

Proyecto Final DAW

11/06/2021

**─**

Javier Luque Rodríguez

Tutorizado por Equipo Educativo 2º DAW

Instituto de Educación Secundaria Campanillas

Técnico Superior de Desarrollo de Aplicaciones Web

[**Introducción**](#_5h7grjf6h2ap) **1**

[Presentación y objetivos](#_wne0r3pae9t) 1

[Contexto](#_bvdsuv86fxp6) 1

[Planteamiento del problema](#_bv1x7mhgbk1y) 1

[Estructura del documento](#_rlwklhdxxjdd) 1

[**Especificaciones de requisitos**](#_f8b5epwbybfk) **3**

[Introducción](#_rnde324z87g8) 3

[Propósito](#_y9p4zmqzjy6h) 3

[Ámbito](#_nfa71lpeyryh) 3

[Definiciones, siglas y abreviaturas](#_imc5ff6z3xp9) 3

[Visión global](#_oy60nm710slg) 4

[Descripción general](#_ba44wk4xmji5) 4

[Perspectiva del producto](#_gu07la6i16z1) 4

[Funciones del producto](#_5664a6djc8b2) 4

[Usuario administrador](#_9pntvcxcd6tv) 5

[Características del usuario](#_2hvn0p2nz76x) 6

[Usuarios no registrados](#_9305nequ29qa) 6

[Usuarios registrados](#_ll6obel8w1t5) 6

[Usuarios administradores](#_wdqtx4hkoktr) 6

[Restricciones](#_ab7wu9su9cpx) 6

[Supuestos y dependencias](#_y3mhxt1z2d83) 6

[Requisitos específicos](#_out94tspnhth) 7

[Requerimientos funcionales](#_28osxboohwvx) 7

[Usuarios no registrados](#_gqpyi61k9wxw) 7

[Usuarios registrados](#_l70y3s675q9h) 8

[Requerimiento de interfaces externos](#_ezyabkmo150p) 13

[Interfaces de usuario](#_25s7umizalye) 13

[Interfaces hardware](#_qkp0yd5bky6a) 13

[Interfaces software](#_5h63x5e2vswj) 13

[Interfaces de comunicaciones](#_up1fwa5uefbl) 13

[Requerimientos de eficiencia](#_g6p29vp7dr52) 13

[Obligaciones del diseño](#_ce0y8ynxxs2s) 14

[Estándares cumplidos](#_c90wku3tjbwn) 14

[Limitaciones hardware](#_7hxyd4e7du8m) 14

[Atributos](#_7lmujjnwy2dk) 14

[Seguridad](#_rzjnh1wu9i7f) 14

[Facilidades de mantenimiento](#_a90vmw7fmpiw) 14

[Portabilidad](#_41lt31tbrdzw) 15

[Otros requerimientos](#_5rvsnqiovcwm) 15

[**Análisis**](#_4tm70g393z4i) **16**

[Introducción](#_q74ho6kekxsr) 16

[Diagramas de clases](#_963c2bwhx4z3) 16

[Diagramas de casos de uso](#_oh2fap4tsi4) 17

[Actores](#_iy9jejge0mit) 17

[Casos de uso del usuario](#_8okkz1j211q5) 18

[Casos de uso del usuario registrado](#_cwzrkhp64s65) 18

[**Diseño**](#_mo9azlr2f4ui) **20**

[Introducción](#_v3ijrv9bmntc) 20

[Capa de presentación](#_7r2a1f7mmdgi) 20

[Capa de negocio](#_el2pbzz2lhds) 21

[Capa de persistencia](#_osu7ixnpuy4y) 21

[**Implementación**](#_o6oe2porb2xm) **23**

[Tecnologías utilizadas en el desarrollo del proyecto](#_af5rl5etjqve) 23

[HTML](#_d1a6z4qu0bk2) 23

[SCSS/SASS](#_7os0kl1uji5) 23

[TypeScript](#_pqpaq77djimy) 24

[Angular](#_azwwvrbxqbco) 25

[Node.js](#_f1zhchv8y6ca) 25

[Express](#_hllh3m9re5zi) 25

[TypeORM](#_avfwnqbpxxgd) 26

[SQL](#_dl96rmmff9) 26

[MySQL](#_p4wzk9hmy587) 27

[Descripción del proyecto](#_nssdmo7pquag) 27

[Capa de presentación](#_3vlwjzndw74j) 28

[Capa de negocio o lógica de la aplicación](#_vj6wy28ndtie) 30

[Capa de persistencia o de datos](#_4pt1zm1g9ofe) 32

[**Evaluación**](#_dom52jrue34q) **34**

[Introducción](#_mltvffl9zm2c) 34

[Validación de CSS](#_8cthlmik3ev8) 34

[Validación de enlaces](#_wuqy8bsdc60h) 35

[Validación de la resolución](#_e8ie6ktkmch3) 36

[Validación de navegadores](#_8ta6gbhe328n) 36

[Validación de la seguridad en el acceso a la zona privada](#_kf937st5khb3) 38

[**Diseño Interfaz**](#_70ax2ru7wnn3) **40**

[Mapa de navegación](#_fji6sz2k9zj9) 40

[Paleta de color](#_e45lo24btx0t) 40

[Elección de la fuente](#_knzjaqkhh3ft) 42

[Contraste fondo con letra](#_reuloqyod5ia) 42

[Equilibrio visual](#_e3pz4lm144dz) 44

[Tensión compositiva](#_hgz7avxzta7a) 45

[Accesibilidad y usabilidad](#_lircvbbw0i78) 45

[**Despliegue de la aplicaciones web**](#_y5st5bnlm5u8) **47**

[Sistema de Control de Versiones (VCS)](#_ushgtx4v4ksz) 47

[Introducción](#_m0isux87q98m) 47

[Git](#_hs425jjzqljx) 47

[Github](#_9n3ita97ft45) 48

[Organización del proyecto](#_gqoc0wng0ifu) 48

[Issues](#_910l2utlt76s) 50

[Ramas](#_827td24s9xk6) 52

[Despliegue en remoto](#_h9q7tgh2xwfx) 56

[Heroku](#_tgf518jknfyh) 56

[Github Pages](#_b5h2mw5ojyv3) 60

[Generación automática de la documentación](#_yvna6n7ta7ad) 63

[**Conclusión**](#_sorleabbxnrd) **66**

[Validación personal del trabajo realizado](#_vhlafsf862f1) 66

[Posibles ampliaciones](#_9gfg9sgfl7hj) 66

[**Bibliografía**](#_56z6th246ul9) **67**

# 

# Introducción

## Presentación y objetivos

En el presente documento se describe el trabajo realizado en el proyecto final del Grado Superior en Desarrollo de Aplicaciones Web. El proyecto consiste en el desarrollo de una aplicación web deportiva que nos permita compartir con otras personas rutas por la montaña. Dicha aplicación está inspirada por otras aplicaciones como [wikiloc](https://www.wikiloc.com/) o [strava](https://www.strava.com).

La aplicación web es accesible desde cualquier navegador mediante Internet y contiene información sobre los usuarios que la utilizan y las rutas que estos realizan. Trata buscar que las personas puedan realizar las rutas que los demás van subiendo.

Aunque en el backend se distinguen entre usuario normal y administrador, en esta primera versión solo se ha implementado el usuario normal en el frontend. Más adelante podremos ver todas las funcionalidades que puede utilizar el usuario.

## Contexto

Para el desarrollo de la aplicación realice una investigación previa de las aplicaciones similares que había en el mercado para saber bien cuales son algunas de las funcionalidades mínimas que quería implementar en el proyecto. Una vez decidí como quería que fuera el proyecto dividí el trabajo en tres partes: desarrollo del backend (API REST), desarrollo del frontend y diseño.

## Planteamiento del problema

El problema que se nos ha planteado es la construcción de una aplicación web deportiva. La funcionalidad de la aplicación, a grandes rasgos, era:

* Los usuarios pueden tener permisos de usuario o administrador.
* Los usuarios que están logueados pueden:
  + Seguir o dejar de seguir a otros usuarios.
  + Crear y actualizar sus rutas, así como poder subir el fichero GPX de la ruta.
  + Descargar las rutas de otros usuarios.
  + Comentar las rutas de otros usuarios.

## Estructura del documento

Este documento se encuentra dividido en una serie de capítulos que corresponden a las distintas etapas que conforman el proceso de desarrollo del proyecto. Estas etapas han sido:

* **Especificación de requisitos**: Se redactó de una manera global una primera visión del proyecto donde señalamos los requisitos que debía cumplir. La finalidad de esta etapa era plantear unos objetivos de cómo se quería que fuera la aplicación final, así como unos objetivos mínimos. En nuestro caso el visto bueno nos lo dieron los profesores del grado superior de Desarrollo de Aplicaciones Web.
* **Análisis**: Se realizó el modelado conceptual de la futura solución mediante el uso de diagramas (diagrama de clases y diagramas de casos de uso). Los modelos ayudan a visualizar como es el sistema, proporcionando plantillas que sirven de guía en la construcción de la aplicación. En esta etapa se especifica qué debe hacer la aplicación pero no cómo debe hacerlo.
* **Diseño**: Se utilizaron los elementos y modelos obtenidos durante el análisis para transformarlos en mecanismos que puedan ser utilizados en un entorno web con las características y condiciones que establecen este tipo de entornos. Se diseñaron todos los niveles de los que consta la aplicación(nivel de presentación, nivel lógico y nivel de persistencia).Tanto la etapa del análisis como la del diseño están desprovistas de código. Un buen análisis y un buen diseño son la mejor forma de llegar a producir software de calidad.
* **Implementación**: Se utilizaron los elementos obtenidos en el diseño para permitir la elaboración del producto o prototipo funcional, es decir, que puede ser puesto en marcha y sometido a pruebas.Para ello se consideraron las diversas tecnologías que han intervenido en la elaboración de dicho producto. Todo lo desarrollado en las etapas del análisis y del diseño, se tradujó a código.
* **Evaluación y pruebas**: Esta fase se centró en la comprobación del correcto funcionamiento del producto desarrollado mediante una serie de pruebas.Tras estas tareas pudimos dar por concluido el proyecto, por lo que en último lugar mostramos las conclusiones obtenidas y listamos la bibliografía utilizada durante la realización del mismo.

# Especificaciones de requisitos

## Introducción

### Propósito

La especificación de requisitos consiste en definir cuáles son los requerimientos que debe tener la aplicación que se va a desarrollar y describir las funcionalidades que tiene el usuario a lo largo de la aplicación.

### Ámbito

El desarrollo de la aplicación web busca ofrecer diversos servicios y funcionalidades que permitan al usuario compartir diversas rutas con el fin de fomentar el deporte y el conocimiento de nuevos caminos que recorrer. Por tanto, en la aplicación podemos encontrar dos áreas bien diferenciadas. Una de ellas consistiría en la parte pública donde los usuarios solo pueden acceder si no están logueados, ya que aquí podrán loguearse o registrarse en la aplicación. Por otro lado, tenemos la parte que sí requiere que el usuario esté logueado, donde se encuentran los servicios principales y casos de uso más relacionados con las rutas.

