



Tutor: Álvaro Domínguez Cabrera

Índice

1. Resumen.....	3
2. Justificación.....	3
a. Características generales.....	3
b. Restricciones generales.....	3
c. Aspectos a cubrir, junto con los que no se van a tratar.....	4
d. Estudio de las prestaciones de la herramienta.....	4
3. Justificación de la tecnología empleada.....	4
4. Requerimientos hardware y software.....	5
5. Análisis y diseño.....	7
a. Diagrama de casos de uso.....	7
b. Diagrama de clases.....	8
c. Descripciones de las bases de datos.....	8
7. Implementación.....	11
8. Evaluación y prueba.....	12
9. Manual de estilos:.....	13
a. Mockup.....	13
b. Criterios de accesibilidad.....	23
c. Criterios de usabilidad.....	23
d. Tipografía.....	24
e. Mapa de colores y elementos donde se aplica.....	24
f. Dispositivos / vistas y tecnología usada.....	25
10. Softwares utilizados.....	26
11. Mejoras posibles y aportaciones.....	27
12. Bibliografía.....	27

1. Resumen

Anime API es una REST API monorrepo multipaquete, desarrollada con el patrón de arquitectura MVC. El objetivo es llegar a aquellos desarrolladores que quieran realizar un proyecto a través de una API, ya sea como hobby o para aprender. La API puede hacer las operaciones básicas conocidas como CRUD (crear, leer, actualizar y eliminar). En este proyecto se ha realizado con la intención de desarrollar una API, aprender los conceptos relacionados y de paso aprovechar para aprender y desarrollar con TypeScript y Express. Además, de ofrecer un API para animes, las cuales no he conseguido encontrar ninguna en español y casi ninguna en inglés con estas características.

También se ha tenido en cuenta realizar una web con Astro, React y TypeScript. Su funcionalidad es poder explicar todo lo relacionado a la hora de poder hacer peticiones, cuales son sus endpoints y qué respuesta debería devolver la API según la petición. Además, se ha incluido una página la cual permite hacer unos primeros pasos para entender cómo funciona y así tener una forma más gráfica para aprender sus endpoints.

2. Justificación

a. Características generales

La API REST de animes, géneros y usuarios se ha desarrollado con el propósito de proporcionar una plataforma sencilla y accesible para la gestión y poder consultar información relacionada con animes. Las características generales son:

- Interfaz RESTful: la API sigue los principios REST, ya que, garantiza una fácil integración con otros sistemas.
- CRUD: Soporta operaciones de lectura para animes, géneros y usuarios, y de creación, lectura, actualización y eliminación para animes y usuarios.
- Documentación: en la web creada se basa en explicar el funcionamiento y cómo deben hacerse las peticiones.

b. Restricciones generales

Durante el desarrollo del proyecto se han identificado las siguientes restricciones:

- Limitación de conocimientos: Al iniciar un proyecto con software desconocido, hubo que leer mucha documentación y realizar varias pruebas.
- Limitación temporal: al ser un proyecto que ha tenido que ser realizado en dos meses y medio, y si le sumamos el punto anterior mientras se hacía las prácticas, se ha notado una falta de tiempo.
- Compatibilidad de navegadores: aunque se intentado alcanzar al 100% de los navegadores, por desgracia se ha llegado hasta un 97%. No se garantiza compatibilidad con versiones muy antiguas o con navegadores poco conocidos.

c. Aspectos a cubrir, junto con los que no se van a tratar

Aspectos a cubrir:

- Gestión de animes: Creación, consulta, actualización y eliminación de registros de animes.
- Gestión de géneros: Administración de géneros, permitiendo asociarlos a los animes para luego poder hacer una búsqueda a partir de un género.
- Gestión de usuarios: Creación, consulta, actualización y eliminación.
- Documentación de la API: descripción detallada de cada endpoint, métodos disponibles y ejemplo de uso.
- Explorar la API: permite hacer llamadas a través de una interfaz gráfica.

Aspectos que no se van a cubrir:

- Paginación de los animes, géneros o usuarios.
- Autenticación.
- Inicio de sesión: pueden hacer un número ilimitado de consultas.

d. Estudio de las prestaciones de la herramienta

En el estudio previo a realizar el proyecto, me di cuenta que las herramientas existentes no son muy claras en la documentación. Por lo tanto, me puse a pensar cómo podía hacer que no ocurriese lo mismo en la mía. Fue ahí cuando decidí hacer una documentación diferencial, donde cualquiera que quisiera utilizar la API tuviese una línea de aprendizaje exponencial.

Además se le añadió una página concreta (Explorador de la API) para poder ver de una forma más vistosa cómo utilizar la API, ya que, ninguna de las herramientas existentes vistas traen una opción así. También, en la documentación viene bien explicada la utilización de la página para que sea clara y concisa.

Otra prestación añadida, por si alguien quiere probar el código, es obtener todos los datos. Es decir, a partir de un archivo con extensión JSON (animes, géneros y usuarios) se pueden obtener los datos e introducirlo en las bases de datos tanto en la de MySQL como en MongoDB.

3. Justificación de la tecnología empleada

Tecnologías empleadas:

- Node.js: opción popular para desarrollar APIs debido a su velocidad y un gran ecosistema de paquetes.
- Express: la facilidad para el desarrollo de aplicaciones web y APIs con una estructura minimalista y flexible.
- TypeScript: añade tipado estático al código JavaScript, mejorando la seguridad y mantenibilidad del código. Permite detectar errores en tiempo de compilación y facilita la escalabilidad de la API.

- ts-node-dev: levantar el servidor sin necesidad de transpilar manualmente el código, mejora la productividad y acelera el ciclo de desarrollo.
- cors: permitir solicitudes de recursos entre diferentes dominios.
- dotenv: facilitar la gestión de variables de entorno.
- MongoDB (driver): paquete de npm para facilitar la conexión y operación de la base de datos desde Node.js.
- MongoDB Compass: proporcionar una interfaz gráfica para gestionar y visualizar la base de datos en local. Facilita las tareas de administración y desarrollo.
- MongoDB Atlas: base de datos en la nube altamente escalable.
- MySQL: gestionar las bases de datos relacionales en local.
- mysql2: paquete de npm, que proporciona una interfaz eficiente y moderna para conectar y realizar operaciones con bases de datos MySQL desde Node.js.
- uuid: validar identificadores únicos universales (UUID), así se verifica que las ID cumplan con el patrón esperado.
- zod: permite definir y validar esquemas de datos de manera declarativa, garantizando que la entrada de animés y usuarios cumplan con las especificaciones deseadas.
- Astro: framework web agnóstico a la UI y utiliza arquitectura basada en componentes. Además, no carga JavaScript por defecto, es decir, por defecto al cliente no se envía JS.
- React: su enfoque en componentes que permite desarrollar web dinámicas.
- react-toastify: para implementar notificaciones emergentes.
- tailwindcss: framework CSS que proporciona clases de utilidad y acelera el desarrollo.
- git: esencial para el control de versiones.
- GitHub: para alojar el proyecto.
- npm: gestionar los paquetes.
- npx: instalar funcionalidades en Astro.
- Render: plataforma para desplegar en la nube la API.
- Vercel: plataforma para desplegar en la nube la web.
- Postman: utilizada para probar la web y la API, permitiendo realizar y automatizar pruebas.
- Animaciones Tailwind Midudev: utilizado para implementar animaciones con tailwind.

4. Requerimientos hardware y software

Requerimientos hardware y software en el cliente:

- Hardware:
 - Un dispositivo, ya sea, móvil, tableta u ordenador.
 - Conexión de red.
- Software para utilizar la web:
 - Sistema operativo.
 - Navegador.

