica respondiendo al cliente con los recursos solicitados (archivos HTML, CSS, JavaScript y datos).

La comunicación entre cliente y servidor se realizará mediante protocolos de red HTTP.

3.2 Tecnologías utilizadas

Para el apartado de tecnologías utilizadas, se dividirá entre tecnologías de diseño y tecnologías de desarrollo.

Por lo que repercute al diseño, se ha centrado en utilizar una herramienta llamada **Figma** basada en diseño de interfaces de usuario. Esta herramienta permite crear prototipos de las páginas web del proyecto y desarrollar componentes definiendo su estructura y apariencia visual o estilo.

Por otra parte, las tecnologías de desarrollo que se usarán en este proyecto son:

StencilJS, un framework que se usará para implementar los diseños creados en Figma de los componentes de la biblioteca. Permite crear componentes web reutilizables y seguir los estándares web W3C.

Para desarrollar los componentes en StencilJS, se han utilizado principalmente 5 lenguajes:

- **HTML:** Los Web Components basan su estructura en este lenguaje y se utiliza para definir su contenido. Es necesario conocer más allá de las nociones básicas para implementar los diseños creados en Figma haciendo uso de StencilJS.

- **CSS:** Es el lenguaje que permite definir la apariencia visual de los Web Component mediante estilos. En este proyecto, se hace uso de un CSS puro sin utilizar CSS preprocesados como podrían ser Sass o Less. Para poder aplicar correctamente los estilos aplicados en el diseño en Figma se debe tener conocimiento de este lenguaje ya que muchos estilos se superponen unos con otros a la hora de definir cada uno de los elementos que forman un componente.

- **TypeScript:** Es un superconjunto de JavaScript que agrega características de tipado estático al lenguaje y en este caso es utilizado para la construcción de la lógica de componentes en StencilJS. Para hacer uso del lenguaje se debe tener conocimientos básicos en JavaScript y conocimientos en lenguajes tipados. Para adquirir los conocimientos necesarios de sintaxis de este lenguaje se puede acceder a la documentación proporcionada por TypeScript.

- **JSX:** Es una extensión de sintaxis para poder escribir código HTML dentro de archivos de tipo JavaScript o Typescript para facilitar la creación de componentes con StencilJS. Tanto la creación como manipulación del DOM se vuelven más sencillas al utilizar sintaxis HTML dentro de los componentes.

- **Markdown:** Es un lenguaje de marcado ligero que se utiliza para dar formato a documentos de texto plano. Principalmente se encuentra en archivos README en los repositorios con código fuente con contenido acerca del uso de ese código, funcionalidades, descripciones, etc.
    - Se utilizará este lenguaje para construir los archivos README de cada componente junto con un README global de la aplicación web de muestra y documentación de componentes.

El **editor de texto** utilizado para desarrollar junto con el framework de StencilJS y los lenguajes mencionados anteriormente ha sido **Visual Studio Code**, ya que permite navegar entre los distintos archivos de una forma sencilla. A su vez, se pueden añadir extensiones como Prettier para el formateo del código, mejorando así la apariencia y organización de este y otras extensiones que permiten localizar errores de código mediante notificaciones. También dispone de un terminal con el que usar los comandos necesarios para construir el proyecto y ejecutarlo entre otras opciones.

La elección de este editor de texto ha sido debido a ser uno de los más completos y utilizados actualmente junto con una experiencia previa de uso del mismo.

Las tecnologías usadas para la gestión de repositorio y control de versiones en este proyecto son GitHub y SourceTree.

GitHub es una plataforma que permite alojar repositorios Git basada en la nube y es una de las opciones más populares para el desarrollo de software. Permite que tu código sea accesible por todos a través de tu perfil y puedes clonar proyectos mediante un comando de consola haciendo uso de Git. También tiene una interfaz de usuario intuitiva y dispone de una comunidad enorme de desarrolladores.

Por otra parte, se usará de **SourceTree** vinculado al proyecto ubicado en el repositorio de GitHub. Esta herramienta facilita el uso de Git permitiendo controlar la gran mayoría de los comandos Git vía una interfaz intuitiva, pudiendo realizar acciones como crear y fusionar ramas, confirmar cambios, resolver conflictos, explorar historial de versiones, etc.

Al tener esta herramienta vinculada al proyecto en GitHub, cada cambio que se haga desde SourceTree se verá reflejado en la nube. Se ha elegido SourceTree debido a haberlo utilizado anteriormente en otro proyecto y ya conocer su funcionamiento y ser una herramienta popular, pero se podía haber utilizado cualquier herramienta con una interfaz de usuario para controlar Git o el propio Git vía consola de comandos.

Por último, se ha empleado **NPM** (Node Package Manager) que es un administrador de paquetes para el entorno de ejecución explicado en el apartado 3.1 Arquitectura de la aplicación. Es una herramienta que permite el desarrollo de aplicaciones web y de servidor en un ecosistema de Node.js y se utiliza para instalar, administrar y compartir paquetes de código reutilizable en lenguaje JavaScript.

Node.js será utilizado como herramienta para la compilación de componentes, gestión de dependencias, inicio de un servidor de desarrollo donde se verá la página web de muestra y documentación de componentes y también para la generación de una versión optimizada para producción. Todo esto a través del administrador de paquetes NPM.

3.3 Control de versiones

Antes de empezar el proyecto, en el alta del trabajo de final de grado, se explicaba que este proyecto iba a seguir un desarrollo incremental, ya que la tecnología de Web Components es relativamente novedosa y se empezaría con el desarrollo de componentes sencillos y posteriormente se irían refinando.

Para ello se dispone de un repositorio monorepo en la nube haciendo uso de Github, haciendo uso de Git para cada tarea con la herramienta SourceTree y pudiendo ver la aplicación web del proyecto desplegada en Netlify accesible para todo el mundo.

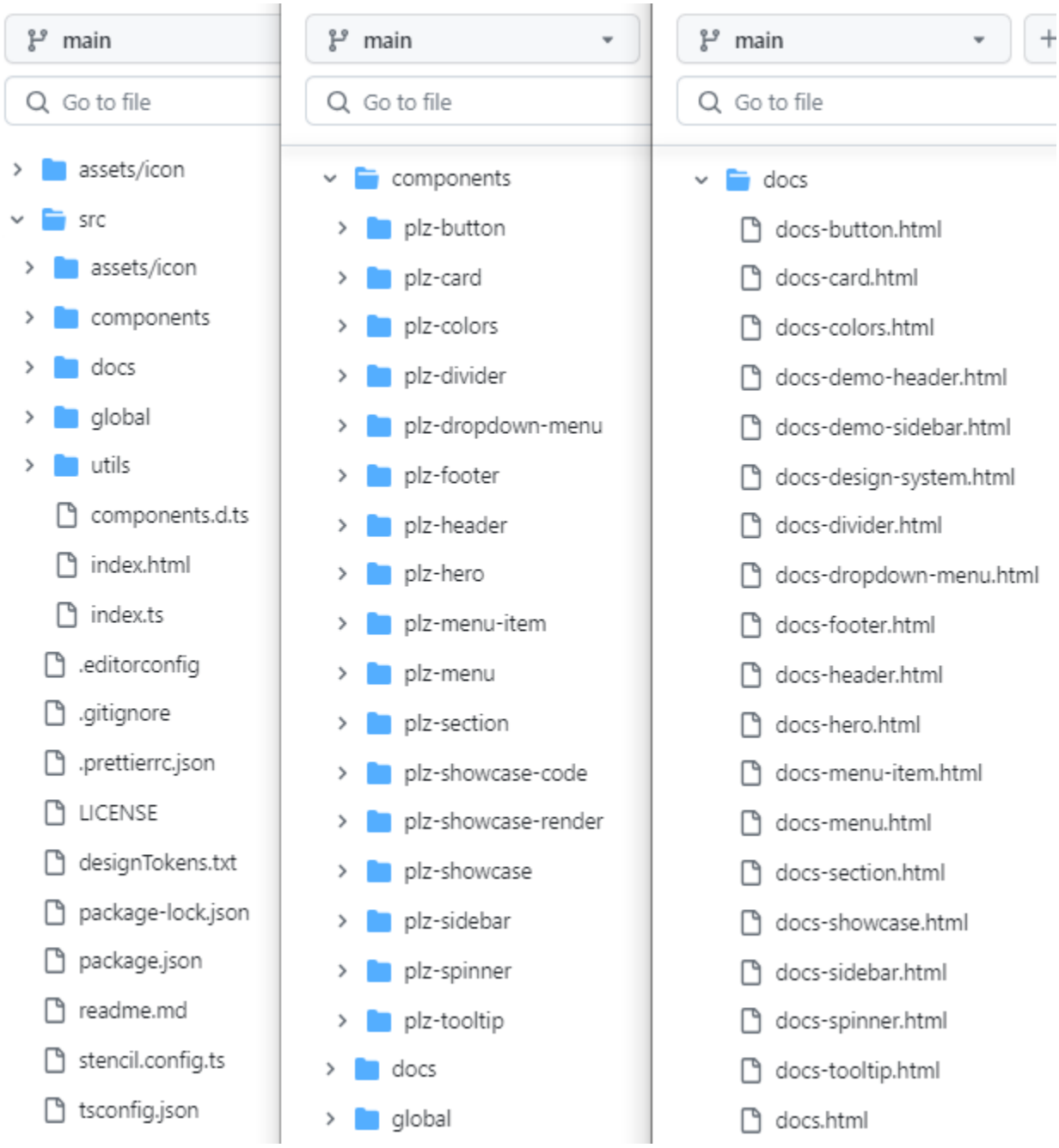
A continuación, se explicará cada uno de estos puntos.

3.3.1 Repositorio

En cuanto al repositorio, se ha visto conveniente utilizar un proyecto monorepo. Este término se refiere a una estructura de gestión de código en el que todos los componentes y proyectos se almacenan en un mismo repositorio. En este caso, se tendría en el repositorio la librería de componentes y la aplicación web donde se muestran y explican los componentes y su uso con ejemplos de código y visuales en un mismo repositorio.

La estructura del proyecto quedaría tal y como se muestra en las imágenes a continuación:

1-Estructura Monorepo



Se ha considerado utilizar un monorepo principalmente por la ventaja del mantenimiento unificado. Teniendo tanto la librería de componentes, como la aplicación de documentación en un mismo repositorio se puede actualizar tanto los componentes como la documentación a la vez que se realiza una modificación, sin tener la necesidad de cambiar de repositorio Git para realizar el cambio. De esta manera se evita que cualquier modificación en los componentes pueda no quedar reflejada en la aplicación web de documentación.

También se simplifica el flujo de desarrollo, al no tener que cambiar configuraciones del entorno cuando se tenga que trabajar en distintas secciones del proyecto. Esta peculiaridad permite aumentar la eficiencia y reducir la aparición de errores.

3.3.2 Github y SourceTree

GitHub es una plataforma que permite alojar código fuente de tus proyectos en línea. También permite la colaboración basada en Git, donde puedes administrar los proyectos y permite opciones de seguimiento de problemas y más herramientas de desarrollo colaborativo.

Los repositorios en GitHub son repositorios Git y por lo tanto, puedes aplicar todos los comandos y operaciones de Git estándar.

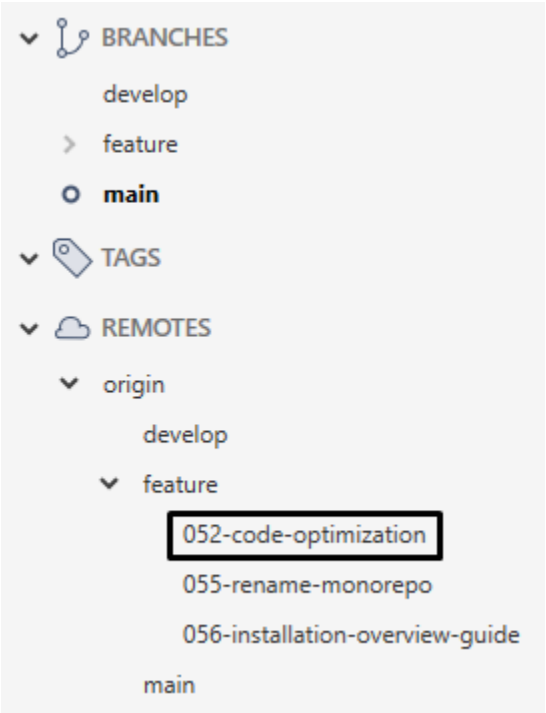
Por otra parte, se dará uso a SourceTree que es una herramienta gratuita que proporciona una interfaz gráfica de usuario orientada a Git. Con esta herramienta se puede realizar las acciones que se haría vía comando de una forma rápida como podría ser creación de ramas, fusión de ramas, gestión de conflictos y muchas más mediante clicks.

Se decidió utilizar esta combinación de tecnologías debido a la facilidad de la gestión de ramas y control de versiones que proporciona clonar el repositorio GitHub en local y utilizar una interfaz gráfica de usuario, en este caso SourceTree, vinculada al repositorio en la nube alojado en GitHub. De esta forma se pueden realizar todas las acciones Git sin tener que acceder a Github, ya que las acciones llevadas a cabo en SourceTree se verán reflejadas directamente en el repositorio en la nube.

Para mantener un buen control de versiones se decidió seguir el modelo que se utiliza popularmente en el mundo del desarrollo frontEnd. Se dispondrá de la rama principal con nombre “main”, que será la rama de producción. Esta rama será la que esté vinculada a la página web de documentación ubicada en Netlify (explicado en 3.3.3 Netlify) y siempre deberá de revisarse a detalle antes de hacer cualquier fusión de ramas que el contenido no genere conflictos.

Luego estará la rama de desarrollo con el nombre de “develop”, que será la rama origen de todas las “ramas función” o ramas feature como se llamará en esta sección. De la rama develop saldrán las ramas feature, que serán tareas concretas para implementar componentes, secciones de documentación, estilos, etc. y se utilizará una estructura numérica seguida de una breve descripción para poder identificar de un vistazo qué se está desarrollando en esta feature.

2-Ramas SourceTree

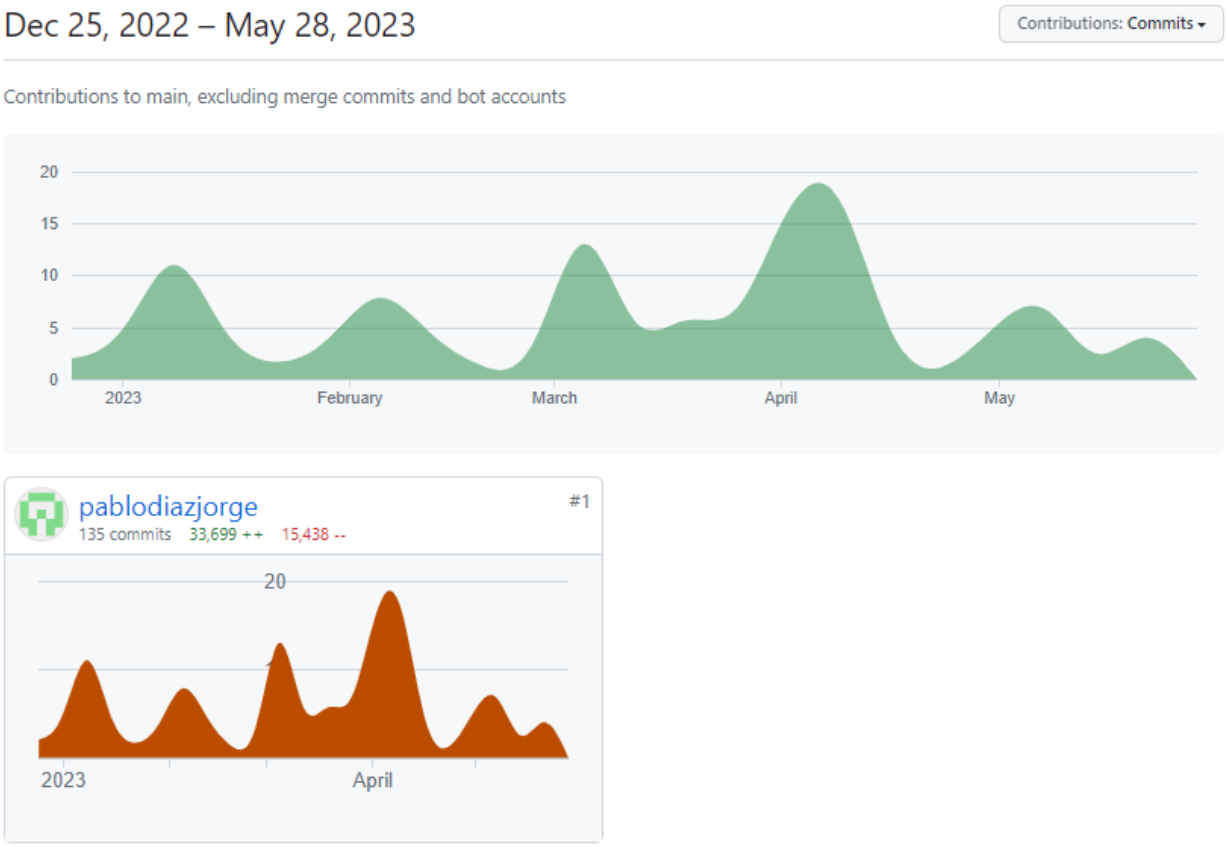


En esta imagen se puede ver en la sección "Branches" las ramas en local y en la sección "Remotes" las ramas en la nube. También está el ejemplo de la nomenclatura seguida a lo largo del proyecto.

Como ya se ha comentado al inicio del proyecto, se propuso un desarrollo incremental debido a la complejidad de la relativamente novedosa tecnología que son los Web Components. Se ha seguido este tipo de desarrollo, avanzando con features, en un inicio construyendo los componentes necesarios para la landing-page y luego los necesarios para la sección de documentación de cada componente. Una vez se desarrolló la primera versión con las funcionalidades necesarias para la web palaze, se empezó a desarrollar más funcionalidades y características de los componentes. Estas características y funcionalidades están todas explicadas en la propia documentación de cada componente en la página web.

A lo largo del proyecto se han creado un total de 56 features y un total de 135 commits a lo largo del periodo del 25 de diciembre del 2022 al 28 de mayo del 2023 como se mostrará en la siguiente imagen extraída del repositorio GitHub.

3-Github commits



Para más información sobre los commits en el proyecto, se puede consultar este enlace.

3.3.3 Netlify

Netlify es una plataforma que permite el alojamiento y despliegue de sitios web. El objetivo de usar esta plataforma en el proyecto es disponer de una versión online desplegada de la documentación de los componentes.

Esta plataforma es intuitiva y fácil de usar, por lo que la curva de aprendizaje es prácticamente mínima.

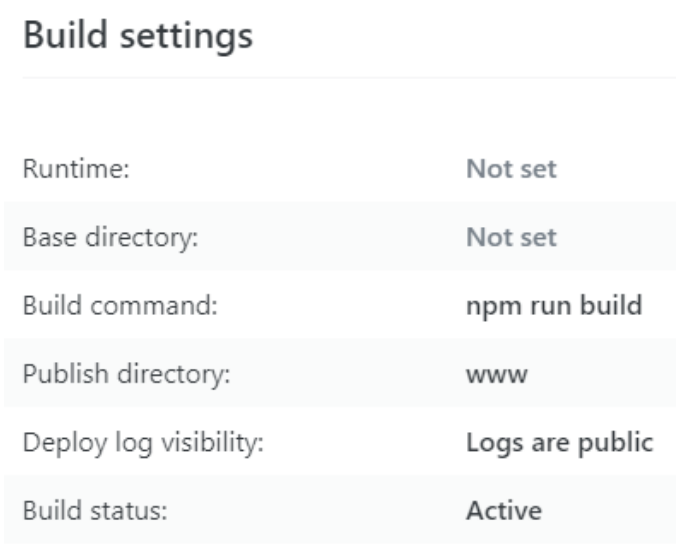
Uno de los factores que hizo clave la elección de Netlify es la integración con plataformas de control de versiones. En este caso, permite vincular el repositorio de GitHub, concretamente la rama de producción (main) y mantener continuamente actualizada la página web, ya que cuando se realicen cambios en la rama, se volverá a construir y desplegar con esos mismo cambios.

Por otra parte, cabe recalcar que, al estar utilizando una versión gratuita, la carga de la web se vuelve un poco lenta, variando la velocidad dependiendo de la afluencia de interacciones en los servidores de la plataforma en el momento y no permite alojar muchos datos. En este caso, ya que es un proyecto de final de grado, no se ha visto necesario contratar un servicio e invertir dinero, por lo que se ha optado por una versión gratuita, conociendo sus limitaciones.

En cuanto a la implementación del despliegue, el primer paso sería crearte una cuenta en la página web de la plataforma y vincular tu cuenta de GitHub para disponer de tus repositorios.