### Definiciones, siglas y abreviaturas

* **Aplicación web**: Conjunto de archivos electrónicos y páginas web referentes a un tema en particular que incluye una página inicial de bienvenida, con un nombre de dominio y dirección en Internet específicos.
* **Interfaz**: Parte del programa informático que permite el flujo de información entre varias aplicaciones o entre el propio programa y el usuario
* **Navegador**: Permite al usuario recuperar y visualizar páginas web a través de Internet.
* **DAW**: Grado Superior en Desarrollo de Aplicaciones Web.
* **MVC**: Modelo Vista Controlador, Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.
  + El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
  + La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
  + El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestiona el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

### Visión global

A continuación, se hará una descripción general del sistema desarrollado con sus funciones, características del usuario y otros componentes de la aplicación y dependencias. Además, también se explicará una especificación detallada de los requisitos detectados.

## Descripción general

### Perspectiva del producto

La aplicación que se ha desarrollado busca divulgar rutas deportivas realizadas en distintos tipos de actividades, siendo los mismos usuarios los encargados de subir y mantener actualizadas dichas rutas. La aplicación permitirá que se acceda desde cualquier navegador web que tenga conexión a internet, por tanto, es independiente del sistema operativo, lo cual permitirá llegar a un mayor número de personas.

### Funciones del producto

A continuación, se muestran las distintas funciones que engloba la aplicación, según el tipo de usuario que se encuentre conectado.

**Usuario anónimo:**

* **Autenticación**: se muestran un par de cajas de texto donde el usuario podrá introducir su email y contraseña para poder acceder a la parte privada de la aplicación. Al hacer click en el botón de login, si la información es correcta se redirigirá al usuario a la parte privada.
* **Registrarse**: se muestran cuatro cajas de texto donde el usuario deberá introducir, su email, contraseña, nombre y apellidos. Una vez introducidos si la información es correcta se habrá creado una cuenta para el usuario y este ya podrá acceder a la aplicación.

**Usuario registrado y logueado**:

* **Obtener listado de rutas**: se muestra un listado de todas las rutas existentes en la aplicación.
* **Obtener listado de rutas de usuarios que el usuario sigue**: muestra un listado de las rutas que han subido únicamente los usuarios a los que el usuario logueado sigue.
* **Obtener listado de rutas de un usuario**: rutas subidas por únicamente un usuario.
* **Obtener listado de usuarios**: se muestra un listado con los nombres de todos los usuarios que forman parte de la aplicación.
* **Seguir a un usuario**: el usuario puede seguir a otros usuarios para mantenerse actualizados de las rutas que va subiendo dicho usuario.
* **Dejar de seguir a un usuario**: el usuario puede dejar de seguir a otro usuario.
* **Ver información de un usuario**: se mostrará la información relativa a un usuario.
* **Buscar un usuario**: el usuario puede buscar a otros usuarios que también se encuentren registrados en la aplicación.
* **Crear una ruta**: permite al usuario dar de alta una ruta nueva.
* **Editar una ruta**: el usuario puede editar una ruta que ya haya subido.
* **Borrar una ruta**: el usuario puede borrar las rutas que él haya subido.
* **Ver información de una ruta**: al acceder a una ruta en concreto nos mostrará toda la información relativa a la ruta.
* **Descargar el fichero de la ruta**: el usuario puede descargar el fichero vinculado a una ruta.
* **Comentar una ruta**: el usuario puede comentar tantas veces como quiera en una ruta.
* **Comentar un reto**: el usuario puede comentar tantas veces como quiera en un reto.
* **Puntuar una ruta**: el usuario puede puntuar una ruta del 1 al 5 en función de lo que le haya gustado.
* **Puntuar un reto**: el usuario puede puntuar una ruta del 1 al 5 en función de lo que le haya gustado.
* **Listado de comentarios**: el usuario puede ver todos los comentarios que tiene una ruta.
* **Unirse a un reto**: para llevar un seguimiento de los retos que lleva el usuario puede unirse a un reto.
* **Completar un reto**: el usuario puede marcar un reto que está siguiendo como completado.
* **Borrar comentario**: el usuario puede borrar los comentarios que haya hecho en una ruta.
* **Cerrar sesión**: en la barra de navegación el usuario tiene una opción para poder cerrar la sesión. Una vez cerrada redirigirá al usuario a la página de login de nuevo.

### Usuario administrador

* **Crear una actividad**: el administrador puede crear actividades nuevas.
* **Modificar una actividad**: el administrador puede modificar una actividad.
* **Obtener listado de actividades**: el usuario puede obtener un listado de todas las actividades.
* **Crear usuarios**: el administrador puede crear nuevos usuarios.
* **Actualizar usuarios**: el administrador puede modificar la información de un usuario.
* **Borrar usuarios**: el administrador puede borrar a otros usuarios.
* **Dar permisos de administrador**: los administradores pueden dar permisos de administrador a otros usuarios.
* **Quitar permisos de administrador**: los administradores pueden quitar permisos de administrador a otros administradores.

### Características del usuario

En la aplicación podemos diferenciar entre dos tipos de usuarios: los usuarios registrados y los usuarios no registrados.

#### Usuarios no registrados

Estos usuarios solo podrán acceder a la zona donde poder hacer login y registrarse.

#### Usuarios registrados

Estos usuarios son los que tienen acceso a la aplicación y, por tanto, los que harán uso de las funcionalidades de la aplicación.

#### Usuarios administradores

Estos usuarios son los que tienen acceso a la aplicación y se encargan mantener la aplicación y ofrecer un buen soporte a los usuarios registrados..

### Restricciones

Al tratarse de una aplicación web, se requiere de un dispositivo electrónico, como un teléfono, tablet u ordenador, que tenga acceso a un navegador y a internet.

### Supuestos y dependencias

La aplicación desarrollada ofrece un soporte multiplataforma ya que trabaja al margen de cualquier hardware o software. Sin embargo, eso no quiere decir que no tenga ninguna dependencia. El servidor web donde se aloje nuestra aplicación debe ser capaz de soportar una serie de dependencias, que son: MySQL, Node.js y NPM. Veamos una breve descripción de cada una de estas:

* **Node.js** es un entorno de tiempo de ejecución de JavaScript (de ahí su terminación en .js haciendo alusión al lenguaje JavaScript). Este entorno de tiempo de ejecución en tiempo real incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript.
* **NPM**: npm es el sistema de gestión de paquetes por defecto para Node.js.
* **MySQL**: es un sistema de gestión de bases de datos relacional desarrollado por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo.

## 

## Requisitos específicos

### Requerimientos funcionales

Dado que no se ha implementado en esta primera versión todas las caracteristicas descritas en la descripción general de la aplicación, este apartado vamos a ver las diversas funcionalidades que ofrece esta primera versión de la aplicación web, pero con una descripción más detallada. Estas funciones se encuentran clasificadas según el tipo de usuario que vaya a acceder a la aplicación.

#### Usuarios no registrados

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Autenticación |
| Propósito | Acceder a la aplicación |
| Entrada | Email y contraseña |
| Proceso | Comprueba que el email y la contraseña sean correctos |
| Salida | Api token o aviso de que los datos enviados no son válidos |

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Registrarse |
| Propósito | Crear una cuenta para la aplicación |
| Entrada | Email, contraseña y nombre |
| Proceso | Almacena en la base de datos el nuevo usuario |
| Salida | Respuesta de error o éxito |

#### 

#### Usuarios registrados

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Obtener listado de rutas |
| Propósito | Obtener un listado de todas las rutas que existen en la aplicación |
| Entrada |  |
| Proceso | Obtiene todas las rutas de la base de datos |
| Salida | Listado de las rutas |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Obtener listado de rutas de los usuarios a los que sigue |
| Propósito | Obtener un listado de todas las rutas de los usuarios a los que sigue el usuario logueado |
| Entrada |  |
| Proceso | Obtiene todas las rutas de la base de datos de los usuarios a los que sigue el usuario |
| Salida | Listado de las rutas de los usuarios a los que sigue |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Obtener listado de rutas de un usuario |
| Propósito | Obtener un listado de todas las rutas de un usuario |
| Entrada | id del usuario |
| Proceso | Obtiene todas las rutas de la base de datos del usuario |
| Salida | Listado de las rutas del usuario |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Seguir a un usuario |
| Propósito | Seguir a un usuario |
| Entrada | id del usuario a seguir |
| Proceso | Crea una nueva fila en la base de datos relacionando al usuario a seguir y al logueado |
| Salida | Respuesta de error o éxito |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Dejar de seguir a un usuario |
| Propósito | Dejar de seguir a un usuario |
| Entrada | id del usuario a dejar de seguir |
| Proceso | Elimina la fila de la base de datos correspondiente |
| Salida | Respuesta de error o éxito |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Ver información de un usuario |
| Propósito | Ver toda la información relativa a un usuario |
| Entrada | id del usuario |
| Proceso | Busca en la base de datos el usuario con el id que se pasa como parámetro |
| Salida | Devuelve el usuario vinculado al id |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Buscar un usuario |
| Propósito | Buscar a un usuario por su nombre |
| Entrada | Nombre |
| Proceso | Busca en la base de datos al usuario o usuarios que encajen con el nombre |
| Salida | Listado de usuarios que coinciden con la búsqueda |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Crear una ruta |
| Propósito | Permitir al usuario dar de alta una ruta nueva. |
| Entrada | Id de la ruta a actualizar, nombre de la ruta, descripción, dificultad, elevación ganada, elevación máxima (opcional), elevación mínima (opcional), país, provincia, ciudad, archivo gpx de la ruta (opcional) e id de la actividad. |
| Proceso | Actualiza en la base de datos una ruta |
| Salida | Respuesta de éxito o error |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Borrar una ruta |
| Propósito | Eliminar una ruta |
| Entrada | Id de la ruta a borrar |
| Proceso | Elimina de la base de datos una ruta |
| Salida | Respuesta de éxito o error |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Ver información una ruta |
| Propósito | Ver toda la información relacionada a una ruta |
| Entrada | Id de la ruta |
| Proceso | Obtiene la ruta de la base de datos relacionado al id |
| Salida | La ruta o una respuesta de error |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Descargar el fichero de una ruta |
| Propósito | Descargar el fichero gpx que tiene una ruta |
| Entrada | Ruta del fichero |
| Proceso | Obtiene el fichero gpx buscando en el storage la ruta indicada |
| Salida | Fichero gpx o error |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Comentar una ruta |
| Propósito | Comentar una ruta |
| Entrada | id de la ruta, comentario |
| Proceso | Inserta en la base de datos un nuevo comentario de una ruta |
| Salida | Respuesta de éxito o error |