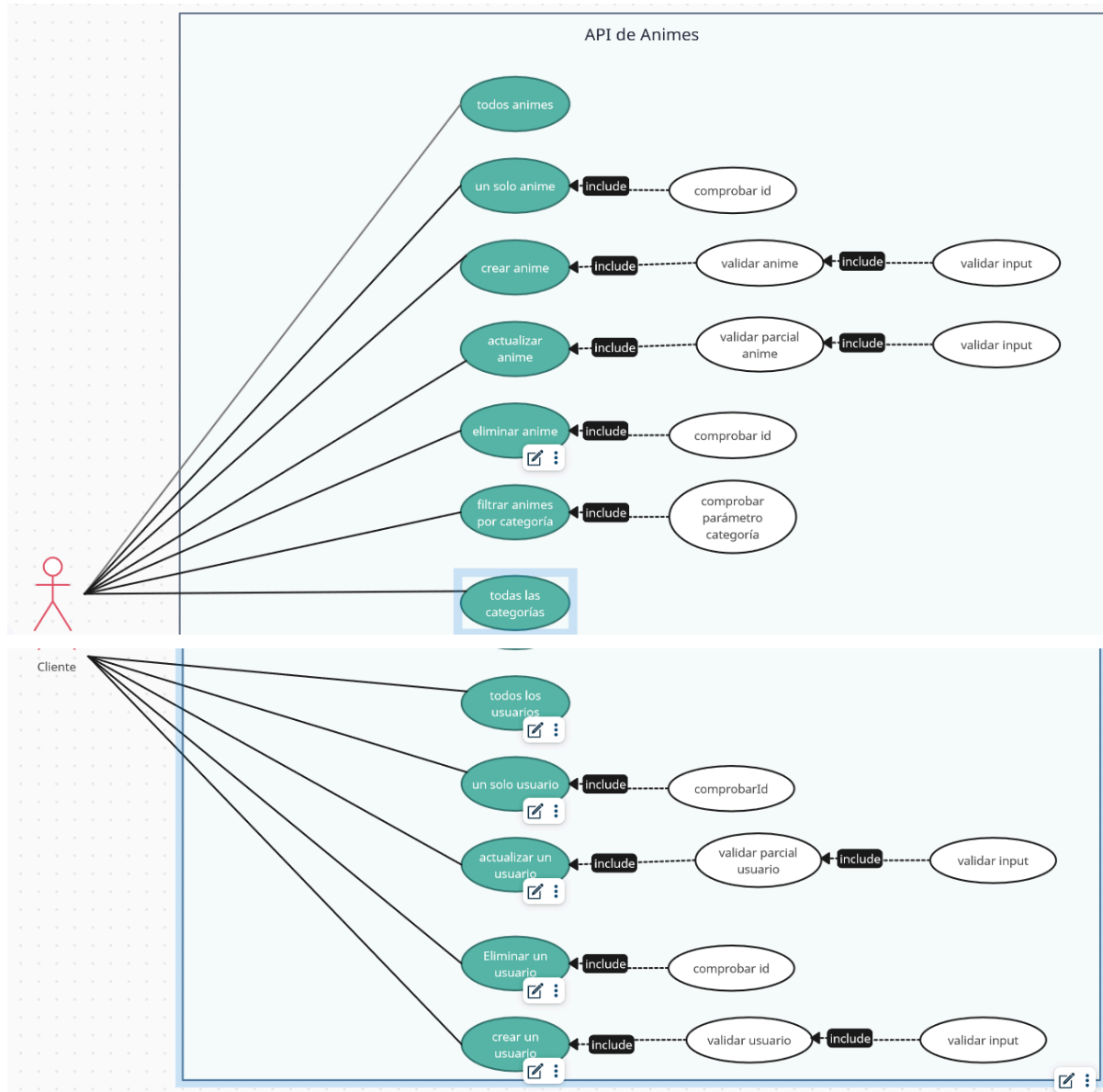
- Software para utilizar la API:
 - Sistema operativo.
 - Editor de código.
 - Terminal.
 - Cualquier otro que pueda hacer peticiones.

Requerimientos hardware y software en el servidor (Vercel y Render):

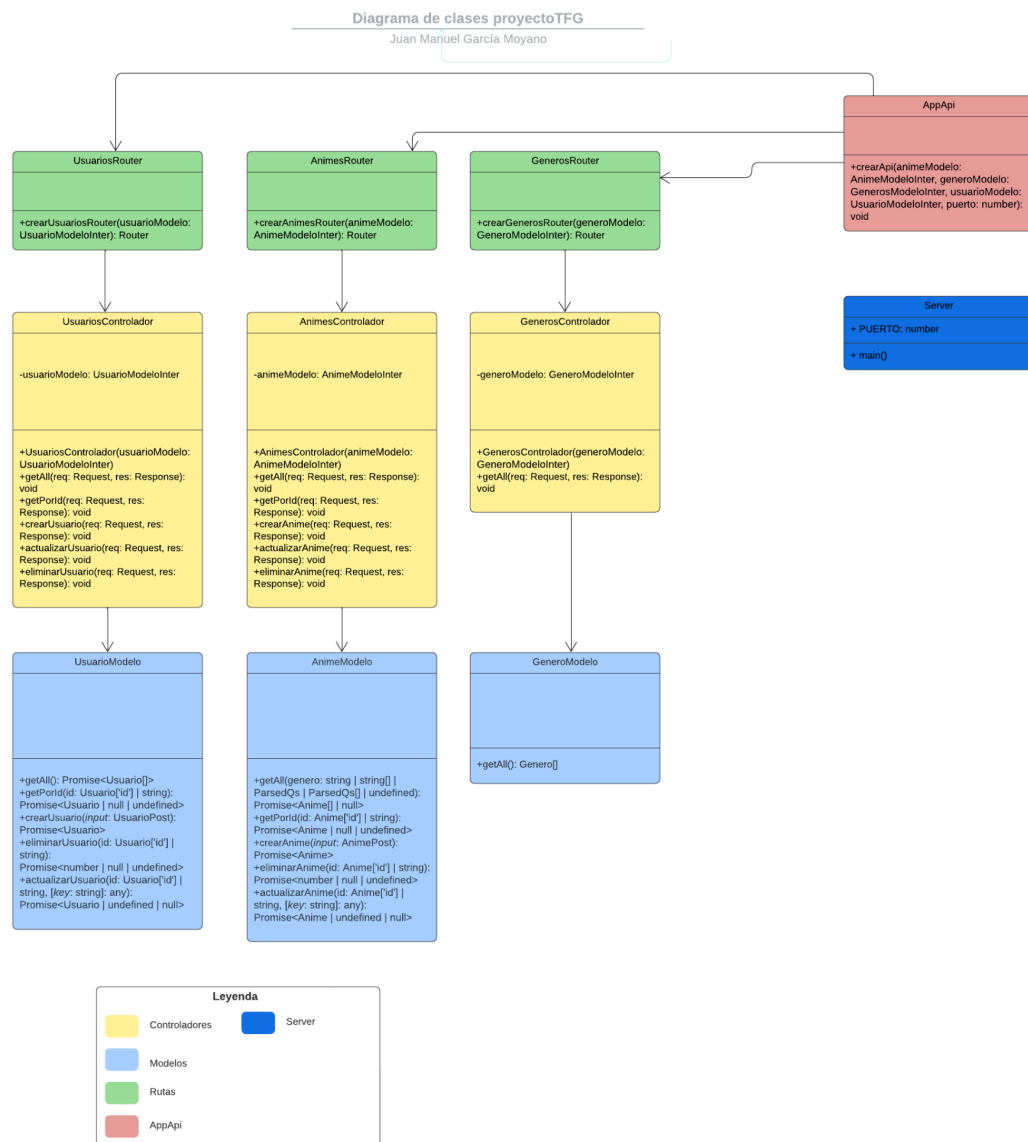
- Hardware:
 - Almacenamiento.
 - Conexión de red.
 - Sistema operativo.
- Software:
 - Node.js versión 20.x
 - npm 10.x