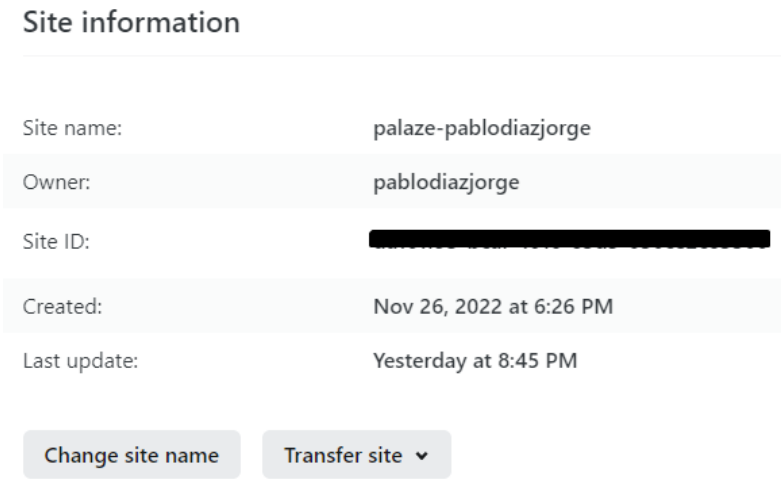
Una vez hecho esto deberás crear un "nuevo sitio web". En esta opción te aparecerán configuraciones que deberás completar para determinar cómo se construirá la web. Para ello debes conocer cómo se despliega tu web dependiendo del entorno en el que te encuentres, en este caso, la distribución que se utiliza se llama "www" y se construye con el comando npm run build, por lo que se tendrá que especificar como se ve en la imagen a continuación.

4-Netlify build settings



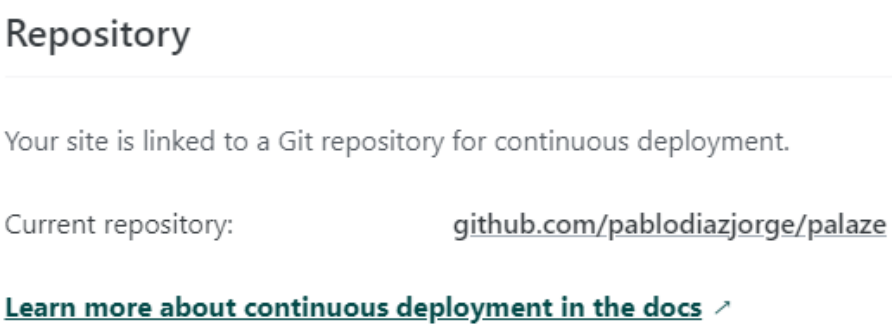
Otra configuración que se deberá especificar son el nombre de dominio de la página que en este caso será el siguiente:

5-Netlify domain settings

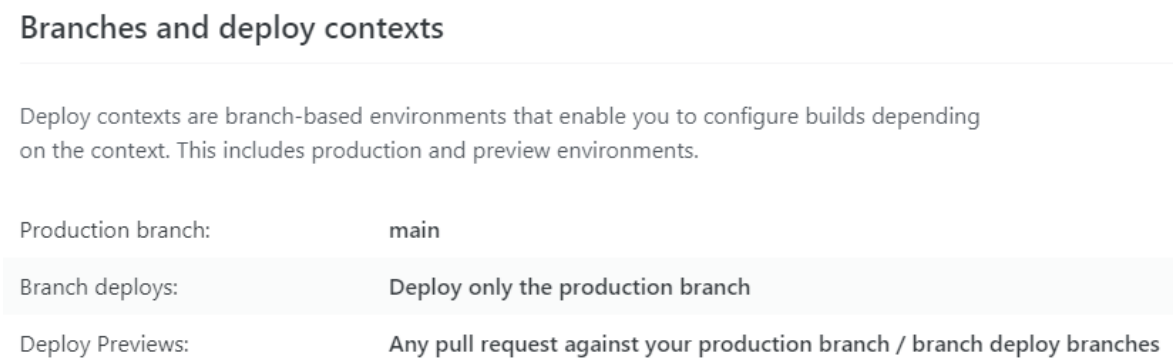


Por último, habrá que definir la configuración del repositorio de GitHub y la configuración de la rama a la que apuntará el despliegue:

6-Netlify repository settings



7-Netlify branches settings



Tras realizar estas configuraciones se hallará la página web Palaze desplegada online, a la que puedes acceder mediante este enlace.

3.4 Publicación y Distribución

3.4.1 Distribución

Una distribución es el resultado de compilar y empaquetar código, en este caso la librería de componentes o la página web de documentación. Es el conjunto de archivos generados para distribuir y utilizar en las aplicaciones web.

Una de las características más importantes de StencilJS es la capacidad de generar distintas distribuciones dependiendo del uso de los componentes. Se pueden especificar como "output target types" en el archivo de stencil.config.ts.

Existen un total de tres distribuciones proporcionadas por StencilJS:

- **www:** Es la distribución por defecto, despliega una aplicación web en un servidor HTTP y es la distribución con web completa que se utilizará para mostrar la documentación de los componentes.
- **dist:** Es una distribución empleada para crear una biblioteca de componentes ya que puede ser importado y utilizado en otros proyectos.
- **dist-custom-elements:** Es la distribución empleada para generar un paquete de custom elements que podrán utilizarse en cualquier proyecto sin dependencias de StencilJS. Serán componentes que se puedan utilizar sin framework o con cualquier tipo de framework.

El proyecto dispone de estas tres distribuciones para que los desarrolladores que utilicen la librería palaze puedan disponer de los componentes de la forma que más se adecue a sus necesidades.

3.4.1.1 www

La distribución "www" es la opción por defecto. Desplegará una aplicación web en un servidor HTTP y es especialmente útil para implementar prerenderización y service workers.

Es recomendable especificar dentro del archivo de stencil.config.ts la distribución dentro de la sección **'outputTargets:[ ]'**, pero si no está definida otra distribución, esta se aplicará por defecto.

Esta distribución se utiliza en el proyecto para construir la aplicación web de documentación sobre la librería de componentes y desplegarla junto a los componentes. Se incluirán todos los archivos CSS, HTML, JavaScript y recursos como imágenes entre otros.

Como se ha mencionado anteriormente, la opción de prerenderizado ayudará a desplegar contenido estático con mejor rendimiento para mejorar la experiencia de usuario.

En este caso, no se hará uso de los service workers ya que no se tendrá ninguna funcionalidad de experiencia offline cuando la red no está disponible. A su vez, se deberá definir las rutas de los archivos html para que puedan construirse y empaquetarse en la distribución como se observa a continuación.

8-Distribución www

```
24        {
25          type: 'www',
26          serviceWorker: null, //disable service workers
27          copy: [
28            { src: 'docs/docs.html', dest: 'docs.html' },
29            { src: 'docs/docs-button.html', dest: 'docs-button.html' },
30            { src: 'docs/docs-card.html', dest: 'docs-card.html' },
31            { src: 'docs/docs-divider.html', dest: 'docs-divider.html' },
32            { src: 'docs/docs-hero.html', dest: 'docs-hero.html' },
33            { src: 'docs/docs-menu.html', dest: 'docs-menu.html' },
34            { src: 'docs/docs-menu-item.html', dest: 'docs-menu-item.html' },
35            { src: 'docs/docs-tooltip.html', dest: 'docs-tooltip.html' },
36            { src: 'docs/docs-spinner.html', dest: 'docs-spinner.html' },
37            { src: 'docs/docs-design-system.html', dest: 'docs-design-system.html' },
38            { src: 'docs/docs-section.html', dest: 'docs-section.html' },
39            { src: 'docs/docs-footer.html', dest: 'docs-footer.html' },
40            { src: 'docs/docs-demo-sidebar.html', dest: 'docs-demo-sidebar.html' },
41            { src: 'docs/docs-demo-header.html', dest: 'docs-demo-header.html' },
42            { src: 'docs/docs-header.html', dest: 'docs-header.html' },
43            { src: 'docs/docs-sidebar.html', dest: 'docs-sidebar.html' },
44            { src: 'docs/docs-dropdown-menu.html', dest: 'docs-dropdown-menu.html' },
45            { src: 'docs/docs-colors.html', dest: 'docs-colors.html' },
46            { src: 'docs/docs-showcase.html', dest: 'docs-showcase.html' },
47          ],
48        },
```

3.4.1.2 Dist

La distribución "dist" se utiliza para generar una biblioteca de componentes que puede ser auto-cargable de forma diferida, también conocido como self-lazy loading. Por otra parte, el paquete generado es "treeshakable". A continuación, se definirá estas dos características que proporciona StencilJS para este tipo de distribución con más detalle.

- **Lazy loading** (carga diferida): Permite cargar únicamente los componentes que se utilicen en tu página web. Esto permite poder agregar una etiqueta script a cualquier página para disponer de la librería de componentes al completo lista para usar, pero optimizando el rendimiento de tu página, ya que solo se cargarán los componentes que utilices, la carga de componentes funciona bajo demanda. Esto funciona de tal forma que cada componente se empaqueta como un módulo independiente, de ahí la reducción del tiempo de carga inicial, la optimización de recursos y la mejora de la experiencia de usuario que proporciona.

- **Tree shaking:** Es una técnica de construcción de paquetes que permite eliminar código que no se utiliza en una aplicación. En los componentes se aplica de tal manera que solo los componentes importados y utilizados en el proyecto son construidos en el paquete final de la biblioteca.

  StencilJS analiza el código de estos componentes que se está utilizando y a partir de esta información elimina los componentes a los que no se les da uso para que el paquete final cargado en la construcción de tu página esté optimizado y sea más ligero.

Al igual que la distribución anterior, se debe definir la distribución dentro del archivo de stencil.config.ts y dentro de la sección **'outputTargets:[ ]'** como se muestra a continuación.

9-Distribución dist

```
6       outputTargets: [
7         {
8           type: 'dist',
9           esmLoaderPath: '../loader',
10        },
```

3.4.1.3 Dist-custom-elements

La distribución dist-custom-elements se utiliza para generar un paquete de custom elements que extienden directamente de HTMLElement. Esto proporciona funciones para definir los elementos en el Custom Element Registry, que es un registro para definir que estos elementos son custom elements y por lo tanto podrán utilizarse como Web Components personalizados.

A diferencia de la distribución explicada anteriormente (dist), dist-custom-elements no dispone de lazy loading ni de tree shaking por lo que si se quiere esta tecnología, se tendrá que hacer referencia a la distribución dist o tener un framework que se encargue del empaquetado y el registro de elementos personalizados de forma independiente.

Sin embargo, si no requieres de esta tecnología puedes utilizar los componentes perfectamente en un entorno con framework o sin framework.

Al igual que las distribución anteriores, habrá que definirla dentro del archivo de stencil.config.ts y dentro de la sección **'outputTargets:[ ]'** como se muestra a continuación.

En este caso, debido al componente plz-button que utiliza cuatro imágenes predefinidas, se deberá utilizar la etiqueta **'copy:[ ]'** para seleccionar la ruta donde se encuentran las imágenes y poder hacer el empaquetado del componente correctamente, ya que si no se hiciera esto, al utilizar la librería desde otro proyecto, no tendrías acceso a las imágenes, al compilar el proyecto se intentaría obtener las imágenes mediante su ruta pero no las encontraría.

10-Distribución dist-custom-elements

```
11        {
12          type: 'dist-custom-elements',
13          generateTypeDeclarations: true,
14          copy: [
15            { src: 'assets/icon/Icon-github.svg', dest: 'assets/icon/Icon-github.svg' },
16            { src: 'assets/icon/Icon-linkedin.svg', dest: 'assets/icon/Icon-linkedin.svg' },
17            { src: 'assets/icon/Icon-mail.svg', dest: 'assets/icon/Icon-mail.svg' },
18            { src: 'assets/icon/Icon-palaze.svg', dest: 'assets/icon/Icon-palaze.svg' },
19          ],
20        },
```

3.4.2 Publicación en NPM

Node Package Manager, también conocido como NPM, es un administrador de paquetes que permite publicar, compartir y distribuir bibliotecas con código fuente en un ecosistema JavaScript.

Para este proyecto se decidió publicar la librería de componentes con este paquete debido a su popularidad e instalación sencilla. Al publicar tu librería en NPM, los usuarios la pueden instalar fácilmente mediante mediante una línea de comandos, lo que lo vuelve mucho más accesible.

Para poder publicar un paquete de código fuente en NPM primero deberás registrarte en su página oficial. Una vez hecho esto hay varias formas de publicar tu paquete, lo puedes hacer directamente desde la página de GitHub o como se ha hecho en el caso de este proyecto, entrar en el entorno de desarrollo vinculado a tu repositorio GitHub y ejecutar las siguientes líneas de comando.

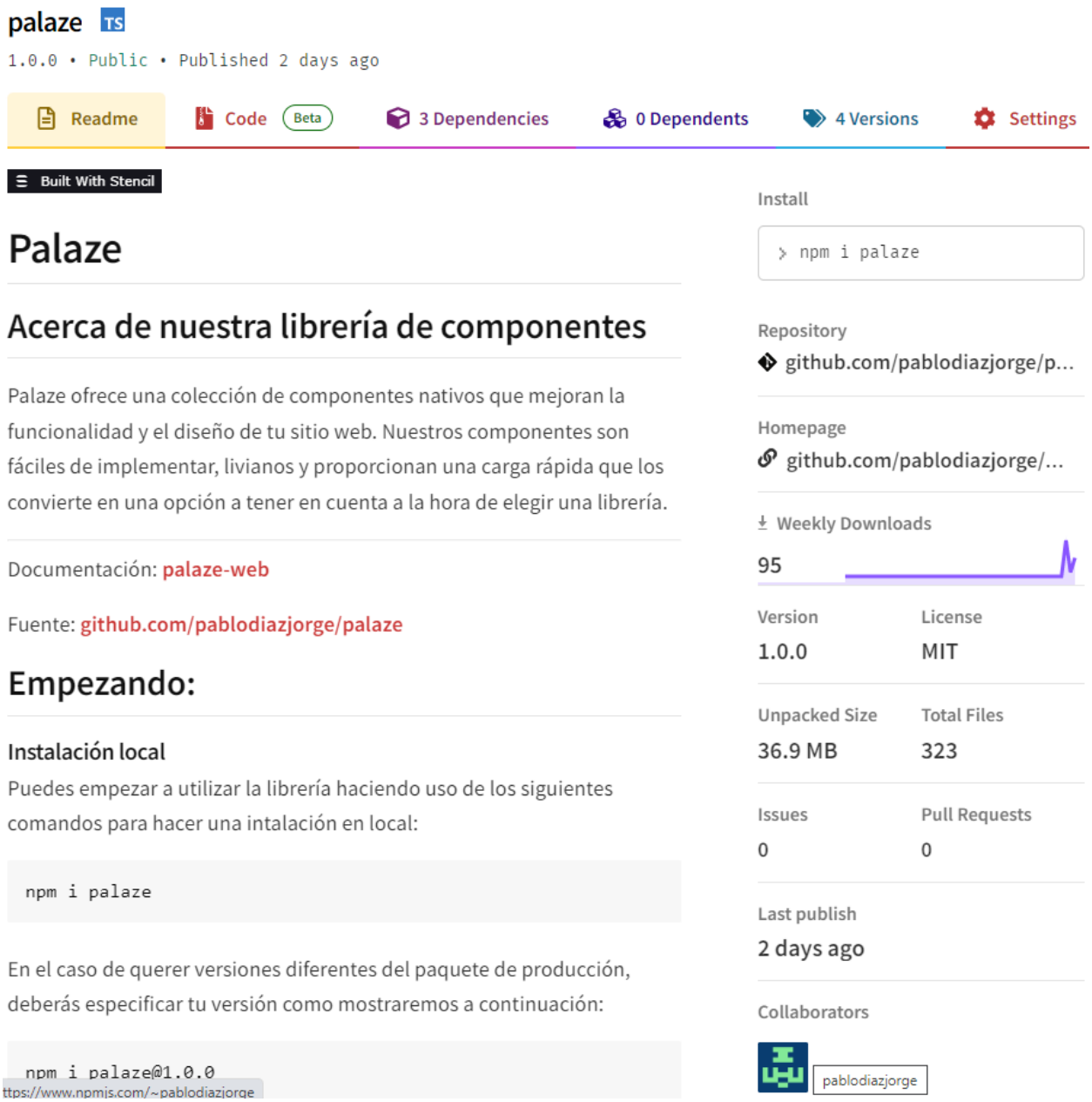
Deberás iniciar sesión mediante el comando "`npm login`". En el terminal te pedirá información de inicio de sesión, concretamente usuario y contraseña.

Una vez hayas iniciado sesión correctamente se procederá a publicar la librería con el comando "`npm publish --access public`". En este caso, al tener verificación en dos pasos para tener más seguridad sobre el repositorio, se deberá proporcionar otro código complementario para completar la publicación del paquete.

Recalcar que habrá que cambiar el archivo package.json del proyecto para definir la versión del paquete de web components.

A continuación, en la imagen se puede ver el paquete palaze publicado y con su versión de release 1.0.0.

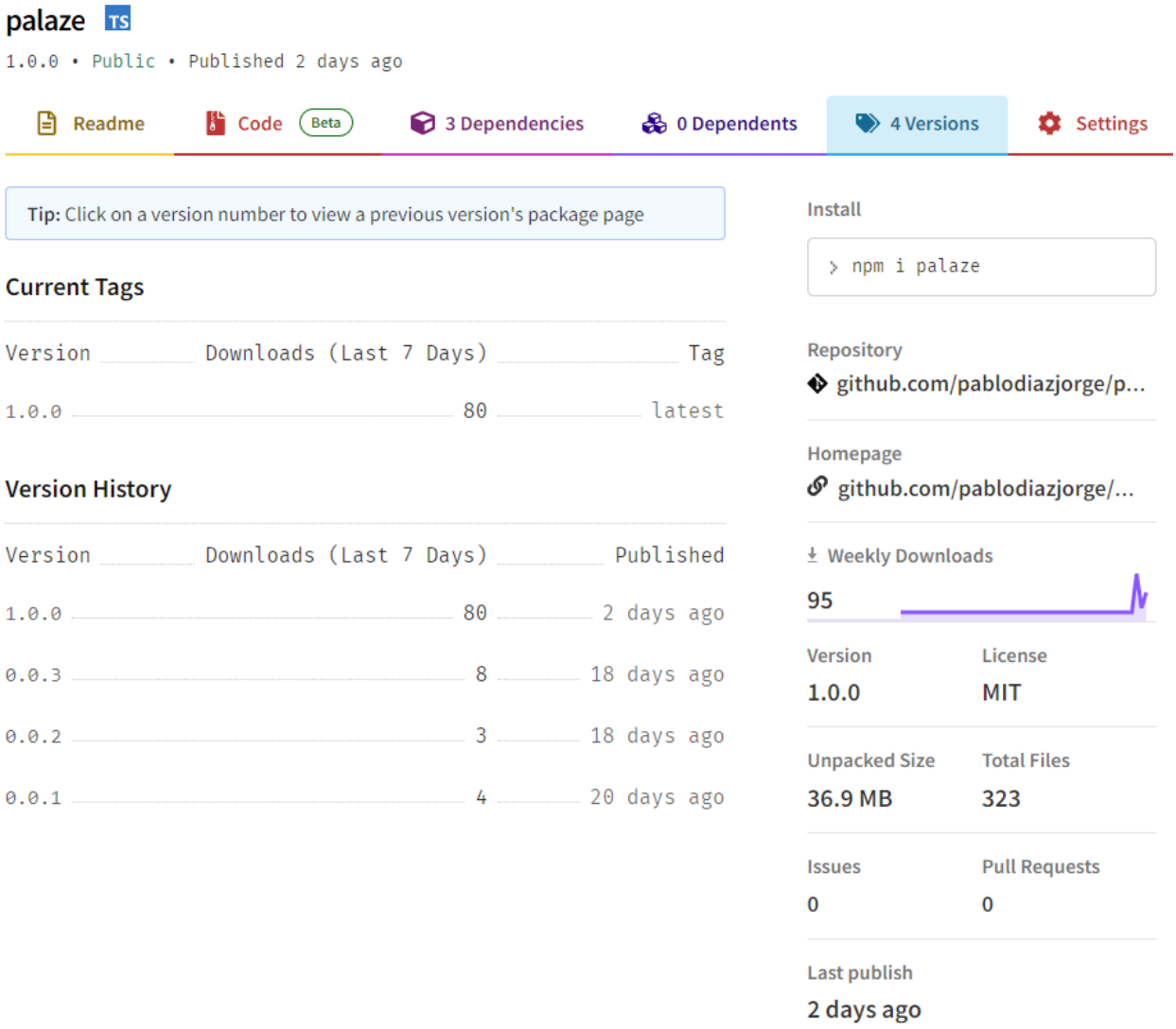
11-Publicación npm



Como se puede observar en la imagen, dentro del paquete NPM aparece información de la librería y una iniciación sobre el uso e instalación de los componentes. También aparece un link a la página web de documentación y un enlace al repositorio de GitHub.