#### 

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Obtener listado comentarios |
| Propósito | Obtener un listado de todos los comentarios que tiene una ruta |
| Entrada | id de la ruta |
| Proceso | Obtiene todos los comentarios de una ruta de la base de datos |
| Salida | Listado de comentarios |

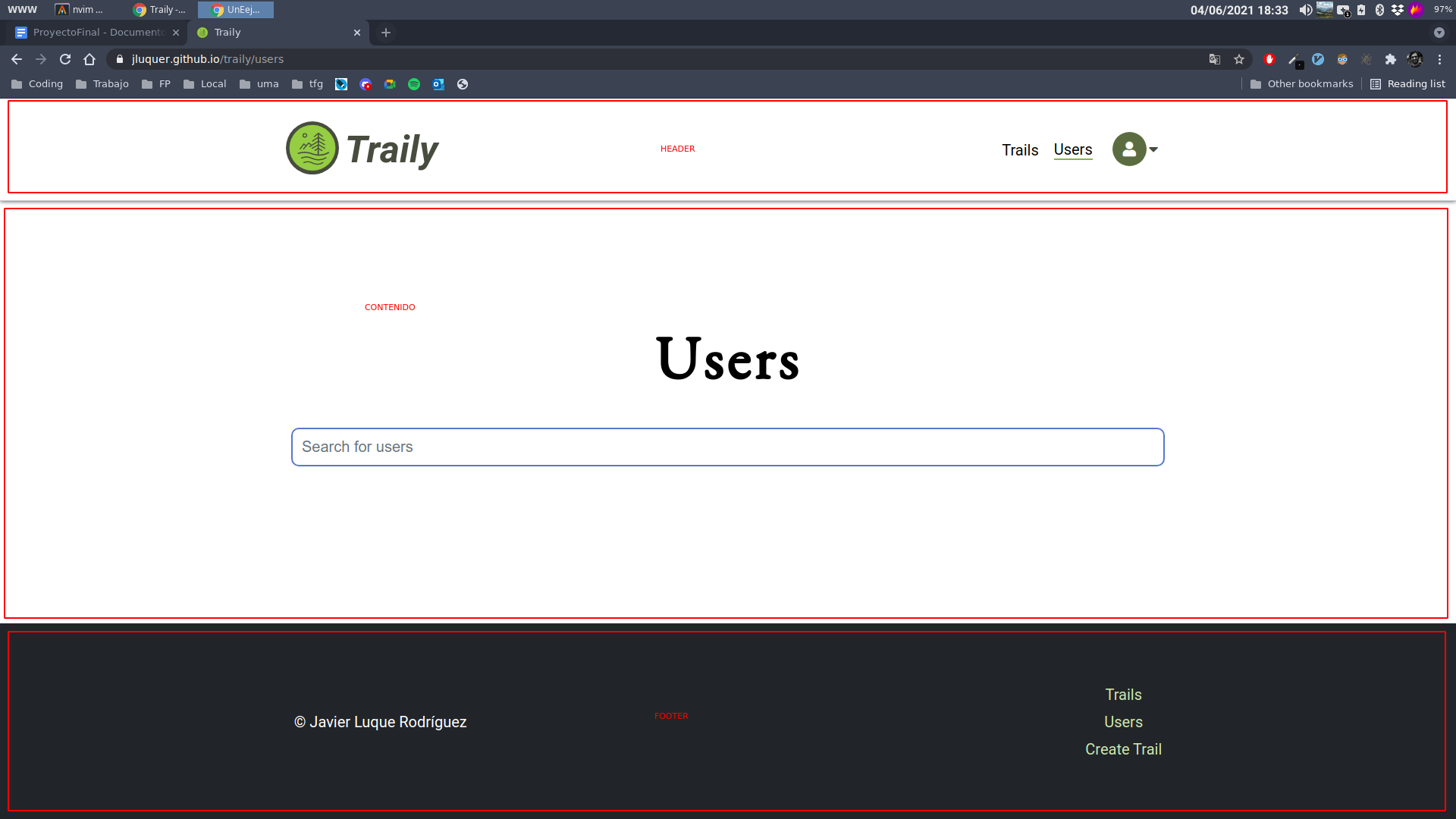
|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Borrar comentario |
| Propósito | Borrar un comentario de una ruta |
| Entrada | id del comentario |
| Proceso | Borra el comentario de la base de datos con el id |
| Salida | Respuesta de éxito o error |

|  |  |
| --- | --- |
| Apartado | Descripción |
| Título | Cerrar sesión |
| Propósito | Cerrar la sesión que el usuario tiene abierta |
| Entrada |  |
| Proceso | Elimina el api token que hay almacenado en el localstorage del navegador |
| Salida |  |

### Requerimiento de interfaces externos

#### Interfaces de usuario

A continuación, se muestra una captura de pantalla en la que podemos ver las distintas zonas que conforman la interfaz gráfica con la que interactúa el usuario que haga uso de la aplicación.



#### Interfaces hardware

Al tratarse de una aplicación web, se podrá visualizar sobre cualquier sistema operativo.

#### Interfaces software

La aplicación funcionará en cualquier dispositivo electrónico que pueda hacer uso de un navegador web y conexión a Internet.

#### Interfaces de comunicaciones

Las comunicaciones se efectúan siguiendo el protocolo HTTP mediante conexiones TCP/IP.

### Requerimientos de eficiencia

Debido a la metodología empleada en el diseño de la aplicación, la aplicación tiene un rendimiento independiente del hardware y software utilizado. Los factores que más pueden influir son las velocidades de conexión del cliente y del servidor. En cuanto al servidor, actualmente la mayoría cuentan con velocidades de subida más que suficientes para nuestras necesidades. Respecto al cliente, las velocidades ofrecidas actualmente por los Proveedores de servicios de Internet (ISP) también son más que suficientes para la correcta transmisión de datos entre el cliente y el servidor.

Otro aspecto que se ha tenido en cuenta es la gran variabilidad del tamaño de las pantallas que hay hoy en día, teniendo en cuenta que el teléfono móvil es el medio para visualizar páginas webs actualmente. Por tanto, el diseño de la aplicación se trata de un diseño mobile first y responsive, que se adapta a cualquier tipo de pantalla donde se quiera visualizar la aplicación.

### Obligaciones del diseño

#### Estándares cumplidos

Se han intentado cumplir los estándares de cualquier web con acceso seguro, creando un sistema de autenticación para que nadie pueda acceder a una zona de la web a la que no tiene permiso de acceso. El idioma elegido para la presentación de las páginas ha sido el inglés.

#### Limitaciones hardware

Al tratarse de una aplicación web no se requiere un hardware específico. El servidor que albergará la base de datos del sistema deberá permanecer disponible las 24 horas, puesto que este host será quien atienda las peticiones de lectura y escritura de los usuarios que accedan a la aplicación. Hoy en día son muchos los servicios de hosting que hay gratuitos en la red, sin embargo, muchos de estos en ocasiones pueden sufrir caídas con mayor frecuencia.

### Atributos

#### Seguridad

La seguridad es un componente fundamental en el portal. La administración del sitio web está sujeta a la identificación satisfactoria del usuario en la aplicación, de forma que ningún usuario anónimo o visitante pueda, por ejemplo, crear o modificar una ruta. Para asegurar la identidad del usuario se requerirá su email y contraseña que autentica a este en la aplicación. La información acerca de la cuenta, rutas, comentarios y otros se guardará en la base de datos. El proceso de login o autenticación llevará al usuario al dashboard de la aplicación.

#### Facilidades de mantenimiento

El mantenimiento básico de las rutas lo realizarán los usuarios registrados. Sin embargo, cualquier cambio que se desease introducir y requiriese de la modificación de la base de datos, así como implementar nueva funcionalidad, deberá ser llevada a cabo por el programador de la web.

#### Portabilidad

La aplicación ha sido diseñada con tecnología libre, luego podrá ser soportada por cualquier plataforma y sistema operativo. Por lo mismo, se podrá acceder a ella desde cualquier navegador.

#### Otros requerimientos

El portal usa una base de datos MySQL donde se almacena toda la información referida a los usuarios, rutas, comentarios, etc. Las consultas a la base de datos se realizan por parte del servidor web mediante Node.js y su API de acceso a bases de datos MySQL.

Por otro lado, ya hemos comentado que para poder introducir datos en la base de datos se debe ser el usuario registrado. Al acceder un usuario anónimo deberá crear una cuenta nueva para poder realizar modificaciones en los datos que tenemos en la base de datos.

# 

# Análisis

## Introducción

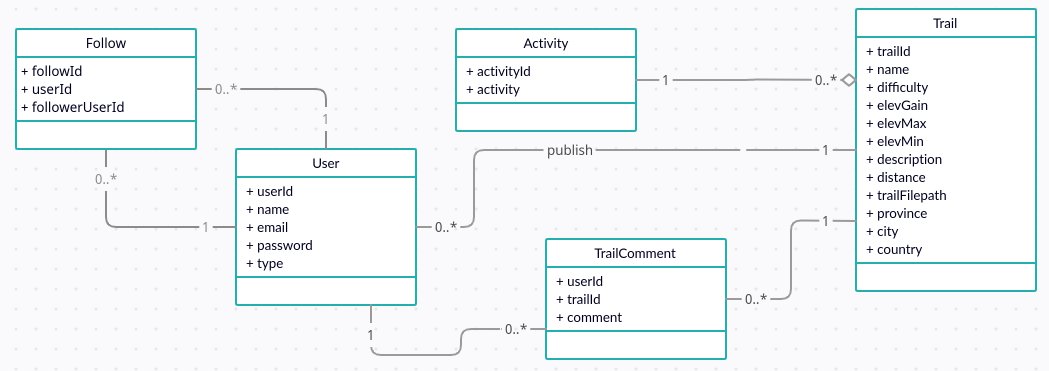
Para realizar el análisis de esta aplicación web se ha optado por seguir las recomendaciones definidas por UML (Unified Modeling Language o Lenguaje Unificado de Modelado en castellano).

Este modelo dispone de multitud de diagramas que ayudan a comprender la complejidad del futuro sistema, permitiéndonos plasmar en un lenguaje estándar aquellas funcionalidades, requisitos y demás características que hemos detectado en el sistema. Aunque UML define una gran cantidad de diagramas para representar los distintos aspectos del desarrollo de la aplicación, en nuestro caso nos hemos centrado en dos tipos de diagramas: el diagrama de clases y el diagrama de casos de usos.