5. Análisis y diseño

a. Diagrama de casos de uso



b. Diagrama de clases



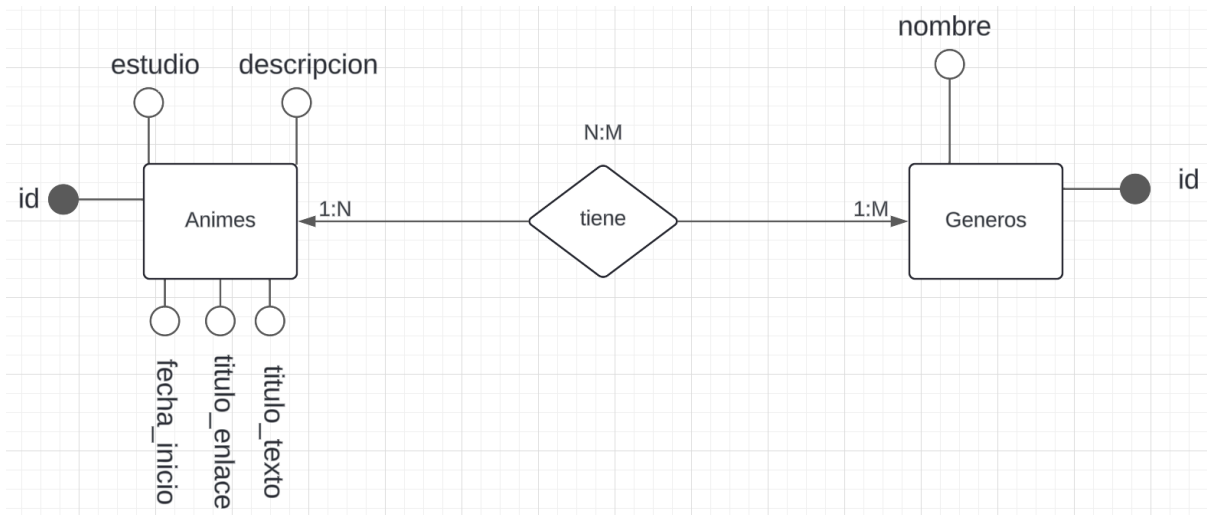
c. Descripciones de las bases de datos

La API se ha realizado con 3 bases de datos, las cuales se van a poder cambiar entre ellas mediante inyección de dependencias. Las bases de datos van a ser las siguientes:

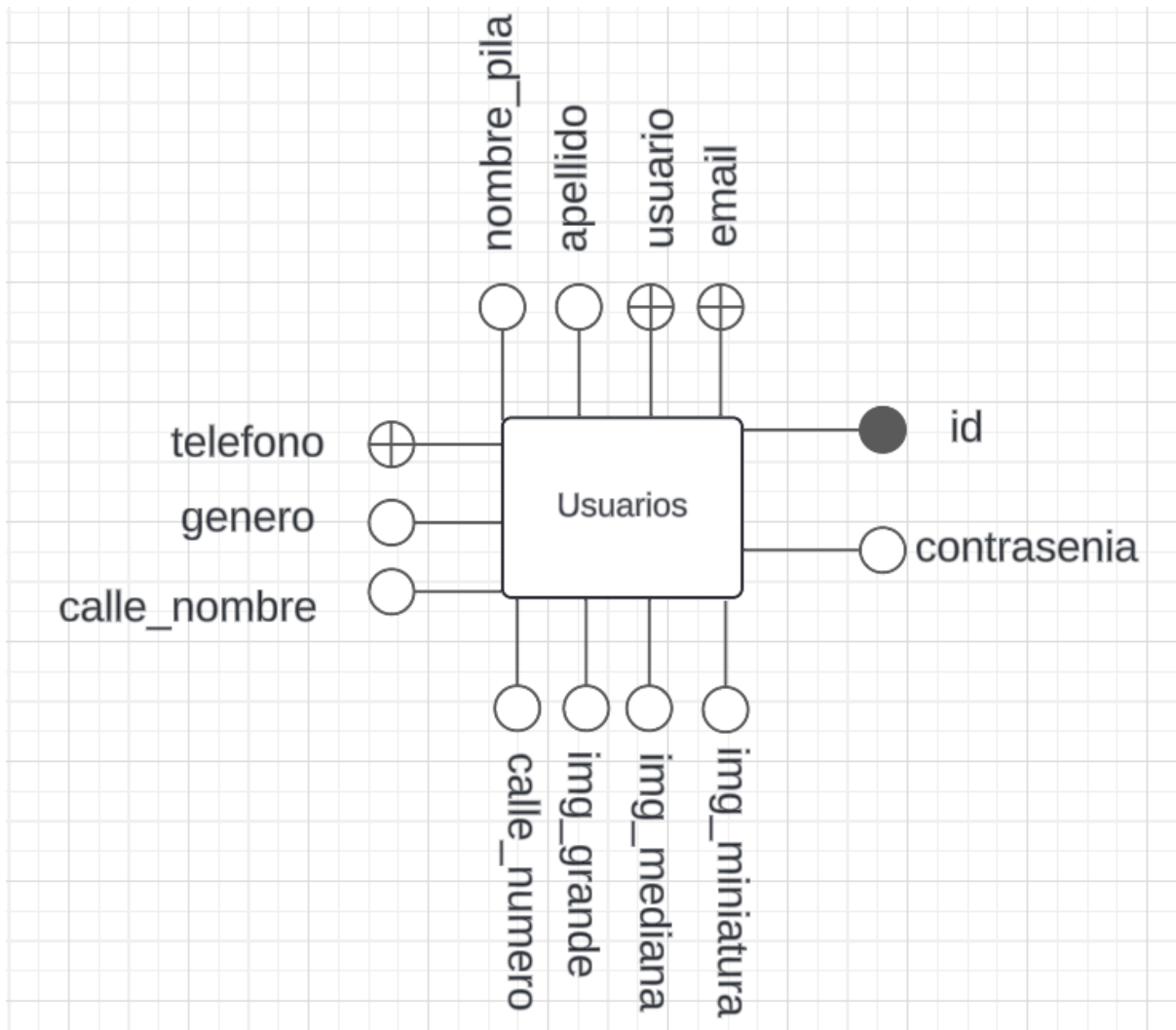
- Local
- Relacional
- No relacional

Pasamos a describir cada una de las bases de datos:

- Local: tiene 3 ficheros JSON y cada uno, un array con objetos que a su vez tendrá animes, géneros o usuarios.
- SQL: se crearán 4 tablas una para los animes, otra para los géneros, otra para la relación entre animes / géneros y otra para los usuarios.



Por último los usuarios, que será otra tabla.



- c. NoSQL: una base de datos noSQL, los datos se almacenan en documentos, por lo tanto hay tres documentos con animes, géneros o usuarios. Cada elemento de los documentos tendrá un `_id`, el cuál mongoDB lo genera automáticamente.

i. Estructura del documento de Anime:

```
_id: ObjectId('664f745bb3aa1dc586cf832d')
id: "59b57db3-122c-4c47-9201-7e8ca06e3d28"
estudio: "Wit estudio"
▶ generos: Array (5)
descripcion: "A medida que el mundo está en medio de una revolución industrial, apar..."
▶ titulo: Object
fechaInicio: "08/04/2016"
```

• Campos:

- id: identificador único del anime.
- estudio: Nombre del estudio de animación.
- descripcion: Descripción detallada del anime.
- fecha_inicio: Fecha de estreno del anime.
- titulo_enlace: Enlace para más información sobre el anime.
- titulo_texto: Título del anime.

ii. Estructura del documento de Género:

```
_id: ObjectId('664f7519b3aa1dc586cf836f')
id: "ebd8e888-ea8f-46f0-9bf2-4b59a85bfaa6"
nombre: "Misterio"
```

• Campos:

- id: identificador único del género.
- nombre: Nombre del género.

iii. Estructura del documento de Usuario:

```
_id: ObjectId('664f7313b3aa1dc586cf830f')
id: "3fd2718e-d2b3-4e1c-a6a6-1d8875144b95"
▶ nombre: Object
usuario: "redlion811"
email: "savitha.pai@example.com"
contrasena: "micheal"
telefono: "7561111311"
genero: "female"
▶ calle: Object
▶ imagen: Object
```

6. Campos:

- a. id: identificador único del usuario.
- b. nombre_pila: Nombre de pila del usuario.

- c. apellido: Apellido del usuario.
- d. usuario: Nombre de usuario para el login.
- e. email: Dirección de correo electrónico.
- f. contraseña: Contraseña para el login.
- g. telefono: Número de teléfono de contacto.
- h. genero: Género del usuario.
- i. calle: Objeto que contiene el número y nombre de la calle.
- j. imagen: Objeto que contiene tres imágenes que son grande, mediana y miniatura.

7. Implementación

Para el maquetado (CSS) se ha utilizado Tailwind un framework CSS. Además, astro implementa la etiqueta `style` que detecta el CSS e incluirá los estilos automáticamente, por lo tanto, no se ha utilizado por defecto hojas de estilos. Hay que decir que se han creado dos para los componentes de React (explorador y documentación).