Antes de disponer de la primera versión de release, se hicieron pruebas con el paquete desde distintos proyectos. Una vez revisado y arreglado algún error de repercusión mínima que podía quedar en el paquete, se hizo una actualización del paquete. A continuación, se puede ver una imagen de las versiones del paquete.

12-Publicación npm versiones



Para obtener más información puedes acceder al paquete publicado en NPM mediante este enlace.

3.4.3 Instalación de la librería y uso

La instalación de la librería se puede hacer en local o de forma online mediante una referencia a la librería.

Para la instalación local se deberá de hacer uso del comando:

```
npm i palaze
```

En el caso de querer versiones anteriores del paquete de producción, deberás especificar tu versión como se mostrará a continuación:

```
npm i palaze@0.0.3
```

Una vez instalado el paquete, deberás añadir unas líneas de código en tu página para poder hacer uso de los componentes. A continuación, se muestra cómo hacerlo:

```
<script type="module"
src="/node_modules/palaze/dist/palaze/palaze.esm.js"></script>
```

Será trabajo del desarrollador que haga uso de la librería de ajustar las configuraciones para su proyecto en concreto.

En el caso del ejemplo, se almacena dentro del fichero "node_modules/palaze/". En esta ubicación se encontrará la instalación del paquete.

Este ejemplo es de cómo implementar la librería desde la distribución dist, para implementarla desde la distribución dist-custom-elements se deberá de configurar dependiendo de tu entorno de desarrollo de proyecto. Para un caso genérico, se debería de poder importar un componente único para su uso mediante:

```
import { defineCustomElement } from 'palaze/dist/components/plz-
button';
```

La ruta y uso del componente variará dependiendo del entorno de desarrollo del proyecto en el que te encuentres, para más información se puede consultar este enlace donde se detalla esta peculiaridad en la documentación de StencilJS.

Por otra parte, se encuentra la instalación online vía CDN, este tipo de instalación es el más sencillo, ya que para poder disponer de la librería no se necesita hacer uso de ningún comando ni instalar nada, todo será online vía referencia al paquete.

A continuación, se muestra un ejemplo de cómo instalar o implementar la librería vía CDN:

```
<script type="module"
src="https://cdn.jsdelivr.net/npm/palaze/dist/palaze/palaze.esm.
js"></script>
```

Para ver una demo online del uso de la librería podrás acceder a un CodePen vía el siguiente enlace.

En cuanto al uso de componentes de la librería palaze, se utilizan como elementos HTML nativos. Para hacer uso de ellos debes conocer sus atributos, slots y funcionalidades. Para conocerlos mejor deberás acceder a la documentación de cada componente ubicada en la página palaze-web, donde se explica detalladamente cada componente y todo lo que necesitas saber para empezar a desarrollar tus proyectos haciendo uso de los componentes palaze.

A continuación, se muestra un ejemplo de cómo se usaría el componente plz-button en una de sus variantes:

```
<plz-button color="black" hover-color="#495057"  corners="low-
rounded">Docs</plz-button>
```

En este código se puede apreciar la etiqueta que caracteriza al componente "plz-button" junto con sus atributos, cuyas diferentes características y opciones vendrán definidas en palaze-web.

4 Diseño e implementación

En esta sección se explicará todo el proceso de creación de un web component, desde un diseño previo de la web, a partir de este un diseño de web components y para finalizar el desarrollo de estos mediante las tecnologías seleccionadas siguiendo las recomendaciones de buenas prácticas (best practices).

Debido a que los diseños pueden llegar a ocupar espacio superior a una hoja A4 convencional, se dividirá durante las siguientes secciones para que las explicaciones se vuelvan más amenas. Si se desea consultar el entorno de diseño de Figma donde se encuentran todos los diseños creados para este proyecto, se puede acceder mediante este enlace.

4.1 Idea y representación gráfica

El primer paso a tomar después de haber aprendido sobre la tecnología Web Components y decidido las herramientas y tecnologías que se utilizarán para el proyecto es investigar acerca de las bases de una web de presentación y una web de documentación.

Una vez analizado varios diseños de este tipo de webs se ha construido la primera idea acerca de cómo representar estas dos páginas de la aplicación web palaze. Posteriormente a partir de estos bocetos se hará el diseño final.

Mientras se realizó el proceso de análisis y planteamiento, se pensó en varios nombres para la librería de componentes, decidiendo finalmente llamarla **"Palaze"**.

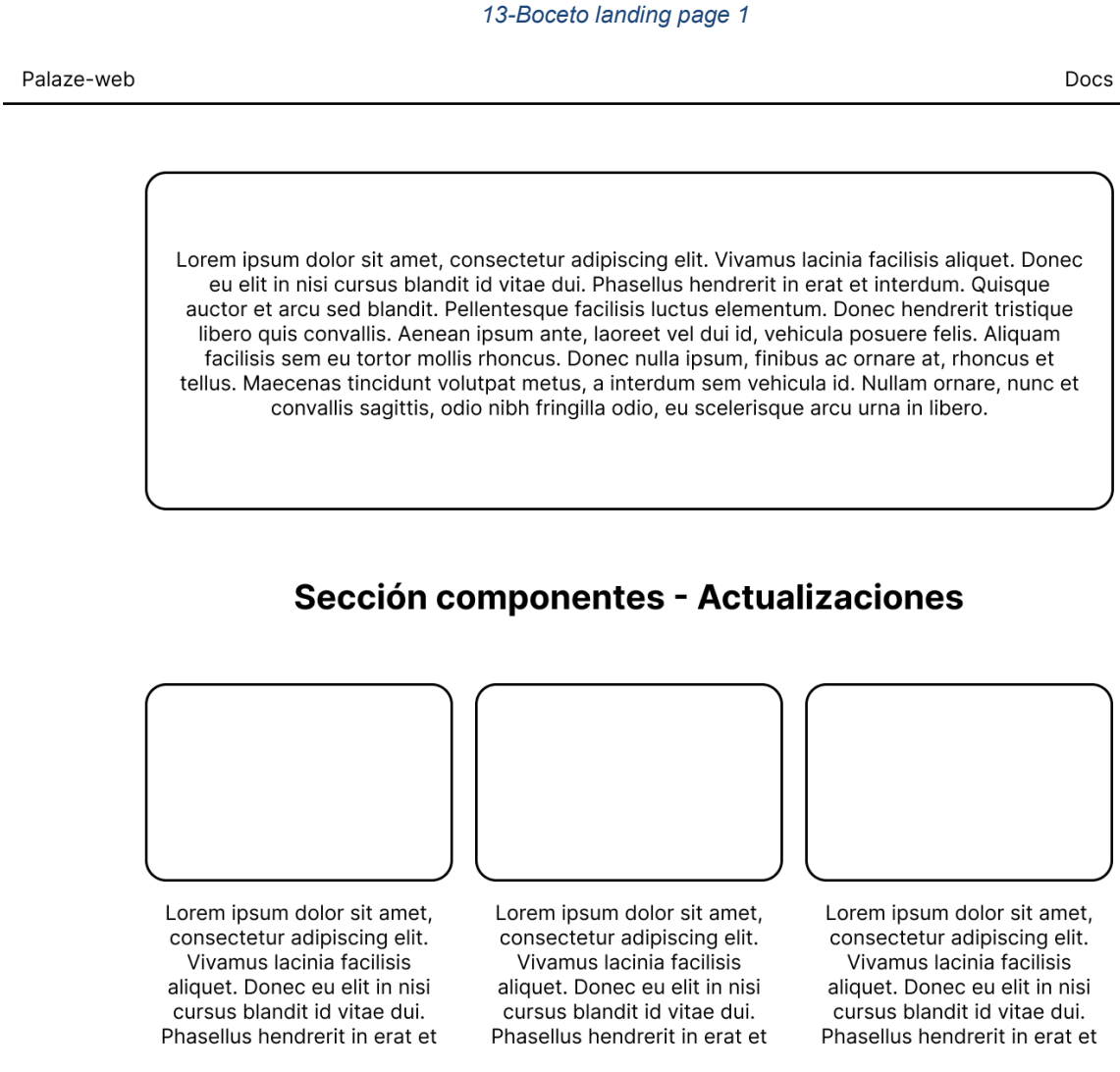
Para mostrar el primer boceto de la página de presentación de la web (landing page) se dividirá en partes:

Header, estará situado en la parte superior de la página siempre accesible por el usuario, habrá una barra de navegación con enlaces a la documentación, al repositorio GitHub, etc.

Sección Hero, será lo primero que habrá al entrar a la página con un título llamativo y una descripción sobre el producto, en este caso la librería de componentes.

Sección de últimas actualizaciones: se mostrarán mediante cards información con un link a las últimas actualizaciones que se han producido en la librería de componentes.

13-Boceto landing page 1

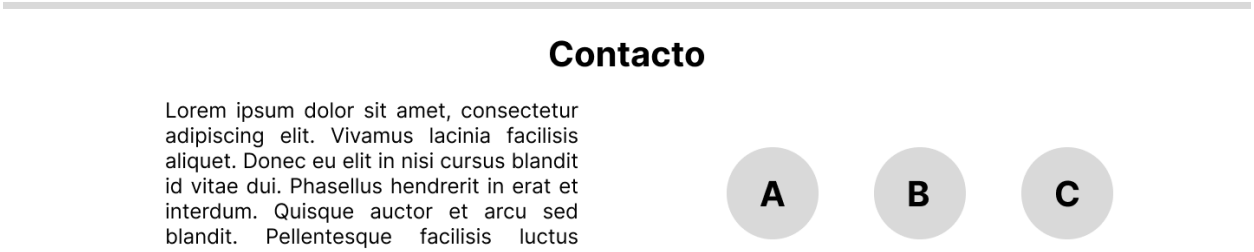


A continuación, habrá un separador para dividir un poco el contenido mostrado anteriormente y la siguiente **sección acerca del proyecto**, donde habrá un título y un texto descriptivo amplio contando información relevante de este.

14-Boceto landing page 2

Por último, habrá un pié de página (footer) donde se encontrará información de motivo social con un par de iconos con enlace a las diferentes redes.

15-Boceto landing page 3



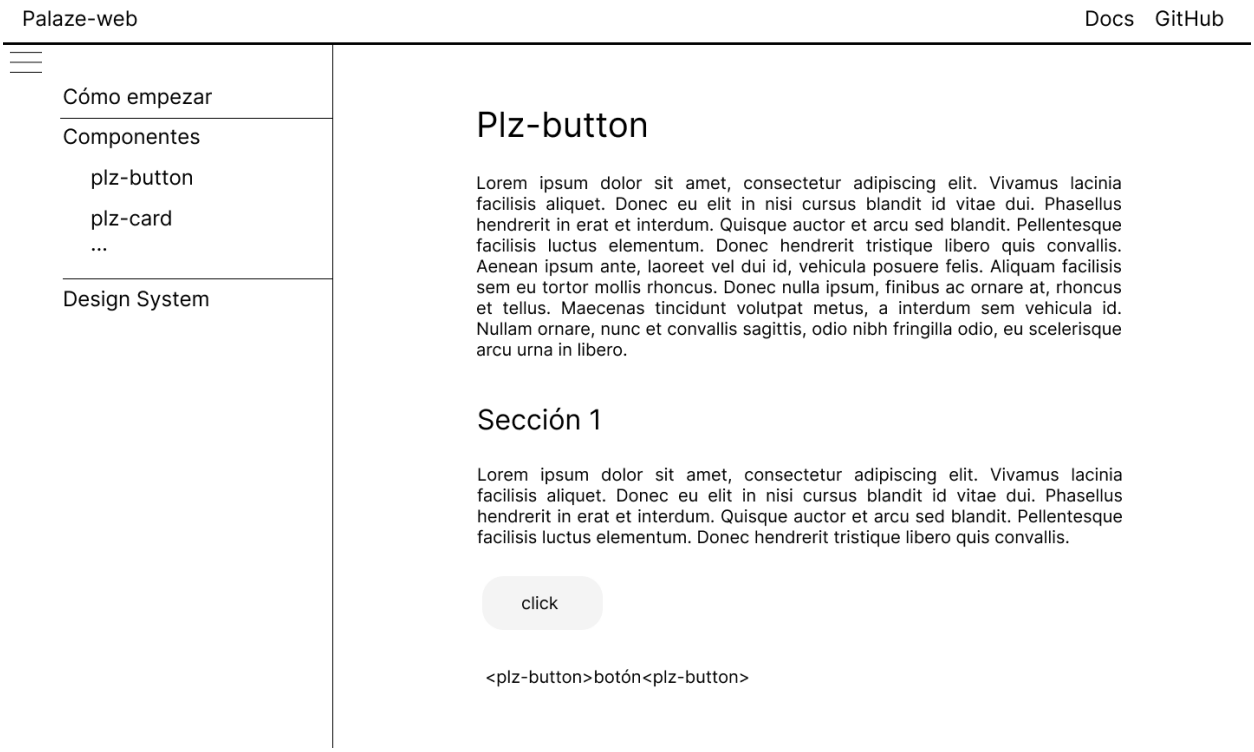
El boceto de la página de documentación de los web components (docs page) constará de:

Header, estará situado en la parte superior de la página siempre accesible por el usuario, habrá una barra de navegación con enlaces a la documentación, al repositorio GitHub, etc.

Sidebar, siempre visible al entrar en la página. Será una navegación con estilo parecido a un índice con el que podrás acceder a cada apartado de documentación de la librería, ya sea general o concreta de los componentes. Dispondrá de un botón con forma de acordeón que permitirá la función de esconder o mostrar el sidebar cuando se haga click sobre este.

Contenido de documentación, toda la información relevante acerca del componente, uso de este con ejemplos gráficos y toda clase de detalles. No se desarrolló el boceto de esta sección en profundidad debido a que faltaba analizar los distintos componentes junto con sus características y funcionalidades.

16-Boceto docs



Con estos primeros bocetos de la página de presentación y la página de documentación, era momento de determinar una estrategia para el diseño a un nivel más profesional haciendo uso de la herramienta Figma.

4.1.1 Estrategia

A partir de los bocetos expuestos anteriormente, habiendo realizado pruebas y leído documentación sobre material de diseño y cómo utilizar la herramienta Figma, se definirá la estrategia.

Se utilizará la metodología llamada Atomic Design. Se seguirá la estructura inversa diseñando primero los elementos más grandes y luego ir descomponiendo en elementos más pequeños.

En este proceso de diseño, no se seguirá al pie de la letra todas las partes de la metodología, se dejará un margen para que los desarrolladores tengan decisiones en cuanto a cómo aplicar la funcionalidad y añadir características en base al diseño.

Primero, a partir del boceto inicial, se diseñará la página final teniendo en cuenta el diseño general que se quiere aplicar, manteniendo una estructura y una disposición correcta de los elementos.

La página de presentación seguirá una distribución sencilla pero muy colorida, buscando el factor atractivo para captar atención.

Sin embargo, la página de documentación seguirá un estilo minimalista, buscando la tranquilidad haciendo uso de colores sólidos y contrastes fuertes, ya que se usará principalmente para una lectura más compleja acerca del uso de los componentes, funcionalidades y características de la librería.

El siguiente paso es dividir este diseño de página final en elementos más pequeños, en este caso al utilizar tecnología Web componentes, se divide en componentes. Para ello hay que observar elementos en la página que se repitan o que puede ser conveniente disponer de ellos como componentes.

Una vez hecho esto, diseñar posibles variaciones de los componentes extraídos y aplicar diferentes estilos a los diseños manteniendo la cohesión entre los diseños.

Por último, identificar las características comunes entre los componentes que formarán el conjunto de tokens de diseño. Estos tokens serán aplicados por el desarrollador para la creación de los componentes.

Recalcar que el diseño se llevará a cabo en unas dimensiones de pantalla de ordenador de escritorio, pero al hacer el desarrollo de la aplicación web, se buscará que tanto la aplicación web como los componentes se adapten a cualquier tamaño de pantalla de diferentes dispositivos.

4.1.2 Diseño en figma de la web

Como en el apartado donde se han expuestos los bocetos, se divide el diseño en partes para poder explicarlo mejor empezando por el diseño de la página de presentación o landing page y luego el diseño de la página de documentación.

En base al boceto, se ha decidido emplear un enfoque colorido para la página de presentación, haciendo uso de paletas cromáticas, pero limitando el rango a cinco por color (en la sección 4.1.4 Tokens de diseño, se profundiza en las elecciones del diseño).

17-Diseño landing page 1

Como se puede observar, se ha definido claramente dos de los elementos del boceto, por una parte el header o encabezado que ahora consta de un panel de navegación a la izquierda con un icono junto con el nombre de la web y otro panel de navegación a la derecha, con enlaces a la misma página como pueden ser Contacto y Acerca de, con los que puedes acceder a distintas secciones y otros dos enlaces a la página de documentación de componentes, Docs, y a la página donde se encuentra el código fuente, Github.

El icono de la página web se ha diseñado haciendo uso de la inteligencia artificial Midjourny. Mediante un comando descriptivo se le pidió un logo con unas características concretas y después de varios intentos se eligió uno. Posteriormente se trasladó el archivo a la herramienta Photoshop para refinar la imagen eliminando elementos innecesarios y por último se procesó en la herramienta Figma para convertirlo en un archivo .svg, ya que es un tipo de archivo vectorial más adecuado para la carga en páginas web. El resultado es el siguiente:

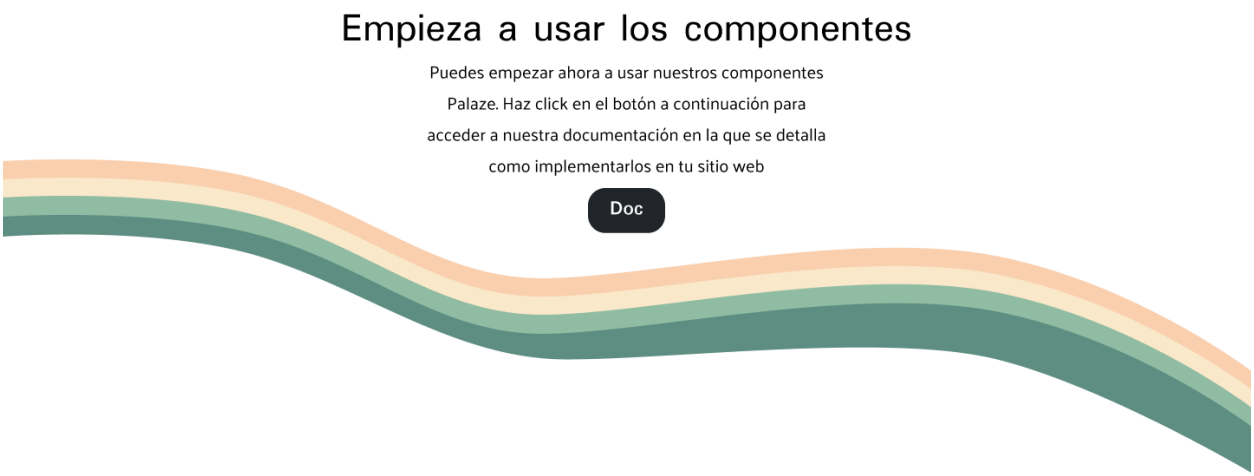
18-Icono palaze

Por otra parte, se encuentra la sección hero, en la que finalmente se ha decidido optar por una imagen con patrones de gamas cromáticas también creada siguiendo el mismo proceso

que el icono de la web y un contenedor en el centro con fondo blanco para evitar problemas con la lectura, siguiendo así las reglas de accesibilidad web. Dentro de este contenedor se ubica un título con el nombre de la web en grande y a continuación un texto descriptivo que trata de ser atractivo para captar la atención con una alineación al centro.

Justo después, continuando la página, se ha añadido una sección que no aparecía en el boceto. Se ha visto conveniente añadir un texto llamativo para incitar a entrar a probar la librería palaze junto con un botón para acceder a la documentación.