## Diagramas de clases

El diagrama de clases describe la estructura del sistema mostrando las relaciones (asociación, agregación y composición) entre los distintos elementos que conforman la aplicación, expresados estos como clases con sus atributos. A continuación describiremos brevemente dichas clases:

* **Usuario (User)**: representa a los usuarios que harán uso de la aplicación.
* **Ruta (Trail)**: contiene la información acerca de una ruta.
* **Comentario de la ruta (TrailComment)**: contiene la información acerca del comentario de una ruta.
* **Actividad (Activity)**: representa una actividad deportiva que se pueda realizar, por ejemplo, salir a correr, montar en bici, etc.
* **Siguiendo** **(Follow)**: contiene la información relacionada sobre qué usuario sigue a cual.



## Diagramas de casos de uso

El diagrama de caso de uso es un tipo de **diagrama UML de comportamiento** y se usa frecuentemente para analizar varios sistemas. Permiten visualizar los diferentes tipos de roles en un sistema y cómo esos roles interactúan con el sistema. Los diagramas de casos de uso se utilizan para reunir los requisitos de uso de un sistema. Dependiendo de sus necesidades, puede utilizar esos datos de diferentes maneras. A continuación se presentan algunas formas de usarlas:

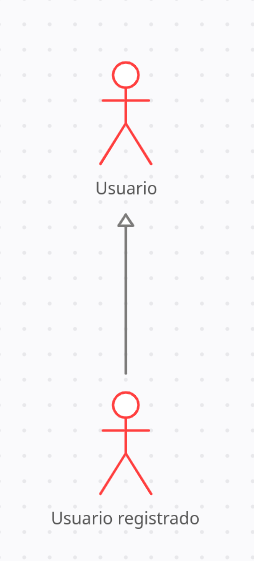
* **Identificar las funciones y la forma en que los roles interactúan con ellas**: el propósito principal de los diagramas de casos de uso.
* **Para una visión de alto nivel del sistema**: especialmente útil cuando se presenta a los administradores o a las partes interesadas. Se pueden destacar los papeles que interactúan con el sistema y la funcionalidad proporcionada por el sistema sin profundizar en el funcionamiento interno del sistema.
* **Identificar los factores internos y externos**: esto puede parecer simple pero en grandes proyectos complejos un sistema puede ser identificado como una función externa en otro caso de uso.

Los diagramas de casos de uso tienen tres elementos:

* **Actores**: Son los usuarios del sistema. Un actor puede ser una persona, un conjunto de personas, un sistema hardware o un sistema software. Los actores representan un rol, que puede desempeñar alguien que necesita intercambiar información con el sistema.
* **Casos de uso**: Un caso de uso describe una forma concreta de utilizar parte de la funcionalidad de un sistema. La colección de todos los casos de uso describe toda la funcionalidad del sistema.
* **Comunicación entre actores y casos de uso**: Cada actor ejecuta un número específico de casos de uso en la aplicación. Por eso decimos que hay comunicación entre actores y casos de uso.

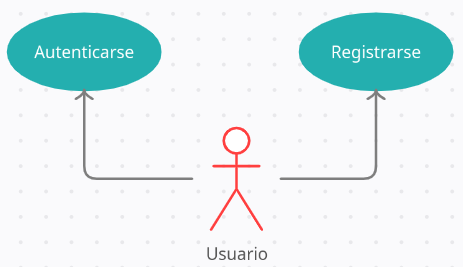
### Actores

Los usuarios son todas aquellas personas que acceden a la aplicación sin necesidad de autentificarse. Por otro lado, tenemos los usuarios registrados que son los usuarios que sí que tienen una cuenta en nuestra aplicación y que acceden a nuestra aplicación mediante su email y contraseña.



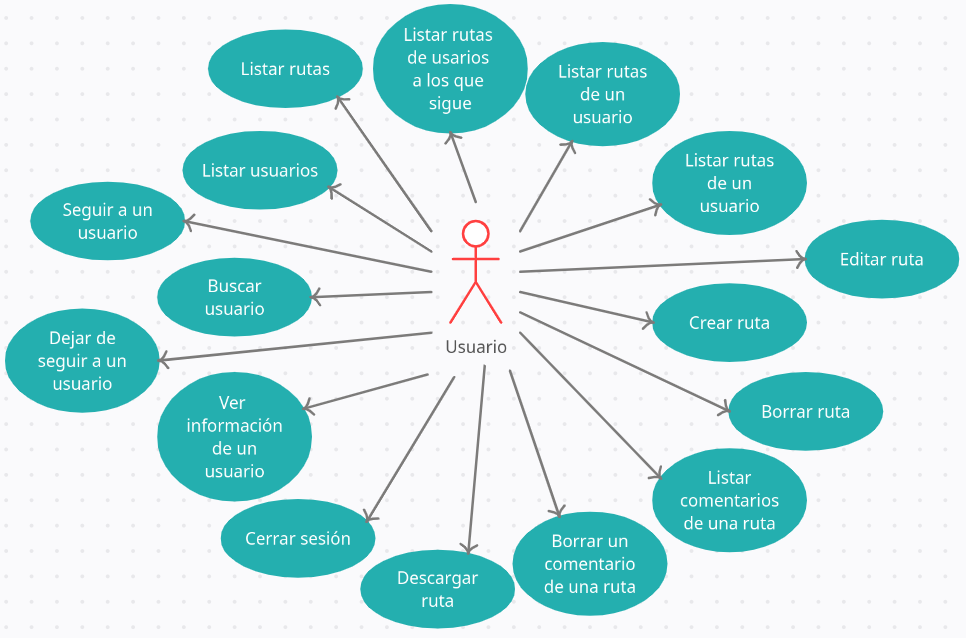
### Casos de uso del usuario

Este tipo de usuario al no tener una cuenta registrada las acciones que podrá hacer son super limitadas. Estos usuarios solo podrán loguearse o registrarse para poder hacer uso de las funcionalidades que ofrece la aplicación.



### Casos de uso del usuario registrado

El usuario registrado y logueado en la aplicación puede obtener listado de rutas, obtener listado de rutas de usuarios que el usuario sigue, obtener listado de rutas de un usuario, obtener listado de usuarios, seguir a un usuario, dejar de seguir a un usuario, ver información de un usuario, buscar un usuario, crear una ruta, editar una ruta, borrar una ruta, ver información de una ruta, descargar el fichero de la ruta, comentar una ruta,obtener listado de comentarios, borrar comentario y cerrar sesión.



# Diseño

## Introducción

El diseño es el proceso que extiende, refina y reorganiza los aspectos detectados en el proceso de modelado conceptual para generar una especificación rigurosa del sistema de información siempre orientada a la obtención de la solución del sistema software. Para el diseño del sitio web se ha escogido una arquitectura en tres capas.

La **arquitectura en tres capas** es un tipo de arquitectura usada en la gran mayoría de sistemas. Se suele usar en sistemas que implementan un modelo de negocio como podría ser una tienda online, una aplicación para gestionar ciertos datos, etc. Sin embargo, no es recomendable usarla en sistemas de tiempo real como los de los coches o aviones. Por lo tanto, esta arquitectura en tres capas es perfecta para nuestra aplicación.

Todo sistema que gestiona datos tendrá una base de datos para guardar esos datos y una interfaz de usuario que será con la que interactúan los usuarios. Además, una parte del sistema se encargará de procesar los datos y gestionar lo que se hace con ellos. La arquitectura en tres capas lo que hace es dividir el sistema en tres partes diferenciadas, de tal forma que cada capa solo se comunique con la inferior. Esas tres capas se denominan:

* **Presentación**: En esta capa se crea la interfaz del usuario. Su única función es pasarle las acciones que realice el usuario a la capa de negocio.
* **Negocio o lógica de la aplicación**: En esta capa se gestiona la lógica de la aplicación. Es donde se dice que se hace con los datos. Por ejemplo para una aplicación de gestión de una biblioteca será donde se gestione cuántos préstamos puede tener un usuario, que ocurre si un usuario se retrasa al devolver un libro, etc. Estará conectada con la capa de persistencia para poder realizar sus funciones.
* **Persistencia**: Esta capa se encarga de guardar los datos. Será donde se gestione todo lo relativo a la base de datos y a la creación, edición y borrado de datos de ésta.

Al hacer que cada capa se comunique solo con la inmediatamente inferior, conseguimos que si hay que realizar un cambio no tengamos que volver a modificar todo el sistema por completo. Si por ejemplo tenemos que cambiar la forma en la que se guardan los datos (el tipo de base de datos por ejemplo), solo tendríamos que tocar la capa de persistencia.

### Capa de presentación

La capa de presentación se puede definir como el conjunto de componentes software que implementan la interacción con los usuarios a través de una representación visual de la aplicación, proporcionando a los usuarios una forma de acceder y controlar los datos y los servicios de los objetos. A partir de la interfaz gráfica, el usuario podrá navegar por las distintas páginas para poder obtener toda la información que desee, o aportar en caso de ser necesario.

### Capa de negocio

La capa de negocio se puede definir como el conjunto de componentes software que implementan completamente el comportamiento de las clases del dominio, especificadas en la fase de modelado conceptual. Es en este nivel, por tanto, donde se implementa la funcionalidad de la aplicación web.

Esta capa sirve de enlace entre los niveles de presentación y de persistencia, ya que la capa de presentación no accede a la base de datos directamente, sino que se comunica con la capa de aplicación para demandar el servicio deseado y es la capa de aplicación la que se comunica con la capa de persistencia para recuperar los datos necesarios.

En el proyecto la capa de negocio se corresponde con las clases, es decir, con los ficheros que representan los distintos elementos del diagrama de clases y sus colecciones, almacenadas en el directorio models. Hay dos tipos de estructura, dependiendo de si se trata de la definición de un elemento o la definición de una colección de elementos.

Los elementos son User, Trail, Follow, Activity and TrailComment y tienen la siguiente estructura:

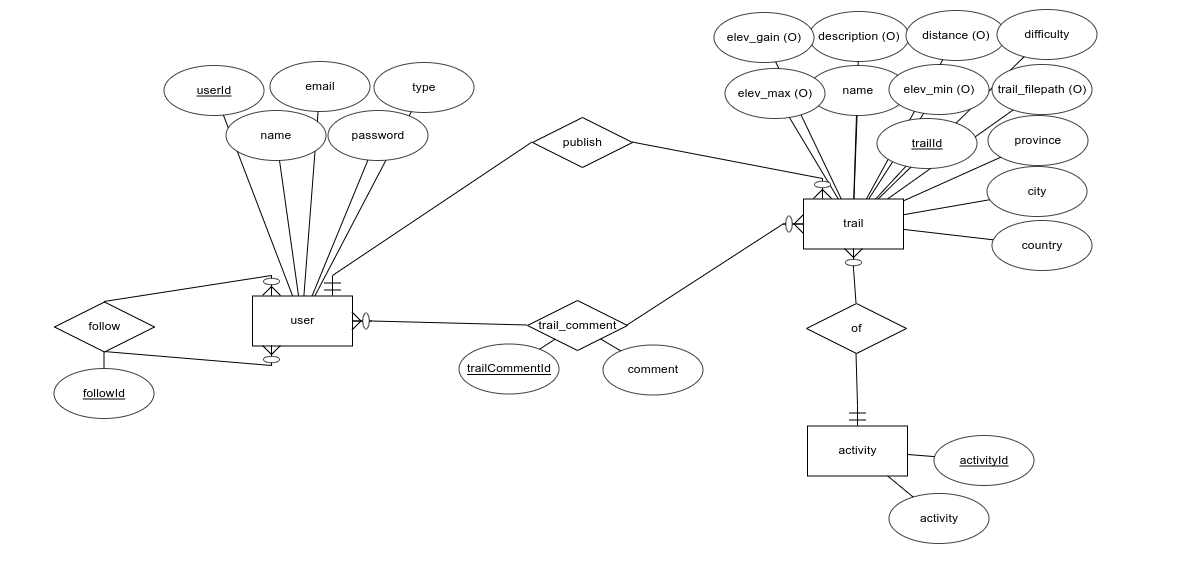
* **Atributos**: Define las propiedades del elemento.

