Informática y comunicaciones

ies azarquiel

Calculadora Web con Reconocimiento de Voz

"Hacer cálculos nunca fue tan fácil"

TRABAJO FINAL DE CICLO

CFGS DESARROLLO DE APLICACIONES WEB

Autor/a: Laura Arellano Torrero

Tutor/a: Tomás Martínez Ruiz

Junio de 2025

Calculadora Web con Reconocimiento de Voz

Laura Arellano Torrero

CFGS DESARROLLO DE APLICACIONES WEB.

ÍNDICE

Capítulo 1: Introducción y Objetivos

En este proyecto voy a desarrollar una calculadora web que se va a poder usar de dos maneras: como una calculadora normal, usando **botones** en la pantalla, y también usando la **voz** para decir las operaciones que queremos hacer.

La interacción mediante botones será similar a las calculadoras tradicionales, lo que ofrece una **experiencia de usuario familiar** a aquellos usuarios que prefieren usar métodos tradicionales.

Por otro lado, la opción de interactuar mediante voz se presenta como una solución innovadora que mejorará la experiencia del usuario, especialmente para quienes buscan una forma más **cómoda y rápida** de realizar operaciones.

La idea ha nacido de observar cómo cada vez más personas interactúan con sus dispositivos utilizando comandos de voz, lo cual no sólo aporta comodidad y rapidez, sino también **accesibilidad** para aquellos usuarios con dificultades visuales.

La aplicación tendrá dos modos de uso: uno **clásico**, mediante botones en pantalla, y otro **innovador**, en el que los usuarios podrán pronunciar las operaciones que desean resolver.

El objetivo principal es crear una aplicación web que sea lo más **intuitiva y accesible posible**, aprovechando las tecnologías actuales del desarrollo web. La **facilidad** de uso es una prioridad, y se espera que tanto el diseño como la interacción de la calculadora sean lo suficientemente **simples** para que cualquier persona pueda usarla sin dificultades. Para llevar a cabo este proyecto, se utilizará **Angular**, un framework basado en TypeScript.

Además, se implementará una **API de reconocimiento de voz**, que será la encargada de escuchar al usuario e interpretar sus palabras, traduciéndose a operaciones matemáticas siempre y cuando el **habla** sea **clara, pausada y comprensible**, lo que permitirá una interpretación adecuada de las instrucciones dadas.

La calculadora se centrará exclusivamente en el funcionamiento en el lado del cliente (frontend), lo cual permite un enfoque más directo y centrado en el comportamiento **visual** e **interactivo** de la aplicación.

Aunque el cliente de este proyecto será ficticio, he imaginado que se podría tratar, por ejemplo, de una **academia online** que necesita una herramienta sencilla y accesible para realizar cálculos rápidos en sesiones de clase, o incluso de una **persona mayor o con discapacidad motora** que necesita una **solución** que no dependa del uso constante del teclado o del ratón.

Este proyecto representa la oportunidad de **profundizar en tecnologías actuales** del desarrollo web, **experimentar** con herramientas modernas y trabajar con APIs externas, todo ello enfocado en crear una solución accesible y orientada al usuario.

Capítulo 2: Especificación de Requisitos

Antes de iniciar el desarrollo de la calculadora web, veo necesario plantear los **requisitos** que debe **cumplir** mi proyecto. A continuación, describiré los **requisitos funcionales** y **no funcionales**, y también adjuntaré **bocetos** del diseño de la calculadora realizados en **Figma**.

Capítulo 2.1: Requisitos Funcionales

Mi aplicación debe cumplir los siguientes requisitos funcionales para alcanzar los objetivos propuestos en el Capítulo 1:

- **Operaciones básicas**: La calculadora debe ser capaz de hacer sumas, restas, multiplicaciones y divisiones.
- **Entrada por voz**: El usuario debe poder dictar operaciones matemáticas sencillas a través de comandos por voz.
- Visualización de **resultados**: Después de cada operación, el resultado debe mostrarse claramente en la pantalla de la calculadora.
- **Activar/Desactivar el micrófono**: El usuario debe poder activar o desactivar el modo de reconocimiento de voz pulsando un botón.
- Gestión de errores: Si el **reconocimiento de voz** no entiende la orden del usuario, debe aparecer un **mensaje de error**.
- Borrar datos: Por último, la calculadora necesita tener la opción de **borrar el contenido** actual para poder hacer una **nueva operación**.

Capítulo 2.2: Requisitos No Funcionales

Los requisitos no funcionales describen **cómo** debe **comportarse** la aplicación:

- Tiempo de respuesta: La aplicación debe ser capaz de realizar los cálculos de manera **rápida y eficiente**, mostrando el resultado de las operaciones en pocos segundos. Esto es fundamental para **evitar** elevados **tiempos de espera** y mantener una **buena experiencia de usuario**.
- Compatibilidad: La calculadora debe ser compatible con **el mayor número posible de navegadores**. Asegurar esta compatibilidad permitirá que la aplicación sea accesible para diferentes usuarios, sin importar el tipo de navegador que estén utilizando.
- Diseño **responsivo**: La interfaz debe adaptarse correctamente a diferentes **tamaños de pantalla**, ya sea en dispositivos móviles, tablets o en ordenadores. Un diseño responsivo mejorará la accesibilidad y la experiencia de uso.
- **Accesibilidad**: El diseño de la aplicación debe emplear colores de **alto contraste** y tamaños de letra adecuados. Esto ayudará a que la herramienta sea inclusiva y esté al alcance de cualquier persona.

- **Experiencia de Usuario** (UX): La aplicación debe ofrecer una interfaz intuitiva y sencilla pensada para usuarios de **todas las edades**. El objetivo es que cualquier persona, sin necesidad de conocimientos técnicos, pueda utilizar la calculadora de forma **rápida** y **sin complicaciones**.
- **Modo claro y modo oscuro**: La aplicación debe contar con la opción de cambiar entre modo claro y modo oscuro, permitiendo que cada usuario elija el **estilo visual** que más le guste.

Capítulo 2.3: Bocetos/prototipos diseñados en Figma

Antes de empezar a programar la calculadora, he diseñado los **bocetos** de la aplicación en Figma. Esta herramienta me ha permitido planificar **cómo** quiero que **se vea** mi aplicación web, qué **colores** se van a usar y cómo se **organizará cada parte** antes de pasar al desarrollo.
Estos bocetos se han diseñado para adaptarse correctamente a diferentes tamaños de pantalla (**ordenador, tablet y móvil**), y para ofrecer dos modos visuales: **modo claro y modo oscuro**, que el usuario podrá alternar según sus preferencias.
A lo largo de este apartado, voy a mostrar las distintas páginas diseñadas: **Inicio, Cómo funciona y Accesibilidad**.
Características comunes de las **tres páginas** diseñadas:
Las tres páginas comparten la misma cabecera, que incluye el **logo** de la aplicación, un **menú de navegación** sencillo que permite navegar entre ellas y un **botón** de cambio de tema (**modo claro/oscuro**) representado por un icono dependiendo del modo que se esté utilizando en ese momento (un sol o una luna).
En cada imagen, el **modo oscuro** se muestra en el lado **izquierdo** y el **modo claro** en el lado **derecho.**
En el diseño responsivo (para tamaño **small)**, el **logo no es visible** y el **menú** de navegación se reduce a un **icono de tres líneas horizontales**.
Por último, en el **pie de página** aparece mi nombre, con el fin de proteger los **derechos de autor** correspondientes.

Página de Inicio:

Esta es una de las **páginas más importantes** de todo el proyecto, ya que no solo es la **primera impresión** que recibe el usuario al acceder a la web, sino que además **contiene directamente la calculadora**, que es el elemento principal y funcional del sitio.