La web tiene un formulario, este ha sido desarrollado para probar la API y que los usuarios puedan hacer pruebas sin necesidad de ejecutar código. En este formulario tienen la opción de escoger entre GET, POST, PATCH y DELETE. Además, tiene una entrada de tipo texto que permite recoger información de la petición (endpoint). También, con los métodos POST Y PATCH pueden añadir parámetros para añadirlos al cuerpo de la petición. Antes de enviar la petición a la API se hacen comprobaciones de coherencia de los datos. Luego la API validará dichos datos mediante un esquema.

Las conexiones a las bases de datos se hacen mediante los drivers pertinentes (mysql y mongodb). Una vez realizada la conexión se hace la consulta requerida, cada base de datos desarrolla la consulta dependiendo del lenguaje. Para la eliminación, creación y actualización, no se ejecutan dichas acciones, lo que se hace es proceder a una simulación de la acción. Las consultas son las siguientes:

- Conseguir todos los elementos: obtiene todos las tuplas de la tabla/documento indicada.
- Conseguir los datos de un anime: se busca una tupla mediante un id.
- Actualizar un elemento: se obtiene el id proporcionado y se consiguen los datos del elemento indicado.
- Eliminar un elemento: mediante el id proporcionado, se obtiene con la consulta los datos del elemento.

Tanto el frontend como el backend tienen un fichero de configuración. En el del frontend se utiliza para indicar la URL que se debe usar y en el backend el puerto que debe usar la API mediante el desarrollo.

8. Evaluación y prueba

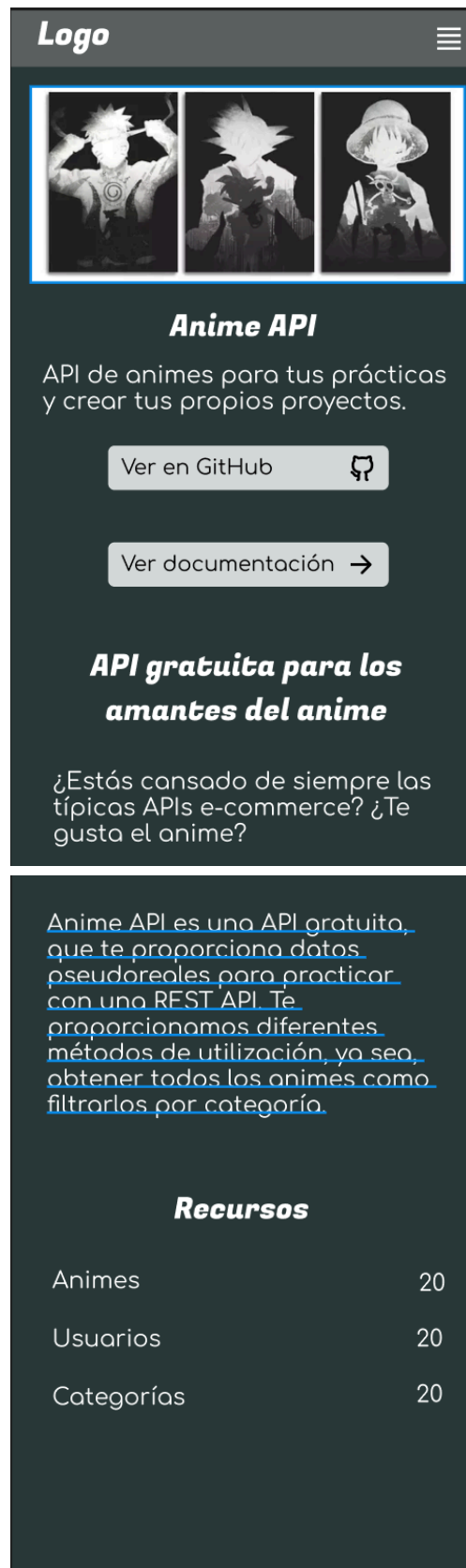
La evaluación de la API REST se llevó a cabo utilizando la herramienta Postman, que permitió realizar pruebas exhaustivas en los diferentes endpoints de la API. Además de unos ficheros con extensión http para que las pruebas fueran más automatizadas. Se han realizado las siguientes evaluaciones y pruebas:

- Pruebas de endpoints:
 - GET: se han probado las solicitudes de obtención de datos de animes, géneros y usuarios.
 - POST: se han realizado pruebas de creación de nuevos registros para animes y usuarios, con datos erróneos para ver el resultado y datos correctos para mostrar que lo crea correctamente.
 - PUT: se han realizado pruebas para la actualización de animes y usuarios, con datos erróneos para ver el resultado y datos correctos para mostrar que lo actualiza correctamente.
 - DELETE: se les ha pasado datos erróneos para comprobar que devuelve el resultado esperado y datos correctos esperando que el resultado sea el esperado, tanto para animes como para usuarios.
- Validación de campos y retroalimentación para garantizar la integridad y consistencia de datos:
 - Tipos de datos: para asegurar que los campos reciban el tipo correcto.
 - Atributos correctos: verificación que el objeto pasado al crear o actualizar un anime o usuario tiene los atributos requeridos antes de procesar la solicitud.
 - Mensajes de error: En caso de errores o validaciones fallidas, se proporciona mensajes error, junto con los códigos de estado HTTP correspondientes.
- Pruebas de la base de datos:
 - Datos de prueba: se utilizaron datos de prueba específicos, tanto correctos como incorrectos, para validar las respuestas de la API. Se simuló con datos válidos y no válidos intencionalmente para verificar que la API respondiera adecuadamente a entradas no válidas.
- Pruebas funcionales y de usuario:
 - Pruebas funcionales de extremo a extremo: se han realizado pruebas completas para asegurarse que todas las funcionalidades principales de la API funcionaran correctamente desde el punto de vista de un usuario final.
 - Pruebas de usabilidad: dos personas realizaron pruebas de usabilidad, utilizando la API para llevar a cabo tareas comunes y proporcionando feedback sobre su experiencia.
- Cobertura de pruebas:
 - A pesar de no haber realizado pruebas unitarias debido a las limitaciones de tiempo, se intentó alcanzar una cobertura completa mediante pruebas manuales y funcionales.
 - Coberturas de código: se realizaron pruebas manuales exhaustivas para intentar cubrir todas las rutas y casos de usos posibles, garantizando que la API respondiera correctamente en diferentes situaciones.


9. Manual de estilos:


a. Mockup

- Móvil:
 - Inicio:



- Documentación:


Logo 

Lugar  Sección que se encuentra

Título

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec lorem metus, tincidunt vitae mi elementum, suscipit maximus tellus.


Petición:


Código de la petición 

Mostrar respuesta

Título

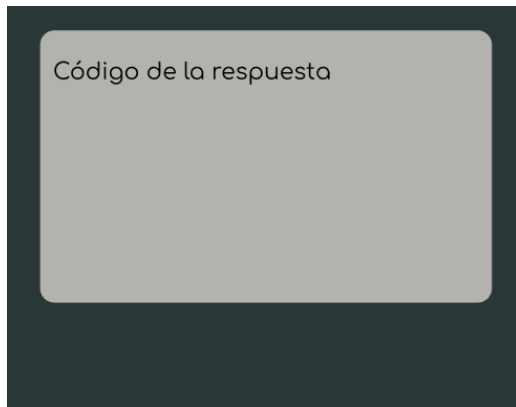
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Donec lorem metus, tincidunt vitae mi elementum, suscipit maximus tellus.