### Capa de persistencia

La capa de persistencia se puede definir como el conjunto de componentes software que proporcionan una serie de servicios que permiten a los objetos del dominio interactuar con su repositorio permanente asociado.

En el proyecto la capa de persistencia se corresponde con la base de datos de la aplicación y las distintas tablas que la conforman. Estas son user, follow, trail, activity, trail\_comment.

A continuación se muestra el diagrama entidad-relación utilizado para la implementación de la base de datos de nuestro proyecto.



# Implementación

## Tecnologías utilizadas en el desarrollo del proyecto

### HTML

HTML (Lenguaje de Marcas de Hipertexto, del inglés **HyperText Markup Language**) es el componente más básico de la Web. Define el significado y la estructura del contenido web. Además de HTML, generalmente se utilizan otras tecnologías para describir la apariencia/presentación de una página web (CSS) o la funcionalidad/comportamiento (JavaScript).

"Hipertexto" hace referencia a los enlaces que conectan páginas web entre sí, ya sea dentro de un único sitio web o entre sitios web. Los enlaces son un aspecto fundamental de la Web. Al subir contenido a Internet y vincularlo a las páginas creadas por otras personas, te conviertes en un participante activo en la «World Wide Web» (Red Informática Mundial).

HTML utiliza "marcas" para etiquetar texto, imágenes y otro contenido para mostrarlo en un navegador Web. Las marcas HTML incluye "elementos" especiales como <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, <div>, <span>, <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li> y muchos otros.

Un elemento HTML se distingue de otro texto en un documento mediante "etiquetas", que consisten en el nombre del elemento rodeado por "<" y ">". El nombre de un elemento dentro de una etiqueta no distingue entre mayúsculas y minúsculas. Es decir, se puede escribir en mayúsculas, minúsculas o una mezcla. Por ejemplo, la etiqueta <title> se puede escribir como <Title>, <TITLE> o de cualquier otra forma.

### SCSS/SASS

Sass (Syntactically Awesome Stylesheets) es un lenguaje de hoja de estilos en cascada que después de sus versiones iniciales, sus creadores han continuado extendiendo Sass con SassScript, un lenguaje de script simple, usado en los ficheros Sass.

Sass es un metalenguaje de Hojas de Estilo en Cascada (CSS). Es un lenguaje de script que es traducido a CSS, SassScript es el lenguaje de script en sí mismo. Sass consiste en dos sintaxis. La sintaxis original, llamada indented syntax («sintaxis con sangrado»), usa la sangría para separar bloques de código y el carácter nueva línea para separar reglas. La sintaxis más reciente, SCSS, usa el formato de bloques como CSS. Este usa llaves para denotar bloques de código y punto y coma (;) para separar las líneas dentro de un bloque. La sintaxis indentada y los ficheros SCSS tienen las extensiones .sass y .scss respectivamente.

CSS3 consiste en una serie de selectores y pseudo-selectores que agrupan las reglas que son aplicadas. Sass (en el amplio contexto de ambas sintaxis) extiende CSS proveyendo de varios mecanismos que están presentes en los lenguajes de programación tradicionales, particularmente lenguajes orientados a objetos, pero este no está disponible para CSS3 como tal. Cuando SassScript se interpreta, este crea bloques de reglas CSS para varios selectores que están definidos en el fichero SASS. El intérprete de SASS traduce SassScript en CSS. Alternativamente, Sass puede monitorear los ficheros .sass o .scss y convertirlos en un fichero .css de salida cada vez que el fichero .sass o .scss es guardado. Sass es simplemente azúcar sintáctica para escribir CSS.

La implementación oficial de Sass es open-source y está escrita en Ruby, sin embargo existen otras implementaciones, incluyendo una en PHP para Drupal.

La sintaxis indentada es un metalenguaje. SCSS es un metalenguaje anidado, lo que es válido en CSS es válido en SCSS con la misma semántica. Sass soporta la integración con la extensión de Firefox Firebug.

SassScript proporciona los mecanismos siguientes: variables, nesting (anidamiento), mixins, y herencia​ de los selectores.

### TypeScript

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. Anders Hejlsberg, diseñador de C# y creador de Delphi y Turbo Pascal, ha trabajado en el desarrollo de TypeScript.​ TypeScript es usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor, o extensiones para programas (Node.js y Deno).

TypeScript extiende la sintaxis de JavaScript, por tanto cualquier código JavaScript existente debería funcionar sin problemas. Está pensado para grandes proyectos, los cuales a través de un compilador de TypeScript se traducen a código JavaScript original.

TypeScript soporta ficheros de definición que contengan información sobre los tipos de librerías JavaScript existentes, similares a los ficheros de cabeceras de C/C++ que describen la estructura de ficheros de objetos existentes. Esto permite a otros programas usar los valores definidos en los ficheros como si fueran entidades TypeScript de tipado estático. Existen cabeceras para librerías populares como jQuery, MongoDB y D3.js, y los módulos básicos de Node.js.

El compilador de TypeScript está escrito asimismo en TypeScript, compilado a JavaScript y con Licencia Apache 2.

TypeScript se incluye como lenguaje de programación de primer nivel en Microsoft Visual Studio 2013 Update 2 y posteriores, junto a C# y otros lenguajes de Microsoft. Una extensión oficial permite a Visual Studio 2012 soportar también TypeScript.

### Angular

Angular (comúnmente llamado Angular 2+ o Angular 2) es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript.

Angular se basa en clases tipo "Componentes", cuyas propiedades son las usadas para hacer el binding de los datos. En dichas clases tenemos propiedades (variables) y métodos (funciones a llamar).

Angular es la evolución de AngularJS aunque incompatible con su predecesor.

### Node.js

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.

### Express

Express es el framework web más popular de Node, y es la librería subyacente para un gran número de otros frameworks web de Node populares. Proporciona mecanismos para:

* Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes caminos URL (rutas).
* Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
* Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.
* Añadir procesamiento de peticiones "middleware".

A pesar de que Express es en sí mismo bastante minimalista, los desarrolladores han creado paquetes de middleware compatibles para abordar casi cualquier problema de desarrollo web. Hay librerías para trabajar con cookies, sesiones, inicios de sesión de usuario, parámetros URL, datos POST, cabeceras de seguridad, etc.

### TypeORM

El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, **ORM**, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM.

**TypeORM** es un ORM que se puede ejecutar en plataformas como **NodeJS**, Cordova, Ionic, React Native, Electron, etc. Se puede utilizar con **TypeScript** y JavaScript (ES5, ES6, ES7, ES8). Su objetivo es ofrecer soporte siempre a las últimas características de JavaScript y proporcionar características adicionales que ayuden a desarrollar cualquier tipo de aplicación que utilice bases de datos, desde aplicaciones pequeñas con unas pocas tablas hasta aplicaciones empresariales a gran escala con varias bases de datos.

TypeORM es compatible con los patrones Active Record y Data Mapper, lo que significa que puede escribir aplicaciones de alta calidad, escasamente acopladas, escalables y fáciles de mantener de la manera más productiva. TypeORM está muy influenciado por otros ORM, como Hibernate, Doctrine y Entity Framework .

### SQL

SQL (por sus siglas en inglés Structured Query Language; en español lenguaje de consulta estructurada) es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.​ Una de sus principales características es el manejo del álgebra y el cálculo relacional para efectuar consultas con el fin de recuperar, de forma sencilla, información de bases de datos, así como realizar cambios en ellas.

SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones. Es un lenguaje declarativo de "alto nivel" o "de no procedimiento" que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación y la orientación a objetos. De esta forma, una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros. SQL también tiene las siguientes características:

* **Lenguaje de definición de datos**: El LDD de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.
* **Lenguaje interactivo de manipulación de datos**: El LMD de SQL incluye lenguajes de consultas basado tanto en álgebra relacional como en cálculo relacional de tuplas.
* **Integridad**: El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.
* **Definición de vistas**: El LDD incluye comandos para definir las vistas.
* **Control de transacciones**: SQL tiene comandos para especificar el comienzo y el final de una transacción.
* **SQL incorporado y dinámico**: Esto quiere decir que se pueden incorporar instrucciones de SQL en lenguajes de programación como: C++, C, Java, PHP, COBOL, TypeScript, etc.
* **Autorización**: El LDD incluye comandos para especificar los derechos de acceso a las relaciones y a las vistas.

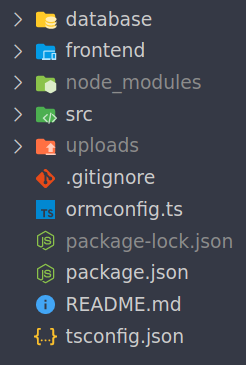
### MySQL

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, todo para entornos de desarrollo web.

## Descripción del proyecto

Antes de empezar a explicar en mayor profundidad cada una de las capas de las que está compuesta el proyecto vamos a ver la estructura de las carpetas por las que está compuesta la aplicación:

* **database**: aquí se encuentran scripts sql para la creación de la base de datos, e imágenes de los diagramas de la base de datos.
* **frontend**: dentro de esta se encuentra toda la parte frontal de la aplicación realizada con Angular.
* **.gitignore**: fichero donde se encuentran listados los ficheros que no queremos que se suban a GitHub.
* **README.md**: contiene información acerca del proyecto, este archivo es el que se previsualiza al entrar en el proyecto en GitHub. Es una forma de documentación de software, usualmente en un archivo de texto plano llamado READ.ME, README.TXT, README.md (para un archivo Markdown), README.1ST o simplemente README.