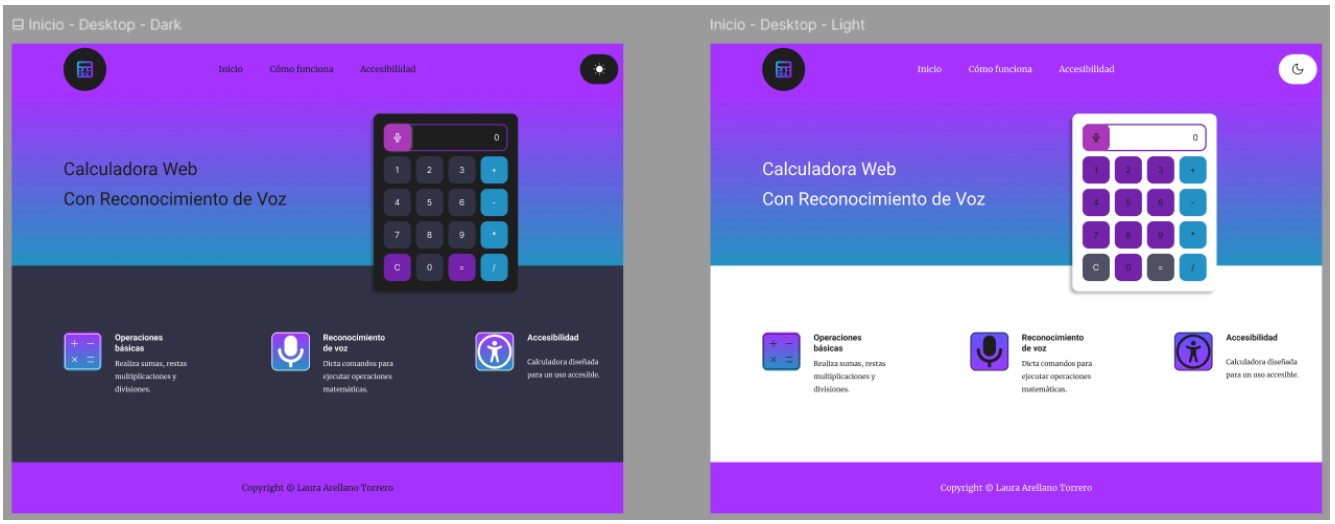
La calculadora se muestra en el **centro** de la pantalla, con un **diseño moderno** que destaca sobre el fondo. Desde aquí, el usuario puede empezar a hacer **operaciones**, ya sea pulsando los botones de los **números** y de los **operadores** o activando el **micrófono** para usar el modo de reconocimiento de voz.

Debajo de la calculadora he añadido **tres "secciones"** con iconos explicativos, que resumen las funcionalidades de mi calculadora:

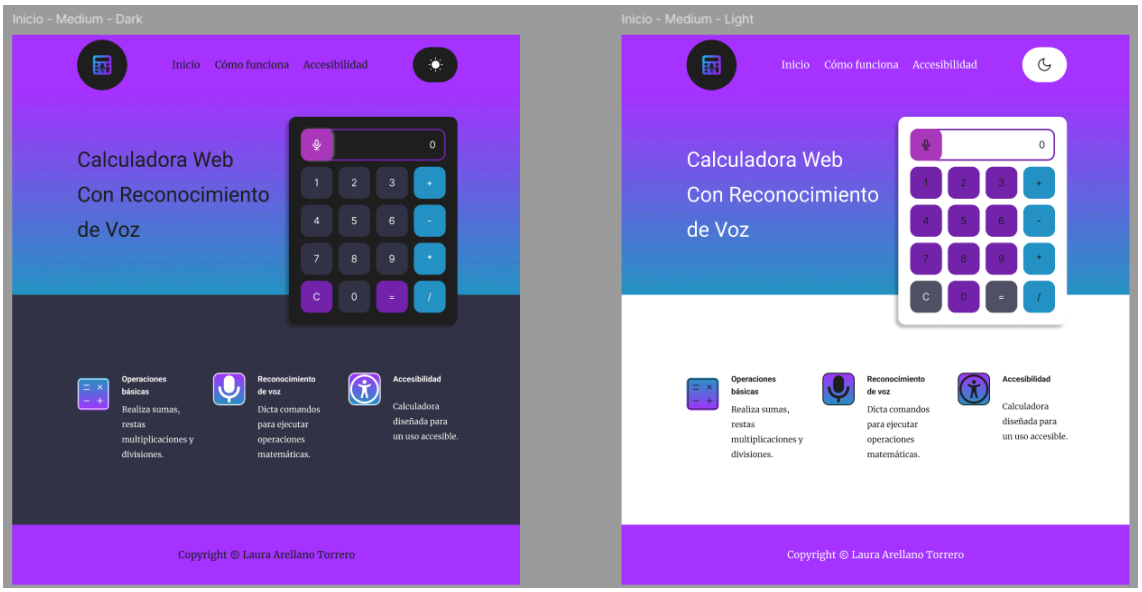
- **Operaciones básicas** (sumas, restas, multiplicaciones y divisiones)
- **Reconocimiento de voz** para dictar operaciones
- **Accesibilidad** (como parte importante del diseño).

A continuación adjunto imágenes de la página de Inicio, diseñada en Figma y adaptada a **distintos tamaños**, en **ambos modos visuales**: oscuro y claro.

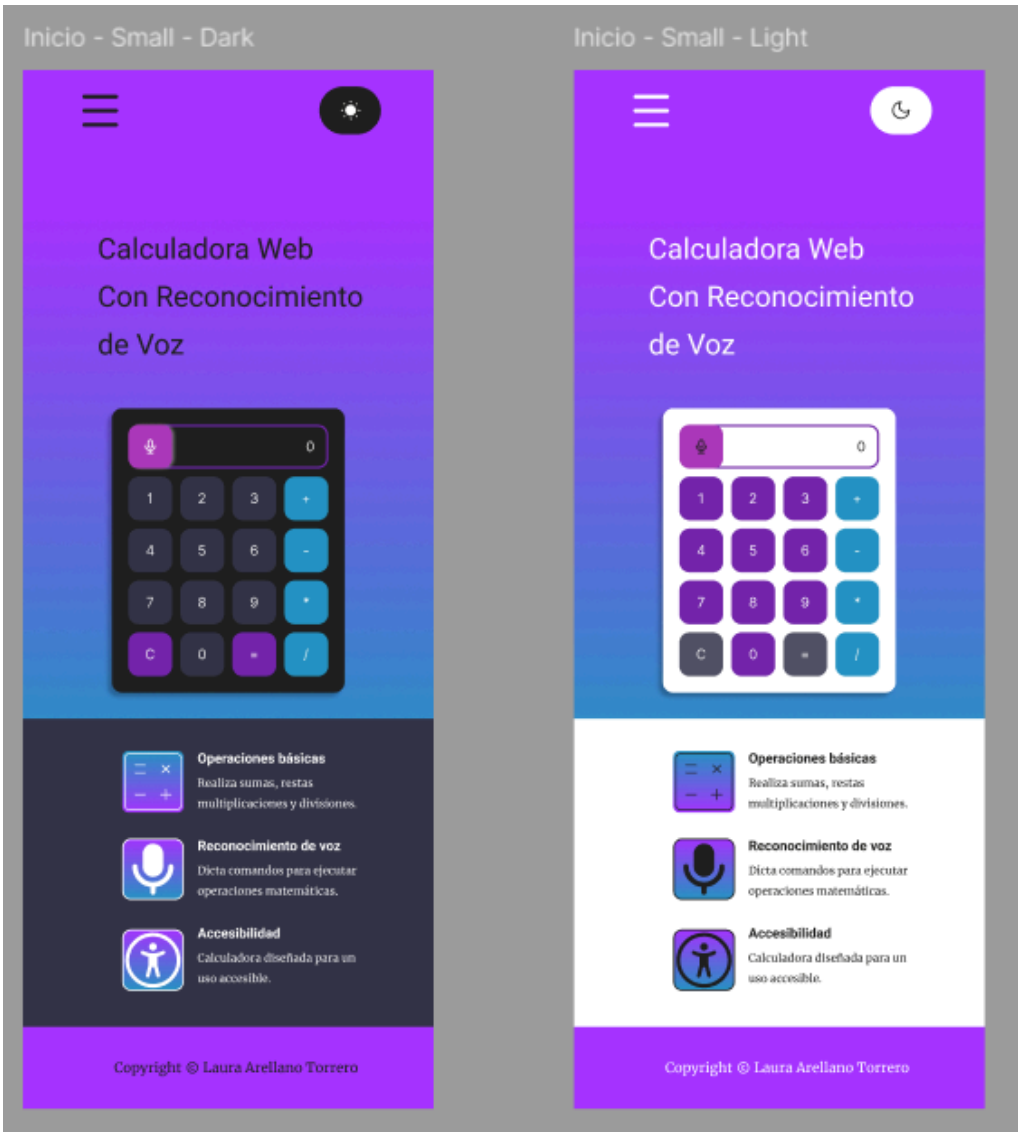
En la siguiente imagen aparece la página **Inicio** en tamaño **Desktop** con un ancho de **1440px**:



A continuación, aparece la página de **Inicio** en tamaño **Medium** con un ancho de **1024px**:



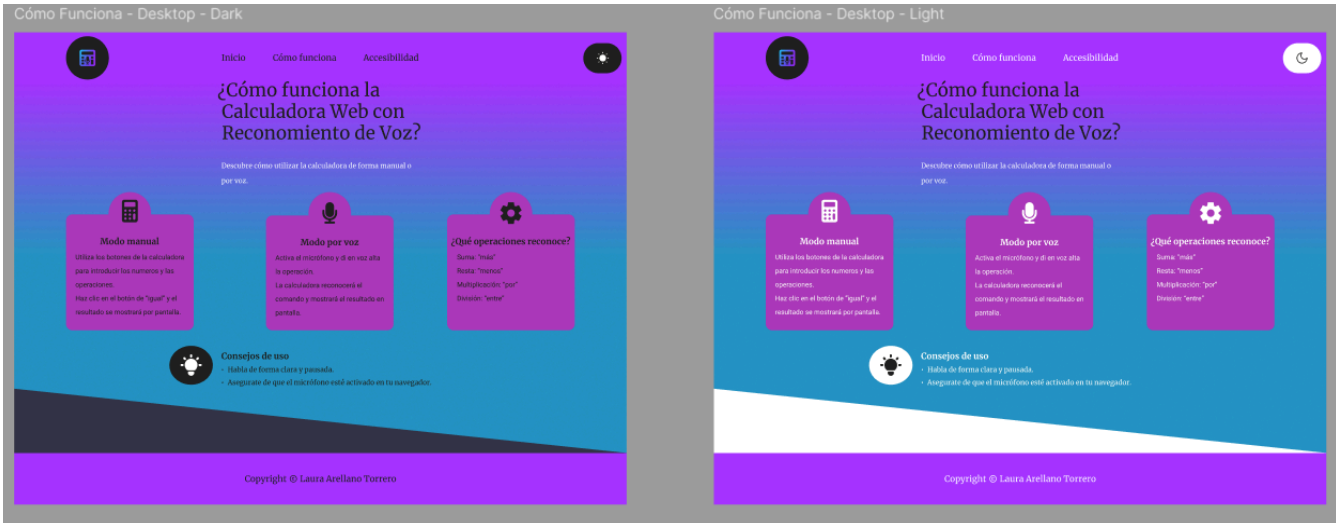
Por último, en la siguiente imagen se puede ver la página de **Inicio** en tamaño **Small** con un ancho de **600px**:



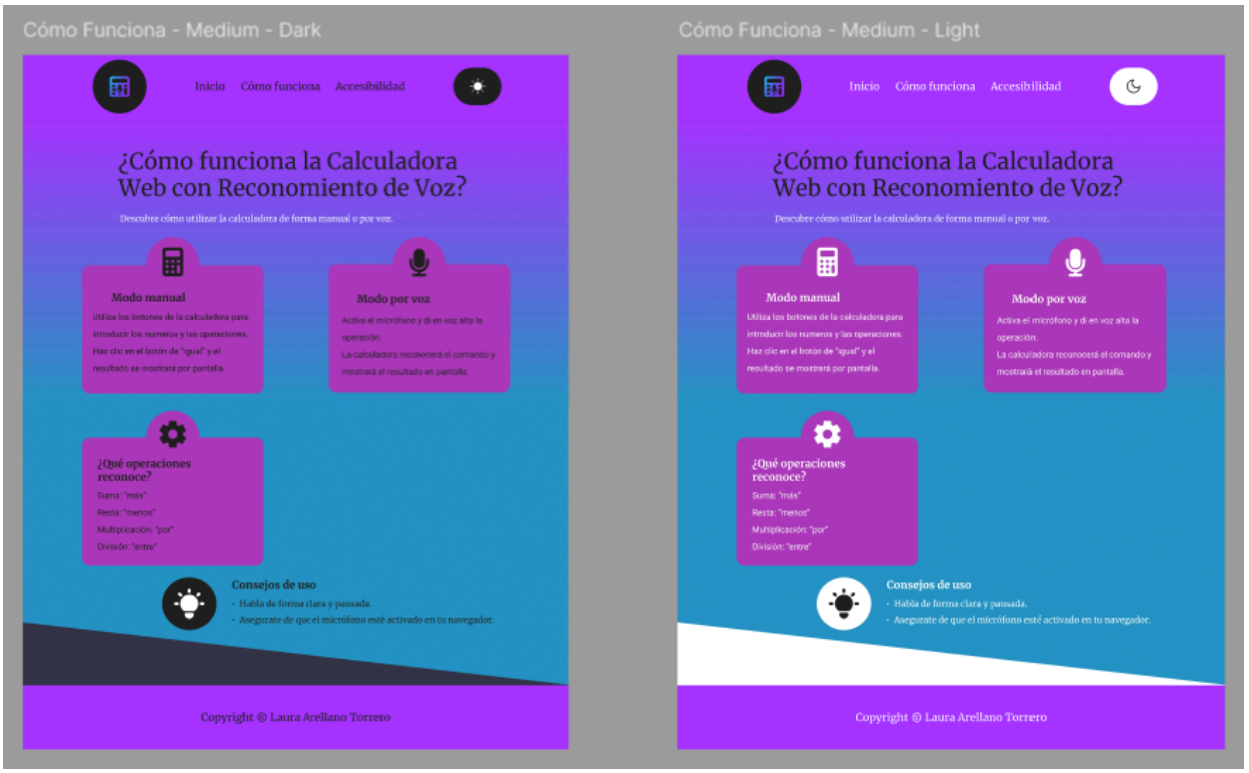
Página Cómo funciona:

Para ayudar al usuario en el uso de la aplicación, he diseñado **tarjetas informativas** en las que explico de forma clara el uso de la calculadora: desde el modo **manual** hasta el uso por **voz**, incluyendo una descripción de las **operaciones** que la calculadora **puede reconocer**.

En la siguiente imagen aparece la página "**Cómo funciona**" en tamaño **Desktop** con un ancho de **1440px**:



Ahora, aparece la página "**Cómo funciona**" en tamaño **Medium** con un ancho de **1024px**:



Por último, en la siguiente imagen se puede ver la página "**Cómo funciona**" en tamaño **Small** con un ancho de **600px**:

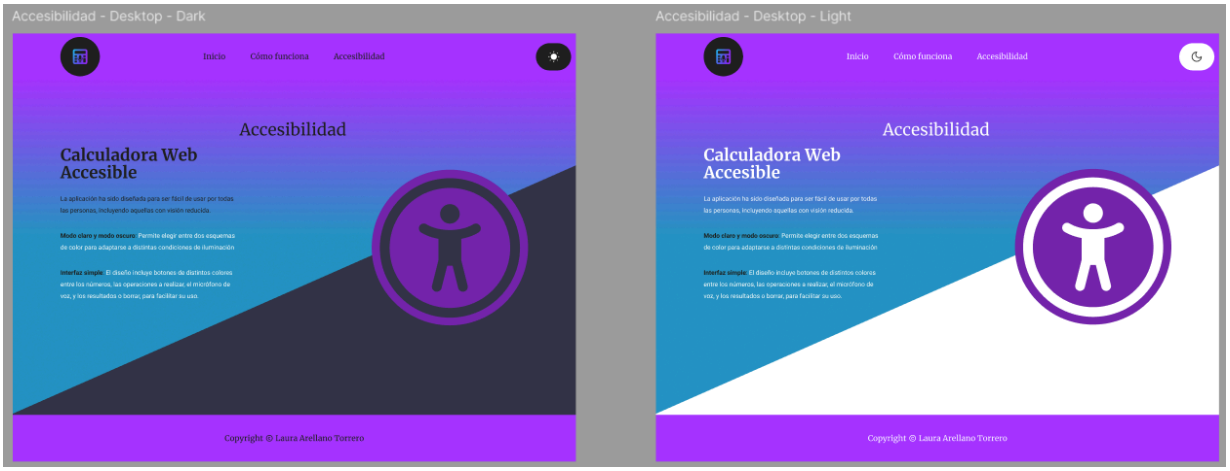
Como se puede observar, en la página "Cómo funciona" he utilizado **iconos** y **textos cortos** descriptivos, favoreciendo una **lectura rápida e intuitiva**.

Este diseño ha sido pensado para explicar el uso de la aplicación lo más **fácil** posible, dando más importancia al **modo por voz,** que puede ser **menos común** para personas con **poca experiencia tecnológica**.

Página Accesibilidad:

He diseñado esta página centrándome en la inclusión, con el objetivo de **garantizar** que cualquier persona, **independientemente** de sus **capacidades visuales** o su **experiencia digital**, pueda utilizar la aplicación fácilmente.