19-Diseño landing page 2

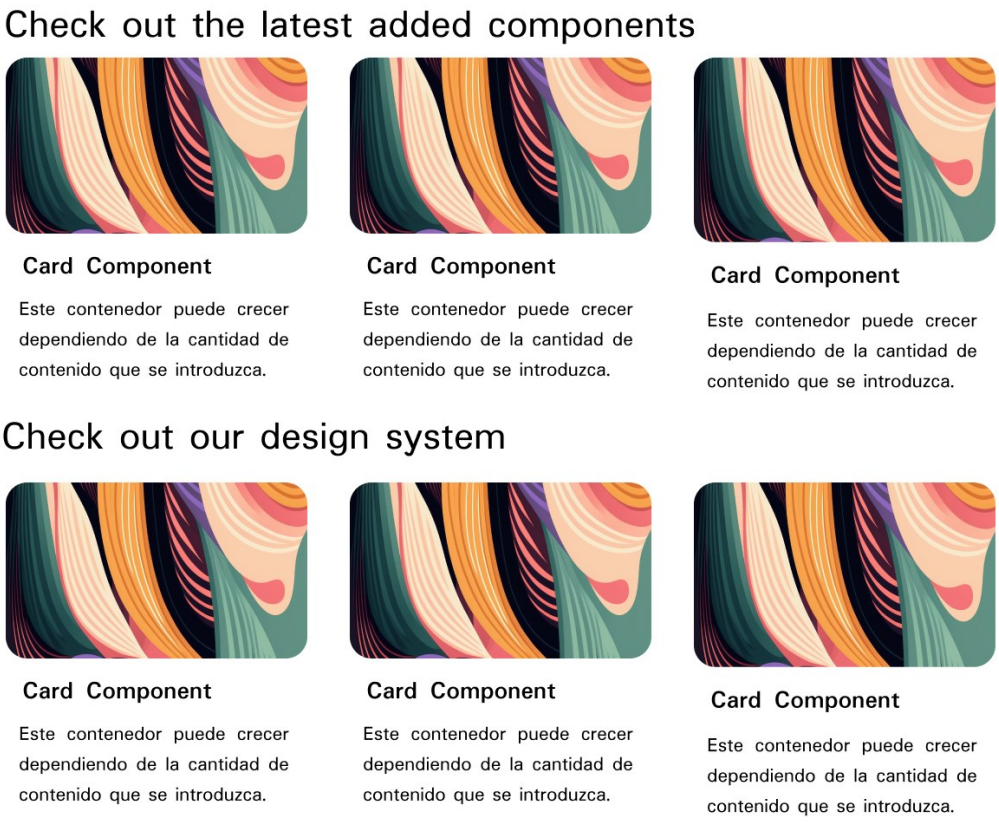


Como se observa, la estructura del contenedor que utiliza el texto sigue la misma composición que en la sección hero.

Para marcar una diferenciación entre el contenido de esta sección y la sección que vendrá a continuación, se ha creado un archivo vectorial con una paleta de colores verdes diseñado y construido con Figma que actuará como un separador.

La siguiente sección corresponde a noticias sobre los componentes, concretamente actualizaciones. También se ha añadido actualizaciones sobre el Design System, todo en forma de tarjetas.

20-Diseño landing page 3



Esta sección no se ha definido al completo debido a que el contenido se irá actualizando en la propia aplicación. En la sección relativa a noticias de componentes habrá un título con "Echa un vistazo a los últimos componentes añadidos" y dentro aparecerán tres elementos en los que habrá un título del componente en el que se han realizado las modificaciones y una breve descripción de los cambios realizados. Cada uno dispondrá de una imagen distinta.

A su vez, en la sección de Design system se seguirá la misma estructura con "Consulta nuestro sistema de diseño" y también aparecerán noticias referidas a este con la misma estructura explicada anteriormente.

21-Diseño landing page 4

Acerca de

Proyecto Palaze

En Palaze, nos enorgullece ofrecer componentes web nativos de calidad, diseñados para mejorar tu experiencia de desarrollo web. Nuestros componentes son fáciles de usar e intuitivos, permitiendo así ahorrar más tiempo y esfuerzo en la creación de tus sitios web de forma profesional.

Además, nuestros componentes son totalmente compatibles con los estándares web actuales, haciendolos así seguros y escalables para adaptarse a tus necesidades de negocio a medida que evolucionan los proyectos. Puedes estar seguro al elegir Palaze ya que es una solución sólida y confiable que te permitirá crear aplicaciones web atractivas y funcionales de forma rápida y eficaz. Debido a la flexibilidad y personalización de los componentes, podrás modificarlos para adaptarlos a tus necesidades específicas o integrarlos a las herramientas que se usen en tus proyectos. Puedes estar seguro de que tus proyectos funcionarán en cualquier dispositivo y navegador.

En resumen, en Palaze nos esforzamos por brindar componentes web nativos originales y con una implementación sencilla. Puedes unirte a nuestra comunidad de desarrolladores y experimentar con el desarrollo tu mismo.

Creador

Pablo Díaz-Jorge García, estudiante de último año de ingeniería informática.
"Desde pequeño siempre me ha gustado la tecnología, me decantaba más por el software que por el hardware. Conforme fueron pasando los años me decidí por enfocar mi trayectoria profesional a la vertiente que más me apasionara, el desarrollo web y concretamente el desarrollo front-end.
En el trabajo de final de grado (tfg), decidí apostar por una tecnología relativamente nueva que son los componentes web nativos. Aquí fue donde empecé a aprender sobre el desarrollo web y desarrollar mis conocimientos sobre los lenguajes CSS, JavaScript y HTML.
Este proyecto empezó como una forma de aprender y empezar a encaminar mi carrera profesional,durante el proceso se han replanteado y rehecho muchas partes una vez desarrolladas, ya que una sola persona tenía que hacer de diseñador y desarrollador.
Después de mucho tiempo empleado y cada vez ir ganando fluidez conforme iba aprendiendo más sobre estos lenguajes y los pequeños trucos para hacer que todos los elementos web se comporten como se esperaría entre ellos, se puede decir que he alcanzado un buen estado del proyecto.
Mi idea con este proyecto es no dejarlo solo como un trabajo de final de grado, sino continuar con su desarrollo conforme pase el tiempo, aprendiendo al crear nuevos componentes y funcionalidades."

Para separar la sección de actualizaciones y novedades se empleará otro separador, que es un archivo vectorial con paleta de colores naranjas que, como el anterior separador verde, ha sido diseñado y construido con Figma.

Seguido del separador se ubica otra sección llamada Acerca de en la que se distribuye la información en dos columnas, en la primera se cuenta información acerca del proyecto palaze e información llamativa de la librería y en la segunda columna, se expone información sobre el creador y cómo surgió esta idea.

Para finalizar, en la parte inferior de la página de presentación habrá un pié de página con referencias a enlaces externos como linkedin, GitHub y un email con el que contactar mediante iconos como se definió en el boceto inicial. También aparece un icono con el logo de palaze con

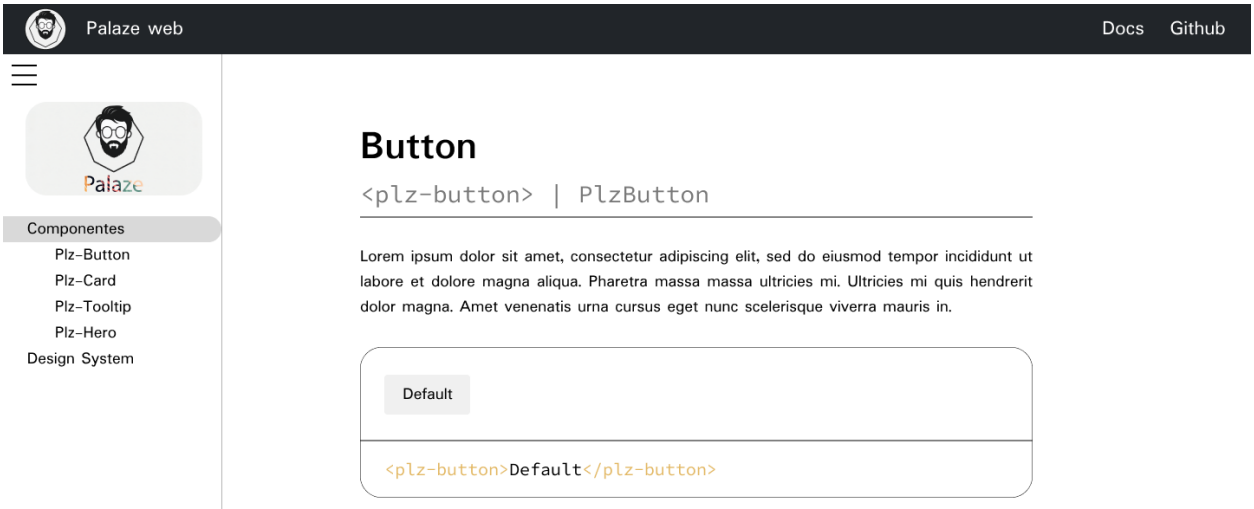
el que poder acceder al inicio de la página y una breve frase descriptiva para finalizar y dejar una buena impresión.

Se ha decidido seguir con una estructura similar a la que tendría la sección hero, pero buscando una paleta de colores menos llamativa.

22-Diseño landing page 5

Se continua con el diseño de la página de documentación, en este caso, no será un diseño final sino una plantilla. Se utilizará esta plantilla para la documentación de cada uno de los componentes. A continuación, se explicará el diseño y las posibles aportaciones.

23-Diseño plantilla docs 1



De un primer vistazo se ubica un header o encabezado que ahora consta de un panel de navegación a la izquierda con un icono junto con el nombre de la web y otro panel de navegación a la derecha, con un enlace a la misma documentación (Docs) y a la página donde se encuentra el código fuente (Github).

También se puede apreciar un sidebar, con distinciones respecto al boceto inicial, ahora se dispone de una imagen con el logo de palaze y el nombre de la librería en la parte superior que también ha sido realizada como archivo vectorial haciendo uso de las herramientas Figma y Photoshop.

24-Icono palaze sección

A continuación de esta imagen se tiene la navegación que no ha sufrido muchos cambios, sigue teniendo la misma estructura.

El botón con forma de acordeón se mantendrá fijo y permitirá esconder y mostrar el sidebar como funcionalidad.

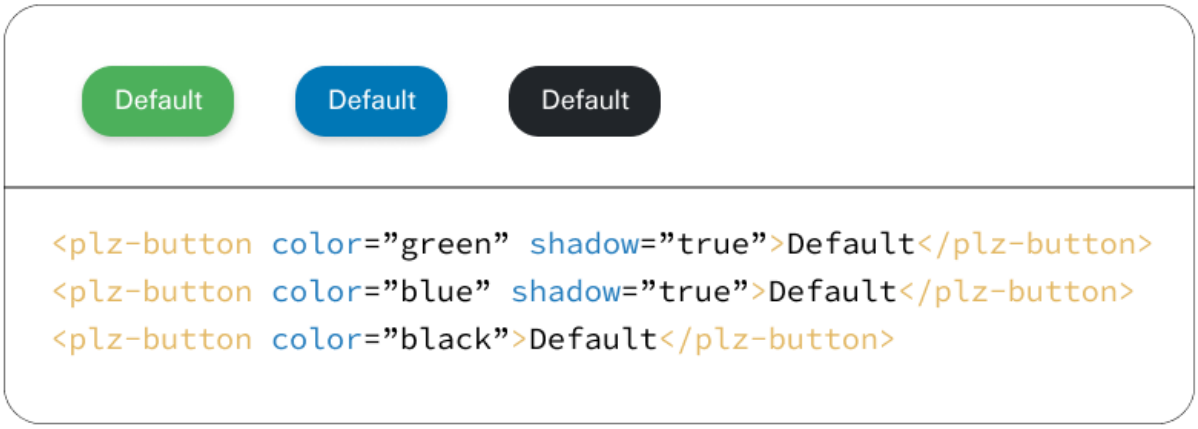
Por otra parte, la sección principal, donde se encontrará el contenido siempre irá introducido por un título con el nombre del componente seguido del nombre de etiqueta y de clase del componente. A su vez, habrá una línea que actuará como separador de contenido para posteriormente disponer de una descripción del componente.

Debajo habrá dos contenedores en el que aparece el componente renderizado y un ejemplo de cómo se ha usado el componente.

Ejemplos

Variantes del componente button

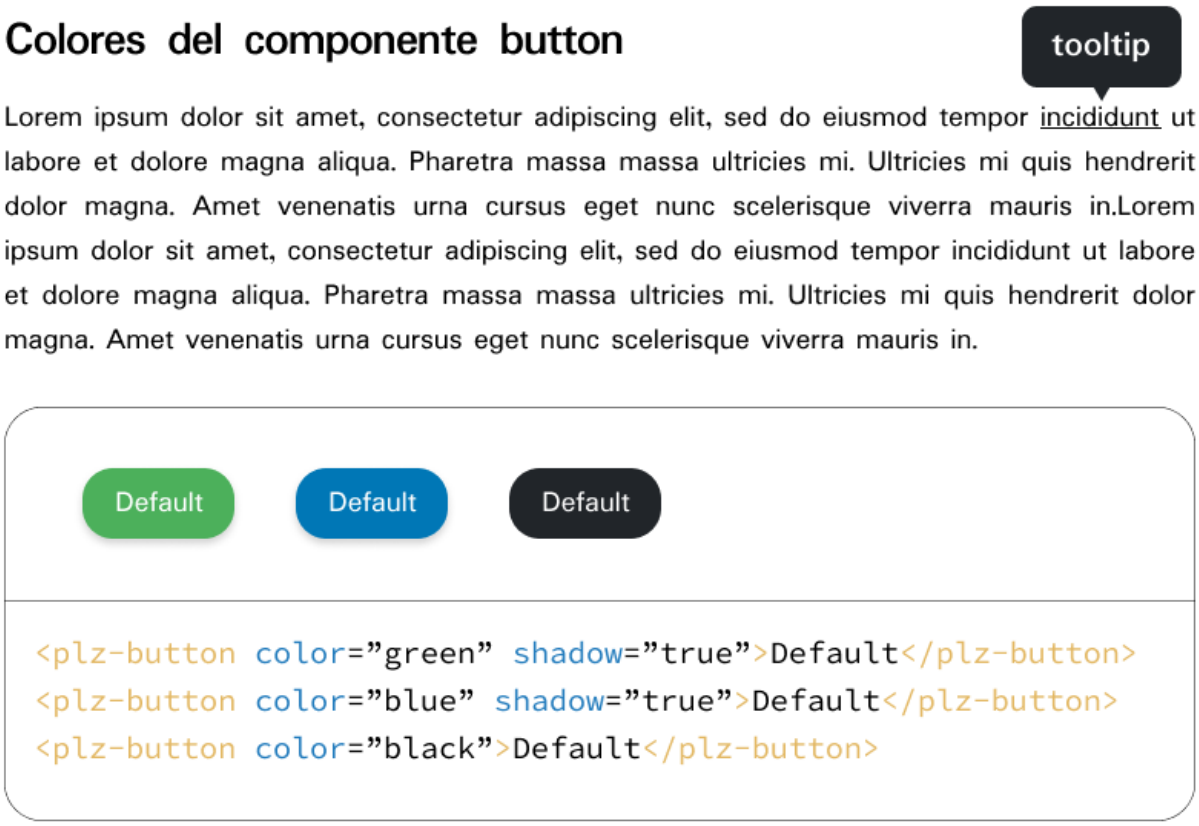
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Pharetra massa massa ultricies mi. Ultricies mi quis hendrerit dolor magna. Amet venenatis urna cursus eget nunc scelerisque viverra mauris in.Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Pharetra massa massa ultricies mi. Ultricies mi quis hendrerit dolor magna. Amet venenatis urna cursus eget nunc scelerisque viverra mauris in.



Siguiendo con el contenido principal, después de la demostración y descripción de la función genérica del componente habrá una sección de ejemplos de uso con las variantes de este renderizadas junto con el código necesario para alcanzar ese resultado.

Habrá tantas secciones como el componente lo requiera y se comentarán cómo usar las características, propiedades y atributos relacionados.

También hay que señalar que cuando exista algún concepto que hace referencia a enlaces externos o internos en la propia documentación, se utilizará un bocadillo emergente (tooltip) como se verá en la siguiente imagen.



Y por último, en forma de resumen, habrá una última sección al final de cada documentación de componente reuniendo los atributos y posibilidades de estos para poder ver de un simple vistazo a modo de tabla las opciones del componente.

4.1.3 Diseño de componentes

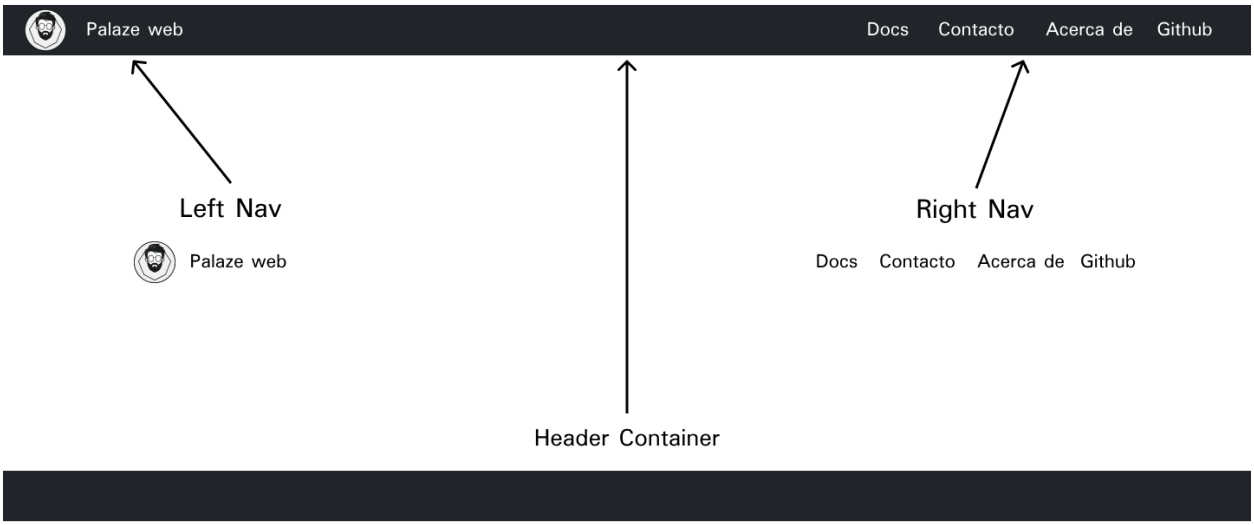
Siguiendo con la estrategia definida anteriormente, el siguiente paso es determinar qué componentes son necesarios a raíz de los diseños tanto de presentación como de documentación.

Debido a que este proyecto sigue una estructura de desarrollo incremental, los primeros componentes que se diseñaron eran prácticamente iguales al diseño anterior sin ninguna variante extra, pero conforme se fueron implementando en forma de código web, se fueron añadiendo más variantes para que pudiera ser más personalizable cuando se usara en otras páginas web.

Todos los detalles técnicos en cuanto a uso del componente, propiedades, atributos, características o funcionalidades del componente estarán explicados en la documentación online.

Se empezará con el primer componente extraído, que es el header o encabezado:

27-Diseño header



Este componente es utilizado en toda la aplicación web palaze. Se puede dividir en un contenedor principal y dos navegaciones, una ubicada en la parte izquierda y otra en la parte derecha.

El siguiente será el componente hero, que tendrá varias variantes.

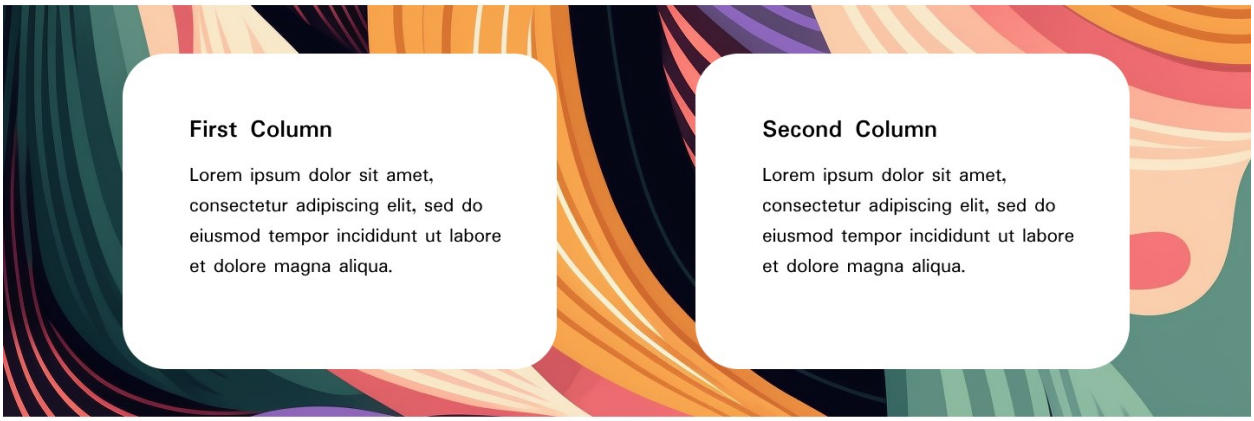
28-Diseño hero default

Esta será la variante por defecto y la que tiene palaze en la página web de presentación, que consta de una imagen de fondo y un contenedor céntrico con contenido alineado al centro.

29-Diseño hero image

Esta variante del componente es una imagen. En algunos casos se puede querer utilizar una imagen con texto incorporado para tu componente y se quiere brindar esta posibilidad.