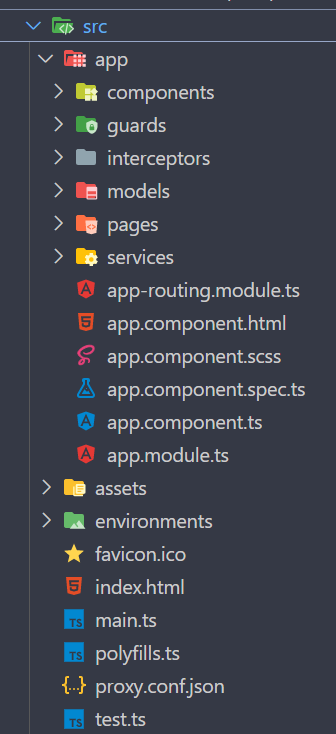
El resto de ficheros que podemos encontrar en el directorio principal del proyecto son todos del backend, vamos a verlo:

* **ormconfig.ts**: aquí es donde introduzco la configuración de TypeORM. Creo las variables relacionadas a la base de datos como el nombre de la base de datos, el usuario, la contraseña y otras configuraciones propias de TypeORM.
* **package.json**: es un archivo de texto en formato json y que contiene información para nuestro proyecto, en el cual especificamos referencias al proyecto como: autor, repositorio, versión, las dependencias y scripts para ejecutar el proyecto, entre otros.
* **package-lock.json**: contiene información del proyecto, así como la versión exacta de cada una de las dependencias de nuestro proyecto. Este fichero permite dar una mayor estabilidad a nuestro proyecto.
* **tsconfig.json**: establece las opciones de configuración del compilador de typescript.
* **src**: aquí es donde se encuentra la lógica de la parte del backend. Es donde se encuentran los controladores, middlewares, clases, etc.
* **uploads**: esta carpeta es utilizada para subir los ficheros gpx que son subidos a la aplicación, es decir, es utilizada a modo de storage.

### Capa de presentación

La capa de presentación corresponde al frontend de nuestra aplicación, el cual está realizado en Angular v11. Al generar el proyecto de Angular con angular cli (herramienta de consola de Angular), nos genera una estructura preestablecida para organizar los ficheros. Veamos un poco como se ha organizado el código de la aplicación.

* **e2e**: carpeta dedicada a testing “end to end”.
* **src**: aquí es donde se encuentra toda la lógica de la parte frontal.
* **angular.json**: este fichero con formato json sirve para configurar angular. Por ejemplo, si queremos implementar un proxy tendremos que indicarle aquí la configuración.

Dado que la carpeta src es la más importante ya que es donde se encuentra todo el código del proyecto. Dentro de src podemos encontrar los siguientes ficheros y carpetas, de los cuales solo comentaremos los más relevantes:

* **index.html**: es el html principal, a partir del cual se cargará el resto de la aplicación.
* **environments**: en esta carpeta guardaremos los archivos de configuración de los diferentes entornos que vayamos a utilizar. Los principales entornos que se utilizan son:
  + **Desarrollo**: Entorno para realizar simulaciones con datos reales antes de enviar la versión definitiva al cliente.
  + **Producción**: Entorno final, aquí la aplicación ya estaría funcionando sin fallos y con datos reales.
* **assets**: carpeta para guardar los diferentes recursos que necesitemos en nuestra aplicación como puedan ser imágenes, iconos, tipos de fuentes o archivos de traducciones si vamos a crear una aplicación con varios idiomas.
* **proxy.conf.json**: un proxy sirve como una puerta de enlace entre nuestra aplicación e internet. Es un servidor intermedio entre el cliente y los servicios a consumir. Por tanto, dentro de este fichero introducimos la configuración del proxy de angular en formato json.
* **app-routing.module.ts**: un módulo de Angular agrupa un conjunto de artefactos de Angular, como son componentes, directivas, pipes y servicios que forman parte de ese mismo módulo. Este fichero, por tanto, agrupa la lógica para gestionar las distintas rutas de la aplicación.
* **app**: en esta carpeta están los componentes, módulos, clases, servicios y archivos de rutas de nuestra aplicación.
  + **app.module.ts**: es el fichero de módulos principal.
  + **services**: dentro de esta carpeta es donde se encuentran los distintos servicios de los que se hacen uso en los componentes de la aplicación.
  + **models**: aquí se encuentran las clases e interfaces a utilizar.
  + **interceptors**: un interceptor inspecciona/modifica las peticiones, osea, lo que va de tu aplicación a la api y también lo que viene de la api a la aplicación. Por tanto, dentro de esta carpeta encontraremos todos los interceptores que queramos configurar para las peticiones.
  + **guards**: aquí es donde se encuentran los distintos guards de nuestra aplicación. Los guards controlan las rutas de navegación entre páginas de la aplicación web, permitiendo o denegando el acceso por ejemplo.
  + **components**: aquí es donde se encuentran todos los componentes compartidos por las distintas páginas de la aplicación. Un componente en Angular es un bloque de código reutilizable, que consta básicamente de 4 archivos: un CSS (en mi casos SCSS), un HTML (también conocido como plantilla o en inglés, template), un TypeScript y uno para testing con las extensión spec.ts.La carpeta app con la que viene Angular por defecto es un componente, aunque un tanto especial. Pero en definitiva también tiene esos mismos archivos. Además, dentro de components, tenemos otra carpeta llamada layouts:
    - **layouts**: esta carpeta sirve para almacenar los componentes que son de la estructura del diseño, por ejemplo, footer o el header. De esta forma sabremos que los componentes que hay en esta carpeta forman parte de la estructura de la página.
  + **pages**: cada ruta de angular está vinculada a un componente mediante el fichero app-routing.module.ts. Por tanto, los componentes que se encuentran en esta carpeta son una página de la aplicación.

### Capa de negocio o lógica de la aplicación

La capa de negocio está formada por el backend, que sería la api donde el frontend realizará las peticiones. La lógica de la api se encuentra en la carpeta src del directorio base. La mayoría de las apis utilizadas a día de hoy son APIs REST (Representational State Transfer- Transferencia de Estado Representacional). Antes de introducir el concepto de REST API, es necesario explicar qué es una API (Interfaz de programación de aplicaciones). Es un conjunto de requisiciones que permite la comunicación de datos entre aplicaciones. Para eso, la API utiliza requisiciones HTTP responsables de las operaciones básicas necesarias para la manipulación de datos. Las principales solicitudes son:

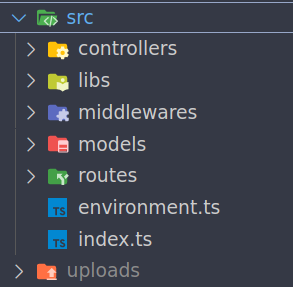
* **POST**: crea datos en el servidor.
* **GET**: lectura de datos en el host.
* **DELETE**: borra la información.
* **PUT**: registro de actualizaciones.

REST es un conjunto de restricciones que se utilizan para que las solicitudes HTTP cumplan con las directrices definidas en la arquitectura. Básicamente, las restricciones determinadas por la arquitectura REST son:

* **Cliente-servidor**: las aplicaciones existentes en el servidor y el cliente deben estar separadas.
* **Sin estado**: las requisiciones se realizan de forma independiente, es decir, cada una ejecuta sólo una determinada acción.
* **Caché**: la API debe utilizar la caché para evitar llamadas recurrentes al servidor.
* **Interfaz uniforme**: agrupa otros cuatro conceptos en los que se determina que los recursos deben ser identificados, la manipulación de los recursos debe ser a través de la representación, con mensajes autodescriptivos y utilizando enlaces para navegar por la aplicación.

Luego, cuando se habla de REST API, significa utilizar una API para acceder a aplicaciones backend, de manera que esa comunicación se realice con los estándares definidos por el estilo de arquitectura REST.

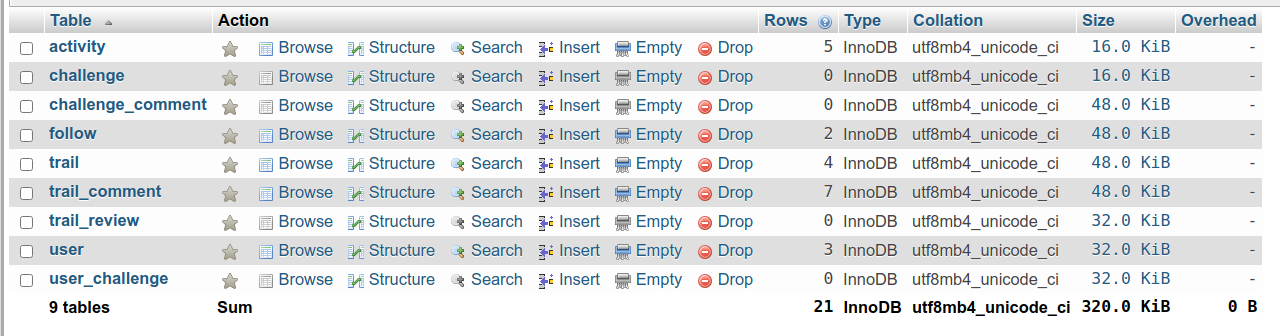
Una vez entendido esto, ya podemos entender bien la estructura MVC que sigue el proyecto, veámoslo:

* **controllers**: aquí se encuentran los distintos controladores de la aplicación.
* **libs**: esta carpeta contiene clases con utilidades varias para la aplicación, de esta forma me permite reutilizar código sin necesidad de repetir la misma función en dos puntos distintos con la misma finalidad.
* **middlewares**: aquí se encuentran los middlewares implementados en la aplicación. Los middlewares son un sistema de software que ofrece servicios y funciones comunes para las aplicaciones. En general, el middleware se encarga de las tareas de gestión de datos, servicios de aplicaciones, mensajería, autenticación y gestión de API. En el caso de nuestra api, el middleware actúa como intermediario, de esta manera antes de hacer uso del controlador, el middleware procesa la petición y comprueba si cumple una serie de requisitos para permitir el uso del controlador. Por ejemplo, si queremos crear una nueva ruta, el usuario debe estar logueado y por ello debe pasar un token a nuestra api para que ésta permita realizar dicha acción. El middleware se encarga de procesar la petición y comprobar si el token es válido, de esta manera ahorramos repetir código.
* **models**: contiene las clases de las que está compuesta la api.
* **routes**: aquí se encuentran las rutas que tiene nuestra api. En el fichero de ruta es donde especifico el middleware que utiliza cada endpoint y que función del controlador utiliza. En la siguiente captura podemos ver un ejemplo de uno de los ficheros de ruta. En la parte remarcada indicamos que el router utiliza la ruta “/create”, entre corchetes pasamos como parámetro un array de los middlewares que utiliza esta ruta y por último indico el controlador y la función del controlador que utiliza este endpoint.