En la siguiente imagen aparece la página "**Accesibilidad**" (**Desktop)** con un ancho de **1440px**:



A continuación, aparece la página "**Accesibilidad**" en tamaño **Medium** con un ancho de **1024px**:

Por último, en la siguiente imagen se puede ver la página "**Accesibilidad**" en tamaño **Small** con un ancho de **600px**:

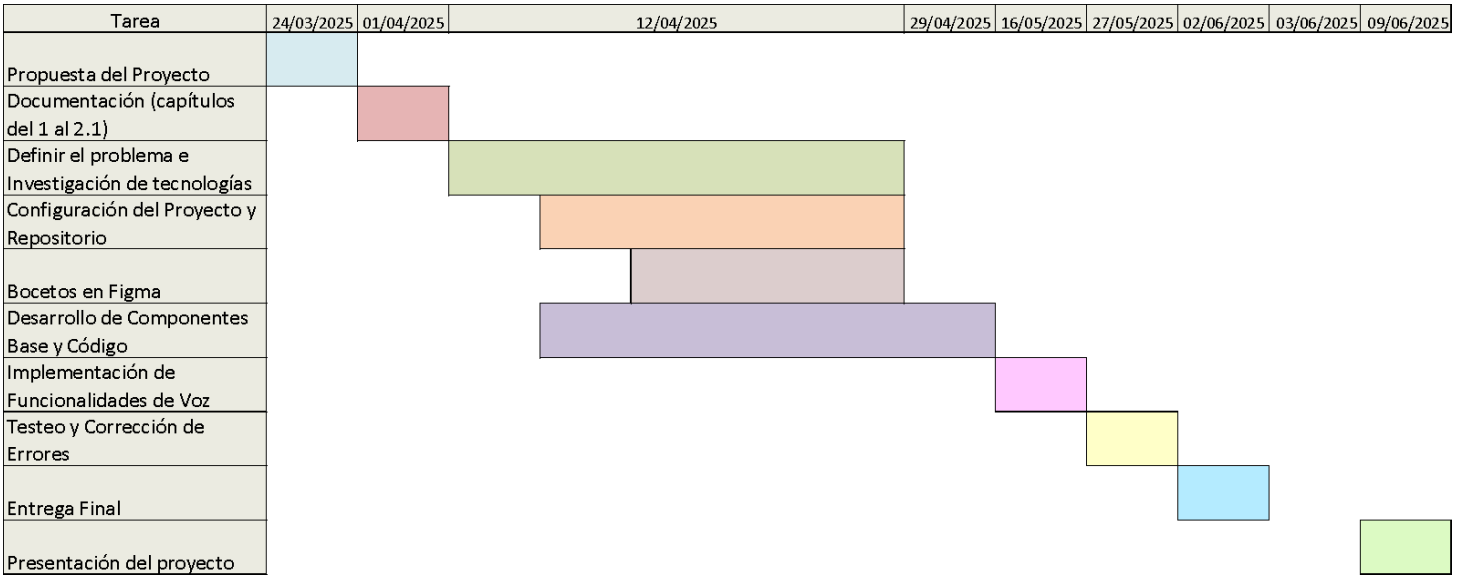
Como se ha podido observar en las imágenes presentadas para cada una de las páginas, el diseño se adapta bien tanto en modo claro como en modo oscuro, manteniendo un buen **diseño responsivo**. Esto asegura una buena **experiencia visual y accesible** sin importar el dispositivo desde que el usuario esté accediendo.

Capítulo 3: Planificación Temporal y Evaluación de Costes

Capítulo 3.1: Planificación Temporal

La planificación temporal de este proyecto tiene como objetivo **estructurar su desarrollo** en **fases ordenadas**, permitiendo **controlar** el progreso y **facilitando la entrega a tiempo**.

He elaborado un **diagrama de Gantt** en Excel que refleja las **tareas principales** a desarrollar, los **plazos previstos** y su relación con las **fechas de entrega**:



Esta planificación está dividida en varias fases, como investigación, diseño, desarrollo, pruebas y entrega, para organizar mejor el trabajo.
Aunque he establecido unos **plazos aproximados**, es posible que durante el desarrollo **surjan imprevistos** que obliguen a hacer pequeños cambios.

El diagrama de Gantt me sirve para visualizar de forma rápida **qué tareas** tengo que realizar en cada momento y **cuánto tiempo** tengo disponible. También he dejado algo de **margen** entre fases **por si alguna tarea lleva más tiempo** del esperado, intentando así **evitar retrasos en la entrega** final.

Capítulo 3.2: Evaluación de Costes

Antes de empezar con el desarrollo de cualquier proyecto web, es importante hacer una estimación de los costes para saber qué **recursos** serían necesarios si este proyecto se hiciera de forma profesional.

PLANIFICACIÓN DE HORAS DE TRABAJO:

Para poder **organizar** mejor el proyecto y la **evaluación de los costes**, he hecho una **estimación** de las horas que me llevaría completar cada parte del desarrollo.

Tarea	**Horas estimadas**
Investigación de tecnologías y planificación	10 horas
Diseño de prototipos en Figma	35 horas
Configuración del proyecto Angular y estructura básica	20 horas
Desarrollo de componentes y funcionalidades principales	40 horas
Integración de reconocimiento de voz	5 horas
Estilos SCSS y diseño responsivo	15 horas
Pruebas, corrección de errores…	5 horas
Documentación	15 horas
TOTAL HORAS DE TRABAJO	145 horas

A continuación, se puede ver una evaluación aproximada de los costes que tendría el desarrollo de mi aplicación web:

EVALUACIÓN DE COSTES

Concepto	**Descripción**	**Coste estimado (€)**
Dominio web	Registro de un dominio .com, .es o parecido	12€/año
Alojamiento web (Hosting)	Plan básico de alojamiento.	50€/año
Diseño (Figma)	Versión gratuita. Si desea la versión PRO, tendrá coste.	0€
Framework Angular	Angular es de código abierto y gratuito	0€
API Reconocimiento de Voz	Web Speech API; no requiere coste.	0€
Control de versiones (GitHub)	GitHub ofrece plan gratuito para proyectos individuales	0€
Visual Studio Code	Software libre. Sin coste.	0€
Horas de desarrollo	Un total de 145 horas de trabajo, 12€/hora.	145 horas x 12€ = 1.740€
Costes adicionales	Imprevistos (actualizaciones, pruebas, corrección de errores…)	20€
TOTAL		**1.822€**

Capítulo 4: Tecnologías utilizadas

En este proyecto he elegido varias **tecnologías** que me han parecido las más **adecuadas** para llevar a cabo el proyecto. A continuación, explico las **principales tecnologías** que estoy utilizando, por qué las he elegido y cómo van a ayudar a que la aplicación funcione de manera correcta.

Lenguajes de programación:

- HTML: **HTML** es ideal para **estructurar contenido** en la web, y es compatible con la mayoría de navegadores. Al ser tan común y fácil de usar, me permite crear la **estructura básica de la página** de manera clara y ordenada. Esto incluye la parte de la **calculadora**, los **botones**, el **área para mostrar resultados** y los **campos de entrada** de las operaciones.
- SCSS (Sass): En lugar de usar CSS tradicional, he trabajado con **SCSS**, que es una versión mejorada de CSS. Este lenguaje me permite **escribir estilos** de forma más **limpia y organizada**, usando **variables**, **funciones** o **anidación de reglas**. Esto hace que el **código** sea mucho **más fácil de mantener**, sobre todo cuando el proyecto empieza a crecer. **Angular** además **lo soporta muy bien**, así que es perfecto para este tipo de aplicaciones web.
- Javascript: **Javascript** es el lenguaje que hace posible que la calculadora sea **interactiva**. Me permite implementar todas las funcionalidades que necesita la aplicación, como hacer los **cálculos** y **recibir comandos de voz**.

Framework:

- Angular (TypeScript): **Angular** es un framework **súper completo y fácil de usar,** ideal para crear aplicaciones web interactivas.

  Lo que más me gusta de Angular es que me permite **organizar todo el código** de manera clara y estructurada, **usando componentes**. Además, al usar **TypeScript**, puedo trabajar con un **sistema de tipos más seguro** que me ayuda a **evitar errores** a la hora de escribir el código. Esto es muy **útil** para **proyectos grandes y complejos**, y me da más control sobre el desarrollo.

API de Reconocimiento de Voz:

- Web Speech API: La **Web Speech API** es la herramienta perfecta para integrar el **reconocimiento de voz**. Gracias a esta API, los usuarios pueden **decir** las **operaciones matemáticas** en lugar de escribirlas, lo que hace que la calculadora sea mucho **más accesible**. Esta API convierte la voz en texto, lo cual me permite interpretar los **comandos de voz** y **convertirlos en operaciones matemáticas**.

Control de versiones y repositorio:

- Git y GitHub: **Git** es una herramienta que permite llevar un **control total del código** y de los **cambios** que se van realizando a lo largo del desarrollo.

  Al usar **GitHub** como **repositorio** en la nube, puedo **almacenar el proyecto** de forma segura y acceder a él desde cualquier lugar. Además, GitHub me permite mantener el proyecto **bien organizado**, ya que puedo hacer **cambios** sin miedo a perder el trabajo anterior. Si algo no funciona, está la opción de **volver a una versión anterior** del código. También permite hacer un seguimiento de la **evolución del proyecto** y colaborar con otras personas si fuera necesario.

Diseño de la interfaz de usuario (UI):

- Figma: **Figma** es una **herramienta de diseño** muy popular y fácil de usar para crear **prototipos o bocetos** de interfaces web.

  Decidí usar Figma para diseñar los bocetos de mi calculadora porque es muy **práctica** y me permite ver cómo quedará la interfaz antes de empezar a programarla. Además, Figma me permite **crear variables** que luego se pueden **exportar** para usar fácilmente a la hora de programar.

Base de Datos:

Este proyecto **no necesita base de datos** porque el funcionamiento es del lado del cliente. La calculadora **no guarda datos**, simplemente resuelve operaciones. Si en un futuro se quisiera añadir la opción de **guardar historial de operaciones** o **preferencias** del usuario, se podría usar una base de datos como **Firebase, MongoDB**…

Otras herramientas y tecnologías:

- NPM (Node Package Manager): NPM es la herramienta que ayuda a gestionar todas las librerías y dependencias del proyecto. También, facilita la instalación y actualización de todas estas librerías de manera rápida y sencilla.