Código de la petición 

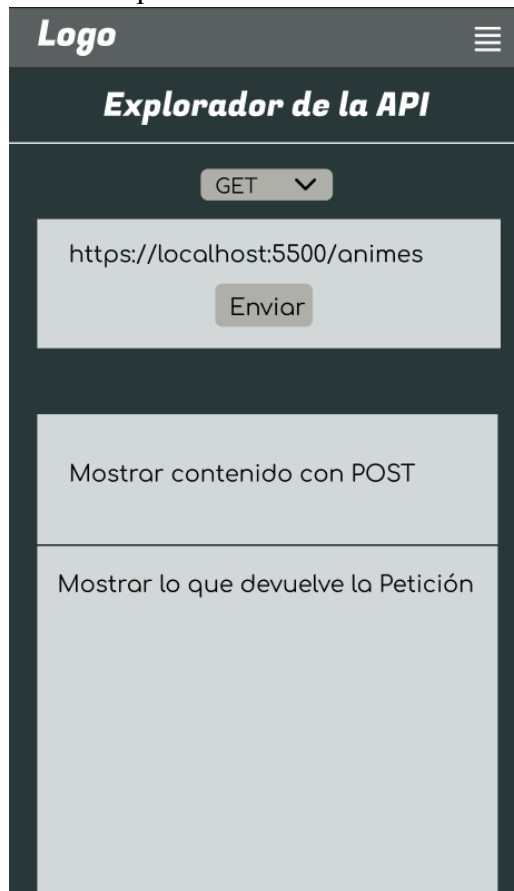
 Texto para advertir

Ocultar respuesta

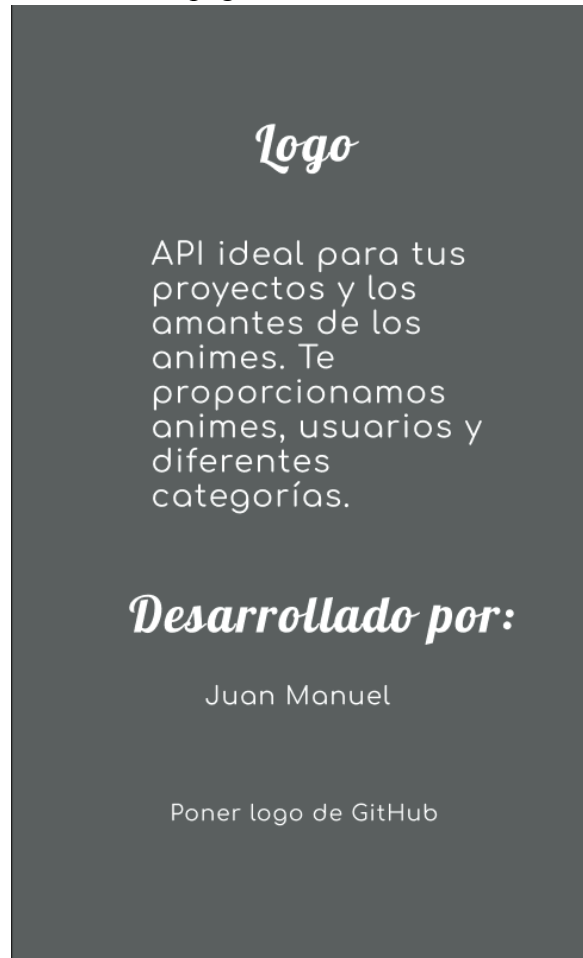
Petición:



- Explorador de la API:



- Pie de página:



- Tableta:
 - Inicio:

Logo[Inicio](#)[Documentación](#)[Explorador de la API](#)



Anime API

API de animes para tus prácticas y crear tus propios proyectos.

[Ver en GitHub](#) [Ver documentación](#)

API gratuita para los amantes del anime

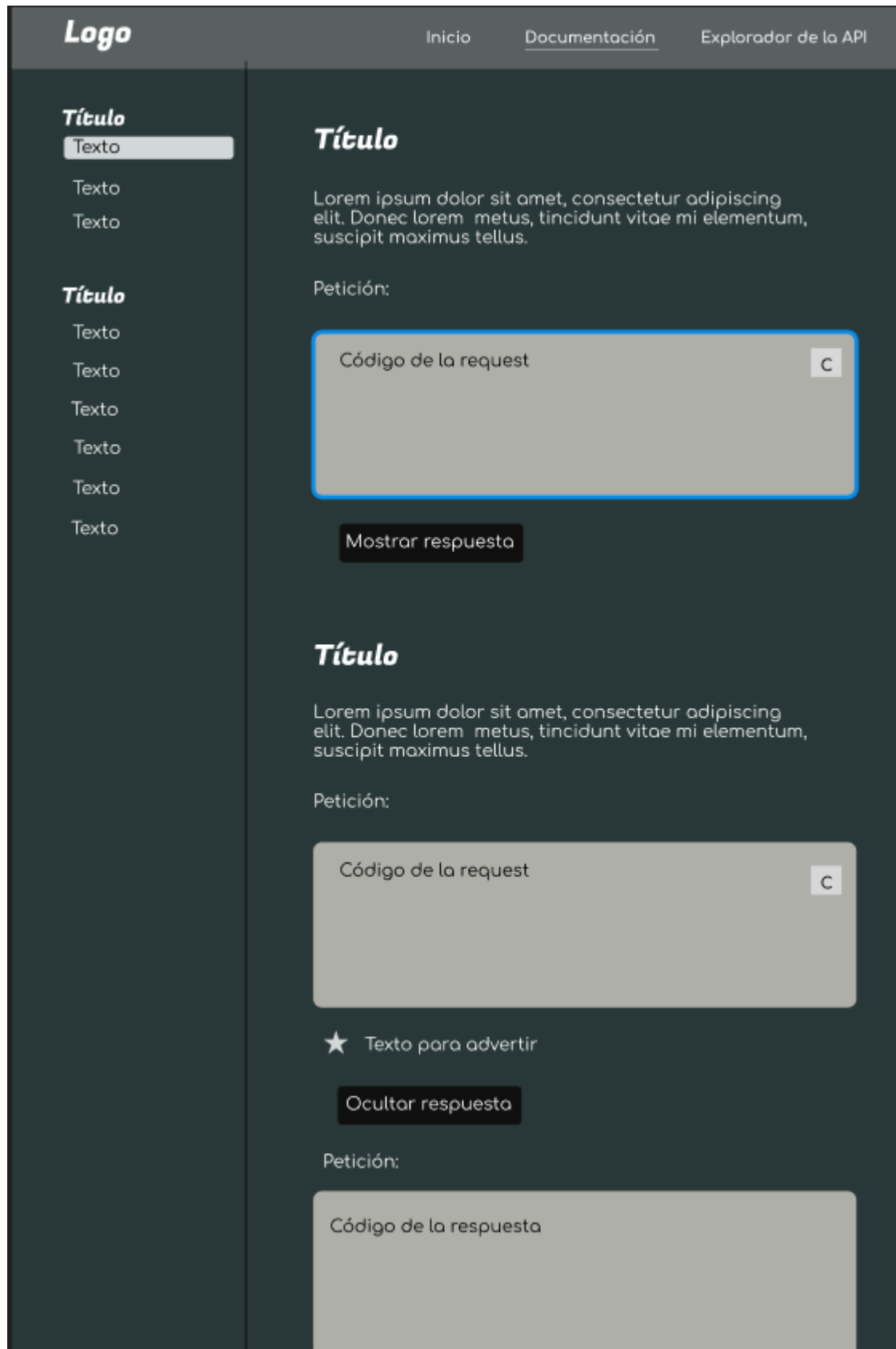
¿Estás cansado de siempre las típicas APIs e-commerce? ¿Te gusta el anime?

Anime API es una API gratuita, que te proporciona datos pseudoreales para practicar con una REST API. Te proporcionamos diferentes métodos de utilización, ya sea, obtener todos los animes como filtrarlos por categoría.