30-Diseño hero two-column



Por último, está la variante de dos columnas, que sigue una estructura similar a la variante por defecto pero con dos contenedores en el que se podrá dividir el contenido. Esta variante deberá de cambiar de columnas a filas dependiendo de las dimensiones del dispositivo donde se abra la aplicación web.

Se sigue con el componente divider o separador.

31-Diseño divider 1

Custom Height – Custom Color

Custom Form: Solid – Dashed – Dotted

Este componente se utiliza para dividir secciones y se quiere que el usuario pueda elegir su color, tamaño y forma.

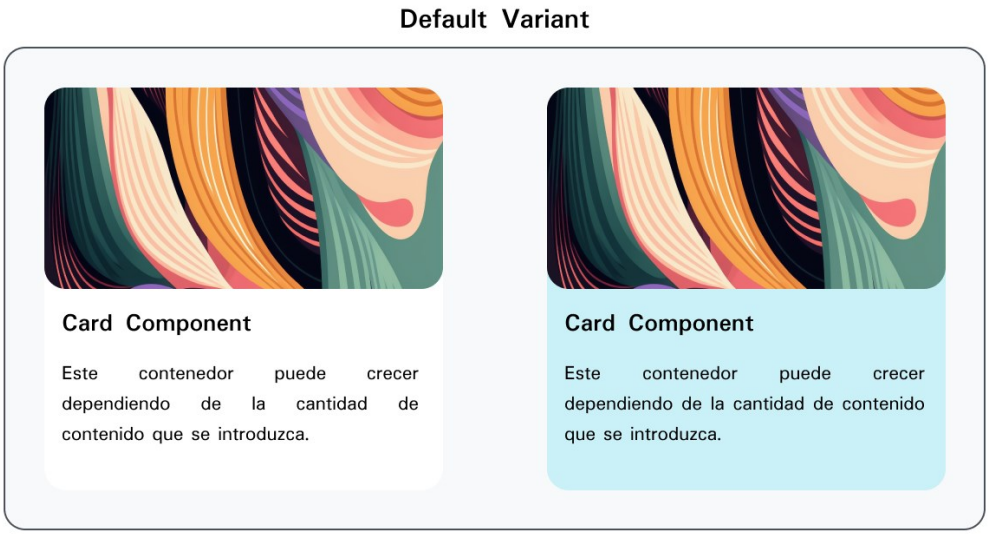
También habrá una opción de incluir un divider de tipo imagen como se usa en la página web de presentación.

32-Diseño divider 2

Custom Image

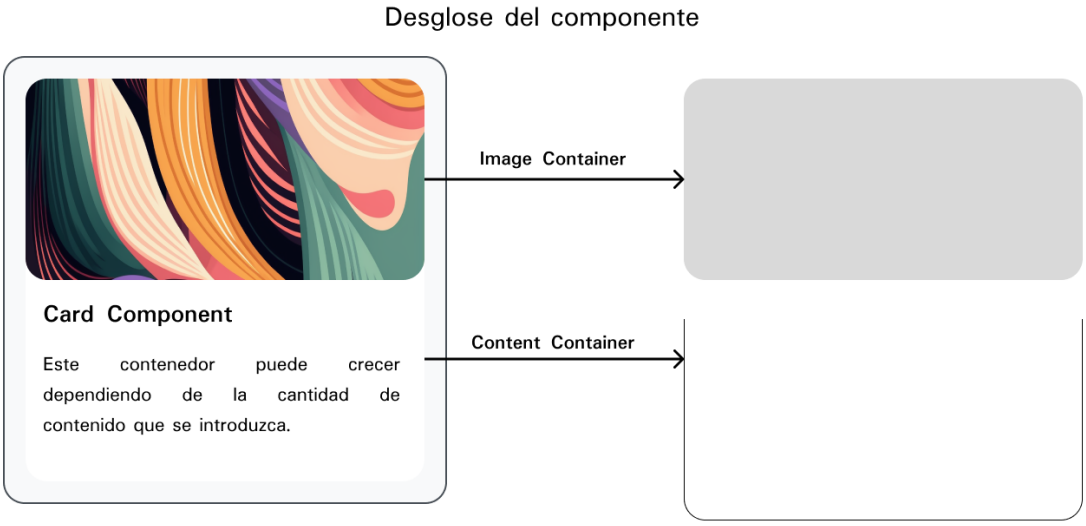
Se continua con un componente card que se ha extraído de la sección de actualizaciones y noticias sobre los componentes. Este componente tendrá las siguientes variantes.

33-Diseño card default



Esta variante será la utilizada por defecto y es la más común dentro del desarrollo web. Consta principalmente de dos contenedores, uno para la imagen y otro para el contenido. También tendrá la funcionalidad de enlace al hacer click sobre el componente y cuando pases el cursor por encima el contenedor de contenido cambiará de color.

34-Diseño card default desglose

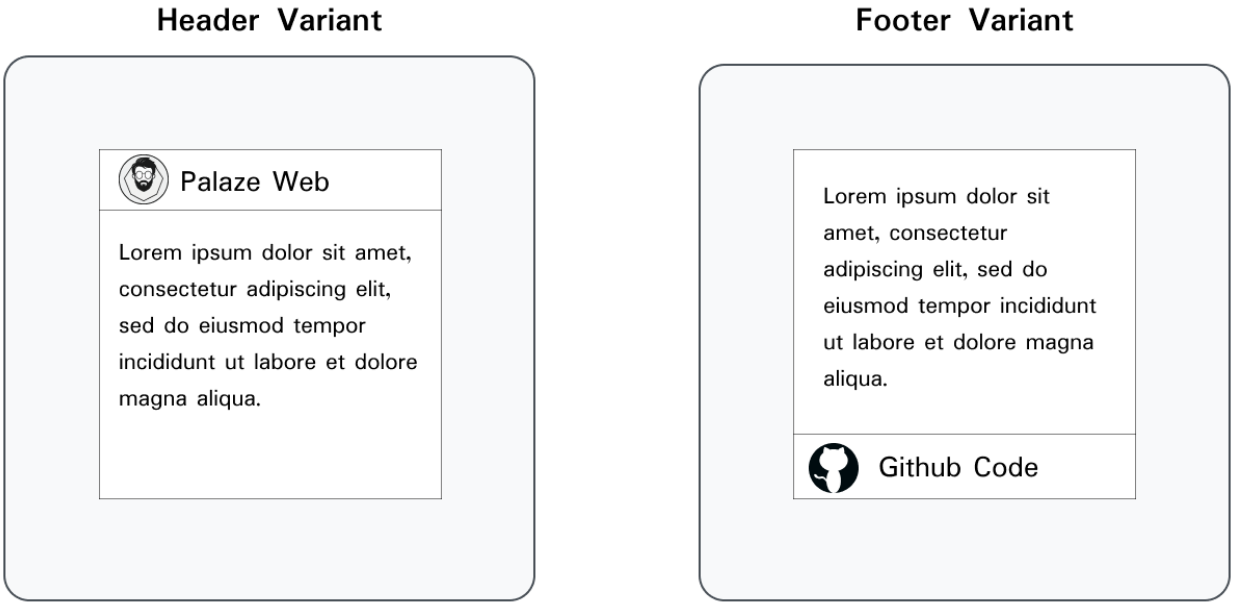


Habrá otra variante simple del componente que consta de un contenedor de contenido con un pequeño borde sombreado para delimitar el tamaño. Esta variable será personalizable por el usuario:

35-Diseño card simple

A continuación, habrá dos variantes, header y footer, ambas están divididas como su nombre bien indica y al igual que la variante simple, son muy personalizables y las delimita un borde sombreado.

36-Diseño card header-footer

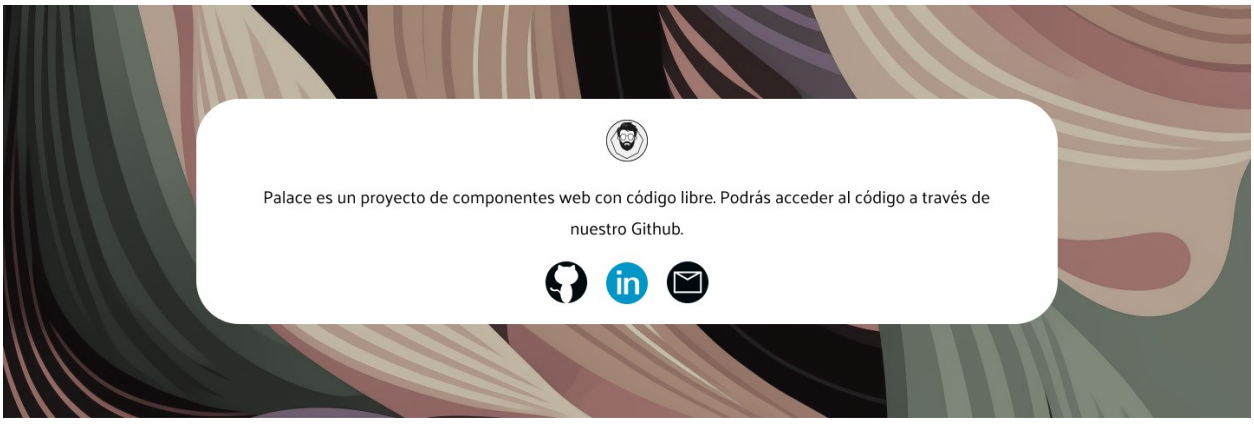


La última variante de este componente se llama imagen y consta de un contenedor con una imagen y un texto por encima como en la imagen siguiente.

37-Diseño card image

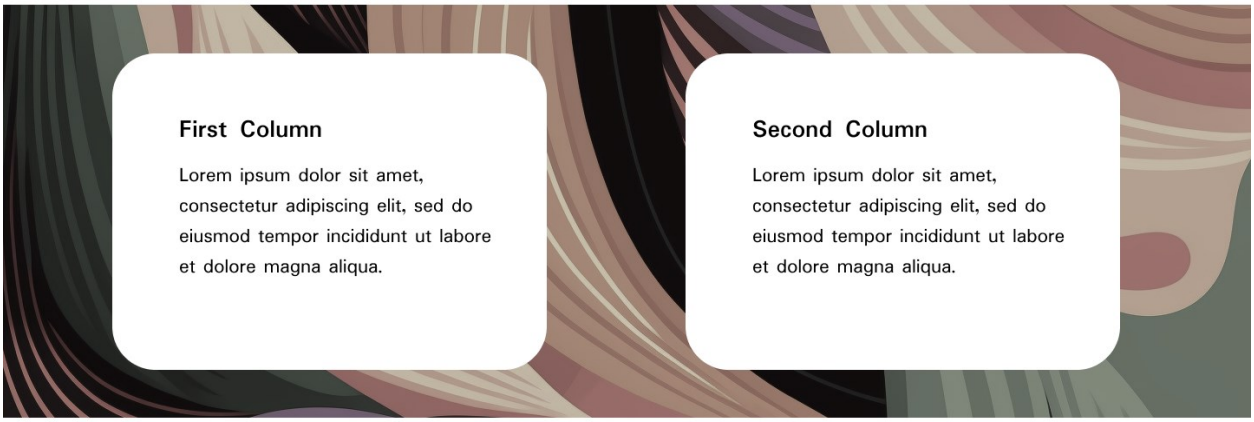
El último componente de la página de presentación será el footer y tendrá dos variantes. La variante block que consta de una imagen de fondo y un bloque de contenido con los elementos alineados al centro.

38-Diseño footer block



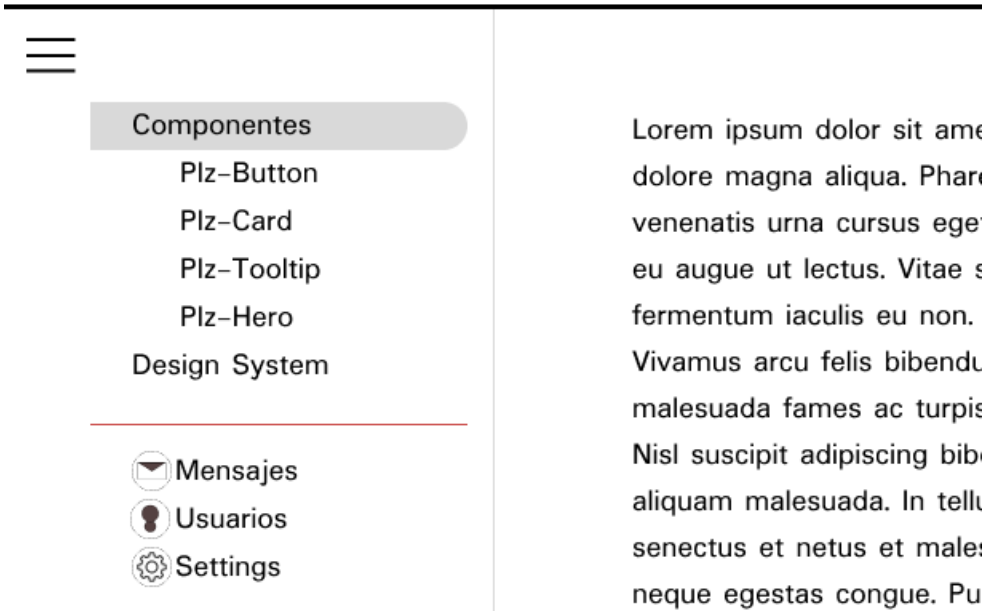
Y la variante de dos columnas en la que se identifica una imagen de fondo y dos bloques de contenido, que al igual que la variante del componente hero con dos columnas, deberá de cambiar de columnas a filas dependiendo de las dimensiones del dispositivo donde se abra la aplicación web.

39-Diseño footer two-column



Siguiendo con la extracción de componentes del diseño se identifica al componente sidebar.

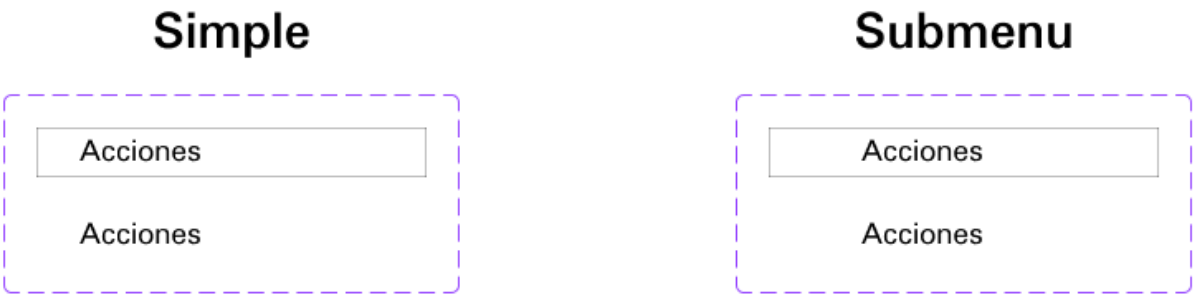
40-Diseño sidebar



Se ha decidido que haya dos componentes más que en conjunto con la funcionalidad del sidebar forman al componente.

Primero se tendría el componente menu-item del que se identificaron varias variantes, que sería cada sección, por ejemplo, mensajes, otro ejemplo podría ser Plz-button, etc.

41-Diseño menu-item 1



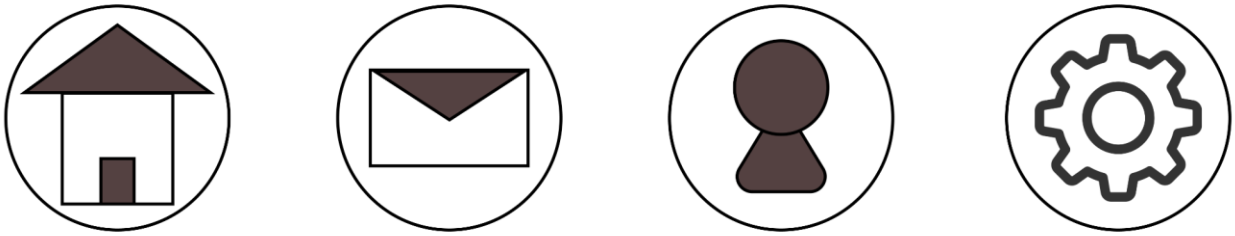
Se tiene la característica de submenú en la que se indenta un poco más que la variante normal. También se dispone de menu-item con borde o sin borde.

42-Diseño menu-item 2

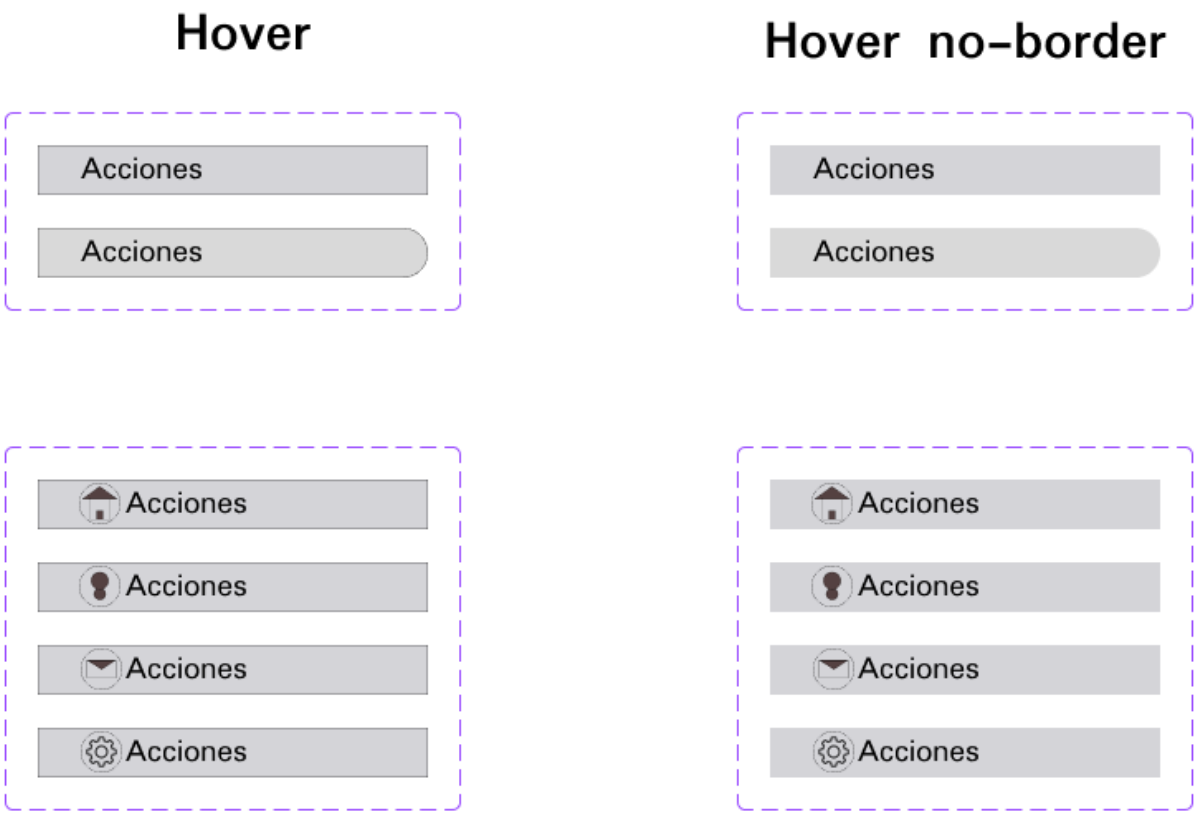
Por otra parte, existe un menu-item con iconos, se dejará al desarrollador que utilice la librería seleccionar una imagen para el icono del componente. Y como en el ejemplo anterior, existen variantes con borde y sin borde.

Estos iconos han sido creados como archivo vectorial haciendo uso de la herramienta Figma.