### Capa de persistencia o de datos

A continuación se muestran las distintas tablas que conforman la base de datos de la aplicación:



Como podemos observar en la captura, la base de datos está compuesta por nueve tablas: activity, challenge, challenge\_comment, follow, trail, trail\_comment, trail\_review, user y user\_challenge. La tabla user, por ejemplo, está compuesta por los siguientes campos:

* **userId**: es la clave primaria de la tabla y hace identificador del usuario.
* **name**: nombre del usuario.
* **email**: email del usuario.
* **password**: contraseña del usuario.
* **type**: tipo de usuario. 0 para admin y 1 usuario normal.

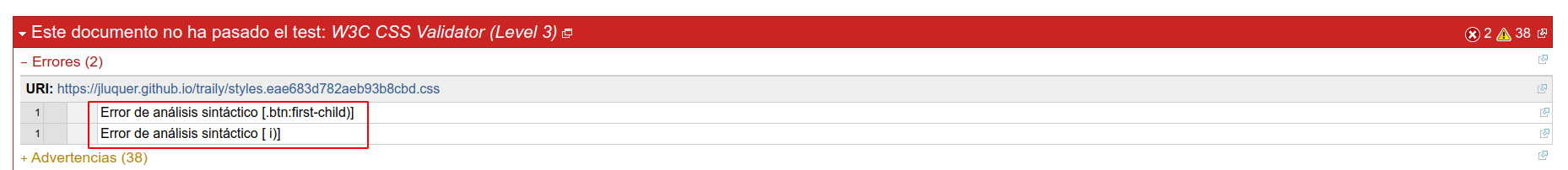
# Evaluación

## Introducción

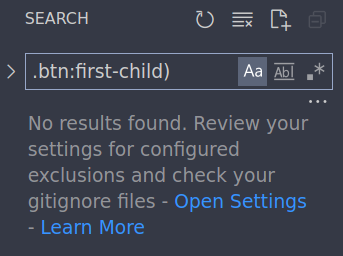
La fase de evaluación y pruebas concluye el ciclo de vida de nuestro proyecto y lo prepara para subirlo al servidor y que se ejecute. Esta fase mide el nivel de calidad que ofrece al usuario la aplicación creada. Las herramientas que vamos a usar para realizar estas pruebas son gratuitas y las podemos encontrar online.

## Validación de CSS

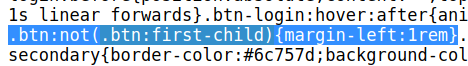
Dado que se ha utilizado el preprocesador sass debemos realizar una prueba de validación css, para ello hemos utilizado la siguiente aplicación web: <https://jigsaw.w3.org/css-validator/>. La validación de errores aunque es errónea tal y como podemos ver en la captura son a causa del preprocesador. Además, estos errores sintácticos son a causa de malas puntuaciones como que falte un ; en el código, y si vamos a las partes correspondientes del código podemos ver que todo está correcto.



Si buscamos en VSCode por ambos erros podemos ver lo siguiente:

* El primer error ni siquiera se encuentra dentro de nuestro código de scss. 

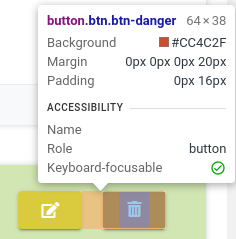
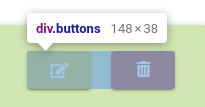
Si buscamos dentro del css compilado (segunda captura) podemos ver que el código está correctamente.



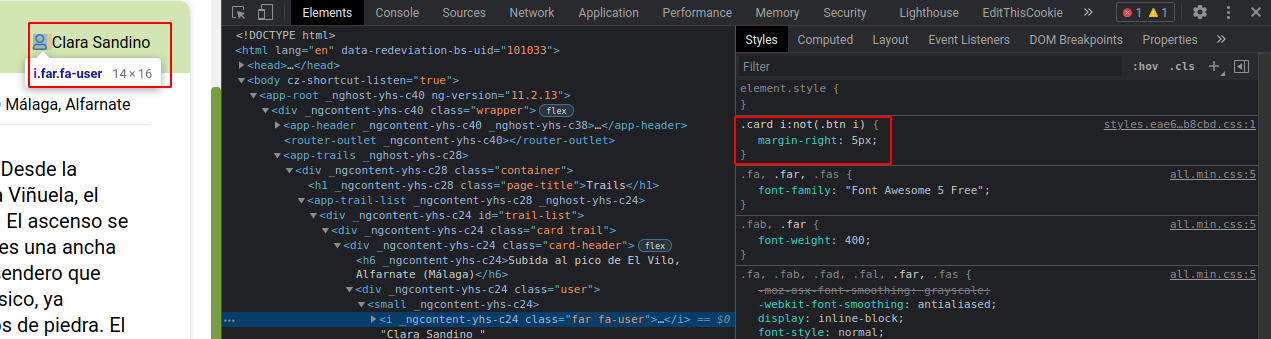
* Lo mismo ocurre con el segundo error tal y como podemos ver en la captura de pantalla. Todo se encuentra correctamente, sin embargo, salta error.



Además, si nos vamos al inspector de elementos del navegador y analizamos la aplicación podemos ver que se está aplicando correctamente. En el primer caso, en la primera captura podemos ver lo que ocupa el div de buttons. Si tuviera un margin left el primer botón, debería tener un mayor tamaño. Y el segundo botón si que tiene un margin left como podemos ver en la segunda captura.

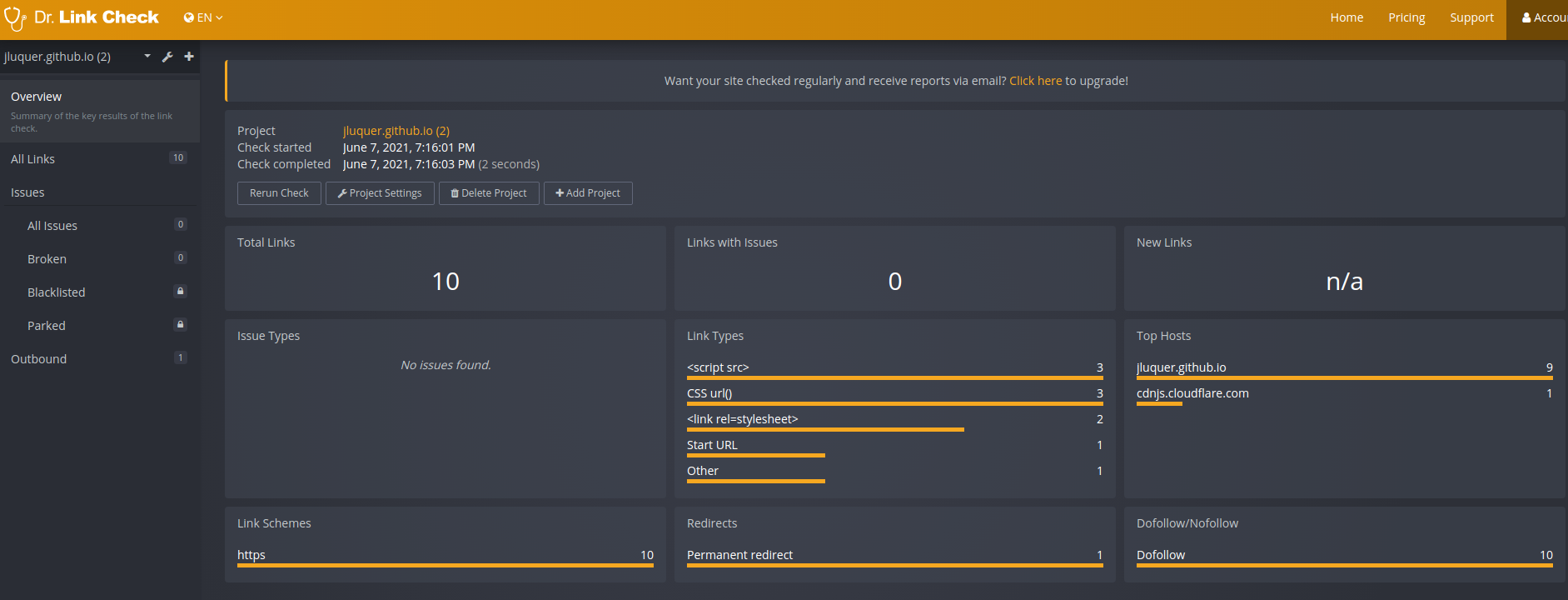


En el segundo error podemos ver que sí que está aplicando el margen tal y como nosotros queremos tal y como podemos ver en la siguiente captura.



### Validación de enlaces

Para validar los enlaces de la página se ha utilizado la siguiente aplicación web: <https://www.drlinkcheck.com/>. Tal y como podemos ver en la siguiente captura nuestra aplicación ha pasado correctamente el test ya que no reporta ningún enlace con problemas.

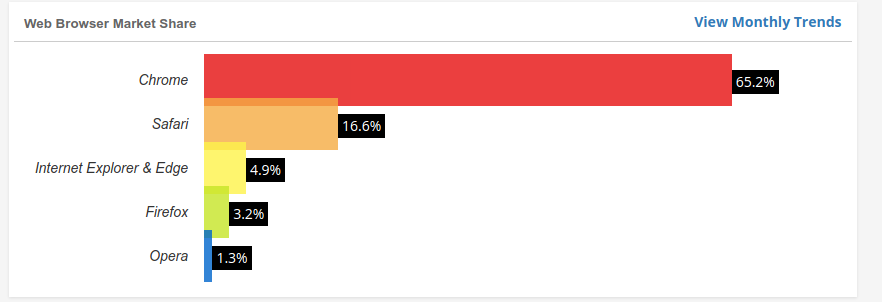


### Validación de la resolución

Uno de los requisitos para la aplicación es que se pudiera visualizar desde cualquier dispositivo. Debido a que hoy en día se sigue un diseño mobile first no se produce ningún descuadre en ninguna de las resoluciones, ya que está realizada para que no ocurra esto y, por tanto, podemos ver todos los elementos colocados adecuadamente.