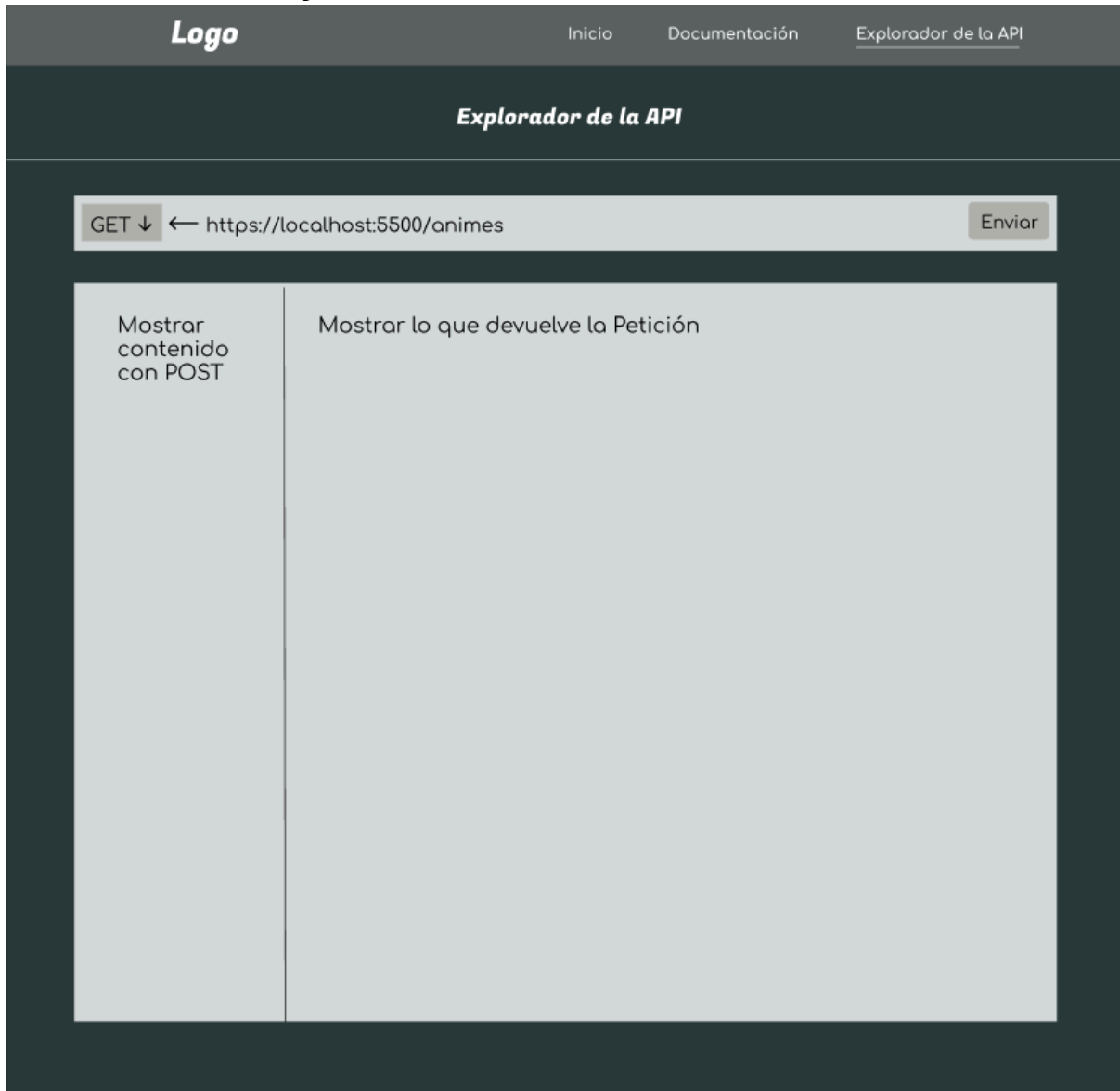
Recursos

Animes	20
Usuarios	20
Categorías	20

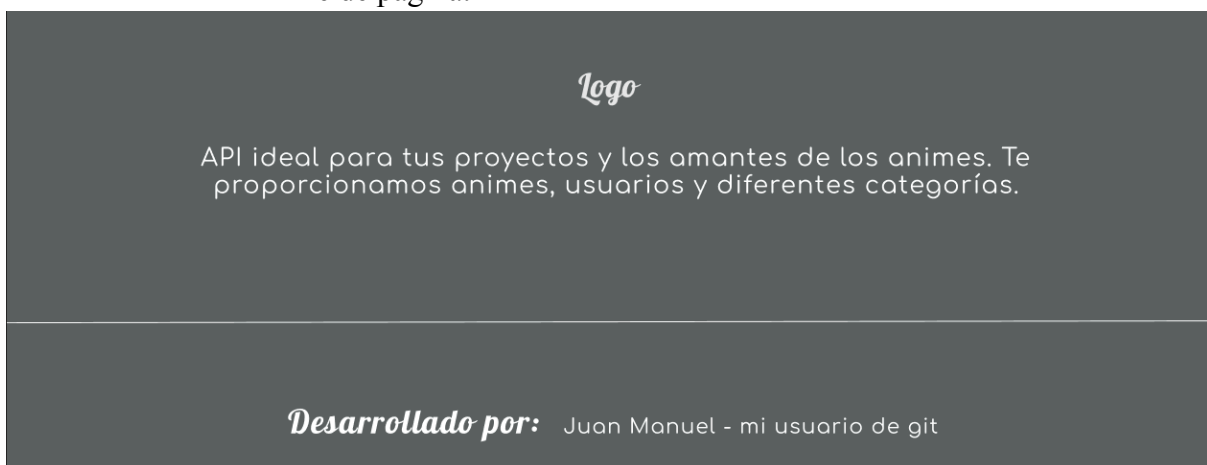
- Documentación:



- Explorador de la API:



- Pié de página:



- Ordenador:
 - Inicio:

Logo

[Inicio](#)[Documentación](#)[Explorador de la API](#)



Anime API

API de animes para tus prácticas y crear tus propios proyectos.

[Ver en GitHub](#) 

[Ver documentación](#) 

API gratuita para los amantes del anime

¿Estás cansado de siempre las típicas APIs e-commerce? ¿Te gusta el anime?

Anime API es una API gratuita, que te proporciona datos pseudoreales para practicar con una REST API. Te proporcionamos diferentes métodos de utilización, ya sea, obtener todos los animes como filtrarlos por categoría.

Recursos

Animes	20
Usuarios	20
Categorías	20

- Documentación:

Logo

InicioDocumentaciónExplorador de la API

Título

Texto

Texto

Texto

Título

Texto

Texto

Texto

Texto

Texto

Texto

Título

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec lorem metus, tincidunt vitae mi elementum, suscipit maximus tellus.

Petición:

Código de la request

C

Mostrar respuesta

Título

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec lorem metus, tincidunt vitae mi elementum, suscipit maximus tellus.

Petición:

Código de la request

★ Texto para advertir

Ocultar respuesta

Petición:

Código de la respuesta

En esta página

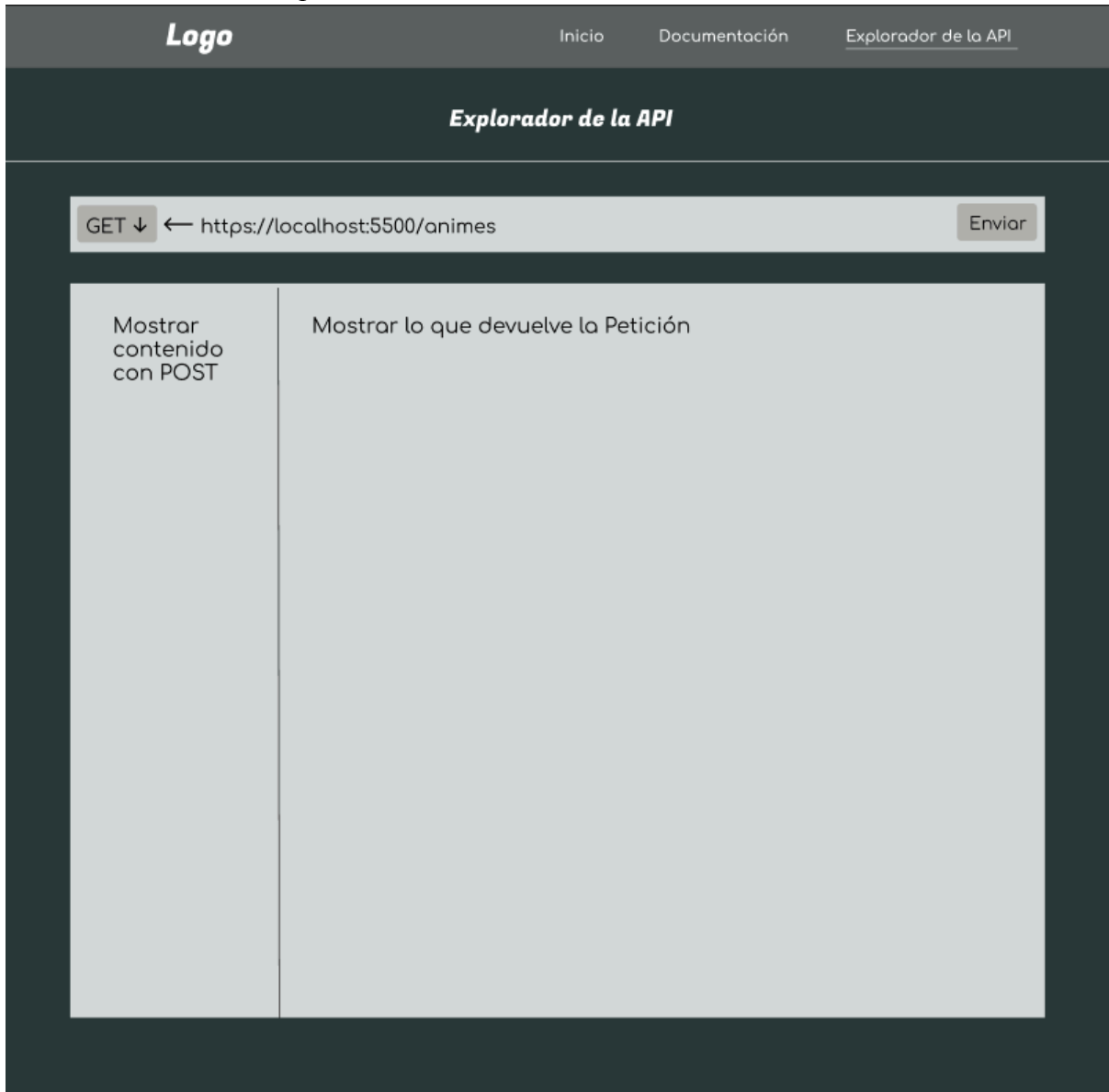
Texto

Texto

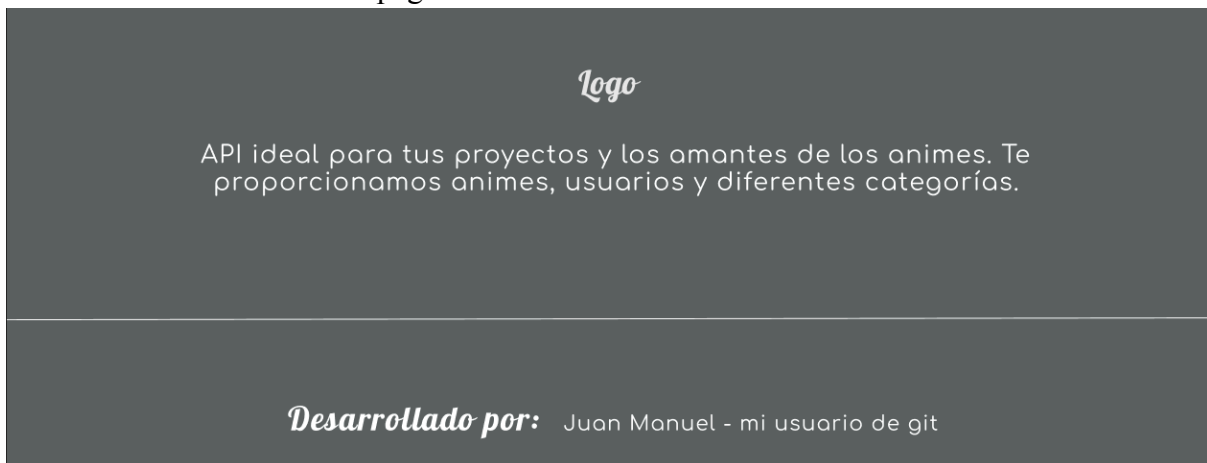
Texto

Texto

- Explorador de la API:



- Pié de página:



b. Criterios de accesibilidad

La web ha seguido los criterios necesarios para conseguir el nivel de conformidad en la norma WCAG 2.1, el nivel AA. Para ello se siguen los siguientes criterios:

- Alternativas textuales: tanto las imágenes como los iconos tienen textos alternativos o descripciones textuales.
- Adaptable: el contenido ha sido estructurado utilizando HTML semántico, para facilitar la navegación y comprensión. Además, se han utilizado atributos aria para mejorar la accesibilidad de elementos dinámicos.
- Distinguible: contraste entre el texto y el fondo. También, el texto puede ser redimensionado hasta un 200% sin pérdida de funcionalidad.
- Accesible por teclado: todas las funcionalidades de la aplicación es accesible mediante teclado y los elementos interactivos como enlaces, botones y campos de formulario tienen indicadores de foco visibles.
- Navegable: los enlaces tienen descripciones claras que indican el destino o acción que realizarán.
- Predecible: la navegación y los elementos interactivos son consistentes en toda la aplicación, facilitando el uso.
- Asistencia en la entrada: los mensajes de errores con claros y específicos, indicando qué problema ocurrió y cómo corregirlo.

Para probar la accesibilidad se ha utilizado la herramienta de W3C. Pasándole una página de la web, te dice los errores de accesibilidad o las precauciones.

c. Criterios de usabilidad

Para este proyecto se han seguido una serie de criterios de usabilidad que aseguran una experiencia de usuario eficiente, efectiva y satisfactoria. Estos criterios se han demostrado en diferentes pruebas realizadas a dos personas. Estas pruebas han sido, por ejemplo, decirle que lean el inicio, que hagan lo que mejor vean en la página de la documentación y que pongan diferentes parámetros en la página del explorador, entre otros. De esta forma se ha conseguido detectar diferentes errores, como al cambiar los géneros, si había un espacio no los modificaba correctamente.

Los criterios de usabilidad seguidos con los siguientes:

- Simplicidad y minimalista: la interfaz de usuario se ha diseñado de manera sencilla y minimalista, consiguiendo así evitar la sobrecarga de información. Se facilita la navegación y el uso de la aplicación. Además, se ha utilizado una jerarquía visual clara, utilizando diferentes tamaños de fuentes y colores.
- Consistencia: todos los elementos de la interfaz mantienen el mismo diseño y comportamiento consistente en toda la aplicación.
- Facilidad de navegación: los menú de navegación están etiquetados y estructurados.
- Feedback del usuario: se proporciona mensajes claros al usuario, por ejemplo, al copiar el código de la petición.

- **Accesibilidad:** la aplicación es responsiva, se asegura que se vea y funciona bien en una variedad de dispositivos y tamaños de pantalla. También, se asegura el uso de aria.
- **Prevención de errores:** se implementan validaciones en los formularios para prevenir errores antes de que el usuario los cometa.
- **Ayuda y documentación:** documentación integrada dentro de la web y se incluyen guías y ejemplos para los nuevos usuarios.

d. Tipografía

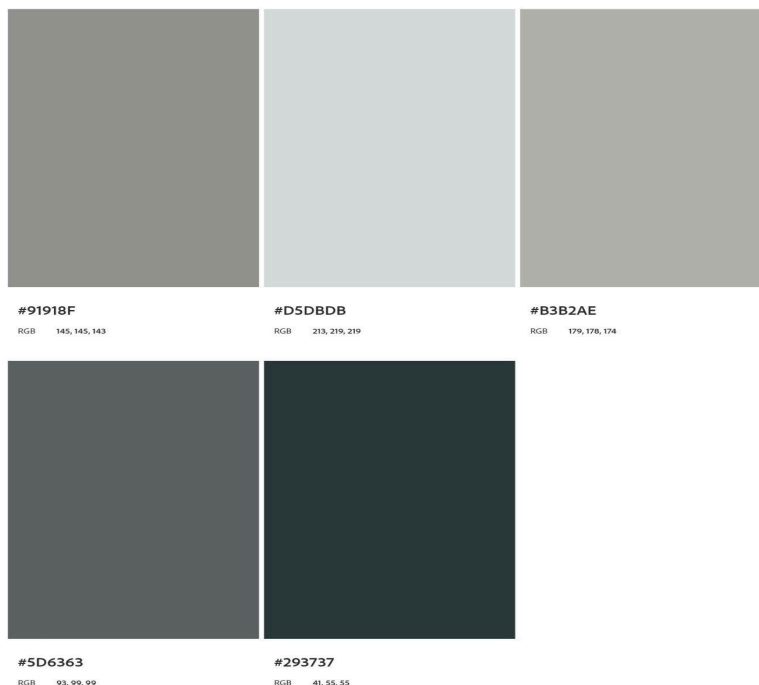
El diseño se ha realizado teniendo en cuenta una idea minimalista pero que llame la atención, por eso, se ha elegido dos tipografías sans-serif, específicamente por sus ventajas. Las razones principales por las que han sido elegidas sans-serif son legibilidad en pantallas, usabilidad y accesibilidad, compatibilidad y rendimiento pero sobre todo ofrece una estética moderna y minimalista.