43-Iconos menu-item



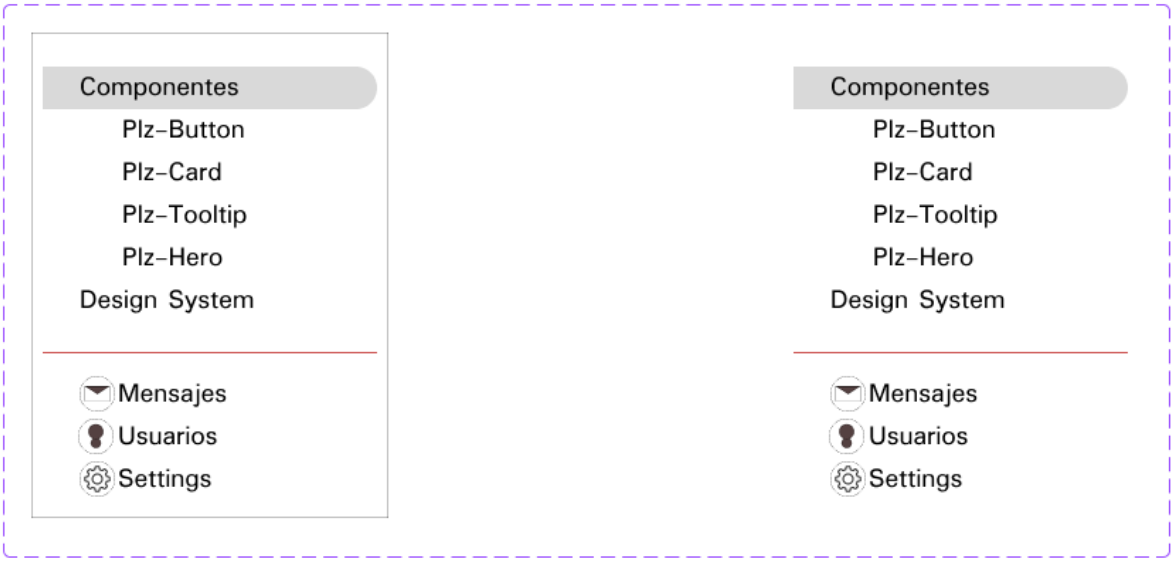
44-Diseño menu-item 3



Por último, existe una variante de menu-item con el final redondo, como si acabara con un semi-circulo en vez de terminar con un cuadrado. También hay versiones con hover y borde y las versiones de hover sin borde.

El otro componente que se utilizará para el sidebar es el componente menu. Este componente agrupará componentes menú-item como se muestra a continuación.

45-Diseño menu

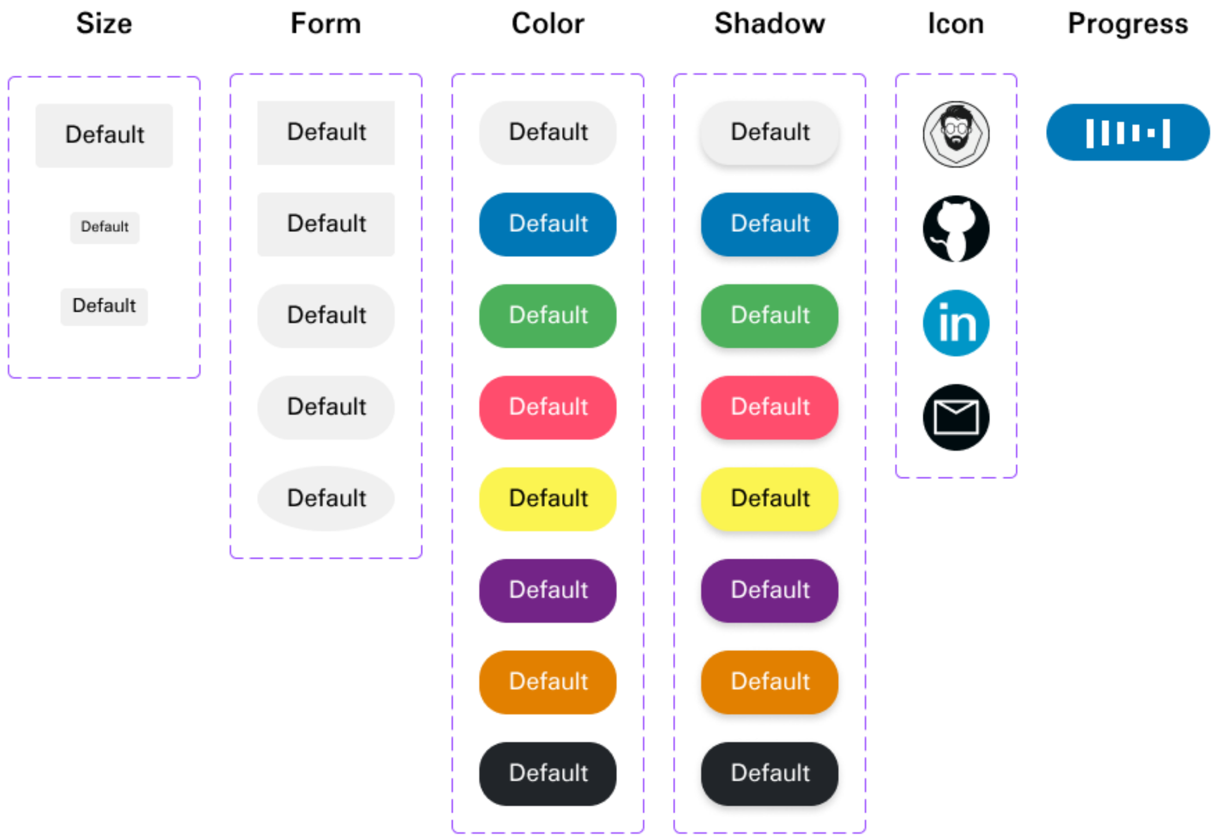


Existen dos versiones, una con borde para delimitar el menú y otra sin borde.

Tanto el componente menu como el componente menu-item se pueden usar separados del componente sidebar para desempeñar funciones en otros casos concretos.

Otro componente será el botón, en el que se dispondrá de una gama de colores predefinidos.

46-Diseño button



Como se puede observar se podrá elegir el tamaño de botón, la forma de su contorno, el color y si quieres que tenga sombra de entre los colores mostrados.

También dispondrás de un botón de progreso en el que habrá una animación de carga que simula una ola y se podrá elegir el color del botón y el de la ola fuera de los predefinidos.

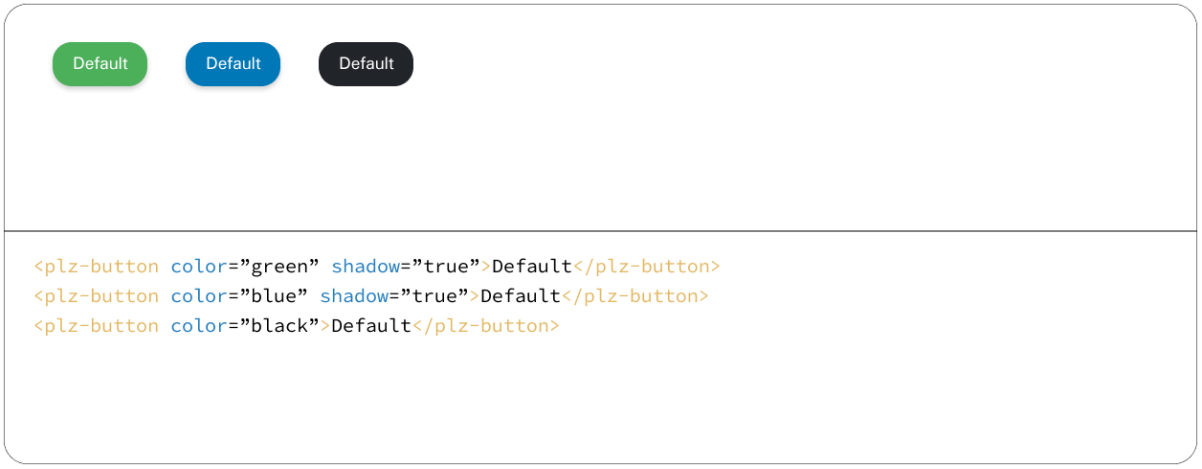
Por último, habrá una variante icono en la que habrá cuatro iconos predefinidos, pero se dará la opción de utilizar una imagen propia para personalizar todavía más el botón.

Los tres iconos, el de linkedin, github y mail, se han realizado mediante las herramientas Photoshop y Figma, como el icono del loco de palaze que se ha mostrado anteriormente.

47-Iconos button

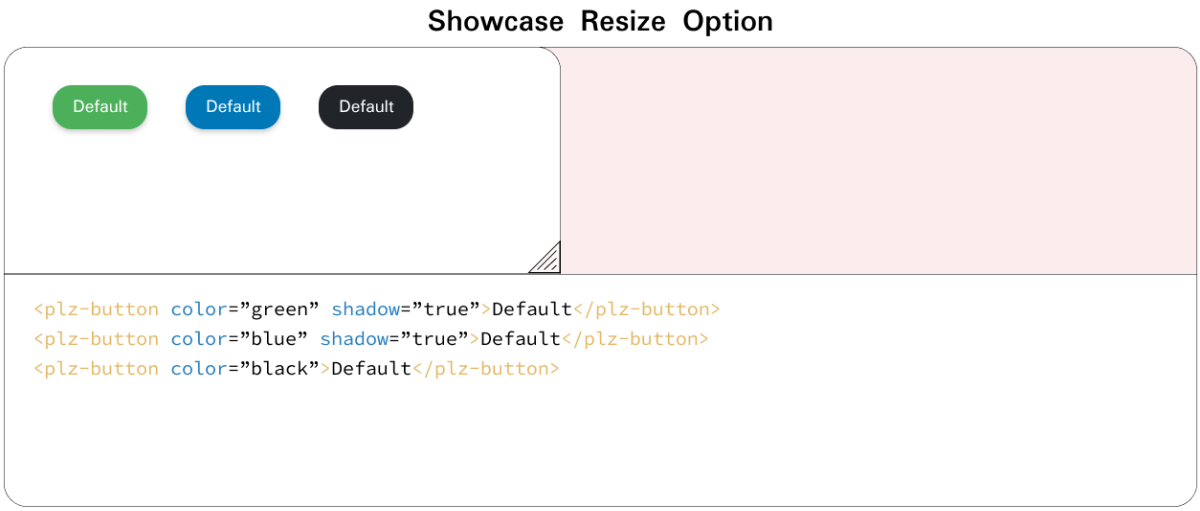
El siguiente componente se llama showcase y tendrá dos contenedores, uno para renderizar un ejemplo de componente y otro para mostrar el código utilizado para renderizarlo. La idea sería que el código se interpretará y se le diera formato y color, pero esta peculiaridad se analizará en el punto 4.2.1 Estudio de viabilidad de diseño.

48-Diseño showcase 1



Este componente tendrá una opción para cambiar el tamaño del contenedor horizontalmente redimensionando como se muestra a continuación:

49-Diseño showcase 2

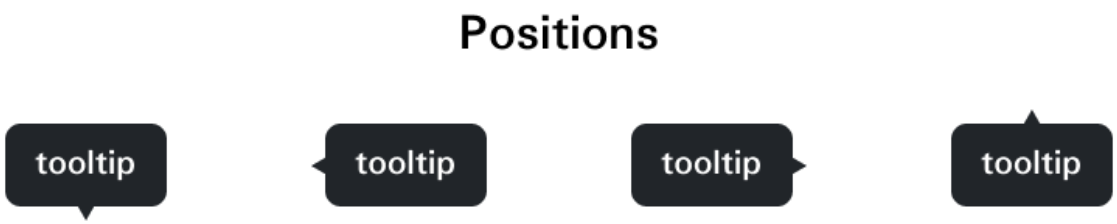


A partir de la necesidad de hacer un componente botón con variante progress, se decidió diseñar un componente spinner con distintas variantes:

50-Diseño spinner

Seguimos con el componente tooltip, que ayudará a añadir información mediante un bocadillo emergente.

51-Diseño tooltip



Se dispone de cuatro posiciones:

- Superior
- Inferior
- Derecha
- Izquierda

También habrá dos variantes, una con función link que al hacer click accedes a un enlace de referencia o la variante con únicamente texto.

El último componente diseñado no aparece en el diseño de página de la documentación ni en el de la presentación, surgió más adelante a partir de la necesidad de mostrar visualmente los tokens de colores asociados a colores del Design System. Al ver que se repite siete veces en una misma sección, se decidió diseñar un componente.

52-Diseño colors

Design System Colors

4.1.4 Tokens de diseño

Siguiendo con los pasos establecidos en la estrategia, una vez extraídos los componentes a desarrollar de los diseños principales, se deberá identificar y definir todas las características comunes aplicadas a el diseño de aplicación web y componentes. Estos tokens serán utilizados posteriormente a la hora de desarrollar los componentes.

Se representarán mediante CSS properties. Para ver una representación gráfica de estos tokens de diseño se puede acceder al apartado de Design System de la documentación online.

La primera característica que se convertirá en tokens de diseño son los colores. En el diseño se decidió adoptar por paletas cromáticas de máximo cinco gammas de un mismo color. Para seleccionar estos colores, se enfocó a colores primarios y luego se fue modificando el color hasta obtener cinco gamas de ese mismo.

Los colores son negro, rojo, amarillo, verde, azul, naranja, morado:

1-Colores

Negro		Rojo		Amarillo	
--plz-color-black-1	black	--plz-color-red-1	#A4133C	--plz-color-yellow-1	#b88900
--plz-color-black-2	#212529	--plz-color-red-2	#FF4D6D	--plz-color-yellow-2	#d3a007
--plz-color-black-3	#495057	--plz-color-red-3	#e67a8d	--plz-color-yellow-3	#fbf451
--plz-color-black-4	#d4d4d8	--plz-color-red-4	#FFB3C1	--plz-color-yellow-4	#ffff7d
--plz-color-black-5	#F8F9FA	--plz-color-red-5	#FFF0F3	--plz-color-yellow-5	#ffffdb
Verde		**Azul**		**Naranja**	
--plz-color-green-1	#081C15	--plz-color-blue-1	#023E8A	--plz-color-orange-1	#ee4a00
--plz-color-green-2	#1B4332	--plz-color-blue-2	#0077B6	--plz-color-orange-2	#e28000
--plz-color-green-3	#4cb05b	--plz-color-blue-3	#0096C7	--plz-color-orange-3	#f9a72f
--plz-color-green-4	#52B788	--plz-color-blue-4	#a7b3e9	--plz-color-orange-4	#fbbd5e
--plz-color-green-5	#D8F3DC	--plz-color-blue-5	#CAF0F8	--plz-color-orange-5	#fde9b0
Morado					
--plz-color-purple-1	#340856				
--plz-color-purple-2	#5d1372				
--plz-color-purple-3	#732487				
--plz-color-purple-4	#a167c9				
--plz-color-purple-5	#fff0ff				

Otro token será la tipografía, en las que se eligen dos fuentes, la principal será Padauk y la de respaldo será sans-serif.

2-Fuente

Fuente	
--plz-font-primary	Padauk, sans-serif

Se ha elegido la fuente Paduk como principal por varias razones:

- Es una fuente diseñada para ser legible en pantallas digitales y tiene mucha claridad y distinción de letras.
- La estética de esta fuente es moderna y limpia, con formas redondeadas y suaves que proporciona una lectura amena y atractiva.
- No es una fuente típica utilizada en todas las páginas web ni dispositivos.

En cuanto a la fuente sans-serif se ha elegido como respaldo porque:

- Es una fuente ampliamente utilizada y compatible con todos los dispositivos.
- Su estética es simple y limpia

Se continua con los tokens para el tamaño de fuente. Para elegir estos tokens, se ha revisado materiales de diseño de otros frameworks y después se han ido probando con textos de ejemplo hasta determinar qué variedad era más adecuada para la web.

3-Tamaño de fuente

Tamaño de fuente	
--plz-font-size-very-small	12px
--plz-font-size-small	14px
--plz-font-size-normal	16px
--plz-font-size-sub-section	20px
--plz-font-size-mid-section	30px
--plz-font-size-below-titles	24px
--plz-font-size-section	48px
--plz-font-size-titles	64px

Para el grosor de la fuente, se ha decidido mantener la simplicidad y utilizar únicamente los estilos regular y negrita.

4-Grosor de fuente

Grosor de fuente	
--plz-font-weight-regular	400
--plz-font-weight-bold	700

Se ha visto que es relevante crear tokens para el espacio entre líneas, ya que cuando se utilizan tamaños de letras grandes, la separación entre líneas puede ser excesiva.

5-Espacio entre línea

Espacio entre línea	
--plz-line-height-smaller	1
--plz-line-height-small	1.4
--plz-line-height-normal	1.8
--plz-line-height-big	2.4

A su vez, por la misma razón que los tokens de espacio entre líneas, puede llegar a suceder con espacio entre letras por lo que se han definido los tokens siguientes:

6-Espacio entre letras

Espacio entre letras	
--plz-letter-space-smaller	-0.48px
--plz-letter-space-small	-0.24px
--plz-letter-space-normal	normal
--plz-letter-space-big	1.2px
--plz-letter-space-bigger	2.4px

Por último, se han definido tokens de formas de bloques que serán aplicados a los elementes de la página como componentes botón, tarjeta, etc. Se ha elegido una amplia variedad empezando desde una forma ovalada o circular hasta una forma de cuadrado, cubriendo así el uso más popular de formas de elementos.

7-Formas de bloques

Formas de bloques	
--plz-border-radius-circle	50%
--plz-border-radius-rounded	9999px
--plz-border-radius-low-rounded	25px
--plz-border-radius-default	4px
--plz-border-radius-square	0px

Estos tokens se utilizarán durante todo el proyecto tanto para la aplicación web como para el desarrollo de la librería de componentes cuando sea necesario.

4.2 Palaze Web Components

4.2.1 Estudio de viabilidad de diseño

En esta sección se revisan todos los diseños de la aplicación web y los componentes extraídos de esta, haciendo hincapié en cómo enfocar el desarrollo de cada componente de forma general y si es necesario realizar modificaciones en la apariencia de estos para poder utilizarlos. También se piensa en la funcionalidad que tendrán los componentes.

En general la viabilidad de los diseños para el desarrollo es correcta, no hay ninguna locura de diseño que no se pueda desarrollar, de hecho, se añadirán más características y funcionalidades siguiendo el desarrollo incremental para que los componentes sean personalizables.

Empezando por el componente button, se seguirá el diseño tal y como se muestra y se añadirá la funcionalidad de link, con la que dependiendo de cómo se utilice el componente, podrá usarse como un botón de acción o un botón de referencia con el que acceder a un enlace.

También se añadirá la posibilidad de incluir cualquier imagen como variante icono a parte de las cuatro propuestas por defecto que serán palaze, github, linkedin y mail.

En cuanto al componente card, en la variante por defecto, se añadirán varias opciones de color para el hover (pasar cursor por encima del elemento) para que sea más personalizable.

Tampoco se limitará el contenido del componente a solo texto, si se quiere introducir un componente botón seguido de un texto y un componente divider, se podrá hacer.

En el componente colors se añadirá una variante custom para que se puedan pasar por atributo cinco colores para crear tu propia paleta de colores.

El componente divider podrá ajustar su espacio respecto otros elementos mediante un atributo del componente, pudiendo así delimitar mejor las separaciones.

Se añadirá una característica para que el componente menu-dropdown pueda mantenerse en un estado de despliegue constante.

El componente spinner será más personalizable, se podrán añadir los colores que se desee para que encaje con el estilo de la página web en el que se utilice. También debido a la complejidad de representar patrones de movimiento con animaciones CSS, el aspecto final de las variantes de spinner no será igual al diseño, pero sí muy similar.

Referente al componente showcase, se ha decidido dividir el componente debido a ciertos factores. Se dejará como componente padre y contenedor de dos componentes hijos al componente showcase. Los dos contenedores que aparecen en el diseño con los componentes renderizados y el código de muestra se han divido en dos:

- Componente showcase-render: Se añadirán características para poder ajustar el contenido dentro del propio componente, márgenes, display, etc.
- Componente showcase-code: Será donde se encuentre el código con el que se ha renderizado los ejemplos en el componente showcase-render. Aquí se plantea hacer uso de dos librerías con peso mínimo para poder hacer realidad el diseño del componente.

  Se pedía que el componente interpretará el código como lenguaje HTML intentándolo y dándole formato. También se pedía que se identificaran los elementos que eran etiquetas, atributos y texto plano para que cambiara de color de forma automática. Para ello se incluirán dos librerías:

  ➔ Prettier: Librería muy popular que permitirá aplicar formato HTML al texto dentro del componente. Se puede configurar para ajustarlo al estilo de página palaze.

  ➔ Hightlight.js: Librería que permitirá interpretar el texto con sintaxis html como lenguaje html y cambiar el estilo que se apreciará visualmente en el resultado final (color, letra, etc). Permite aplicar distintos temas de colores por lo que habrá que mirar cuál es más adecuado según el Design System del proyecto.

- Por último, se ha visto conveniente crear un nuevo componente que permite la distribución de elementos dentro de él en filas y en columnas. Se ha decidido llamarlo componente section y permitirá agrupar contenido de una forma determinada.

Hay componentes que no se han nombrado porque no se ha considerado recalcar ningún problema ni modificación aparente. De todas formas, se puede consultar el resultado final donde se explica toda característica y funcionalidad relevante de los componentes en la documentación web a partir de este enlace.

4.2.2 Listado de componentes

El listado final de componentes que habrá contenidos dentro de la librería será de un total de quince.

Para acceder a una documentación completa acerca de todas las funcionalidades, características, propiedades y atributos, a su vez, de ejemplos de renderizado y código, se deberá acceder a la página de documentación palaze a través de este enlace.