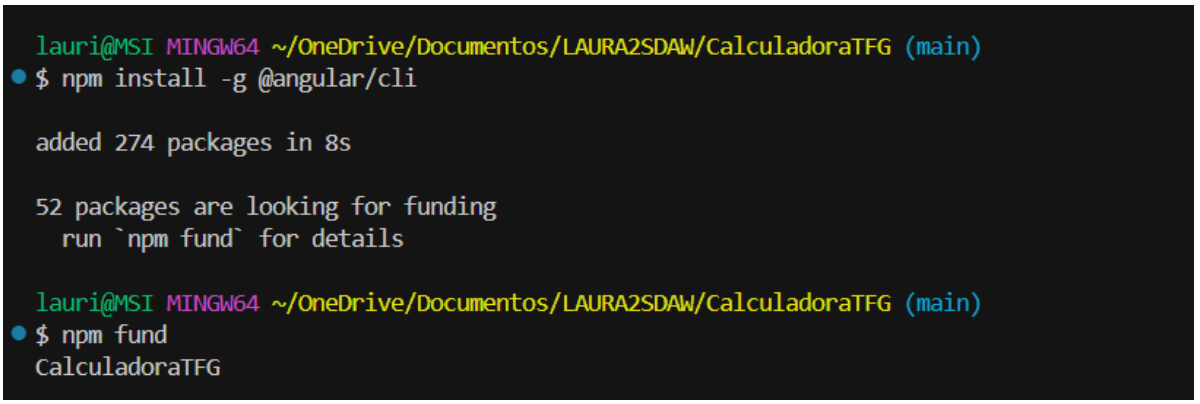
Capítulo 5: Desarrollo e implementación

En este apartado voy a describir cómo ha sido el desarrollo del proyecto, desde la **configuración inicial hasta las pruebas finales**. También voy a comentar algunos problemas que he tenido durante el proceso y cómo los he resuelto.

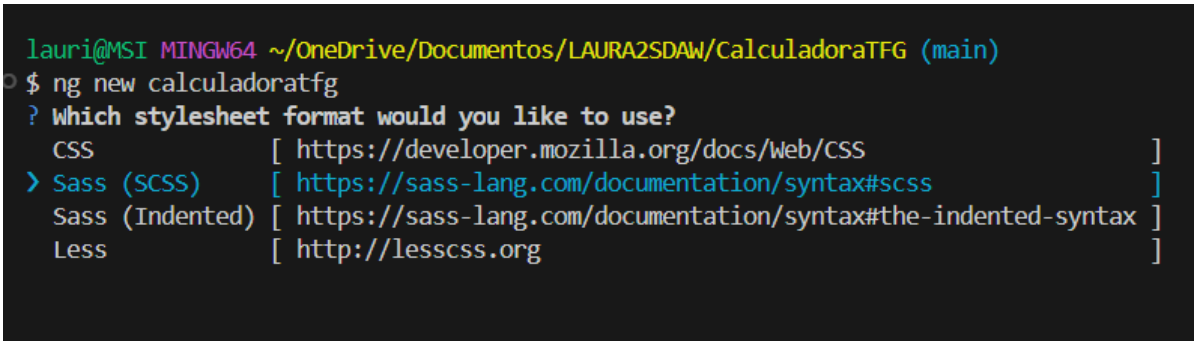
Capítulo 5.1: Preparación del entorno de trabajo

Para comenzar a preparar el entorno de trabajo de este proyecto, el primer paso que he hecho es **instalar Node.js** en la siguiente página: https://nodejs.org/es/download/.

Para continuar, he instalado **Angular CLI** con el siguiente comando: **npm install -g @angular/cli** tal y como muestro en la siguiente imagen:



Luego, he **creado un nuevo proyecto** con Angular, usando el siguiente comando:



Aquí, como se puede ver en la imagen, seleccioné la opción de usar SCSS como preprocesador de estilos, ya que quería trabajar de forma más organizada.

Después, inicié un repositorio de GitHub para ir guardando el progreso del proyecto.

En la siguiente imagen se puede ver qué comandos he utilizado para iniciar el repositorio de GitHub con Visual Studio Code:

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG
$ git init
Initialized empty Git repository in C:/Users/lauri/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/.git/

lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG (master)
$ git remote add origin https://github.com/LauraAre/CalculadoraTFG.git

lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG (master)
$ git branch -M main
```

Capítulo 5.2: Desarrollo de la interfaz

Empecé creando los componentes principales:

-Componente **header**:

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component components/header
CREATE src/app/components/header/header.component.html (22 bytes)
CREATE src/app/components/header/header.component.spec.ts (615 bytes)
CREATE src/app/components/header/header.component.ts (226 bytes)
CREATE src/app/components/header/header.component.scss (0 bytes)
```

-Componente **footer**:

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component components/footer
CREATE src/app/components/footer/footer.component.html (22 bytes)
CREATE src/app/components/footer/footer.component.spec.ts (615 bytes)
CREATE src/app/components/footer/footer.component.ts (226 bytes)
CREATE src/app/components/footer/footer.component.scss (0 bytes)
```

-Componente **theme-mode** (para el cambio de modo claro y modo oscuro):

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component components/theme-mode
CREATE src/app/components/theme-mode/theme-mode.component.html (26 bytes)
CREATE src/app/components/theme-mode/theme-mode.component.spec.ts (637 bytes)
CREATE src/app/components/theme-mode/theme-mode.component.ts (241 bytes)
CREATE src/app/components/theme-mode/theme-mode.component.scss (0 bytes)
```

-Componente **calculadora**:

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component components/calculadora
CREATE src/app/components/calculadora/calculadora.component.html (27 bytes)
CREATE src/app/components/calculadora/calculadora.component.spec.ts (650 bytes)
CREATE src/app/components/calculadora/calculadora.component.ts (246 bytes)
CREATE src/app/components/calculadora/calculadora.component.scss (0 bytes)
```

-Componente **como-funciona-cards** (para las cards informativas en la página "Cómo funciona"):

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component components/como-funciona-cards
CREATE src/app/components/como-funciona-cards/como-funciona-cards.component.html (35 bytes)
CREATE src/app/components/como-funciona-cards/como-funciona-cards.component.spec.ts (694 bytes)
CREATE src/app/components/como-funciona-cards/como-funciona-cards.component.ts (276 bytes)
CREATE src/app/components/como-funciona-cards/como-funciona-cards.component.scss (0 bytes)
```

-Componente **mensaje-error** (para lanzar un mensaje de error si la función de voz de la calculadora no funciona correctamente):

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component components/mensaje-error
CREATE src/app/components/mensaje-error/mensaje-error.component.html (29 bytes)
CREATE src/app/components/mensaje-error/mensaje-error.component.spec.ts (658 bytes)
CREATE src/app/components/mensaje-error/mensaje-error.component.ts (253 bytes)
CREATE src/app/components/mensaje-error/mensaje-error.component.scss (0 bytes)
```

-Componente **inicio** (para la página de Inicio):

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component paginas/inicio
CREATE src/app/paginas/inicio/inicio.component.html (22 bytes)
CREATE src/app/paginas/inicio/inicio.component.spec.ts (615 bytes)
CREATE src/app/paginas/inicio/inicio.component.ts (226 bytes)
CREATE src/app/paginas/inicio/inicio.component.scss (0 bytes)
```

-Componente **como-funciona** (para la página de Cómo Funciona):

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component paginas/como-funciona
CREATE src/app/paginas/como-funciona/como-funciona.component.html (29 bytes)
CREATE src/app/paginas/como-funciona/como-funciona.component.spec.ts (658 bytes)
CREATE src/app/paginas/como-funciona/como-funciona.component.ts (253 bytes)
CREATE src/app/paginas/como-funciona/como-funciona.component.scss (0 bytes)
```

-Componente **accesibilidad** (para la página de Accesibilidad):