Se van a listar las tipografías elegidas y comentar donde se han utilizado:

- **Fugaz One:** se ha utilizado para los encabezados.
- **Comfortaa:** se ha utilizado para el resto del cuerpo, como párrafos, enlaces, ...

e. Mapa de colores y elementos donde se aplica

Para los colores se ha escogido una paleta de grises.



El gris es un color que presenta las sensaciones que estamos buscando:

- Neutralidad y versatilidad.
- Elegancia y profesionalismo: transmite una sensación de elegancia y sofisticación. Además, tiene una apariencia limpia y profesional.
- Legibilidad y contraste.
- Accesibilidad y usabilidad.
- Enfoque en el contenido.
- Flexibilidad en el diseño: nos proporciona un diseño adaptable, minimalista y moderno. Además, nos proporciona flexibilidad en la jerarquía del diseño.

Se va a proporcionar la tableta en 3 formatos:

- Gris neutro: hacer hover en los enlaces de la documentación y GitHub.
 - Hexadecimal: #91918F
 - RGB: 145, 145, 143
 - HSL: 60°, 1%, 56.9%
- Gris claro: fondo del botón de copiar y opciones de documentación.
 - Hexadecimal: #D5DBDB
 - RGB: 213, 219, 219
 - HSL: 180°, 9%, 85%
- Gris mármol: fondo donde se muestra código
 - Hexadecimal: #B3B2AE
 - RGB: 179, 178, 174
 - HSL: 48°, 3%, 70%
- Gris grafito: fondo de la cabecera y en el pie de página.
 - Hexadecimal: #5D6363
 - RGB: 93, 99, 99
 - HSL: 0.50°, 0.03%, 38%
- Gris oscuro cálido: fondo del cuerpo.
 - Hexadecimal: #293737
 - RGB: 41, 55, 55
 - HSL: 185°, 25%, 22%

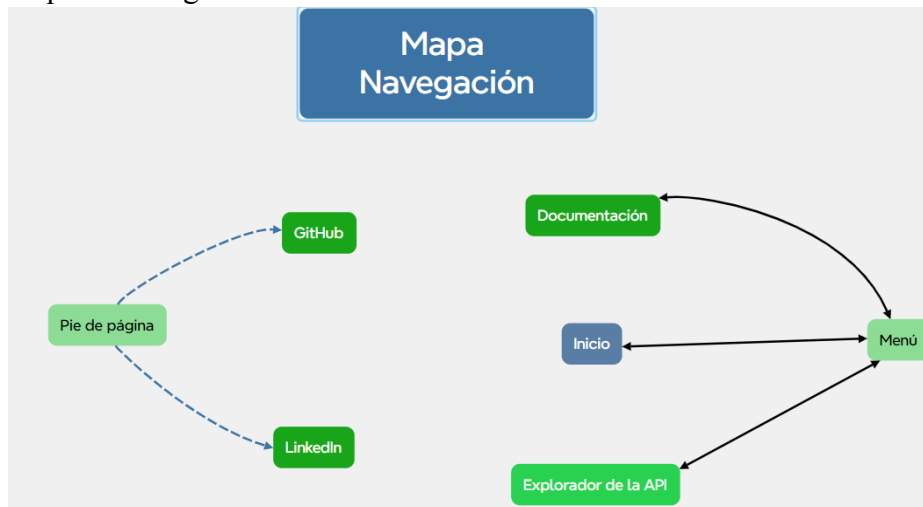
f. Dispositivos / vistas y tecnología usada.

La tecnología que se ha usado para desarrollar el diseño es Figma. Con esta herramienta se han desarrollado vistas para 3 dispositivos, que son ordenador, tableta y móvil. Luego cada uno tiene varias páginas:

- Inicio: muestra una descripción de lo que proporciona la REST API, dos botones, uno para ir al repositorio de GitHub y otro para ver la documentación, y los recursos que proporciona la API.
- Documentación: en esta página se explica las consultas que se pueden hacer, cómo hacerlas y lo que te debe devolver la petición. Además, se explica cómo se tiene que utilizar y hacer peticiones en el Explorador de la API.
- Explorador de la API: de manera gráfica, se pueden hacer peticiones a la API. Esta página es útil para que la línea de aprendizaje sea más vertical que horizontal.

Todo se ha conseguido con la ayuda de CSS y uno de sus framework, Tailwindcss.

Mapa de navegación:



10. Softwares utilizados

Se han utilizado los siguientes softwares:

- node.js: entorno de ejecución.
- express: framework que se ha utilizado para desarrollar la API.
- typescript: me ha servido para añadir tipado, seguridad, mantenibilidad, he conseguido hacer una aplicación escalable y he conseguido mejorar la escalabilidad.
- ts-node-dev: levantar el servidor sin necesidad de transpilar el código.
- cors: paquete de npm que me ha servido para desarrollar el cors de la API.
- dotenv: me ha permitido usar variables de entorno.
- mongodb: paquete de npm, para instalar los driver para luego utilizar mongodb.
- mongodb compass: me ha permitido tener una base de datos NoSQL en local.
- mongodb atlas: utilizado para tener una base de datos NoSQL en la nube.
- MySQL: entorno que me ha permitido crear una base de datos relacional en local (SQL).
- mysql2: paquete de npm, lo he utilizado para conectar y realizar consultas a la base de datos de MySQL.
- uuid: paquete de npm, su función en la API ha sido comprobar si la id que me han pasado es una UUID (string-string-string-string).
- zod: paquete de npm, me ha permitido implementar validaciones para los animes y usuarios.
- astro: framework web, lo he utilizado para desarrollar la web. Lo he elegido porque es agnóstico a UI como React o Vue. Además, me ha permitido utilizar islas, una arquitectura web basada en componentes optimizada.
- react: he desarrollado componentes de forma sencilla y dinámicos.
- HTML5: utilizado para estructurar la web.
- CSS3: utilizado para maquetar la web.
- react-toastify: paquete de npm, me ha servido para lanzar una notificación emergente cuando se pulse un botón de copiar.
- tailwindcss: framework CSS que proporciona clases de utilidades.
- git: software utilizado para el control de versiones.
- GitHub: servicio para alojar en la nube el proyecto desarrollado.

- npm: instalar dependencias y ejecutar scripts.
- npx: instalar las integraciones oficiales de Astro cómo react y tailwind.
- Render: donde se ha desplegado la REST API.
- Vercel: donde se ha desplegado la web de la API.
- Visual Studio Code: editor de código utilizado.
- Postman:herramienta de de pruebas para la API.
- Animaciones Tailwind Midudev

11. Mejoras posibles y aportaciones

Mejoras en funcionalidad:

- Inicio de sesión: se podría implementar un inicio de sesión, y según si el cliente está logueado o no, se debería de poder realizar un cantidad u otra de consultas.
- Autenticación: para los usuarios que estén registrados, podrían identificarse mediante un token.
- Paginación: se ofrece un total de 63 animes, así que se debería de poder paginar mediante una consulta.
- Documentación del código.
- Un vista de administrador, por si se quiere eliminar, modificar o crear animes, usuarios o géneros de una forma más sencilla.

Para la organización del código se ha utilizado la estrategia de gestión de código de monorrepo multipaquete. Además, se ha diseñado mediante el patrón de arquitectura MVC, como se ha mencionado en un punto anterior. Por lo tanto, el proyecto tiene controladores, modelos y una vista.

12. Bibliografía

- [Figma](#)
- [Visual Studio code](#)
- [cors](#)
- [mysql2](#)
- [mongodb](#)
- [Express](#)
- [dotenv](#)
- [ts-node-dev](#)
- [zod](#)
- [uuid](#)
- [react-toastify](#)
- [Astro](#)
- [React](#)
- [Node](#)
- [Tailwind](#)
- [Git](#)
- [GitHub](#)

- [TypeScript](#)
- [Render](#)
- [Vercel](#)
- [Postman](#)
- [MongoDB Compass](#)
- [MongoDB Atlas](#)
- [MySQL](#)
- [Adobe Color](#)
- [Animaciones Tailwind Midudev](#)