Listado de componentes de la librería palaze:

1. Button | plz-button
2. Card | plz-card
3. Colors | plz-colors
4. Divider | plz-divider
5. Footer | plz-footer
6. Header | plz-header
7. Hero | plz-hero
8. Menu | plz-menu
9. Menu-dropdown | plz-menu-dropdown
10. Menu-item | plz-menu-item
11. Section | plz-section
12. Showcase | plz-showcase
13. Sidebar | plz-sidebar
14. Spinner | plz-spinner
15. Tooltip | plz-tooltip

4.2.3 Detalles de implementación a partir del diseño

Antes de empezar esta sección, hay que recalcar que el proyecto es muy extenso, ha habido muchos procesos que formaban parte de la construcción del proyecto y cada uno podría extenderse ampliamente, por ello se ha decidido reducir las explicaciones cogiendo como ejemplo, uno de los quince componentes que forman la librería. A su vez, se ha decidido dejar la explicación del código de la aplicación web a un lado únicamente recordando que se ha utilizado el entorno de desarrollo proporcionado por StencilJS y que la propia aplicación web es un ejemplo de uso de los componentes, haciendo uso de estos con sus etiquetas como elementos HTML normales, pero con las funcionalidades y características propias de la librería palaze.

4.2.3.1 Plz-button

El primer paso a realizar antes de empezar a desarrollar el componente plz-button es analizar su diseño, este paso ya se ha realizado anteriormente en el estudio de viabilidad. Pasado este paso se identificarán los distintos usos del componente, sus características y funcionalidades.

Primero se divide el botón en función de si se utiliza como un elemento en el que al hacer click sobre y acceder a un enlace de referencia o si se utiliza para otro tipo de acciones que definirá el desarrollador que los utilice.

Posteriormente se identifican las variantes que tendrá el componente, en este caso, según el diseño habrá tres variantes, la variante por defecto, la variante icono y la variante progreso.

En la variante por defecto, se deberán definir propiedades de color, hover, sombra, tamaño y forma que posteriormente serán atributos con los que modificar el componente.

En la variante icono, se tendrá que poder elegir entre las cuatro imágenes definidas en el diseño (palaze, github, linkedin, mail). También se deberá poder cambiar el tamaño del icono y tendrá que haber una forma de introducir una imagen personalizada mediante un atributo.

Por último, está la variante progress que tendrá dos estados, en progreso y botón normal, que cambiará su estado en base a una propiedad. Esta variante no tendrá funcionalidad de link, ya que su función es tener una acción.

Una vez concretado todos estos aspectos hay que dirigirse al código, concretamente al archivo plz-button.tsx en el que estará la lógica y renderizado del componente.

En este archivo lo primero que hay que hacer es importar las dependencias:

```
import { Component, h, Host, Prop } from '@stencil/core';
import Github from "../../assets/icon/Icon-github.svg";
import Linkedin from "../../assets/icon/Icon-linkedin.svg";
import Mail from "../../assets/icon/Icon-mail.svg";
import Palaze from "../../assets/icon/Icon-palaze.svg";
```

El primer import son dependencias necesarias de Stencil para poder crear los componentes desde su framework. Los posteriores import son las imágenes de tipo vectorial que se usarán para la variante icono como se ha explicado anteriormente.

Deben definirse como importaciones debido a que cuando los componentes se distribuyan, se pretende que, al modularizarse, se pueda acceder a estas imágenes, cosa que no pasaría si simplemente se indicara la ruta donde se encuentran a la hora de necesitarlas en el renderizado.

Posteriormente habrá que definir el decorador del componente.

```
@Component({
  tag: 'plz-button',
  styleUrl: 'plz-button.css',
  shadow: true,
})
```

Se definirá un tag para el componente con el nombre que se desee ponerle, en este caso se utilizara plz-button que es la abreviación de palaze-button. Se definirá un fichero para los estilos y en este caso se define el estándar de shadow DOM que permitirá encapsular el código JavaScript y código CSS en su propio árbol DOM haciendo que no interactúe con el DOM principal de la web donde se utilice el componente.

A continuación, se definen las propiedades del componente extraídas del estudio anterior:

```
export class PlzButton {
 @Prop() variant?: 'default' | 'icon' | 'progress' = 'default';
//Variantes de botón
  @Prop() size?: 'default' | 'small' | 'big' = 'default';
//Tamaño del botón
  @Prop() color?: 'default' | 'black' | 'purple' | 'blue' | 'green' |
'red' | 'orange' | 'yellow' = 'default';
```

```
//Colores del botón default y progress
  @Prop() corners?: 'default' | 'low-rounded' | 'rounded' | 'square' |
'circle' = 'default'; //Forma
  @Prop() shadowColor?: 'black' | 'purple' | 'blue' | 'green' | 'red' |
'orange' | 'yellow' | '' = '';
//Color de sombra de componente
  @Prop() icon?: string = ''; //nombre de icono o url del icono custom
  @Prop({ mutable: true }) active?: boolean = false;
//Boolean que determina si la variante progress está activada
  @Prop() link?: string = '';
//Url a la que se accede desde el botón
 @Prop() target?: '_self' | '_blank' | '_parent' | '_top' | 'framename'
= '_self'; //Forma de acceder a la url
  @Prop() hoverColor?: string = ''; //Color del hover
  @Prop() colorProgress?: string = 'white';
//Color del spinner de carga de la variante progress
```

En todas las propiedades se ha considerado optar por limitar las opciones debido al control de errores y seleccionar un valor por defecto en la gran mayoría de propiedades.

- **variant (option):** Variantes del componente.
  - default
  - icon
  - progress
- **size (option):** Tamaños del componente.
  - default
  - small
  - big
- **color (option):** Colores del componente.
  - default
  - black
  - purple
  - blue
  - green
  - red
  - orange
  - yellow
- **corners (option):** Forma del componente.

    - default
    - low-rounded
    - rounded
    - square
    - circle
- **shadowColor (option):** Color de la sombra del componente.
    - default
    - black
    - purple
    - blue
    - green
    - red
    - orange
    - yellow
- **icon (string):** nombre o ruta de la imagen en la variante icon.
- **active (boolean):** Determina si la variante progress está en proceso o no.
  Se le añade la opción a la propiedad mutable:true ya que permite modificar la propiedad desde fuera del componente. Se utilizará en junto con funciones web de delay hasta recibir respuesta desde la web por los desarrolladores.
    - false
- **link (string):** Url de referencia a la que se quiere acceder.
- **target (option):** Forma de acceder a la url de referencia.
    - _self
    - _blank
    - _parent
    - _top
    - framename
- **hoverColor (string):** Colores al pasar el cursor por encima del componente.
    - Cualquiera color
- **colorProgress (string):** Color del spinner de carga de la variante progress.
    - Cualquier color

Una vez definidas las propiedades hay que dirigirse a la hoja de estilos en cascada (fichero css) para definir los estilos que se aplicarán durante el renderizado de las distintas variantes del componente.

Primero se definen las custom properties dentro de la clase :host{}

```
:host {
  /* Peso fuente */
  --plz-font-weight-regular: 400;
  /* Tamaño fuente */
  --plz-font-size-small: 14px;
  --plz-font-size-normal: 16px;
  --plz-font-size-sub-section: 20px;
  /* Altura de línea */
  --plz-line-height-normal: 1.8;
  /* Radio borde */
  --plz-border-radius-circle: 50%;
  --plz-border-radius-rounded: 9999px;
  --plz-border-radius-low-rounded: 20px;
  --plz-border-radius-default: 4px;
  --plz-border-radius-square: 0px;
  /*    ## COLORES ##   */
  /* Primarios-gamma-azul */
  --plz-color-blue-2: #0077B6;
  /* Primarios-gamma-verde */
  --plz-color-green-3: #4cb05b;
  /* Primarios-gamma-rojo  */
  --plz-color-red-2: #FF4D6D;
  /* Primarios-gamma-yellow */
  --plz-color-yellow-3: #fbf451;
  /* Primarios-gamma-purple */
  --plz-color-purple-3: #732487;
  /* Primarios-gamma-orange */
  --plz-color-orange-2: #e28000;
  /* Primarios-gamma-negro */
  --plz-color-black-2: #212529;
  /* Primarios-gamma-blanco */
  --plz-color-white: white;

  --wave-color: var(--plz-color-white);
  --bg-color-hover: var(--plz-color-black-4);
}
```

Estas custom properties han sido obtenidas a partir de los tokens de diseño definidos anteriormente. Las últimas custom properties serán utilizadas para cambiar el color dependiendo del contenido que se reciba desde los atributos al llamar al componente. Esto se explicará más adelante cuando se hable del renderizado.

```
/* Quitamos estilo al link */
a {
  text-decoration: none;
  color: inherit;
}

/* Estilo único variante icon */
:host([variant=icon]) {
  display: inline-block;
  height: 50px;
  width: 50px;
}

/* Tamaño único variante icon */
:host([variant=icon][size=small]) {
  height: 40px;
  width: 40px;
}

/* Tamaño único variante icon */
:host([variant=icon][size=big]) {
  height: 60px;
  width: 60px;
}

/* Estilo único variante default */
:host([variant=default]) {
  display: inline-block;
}
```

En la etiqueta a{} se quitarán algunos estilos por defecto como el color azul, el subrayado y cambiaremos el color del texto al color heredado por la página web donde se utilice.

Posteriormente se definirán los tamaños para los distintos tamaños de variante icon, solamente aplicandose si se cumplen las condiciones que aparecen entre paréntesis. También se aplicará el display: inline-block a la variante default para que el componente pueda ir antes, entre o después de otros elementos web si se desea.

También se aplicará el display: inline-block a la variante default para que el componente pueda ir antes, entre o después de otros elementos web si se desea.

A continuación, se aplica un estilo general a la etiqueta button definiendo el tipo de display, alineando el contenido al centro y añadiendo un margen interno del elemento. A su vez se eliminará el borde sobre el contenedor, retiraremos la opción de que el usuario pueda seleccionar el contenido de dentro del botón y se hará que al pasar el cursor por encima cambie a una mano, dando así a entender que se puede hacer click sobre el componente.

```
/* Estilo general para los botones */
button {
  display: inline-block;
  text-align: center;
  padding: 8px 20px;
  cursor: pointer;
  border: none;
  user-select: none;
}
```

Se utilizará en este caso la custom property comentada anteriormente para definir el color hover al pasar el cursor por encima del componente:

```
/* Color del hover para las variantes que lo permitan */
.hover-color:hover {
  background-color: var(--bg-color-hover);
}
```

También se definirá la característica del color del componente mediante esta estructura de código aplicada a cada color propuesto. Se cambiará el color de fondo y el color de la letra del componente. Ejemplo del color rojo:

```
.red {
  background-color: var(--plz-color-red-2);
  color: var(--plz-color-white);
}
```

Igual que se ha definido el color, se definirá la sombra del componente con esta estructura aplicada a cada color propuesto. Se cambiará al componente añadiendo un borde del color elegido y haciendo uso de propiedades css y de box-shadow ajustaremos el radio y densidad de la sombra aplicada sobre el componente. Ejemplo del color rojo:

```
.shadow-red {
  border: 1px solid var(--plz-color-red-2);
  -webkit-box-shadow: 5px 5px 15px -5px rgba(164, 19, 60, 1);
  -moz-box-shadow: 5px 5px 15px -5px rgba(164, 19, 60, 1);
  box-shadow: 5px 5px 15px -5px rgba(164, 19, 60, 1);
}
```

Se definen las clases asociadas al tamaño del componente mediante el siguiente código:

```
/* ## SIZE ## */
/* Distintos tamaños predefinidos de botones */

.default-size {
  font-size: var(--plz-font-size-normal);
  font-weight: var(--plz-font-weight-regular);
  line-height: var(--plz-line-height-normal);
}

.small-size {
  font-size: var(--plz-font-size-very-small);
  font-weight: var(--plz-font-weight-regular);
  line-height: var(--plz-line-height-normal);
}

.big-size {
  font-size: var(--plz-font-size-sub-section);
  font-weight: var(--plz-font-weight-regular);
  line-height: var(--plz-line-height-normal);
}
```

Para determinar el tamaño se cambiará el tamaño de letra y la altura de línea de este.

También se definirá la forma del botón modificando la forma de las esquinas del componente haciendo uso de border-radius y las css properties definidas en el Design System.

```
/* ## BORDER ## */
/* Distintas formas predefinidas del botón */

.default-border {
  border-radius: var(--plz-border-radius-default);
}

.rounded-border {
  border-radius: var(--plz-border-radius-rounded);
}

.low-rounded-border {
  border-radius: var(--plz-border-radius-low-rounded);
}

.square-border {
  border-radius: var(--plz-border-radius-square);
}

.circle-border {
  border-radius: var(--plz-border-radius-circle);
}
```

Para dar forma a la variante icon, se aplicarán las siguientes modificaciones sobre el componente:

```
/* Variante icon forma y formato */
.icon {
  display: inline-flex;
  align-items: center;
  justify-content: center;
  margin: 0;
  padding: 0;
  border: none;
  background-color: var(--plz-color-white);
  border-radius: var(--plz-border-radius-circle);
}
```

Se utilizará un display flex, ya que hace más sencillo el uso del componente con imágenes. Se cambiará la forma del componente a la de un círculo, se quitarán los bordes y los márgenes tanto internos como externos, también se justificará el contenido al centro. Se pondrá el fondo en blanco por si en algún momento no se utiliza correctamente el componente y aparece un mensaje de imagen no encontrada, que haya suficiente contraste para identificarlo.

Por último, en la variante icon, se definirán los tamaños, esta vez para la imagen del componente:

```
/* Tamaño variante icon */
.icon .icon-default-size {
  margin: 0;
  padding: 0;
  height: 50px;
  width: 50px;
}

/* Tamaño variante icon */
.icon .icon-small-size {
  margin: 0;
  padding: 0;
  height: 40px;
  width: 40px;
}

/* Tamaño variante icon */
.icon .icon-big-size {
  margin: 0;
  padding: 0;
  height: 60px;
  width: 60px;
}

/* Estilo global para imágenes, forma circular. Solo se usa en la
variante icon */
img {
  border-radius: var(--plz-border-radius-circle);
  cursor: pointer;
}
```

Por otra parte, para la variante progress se deberá definir dos contenedores, un contenedor padre (playa) y otro contenedor hijo (wave) que contendrá otros seis contenedores.

```
/* ## PROGRESS VARIANT ## */
/* Contenedor padre de la variante wave */
.playa {
  box-sizing: border-box;
  display: flex;
  justify-content: center;
  align-items: center;
  background: transparent;
}

/* Sub-contenedor ola */
.wave {
  margin: 2px;
  width: 4px;
  height: 16px;
  background: var(--wave-color);
  animation: wave 1.5s linear infinite;
}
```

La clase playa define el botón y la clase wave define los elementos de tipo ola con una forma alargada y fina verticalmente como se enseñará en el resultado final del componente. También se le añade una animación de un segundo y medio con estilo lineal y que nunca acaba, es decir, en bucle.

A continuación, se define el delay necesario a cada uno de los seis contenedores hijos para simular la animación de una ola:

```
/* ANIMACIONES, se determina el delay en cada ola para hacer el efecto
deseado al cambiar de escala cada contenedor */

.wave:nth-child(1) {
  animation-delay: 0.4s;
}

.wave:nth-child(2) {
  animation-delay: 0.5s;
}
```

```
.wave:nth-child(3) {
  animation-delay: 0.6s;
}

.wave:nth-child(4) {
  animation-delay: 0.7s;
}

.wave:nth-child(5) {
  animation-delay: 0.8s;
}
```

Ahora solo faltaría añadir la escala de la animación para completar los estilos de la variante progress.

```
/* Animación de ola de la variante wave */

@keyframes wave {
  0% {
    transform: scale(0);
  }

  50% {
    transform: scale(1);
  }

  100% {
    transform: scale(0);
  }
}
```

Una vez definidos todos los estilos es el momento de continuar con el apartado de renderizado del fichero plz-button.tsx. Para mostrar el código a partir de este punto se utilizarán imágenes, ya que se vuelve más complicado leer el código sin las indentaciones que existen en el editor Visual Studio Code.

53-Plz-button code section render 1

```
render() {
  return this.link != '' && (this.variant == 'default' || this.variant == 'icon') ? (
    <Host>
      <a href={this.link} target={this.target}>
        <button
          class={{
            //Clases generales para todas las variantes
            [`${this.size}-size`]: true,
            // Clases  aplicables a la variante default
            ['hover-color']: this.variant == 'default' && this.hoverColor != '',
            [this.color]: this.variant == 'default',
            [`${this.corners}-border`]: this.variant == 'default',
            [`shadow-${this.shadowColor}`]: this.variant == 'default' && this.shadowColor != '',
            // Clases  aplicables a la variante icon
            [this.variant]: this.variant == 'icon',
            [`${this.variant}-${this.size}-size`]: this.variant == 'icon',
          }}
          style={
            //Estilo que se aplicará si existe hover y se usa la variante default
            this.variant == 'default' && this.hoverColor ? { '--bg-color-hover': this.hoverColor } : {}
          }
        >
```

54-Plz-button code section render 2

```
        {
          //Si se usa la variante icon, se renderiza esta sección de código
          //Opciones de iconos predefinidas
          this.variant == 'icon' && this.icon == 'github' ? (
            <img class={`${this.variant}-${this.size}-size`} src={Github} alt={this.icon} />
          ) : this.variant == 'icon' && this.icon == 'linkedin' ? (
            <img class={`${this.variant}-${this.size}-size`} src={Linkedin} alt={this.icon} />
          ) : this.variant == 'icon' && this.icon == 'palaze' ? (
            <img class={`${this.variant}-${this.size}-size`} src={Palaze} alt={this.icon} />
          ) : this.variant == 'icon' && this.icon == 'mail' ? (
            <img class={`${this.variant}-${this.size}-size`} src={Mail} alt={this.icon} />
          ) : this.variant == 'icon' ? (
            //Opción de icono custom
            <img class={`${this.variant}-${this.size}-size`} src={`${this.icon}`} alt={this.icon} />
          ) : null
        }
        <slot></slot>
      </button>
    </a>
  </Host>
```

En las dos imágenes anteriores, se puede ver una serie de condiciones, pero primero analizaremos la condición de entrada y su estructura.

La condición de entrada será cuando el componente tenga funcionalidad de link y la variante del botón sea o default o icon.

La estructura consta de la etiqueta “Host”, que se utiliza para definir el elemento raíz del componente en el que se renderizan los elementos (concepto similar a root en otros lenguajes).

Seguido a esta etiqueta se utilizará la correspondiente a un buen uso de acción link, en este caso la etiqueta “a”, en la que se definirá el link de referencia mediante el atributo href y la forma en que se quiere abrir el enlace con el atributo target. También se definirá la etiqueta “button” dentro de la etiqueta “a”.

A continuación, se definen las clases de estilos que se aplicarán al componente haciendo uso de condiciones para todos los casos, comprobando si se han introducido cambios haciendo uso de los atributos. Como se puede observar en la primera imagen, hay comentarios de código señalando si las clases aplicadas son generales, para la variante default o la variante icon.

Siguiendo con el análisis de código, se aplicará un estilo siempre que se haya elegido la variante icon para el componente. Este estilo aplica clases a la imagen y se utilizan más condiciones para determinar si la imagen que se desea para la variante icon es alguna de entre las predefinidas o si es una completamente distinta.

En el caso de que el componente no sea de variante icon, no se aplicarán los estilos.

Posteriormente habrá una condición con las mismas condiciones salvo que ahora con la funcionalidad de acción y no la de enlace de referencia, por lo que no se hará uso de la etiqueta “a”.

55-Plz-button code section render 3

```
) : this.link == '' && (this.variant == 'default' || this.variant == 'icon') ? (
```

Por último, se definirá la variante progress con sus dos estados, activado o desactivado, cuya propiedad active será quien mande sobre la apariencia visual el componente.

56-Plz-button code section render 4

```
) : this.variant == 'progress' ? (
  <Host>
    <button
      class={{
        //Clases aplicadas a la variante progress
        [`${this.variant}`]: true,
        [`${this.color}`]: true,
        [`${this.size}-size`]: true,
        [`${this.corners}-border`]: true,
        [`shadow-${this.shadowColor}`]: this.shadowColor != '',
      }}
      style={{
        '--wave-color': `${this.colorProgress}`,
      }}
    >
```

57-Plz-button code section render 5

```
        {
          //Dependiendo de si el botón está activado o no, se utilizará texto plano o spinner de carga
          this.active ? (
            <div class="playa">
              <div class="wave"></div>
              <div class="wave"></div>
              <div class="wave"></div>
              <div class="wave"></div>
              <div class="wave"></div>
              <div class="wave"></div>
            </div>
          ) : (
            <slot></slot>
          )
        }
      </button>
    </Host>
  ) : null;
  }
}
```

En este caso, la condición inicial será que la variante sea progress. Después seguirá una estructura similar a la variante default pero sin la opción de link, es decir, no habrá etiqueta “a”.

Se aplicarán las clases siempre que se cumplan las condiciones requeridas y se introducirá el color del componente mediante style.