```
lauri@MSI MINGW64 ~/OneDrive/Documentos/LAURA2SDAW/CalculadoraTFG/calculadoratfg (main)
$ ng generate component paginas/accesibilidad
CREATE src/app/paginas/accesibilidad/accesibilidad.component.html (29 bytes)
CREATE src/app/paginas/accesibilidad/accesibilidad.component.spec.ts (664 bytes)
CREATE src/app/paginas/accesibilidad/accesibilidad.component.ts (254 bytes)
CREATE src/app/paginas/accesibilidad/accesibilidad.component.scss (0 bytes)
```

Capítulo 5.3: Implementación de la lógica de la calculadora

La parte **lógica** de la calculadora la he desarrollado en el archivo calculadora.component.ts utilizando el lenguaje de programación **TypeScript**, a través de Angular.

En ese archivo, implementé las **operaciones básicas** (suma, resta, multiplicación y división) haciendo que con el reconocimiento de voz también sean reconocidas.

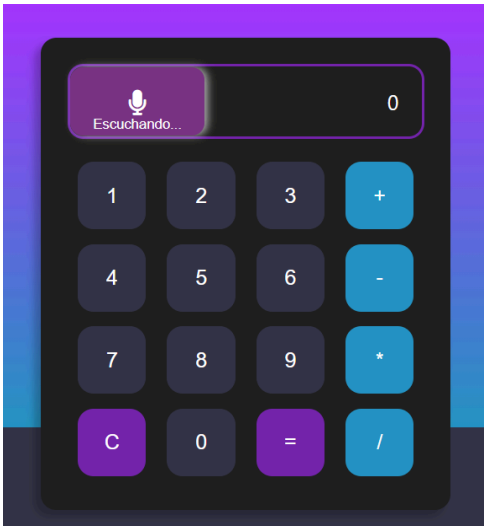
Luego conecté los **botones** a esas **funciones** con Angular en el archivo html de mi calculadora.

Capítulo 5.4: Integración del reconocimiento de voz

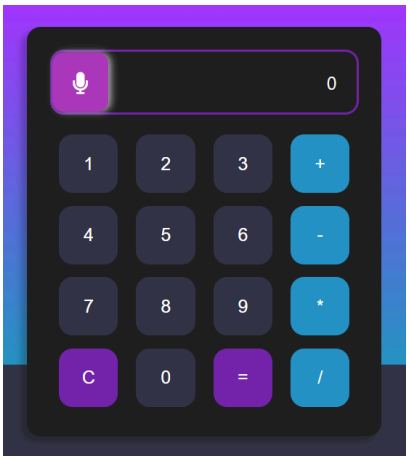
He utilizado una API que se llama **SpeechRecognition**, para interpretar órdenes dichas por voz.

Para **activar** el reconocimiento de voz, la calculadora cuenta con un **icono de un micrófono**, que si le haces **clic una vez**, te sale un mensaje abajo que pone: “Escuchando…” y se **desactiva volviendo a hacer clic** en este mismo.

En la siguiente imagen aparece la calculadora cuando el **micrófono** está **activado**:



En la siguiente imagen aparece la calculadora cuando el **micrófono** está **desactivado**:



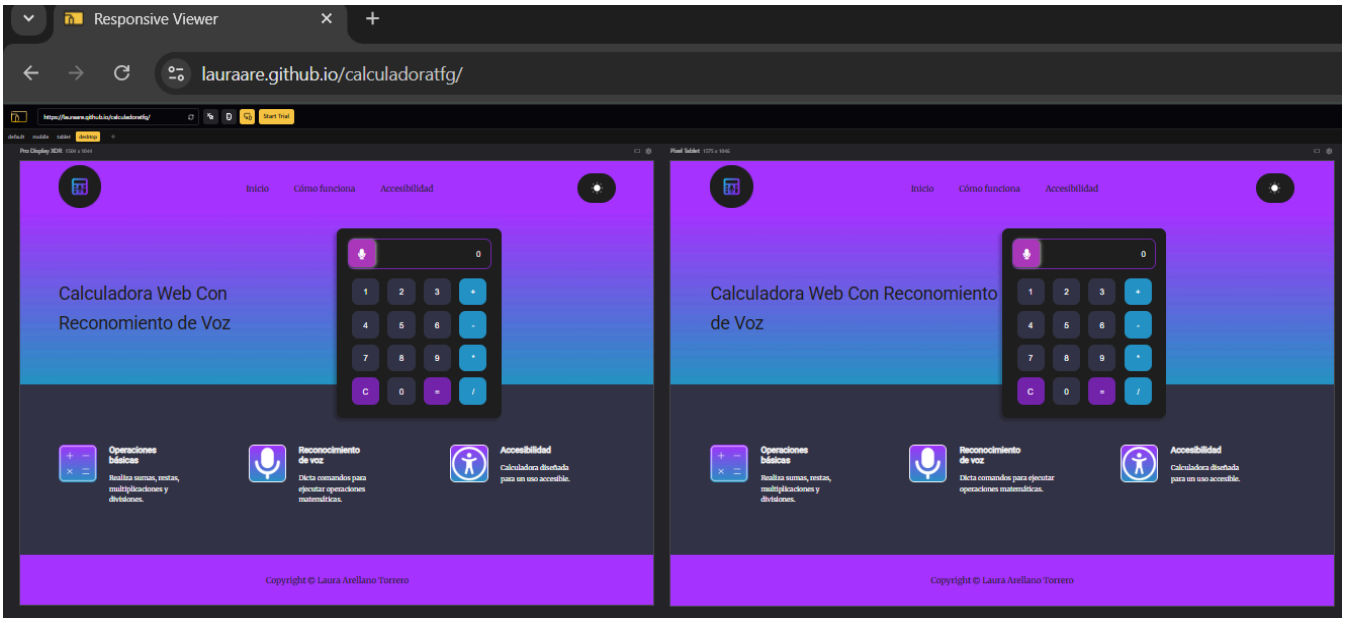
Capítulo 5.5: Diseño responsivo y temas oscuro/claro

El diseño de mi página lo he trabajado con **SCSS** y con **media queries** para hacer mi página responsiva para que la interfaz esté disponible en distintos dispositivos.

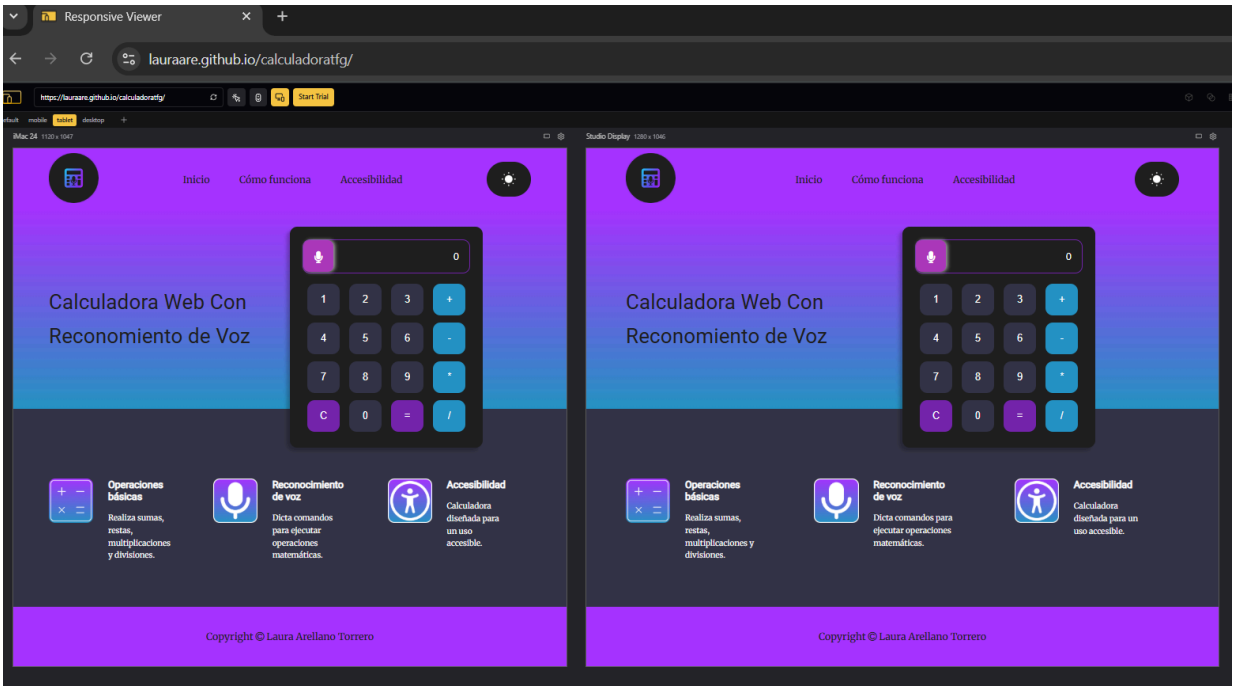
Para comprobar que mi página es responsiva, instalé una **extensión** de Google, que se llama **Responsive Viewer**.

En esta extensión, he añadido 3 “áreas de trabajo”: Desktop, Tablet y Mobile, para mostrar cómo se ven mis páginas en diferentes tamaños:

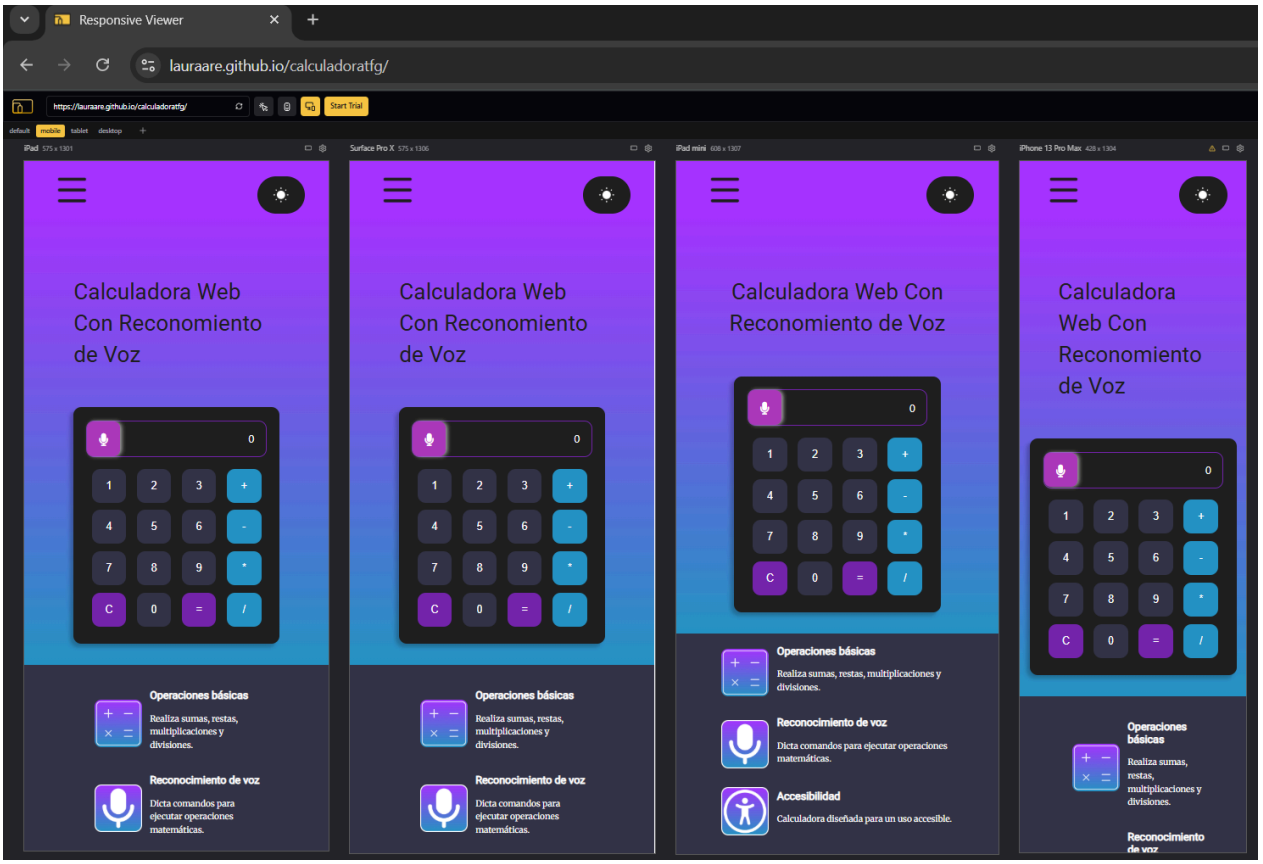
Desktop:



Tablet:



Mobile:



Para el **modo oscuro y el modo claro** se han usado clases llamadas: **light-mode y dark-mode en el archivo general de estilos scss**, y dentro de cada clase se han añadido las propiedades de cambios de colores entre las distintas páginas.

Capítulo 5.6: Problemas encontrados y soluciones aplicadas

Durante el desarrollo de mi aplicación, me encontré con algunos **problemas técnicos que fui resolviendo**. Cada uno resultó ser una oportunidad para **aprender** y **mejorar la lógica** del proyecto. A continuación detallo los **errores más relevantes** que me han ido surgiendo y cómo los fui **solucionando**:

-El primer problema que me surgió fueron los **enlaces** a **mi misma aplicación,** ya que usaba los enlaces de la misma manera que lo había hecho siempre (con a href) pero no me funcionaba, por la **falta de costumbre** con este lenguaje.

La **solución** fue muy sencilla: Aprendí que Angular utiliza su propio sistema de **enrutamiento**, y por eso se deben usar directivas como **routerLink** en lugar de **href**. Esto permite una **navegación más fluida**, sin recargar la página y haciendo que mejore el rendimiento.

En la siguiente imagen se pueden ver ejemplos de cómo usé routerLink en el header:

```
<nav class="item2-menu" [class.active]="menuOpen">
    <a routerLink="/inicio" routerLinkActive="active">Inicio</a>
    <a routerLink="/como-funciona" routerLinkActive="active">Cómo funciona</a>
    <a routerLink="/accesibilidad" routerLinkActive="active">Accesibilidad</a>
</nav>
```

-El segundo problema que me surgió fue el **control del micrófono**. En ciertos momentos, el micrófono se desactivaba de forma inesperada o no se podía detener correctamente.

Para solucionar esto, creé una **variable booleana** con el nombre "**microActivo**" que indica si el **micrófono** está **encendido o apagado**. La función **escucharVoz()** controla esa variable y decide si se debe iniciar (**start**) o detener (**stop**) el reconocimiento de voz. Así se **evita** que el micrófono **se esté usando cuando no se desea usar**.

En la siguiente imagen se puede observar la función escucharVoz():

```
microActivo: boolean = false;
escucharVoz() {
  if (!this.reconocimientoVoz) {
    this.mensajeError = 'Lo sentimos, el reconocimiento de voz no está disponible';
    return;
  }

  if (!this.microActivo) {
    this.reconocimientoVoz.start();
    this.microActivo = true;
    this.mensajeError = '';
  } else {
    this.reconocimientoVoz.stop();
    this.microActivo = false;
  }
}
```

-El siguiente problema que tuve era que cuando el usuario decía frases como: "**dos más dos**", la API **devolvía ese mismo texto**, pero eso **no se podía calcular** directamente.

Para solucionarlo, desarrollé una función llamada **convertirTexto()** que **transforma** los **términos hablados en** sus respectivos símbolos matemáticos: **+,-,*,/**.
También **eliminé espacios innecesarios** para que la operación pueda ser procesada correctamente:

```
// Convierto los operadores de texto escuchados, a símbolos matemáticos
convertirTexto(texto: string): string {
  return texto
    .toLowerCase()
    .replace(/más/g, '+')
    .replace(/menos/g, '-')
    .replace(/por/g, '*')
    .replace(/entre/g, '/')
    .replace(/ /g, ''); // Eliminar espacios en blanco
}
```

-Luego, he integrado una API llamada "**speechSynthesis**" con la idea de que el **resultado** de las operaciones matemáticas sean **dichas en voz alta**. Esta fue una de las últimas funcionalidades que incorporé en mi aplicación web ya que buscaba **facilitar** siempre el **acceso** a los resultados de las operaciones matemáticas a **personas mayores o con alguna discapacidad** motora. Esto al principio me daba algunos **problemas** porque había ocasiones en las que **el sistema no leía el resultado en voz alta**, especialmente en **dispositivos que no soportaban esta API**.

La solución fue añadir una **condición** que comprueba si esta **funcionalidad está disponible en el navegador**. Si **no lo está**, se muestra un **mensaje de error** al usuario indicando que el **navegador** que se está usando **no puede hablar**:

```
decirResultado(resultado: string) {
  if ('speechSynthesis' in window) {
    const mensaje = new SpeechSynthesisUtterance(`El resultado es ${resultado}`);
    mensaje.lang = 'es-ES'; // Idioma de la voz
    mensaje.rate = 1; // Velocidad de la voz
    mensaje.pitch = 1; // Tono de la voz
    window.speechSynthesis.speak(mensaje); // Dice el mensaje (la constante de arriba)
  } else {
    this.mensajeError = 'Lo sentimos, tu navegador no puede hablar.';
  }
}
```

Capítulo 6: Conclusiones y Líneas Futuras

Capítulo 6.1: Conclusión del proyecto

Durante el **desarrollo** de este proyecto, he trabajado principalmente con **Angular**, un **framework de desarrollo frontend**. Esta experiencia me ha permitido no solo construir una aplicación funcional y moderna, sino también **entender en profundidad** cómo se **estructura** una aplicación **en este entorno** y cómo se **diferencia de otras tecnologías** como **React**, con la que ya tenía algo de experiencia previa.

Además, este proyecto me ha dado la oportunidad de trabajar con **nuevas tecnologías** que no había explorado antes, como la **API de reconocimiento de voz** del navegador, o el uso de **speechSynthesis** para la **lectura en voz alta del resultado** de las operaciones.
También me he enfrentado a **retos reales** relacionados con **compatibilidad entre navegadores**, o **gestión de errores del micrófono**, que he tenido que **resolver con lógica** y **creatividad**. Estos desafíos me han aportado un **aprendizaje técnico**,y también me han hecho **mejorar** en la **capacidad de buscar documentación**, **analizar errores** y aplicar **soluciones eficientes**.

Otra conclusión que quiero destacar es que, más allá de la parte técnica, este proyecto me ha ayudado a desarrollar **habilidades de planificación**, **análisis de requisitos** y **toma de decisiones**. Desde el diseño inicial hasta la implementación final, he tenido que pensar como una **desarrolladora profesional**: **estructurar el código**, **cuidar la experiencia de usuario**, **probar funcionalidades**, **corrección de errores**, y mantener siempre un **enfoque claro** hacia los **objetivos** del proyecto.

En resumen, me siento **satisfecha con el resultado** final, y al mismo tiempo motivada para **seguir aprendiendo**, profundizar en frameworks como Angular, y ampliar este proyecto con **futuras mejoras** o incluso **nuevas versiones** que puedan utilizarse en contextos reales.

Capítulo 6.2: Valoración sobre Angular

Angular me ha parecido un framework muy **completo e interesante**. Al estar basado en **TypeScript**, proporciona una estructura que ayuda a mantener el **código ordenado** y fácil de usar, lo que es **ideal** para **proyectos medianos y grandes**. Una de las cosas que más me ha gustado es el **sistema de componentes y módulos**, ya que permite **dividir** la aplicación en **partes independientes y reutilizables**.

También quiero destacar el **sistema de enrutamiento (RouterModule)**, que permite crear una **navegación fluida** entre páginas sin necesidad de recargar la aplicación. Esto no solo **mejora el rendimiento**, sino que también mejora la **experiencia del usuario**.
Las directivas estructurales, como **ngIf o ngFor**, también han sido de gran utilidad para **manipular el DOM**. Permiten **mostrar o esconder elementos**, de una manera clara y rápida.

Uno de los puntos que al principio me costó fue adaptarme a su estructura más rígida y al uso intensivo de **@Component**, **@NgModule**. Venía de trabajar con React, donde la estructura es mucho más flexible y menos marcada, así que este cambio me obligó a **prestar más atención** en la forma en que **organizaba mis archivos** y funcionalidades.
Sin embargo, con el tiempo comprendí que esa "rigidez" o **dificultad**, en realidad es una **gran ventaja en proyectos grandes**, ya que impone buenas prácticas desde el principio y **evita** que el **código** sea demasiado **extenso o menos entendible** con el paso del tiempo.

Otra característica que me ha gustado de Angular es su **integración** con **herramientas modernas de desarrollo** como el **Angular CLI,** que **simplifica** mucho tareas como la **creación de los componentes** con tan solo un comando o la **construcción del proyecto**.

Capítulo 6.3: Angular vs React

Durante el desarrollo de este proyecto con **Angular**, también he tenido presente mi experiencia con **React**, por lo que me ha parecido interesante **compararlos** y **analizar sus principales diferencias**. Aunque ambos se utilizan para desarrollar interfaces de usuario modernas y dinámicas, existen diferencias clave en su estructura de trabajo.

En primer lugar, **Angular es un framework completo**, mientras que **React es una librería de JavaScript** enfocada únicamente en la construcción de interfaces de usuario. Esto implica que **Angular** viene con **muchas funcionalidades integradas** por defecto, como el enrutamiento, el sistema de inyección de dependencias, formularios, validaciones y mucho más. **React**, por el contrario, **requiere que el desarrollador vaya incorporando otras librerías externas** si desea construir una aplicación más compleja (por ejemplo, para el manejo de rutas se suele utilizar React Router).

Una **diferencia importante** es el lenguaje que utilizan por defecto. **Angular** se basa en **TypeScript**, un superconjunto de JavaScript que añade tipado estático y otras características orientadas a la programación estructurada. **React**, en cambio, **está basado en JavaScript, aunque también permite usar TypeScript** si se desea. En mi experiencia, el **uso de TypeScript en Angular** obliga a mantener una **mejor organización del código**, algo que valoro positivamente ahora que he trabajado en un proyecto un poco más grande.

En cuanto a el **aprendizaje**, considero que **Angular es un poco más exigente** al principio. Requiere entender conceptos como módulos, componentes, servicios, directivas estructurales (ngIf, ngFor…) y el sistema de enrutamiento. **React es más ligero** en este sentido, y **permite empezar a programar** una interfaz **con muy poco código**, lo cual lo hace más accesible para principiantes. Sin embargo, esa "**simplicidad**" inicial de React puede **volverse compleja** cuando el **proyecto crece**, ya que depende de muchas decisiones del desarrollador sobre cómo estructurar el código y qué librerías usar.

Otra diferencia importante está en el **manejo del DOM**. **React utiliza el DOM virtual**, lo que permite actualizar la interfaz de forma muy eficiente. **Angular**, en cambio, **trabaja directamente con el DOM real**, aunque lo hace de forma optimizada. En la práctica, no he notado grandes diferencias de rendimiento en este proyecto, ya que no es especialmente pesado, pero es un factor a considerar en aplicaciones con muchas actualizaciones en tiempo real.

En resumen, mi experiencia me lleva a pensar que **React** es ideal para **proyectos más pequeños o rápidos**, donde se busca flexibilidad y agilidad. En cambio, **Angular** es más recomendable para **proyectos medianos o grandes**, donde se necesita una estructura sólida desde el inicio, y donde tener todas las herramientas integradas facilita el mantenimiento.

Para **complementar la explicación** anterior y **facilitar una visión más rápida** y clara de las principales diferencias entre ambos enfoques, a continuación presento una **tabla comparativa** que resume los aspectos más relevantes entre **Angular y React**:

Características	Angular	React
Tipo	Framework completo	Librería centrada en Interfaz de Usuario (UI)
Lenguaje base	TypeScript	JavaScript (aunque puede usarse TypeScript)
Enrutamiento	Incluido	Requiere librería externa (React Router)
Aprendizaje	Compleja al empezar	Más accesible al empezar
Organización del proyecto	Muy estructurado y guiado	Flexible (a elección propia)
DOM	DOM real con optimizaciones	Virtual DOM
Escalabilidad	Muy buena para proyectos grandes	Ideal para proyectos pequeños-medianos
Documentación	Amplia, pero menos flexible	Más soluciones y librerías

Capítulo 6.4: Mejoras futuras del proyecto

A pesar de haber alcanzado un funcionamiento satisfactorio de la aplicación, existen muchos aspectos que podrían mejorarse o ampliarse para ofrecer una experiencia más completa y accesible. A continuación expongo algunas de las mejoras futuras que me gustaría implementar:

-Incorporar expresiones más difíciles en el reconocimiento de voz.

La calculadora está preparada para interpretar operaciones básicas como sumas, restas, multiplicaciones y divisiones. Una mejora importante sería permitir el uso de **potencias, raíces cuadradas o funciones** matemáticas como **seno, coseno o logaritmos**.

-Historial de operaciones realizadas.

Otra funcionalidad interesante sería incluir un sistema de **historial de operaciones**, donde el usuario pueda ver las **últimas operaciones que ha realizado**, **repetirlas** o incluso **editarlas**. Esto podría **almacenarse** localmente mediante **localStorage**, lo que daría más **control al usuario** sin necesidad de una base de datos externa.

-Multilenguaje.

Aunque actualmente el idioma de uso es el español, Angular permite implementar con facilidad **soporte multilenguaje**. Sería buena idea poder ofrecer la aplicación en **inglés o francés**, con **textos y comandos adaptados** para cada idioma. Esto requeriría adaptar tanto la lógica, como la interfaz.

-Integración con bases de datos para estadísticas de uso.

Si en un futuro quisiera **publicar la aplicación** y conocer cómo la **usa** la gente, podría implementar una **conexión con Firebase o MongoDB** para almacenar métricas de uso: **cuántas operaciones se realizan** al día, cuáles son las **más frecuentes**, desde qué dispositivos se accede…

-**Modo accesible para personas con discapacidad.**

El proyecto ya incluye funcionalidades pensadas para la accesibilidad, como la **lectura en voz alta del resultado**. Sin embargo, podría implementarse en un futuro otras mejoras como:

Lectura en voz alta de la página

En la calculadora, **lectura en voz alta** de **cada número** o cada **operando** que se **esté pulsando.**

-Posibilidad de operar con el teclado.

Sería buena idea que la calculadora contase con la posibilidad de poder **escribir con el teclado** de tu propio ordenador o el teclado de tu propio móvil la **operación a realizar.**

-Exportación de operaciones

La última función que me ha parecido interesante podría ser permitir al usuario **exportar el historial de operaciones en formato PDF** o **imprimirlo** directamente desde la aplicación. Esto puede resultar útil en entornos educativos donde se necesita guardar los cálculos realizados.

Capítulo 7: Bibliografía

PÁGINAS WEB CONSULTADAS:

-**Instalación de Node.js** - https://nodejs.org/es/download/ Fecha de consulta: 12/04/2025

-**Configuración de proyecto Angular** - Fecha de consulta: 14/04/2025

VÍDEOS CONSULTADOS:

-**Angular desde 0** -https://youtu.be/f7unUpshmpA?si=_TvlhjAtK8zuy5MX

-**Componentes STANDALONE**: https://youtu.be/FUXM-Qxvfqg?si=l1K8nSSgSxsM8TEN

-**Angular vs React**: https://youtu.be/pqgLl7ja9Tc?si=CVN1OH4C9tMeonn_