La propiedad active por defecto está en false, por lo tanto, siempre accederá a la segunda condición con etiquetas "slot" siempre que no se modifique.

La etiqueta "slot" sirve para introducir cualquier tipo de contenido dinámico, en este caso se utilizará texto, pero en otras condiciones podría pasarse otro tipo de elementos HTML como puede ser una imagen.

Cuando la propiedad active cambie a true, accederá a la estructura de la segunda imagen que forma un conjunto de elemento padre de nombre playa y seis elementos hijos de nombre wave.

En última instancia se crea un fichero readme.md para documentar el componente. Se añadirá una pequeña descripción acerca del componente y una tabla con todos los atributos y las opciones de las que se dispone.

Para ver el resultado final del componente con todas sus características y funcionalidades, junto con los demás componentes desarrollados, se puede acceder a la página web de documentación de palaze mediante este enlace.

4.2.4 W3C y métricas de Lighthouse

El World Wide Web Consortium o W3C plantea unos estándares que siguen los navegadores acerca de la web. En este proyecto se ha prestado especial atención a los estándares propuestos para los web components y se han utilizado siempre que se ha considerado necesario.

Hay uno de los cuatro estándares más relevantes propuestos por el consorcio que no se ha utilizado en la construcción de ninguno de los quince componentes y este es **HTML Templetes**. Este estándar permite crear plantillas facilitando la reutilización de código HTML pudiendo clonarlas en distintas secciones de un componente. En la librería palaze, cada componente tiene variantes que cambian la estructura HTML y por la forma en que está programado no es necesario utilizarlo.

Por otra parte, el estándar **custom-element** es utilizado en todos los componentes, haciendo uso de las características que esto implica como extender la funcionalidad de etiquetas HTML básicas, añadiendo propiedades y atributos convirtiéndolo en un componente propio. También combinando distintas etiquetas sobre un nuevo custom-element o componente, etc.

Otro estándar que se utiliza es el **ES Modulues** que brinda la oportunidad de construir el componente de tal forma que permite modularizar el código y reutilizarlo en partes de la misma

aplicación web, a su vez, estos módulos se vuelven independientes para importar o exportar según lo desees.

Por último, el estándar **Shadow DOM** que permite encapsular y aislar el árbol DOM de los componentes palaze, incluyendo los estilos CSS y toda la lógica JavaScript. Con este estándar se logra que todos los componentes en los que se utiliza queden autocontenidos dentro de su propio árbol DOM, de esta manera están aislados y no intervienen con el árbol DOM de la aplicación web principal donde se utilizarán.

Este último estándar se aplica a todos los componentes excepto al plz-sidebar, en el que existe una peculiaridad. En este componente se busca que el árbol DOM del componente no se aísle del árbol DOM de la web principal ya que existe una funcionalidad toggle que permite que al interactuar mediante un click en el botón acordeón del componente se repliegue (DOM del componente) el componente sidebar reajustando el contenido principal de la página (DOM principal). Esta funcionalidad no sería realizable si el DOM del componente estuviera aislado.

Pasando al siguiente punto, en el desarrollo de la aplicación web y en el desarrollo de los Web Components se ha puesto gran importancia en seguir las normas asociadas a las "best practices" con el uso de las métricas de Lighthouse.

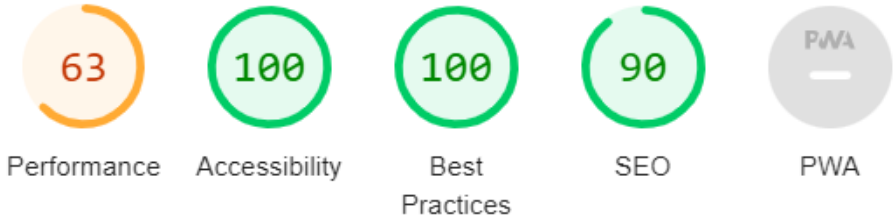
Primero hay que entender qué es Lighthouse y porqué es una herramienta increíblemente útil para el desarrollo web. Esta es una herramienta de código abierto desarrollada por Google que te permite auditar tu página web puntuándola en distintas áreas con una nota del uno al cien. Estas áreas son las que se consideran vitales para una aplicación web bien desarrollada:

- **Performance (Rendimiento):** En esta área se evalúa el rendimiento de la página mediante tiempos de carga, de interactividad, de renderizado, etc.
- **Accessibility (Accesibilidad):** En esta área se identifican problemas que podrían dificultar la navegación a personas discapacitadas de cualquier tipo.
- **Best Practices (Mejores Prácticas):** En esta sección se analizan prácticas de desarrollo web que se consideran buenas como el uso de HTTPS, imágenes optimizadas, etc.
- **SEO:** En esta sección se comprueba que la página esté optimizada para los motores de búsqueda, comprobando su estructura, metadatos, etc.
- **PWA (Progressive Web App):** Esta sección se evalúa si sometes a todo tu proyecto a una evaluación, que no es el caso del proyecto. Se comprobaría si la aplicación web cumple con los estándares para ser una aplicación web progresiva, tener manifiesto de metadatos, service worker, etc.

Todas estas áreas, después de evaluar su sección, propondrán mejoras señalando los elementos que no han aprobado las auditorías. Esta característica es muy útil porque las indicaciones son concretas y siempre acompañadas de enlaces a documentación donde puedes indagar más sobre cómo mejorar en el área problemática.

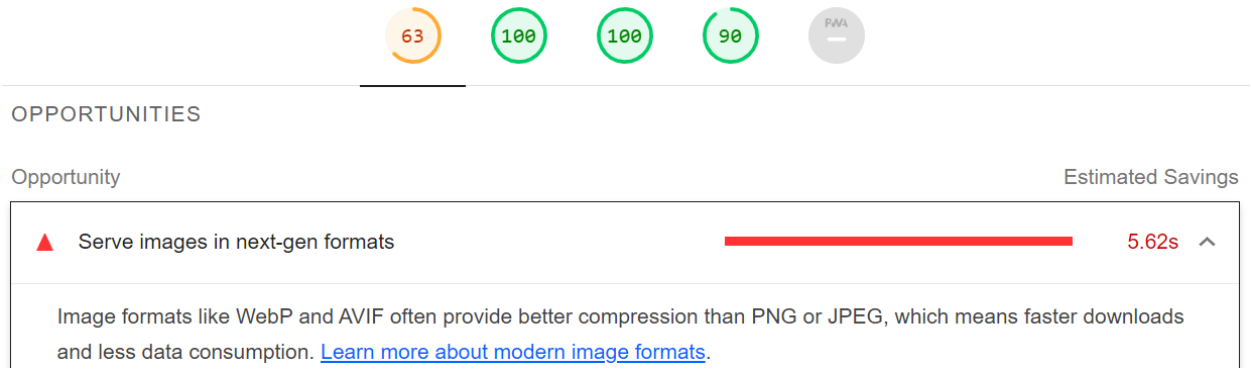
Para hacer más visual el proceso seguido durante el desarrollo de tanto la web como de los componentes se indicará un ejemplo del área de performance:

58-Ejemplo métricas Lighthouse



La puntuación que se muestra a continuación correspondía a una sección en la aplicación web una vez pasada la auditoría. Como se puede ver, la puntuación es buena, salvo el rendimiento de la página que era un poco bajo. A partir de este resultado hay que ir a las sugerencias del área de rendimiento que proponían lo siguiente:

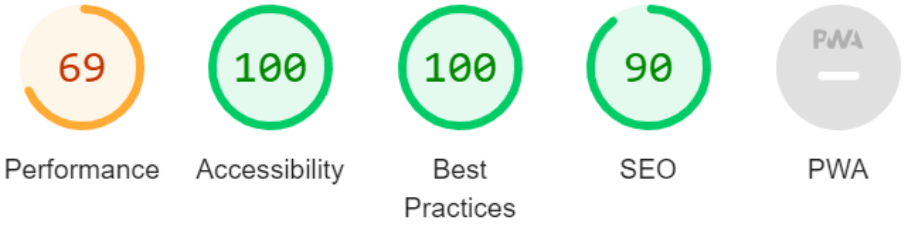
59-Sugerencias métricas Lighthouse 1



Se estaban utilizando unas imágenes que no son recomendadas para el desarrollo web, por lo que habrá que dirigirse a los ficheros donde se encontraban y se busca una forma de convertir el archivo a svg (fichero vectorial) o webp.

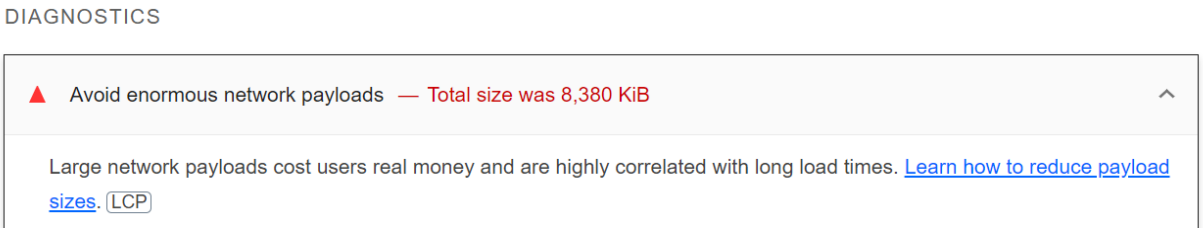
Una vez hecho esto se volvió a pasar la auditoría para ver el resultado y si se había mejorado en algo.

60-Puntuación de mejora de sugerencias de métricas Lighthouse



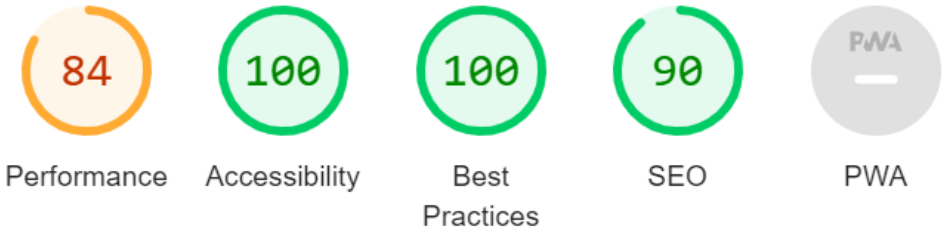
Con este cambio se había pasado de sesenta y tres puntos en rendimiento a sesenta y nueve puntos, que es una mejora significativa. Se sigue con el proceso de detectar cómo mejorar esta puntuación.

61-Sugerencias métricas Lighthouse 2



La sugerencia esta vez es reducir el tamaño de las imágenes y proporciona guías de valores por página web recomendados. Después de comprobar y realizar todas las sugerencias, se alcanza un número de puntuación satisfactorio.

62-Puntuación métricas Lighthouse



Este proceso es el que se ha seguido durante el proceso de desarrollo de la aplicación web y de la biblioteca de componentes, siempre haciendo hincapié en tener la máxima nota posible en el área de Best Practices.

La nota en cada componente y en la aplicación web en el área de Best Practices es de cien puntos y en las otras áreas oscila entre un ochenta y cien puntos. Llegar a estas puntuaciones en la librería de componentes palaze ha supuesto realizar el proceso comentado anteriormente múltiples veces hasta lograr la nota buscada.

Desde un inicio la prioridad ha sido aplicar las métricas de Lighthouse en el área de Best Practices para los Web Components. Las auditorías que se deben aprobar para tener un puntaje de cien en esta área son:

- El uso de HTTPS garantiza la seguridad de comunicación entre usuario y sitio web.
- Evita solicitar el permiso de geolocalización al cargar la página, ya que puede afectar negativamente la experiencia del usuario y la privacidad.
- Evita solicitar el permiso de notificaciones al cargar la página, a menos que sea estrictamente necesario, ya que esto interrumpe la experiencia del usuario.
- Permite que los usuarios peguen contenido en los campos de entrada. Esto mejora la usabilidad y la accesibilidad.
- Muestra imágenes con la proporción de aspecto correcta. Con esto puedes evitar distorsiones que promuevan una mala experiencia de usuario.
- Sirve imágenes con la resolución adecuada para optimizar el rendimiento de la página y reducir tiempos de carga.
- La página tiene el doctype HTML. Es importante que los navegadores puedan interpretar los archivos correctamente.
- Define correctamente el conjunto de caracteres (charset): Es importante que los navegadores puedan interpretar los archivos correctamente.
- Evita listeners de eventos unload: Este tipo de eventos causan problemas de rendimiento y seguridad.
- Evita el uso de APIs obsoletas: Suelen haber vulnerabilidades y no son soportadas por todos los navegadores.
- No hay errores del navegador registrados en la consola: Es importante que nunca haya errores por consola cuando estás navegando por tu web.
- No hay problemas en el panel de Issues en las herramientas de desarrollo de Chrome. En el caso de haberlas, se puede acceder y solucionarlas siguiendo las sugerencias que se proponen.
- La página tiene source maps válidos: Unos source maps bien configurados permiten una mejor depuración y análisis de errores en el código de la web.

Esta ha sido una breve explicación de los aspectos más relevantes del área de best practices, si se desea indagar más sobre estas áreas, se puede acceder a la documentación de Lighthouse referenciada en el punto 29 de la bibliografía o mediante este enlace.

Hay más aspectos dentro de esta área que son de sentido común, como puede ser una buena indentación en el código fuente, seguir unas nomenclaturas descriptivas para nombre, comentar el código, etc.

5 Conclusiones

5.1 Conclusiones personales

La idea de crear una librería de componentes y una aplicación web de documentación de estos vino a raíz de querer enfocar mi carrera profesional como ingeniero informático al sector del desarrollo web, concretamente a la especialidad frontEnd.

Al principio tenía dudas de escoger este tipo de proyecto para mi trabajo de final de grado, ya que suponía muchos retos debido a la falta de conocimiento personal en la mayoría de los lenguajes y herramientas utilizadas para su desarrollo. En este proyecto se abarca todo el conjunto de elementos que tiene un proyecto profesional, empezando desde el diseño y estrategias de la aplicación web, metodologías, desarrollo de una librería, gestión y control de versiones, aplicación de una librería, documentación, distribución, publicación, etc.

El resultado final ha sido satisfactorio, he conseguido alcanzar todos los objetivos propuestos y aprendido de forma incremental a lo largo del proyecto, indagando en conceptos desde la base para entender su funcionamiento. Ha sido un proceso en el que se ha requerido mucho tiempo de investigación, lectura y pruebas hasta alcanzar un resultado óptimo.

Todas estas experiencias me han motivado a buscar retos en mi carrera profesional, buscando dar lo mejor de mí mismo, siendo cada vez más eficiente y productivo ayudándome de nuevas vertientes tecnológicas que vienen para quedarse en el mundo del desarrollo web.

5.2 Futuras líneas de desarrollo

En cuanto a las futuras líneas de desarrollo para este proyecto, me gustaría primero enfocarme en optimizar y añadir funcionalidades al contenido que ya se ha desarrollado:

- Buscar completar una puntuación de cien puntos en la herramienta de Lighthouse para todos los componentes.
- Analizar y añadir funcionalidades y características a los componentes según sea necesario.
- Migrar la aplicación web a otra tecnología que permita la carga de la web de una forma más eficiente.
- Añadir distintos idiomas a la aplicación web.
- Añadir distintos temas a la aplicación web (Ej: Light, Dark).

Por otra parte, una vez completadas las tareas para optimizar lo máximo posible el trabajo ya realizado anteriormente, se podría plantear añadir distintos componentes como podrían ser:

- Componente de alerta (pop up).
- Componentes de tipo formulario.
- Componente sliders (permiten ir cambiando mediante flechas o movimientos de cursor entre distintas opciones).
- Componente color picker (selector de colores).
- Componente de comparador de imágenes.
- Componente de tipo QR (generar y escanear QR con referencias de enlace).

La lista de componentes puede ser muy amplia. Hay que recalcar que siempre se buscaría la creación de componentes en base a una necesidad como se ha realizado en este proyecto.

También habría que ir ampliando el Design System a medida que se siguiera el desarrollo del proyecto.

Considero que este proyecto no ha hecho más que empezar y mi próximo objetivo será seguir investigando y pensando en cómo mejorar, haciendo cada vez más atractiva la librería de componentes y la documentación.

6 Bibliografía

1. Web Components - Web APIs | MDN. (2023, 24 abril). https://developer.mozilla.org/en-US/docs/Web/API/Web_components
2. Ivanovs, A. (2023). The Most Popular Front-end Frameworks in 2023. Stack Diary. https://stackdiary.com/front-end-frameworks/
3. Buckler, C. (2023). A Complete Introduction to Web Components in 2023. Kinsta®. https://kinsta.com/blog/web-components/
4. Stencil Web Component Browser Support | Stencil. (s. f.). https://stenciljs.com/docs/browser-support
5. Moreno, C. (2022). ¿Qué es el Design System y por qué me va a ayudar a escalar mi producto? Product Hackers. https://producthackers.com/es/blog/design-system
6. Nalbert. (2022). Por qué necesitas un sistema de diseño y cómo tus clientes te lo van a agradecer. Plain Concepts. https://www.plainconcepts.com/es/sistema-diseno-guia/
7. Fluent 2 Design System. (s. f.). Fluent 2 Design System. https://fluent2.microsoft.design/
8. Design tokens – Material Design 3. (s. f.). Material Design. https://m3.material.io/foundations/design-tokens/overview
9. Overview - Brand - Atlassian Design System. (s. f.). Atlassian Design System. https://atlassian.design/components/
10. Figma Best Practices. (s. f.). Figma. https://www.figma.com/best-practices/
11. Best Practices Audits - Chrome Developers. (s. f.). Chrome Developers. https://developer.chrome.com/docs/lighthouse/best-practices/
12. Midjourney Quick Start Guide. (s. f.). https://docs.midjourney.com/docs/quick-start
13. Welcome to the Photoshop User Guide. (s. f.). https://helpx.adobe.com/photoshop/user-guide.html
14. JavaScript | MDN. (2023, 1 mayo). https://developer.mozilla.org/en-US/docs/Web/javascript
15. HTML: Lenguaje de etiquetas de hipertexto | MDN. (2023, 13 marzo). https://developer.mozilla.org/es/docs/Web/HTML
16. CSS: Cascading Style Sheets | MDN. (2023, 16 abril). https://developer.mozilla.org/en-US/docs/Web/CSS
17. ::slotted() - CSS: Cascading Style Sheets | MDN. (2023, 13 abril). https://developer.mozilla.org/en-US/docs/Web/CSS/::slotted
18. House, C. (2023). A Complete Guide to CSS Grid | CSS-Tricks. CSS-Tricks. https://css-tricks.com/snippets/css/complete-guide-grid/
19. Stencil - A Compiler for Web Components | Stencil. (s. f.). https://stenciljs.com/docs
20. Get started with Netlify. (2023, 25 abril). Netlify Docs. https://docs.netlify.com/get-started/

21. Quickstart - GitHub Docs. (s. f.). GitHub Docs. https://docs.github.com/en/get-started/quickstart
22. Getting started | npm Docs. (s. f.). https://docs.npmjs.com/getting-started
23. Basic Syntax | Markdown Guide. (s. f.). https://www.markdownguide.org/basic-syntax/
24. Introducing JSX – React. (s. f.). React. https://legacy.reactjs.org/docs/introducing-jsx.html
25. Using JSX for developing Stencil web components. (s. f.). https://masteringionic.com/blog/using-jsx-for-developing-stencil-web-components
26. Shoelace-Style. (s. f.). GitHub - shoelace-style/shoelace: A collection of professionally designed, every day UI components built on Web standards. Works with all framework as well as regular HTML/CSS/JS. GitHub. https://github.com/shoelace-style/shoelace
27. How to use highlight.js. (s. f.). https://highlightjs.org/usage/
28. Creating WebP Images with the Command Line. (2018, 5 noviembre). web.dev. https://web.dev/codelab-serve-images-webp/
29. Lighthouse - Chrome Developers. (s. f.). Chrome Developers. https://developer.chrome.com/docs/lighthouse/

7 Anexo

Para facilitar el acceso a los recursos relacionados con este proyecto, se detallarán a continuación.

1. **Código fuente:** https://github.com/pablodiazjorge/palaze
   El repositorio donde se encuentra todo el código fuente se ubica en GitHub. Tanto la librería de componentes como la aplicación web de presentación, todo en el mismo repositorio monorepo.
2. **Palaze web:** https://palaze-pablodiazjorge.netlify.app/
   La web de presentación palaze está desplegada online en la plataforma Netlify.
3. **Publicación NPM:** https://www.npmjs.com/package/palaze
   La distribución de la librería palaze está publicada mediante NPM, accesible para el uso de cualquiera.
4. **Proyecto prueba en CodePen:** https://codepen.io/BitBlo/pen/PoyLRQq
   Se ha realizado una prueba de uso de los componentes vía CDN como se explica en la memoria en la plataforma CodePen (un editor de código online).