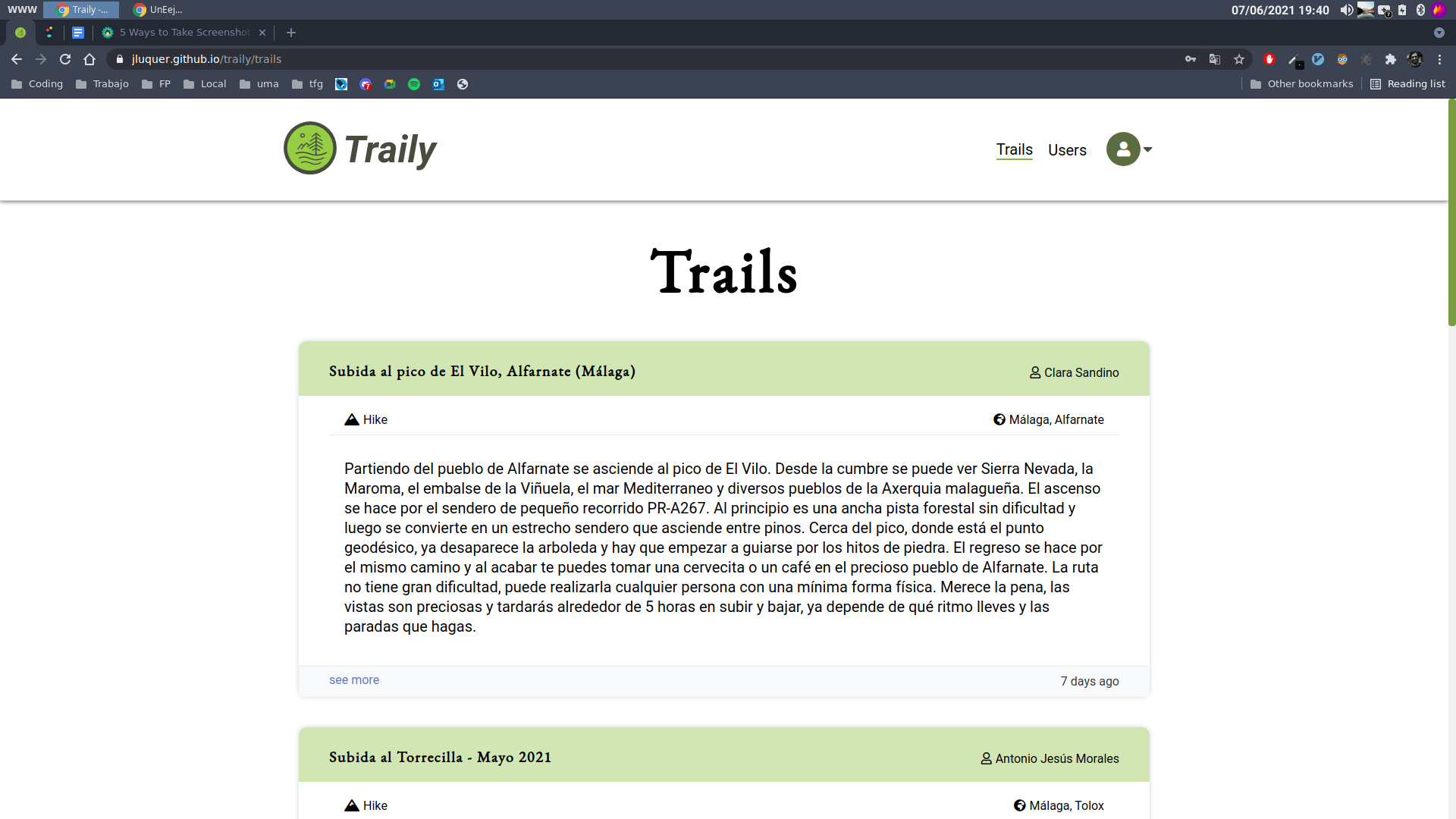
### Validación de navegadores

Dado que no todos los usuarios utilizarán el mismo navegador se ha buscado que la página web se visualice correctamente en los navegadores más populares. En mayo de 2021 los navegadores más utilizados han sido los siguientes:

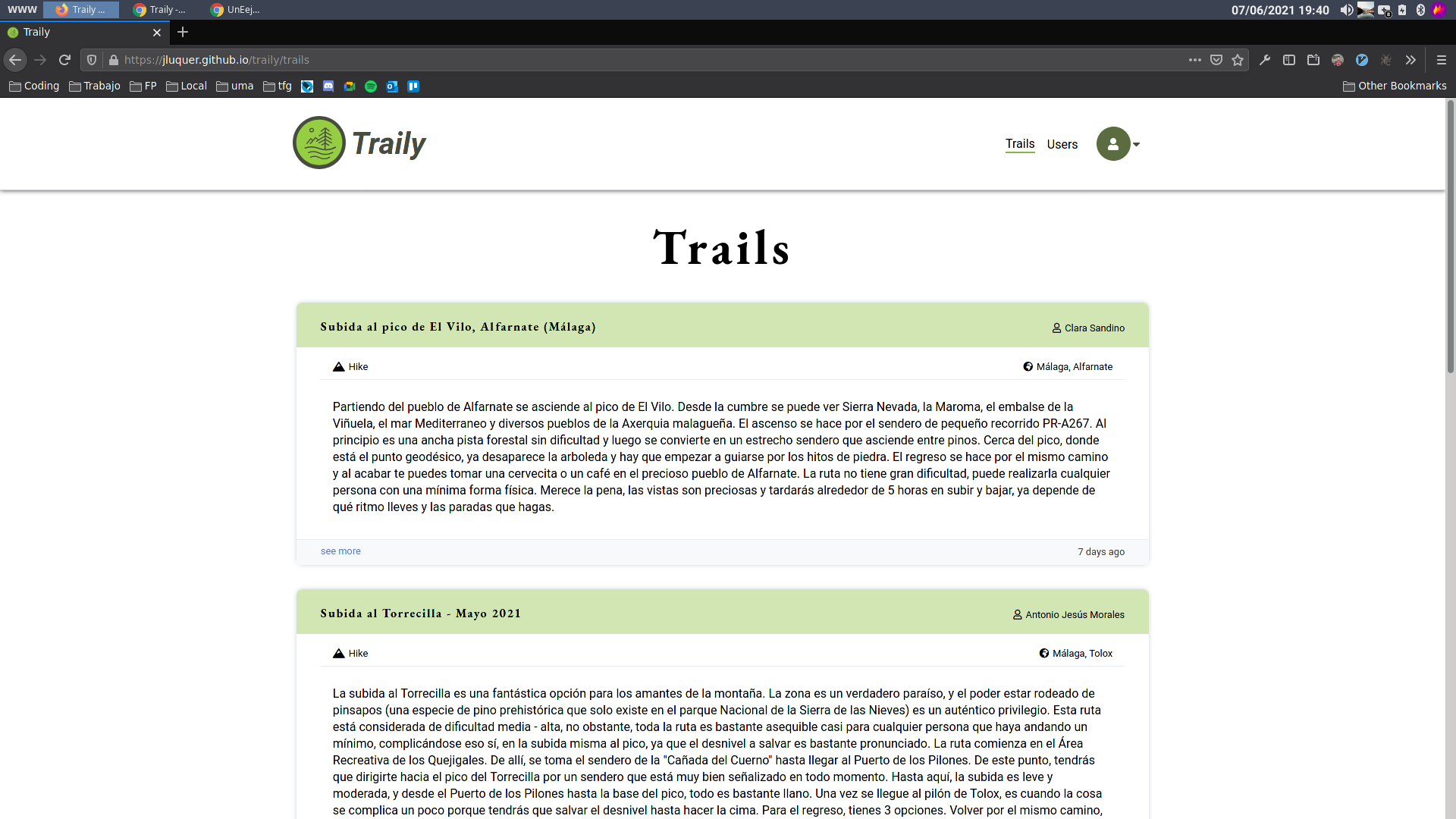


Veamos, por tanto, como ha quedado la página en cada uno de los navegadores:

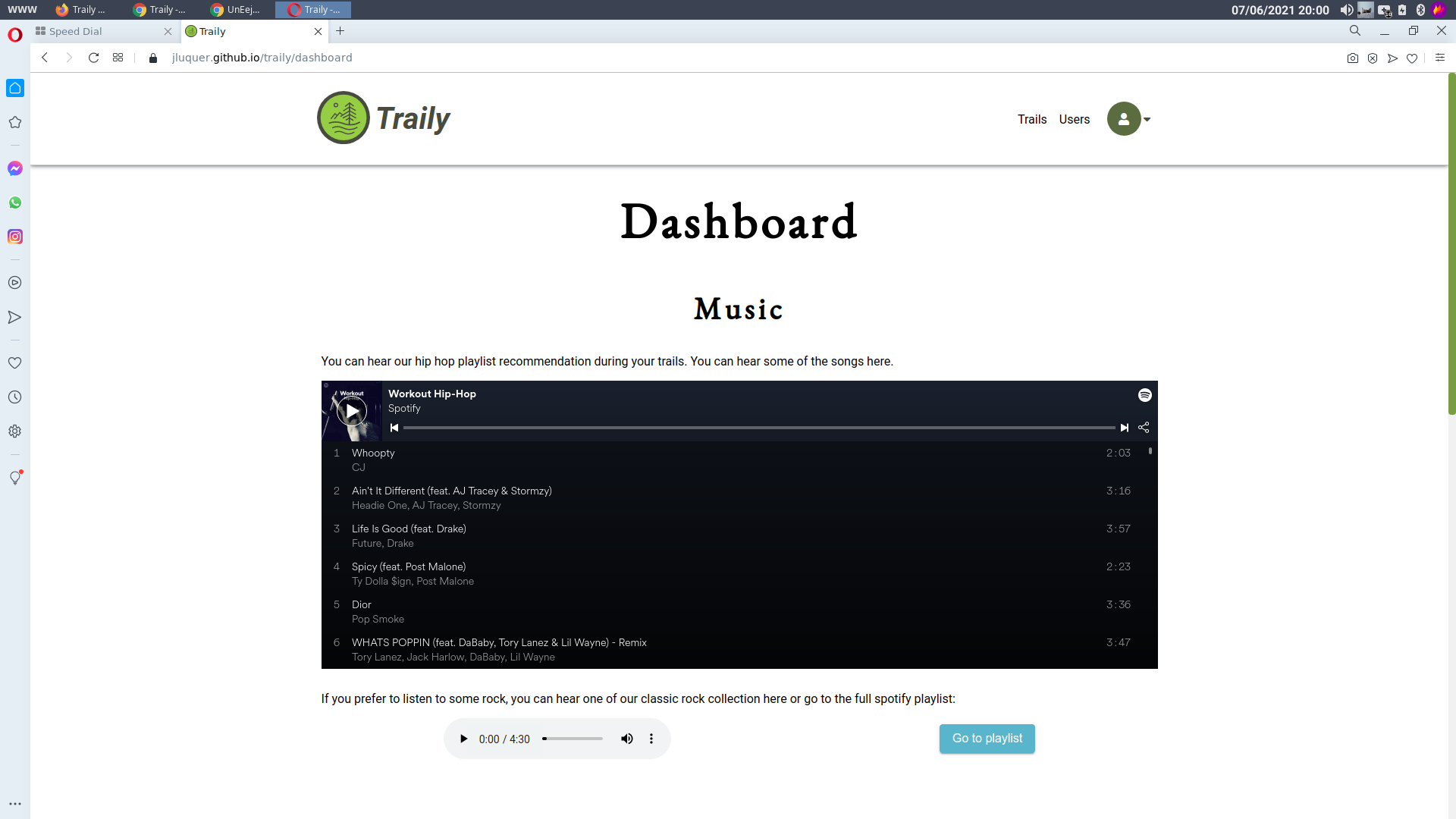
* Chrome



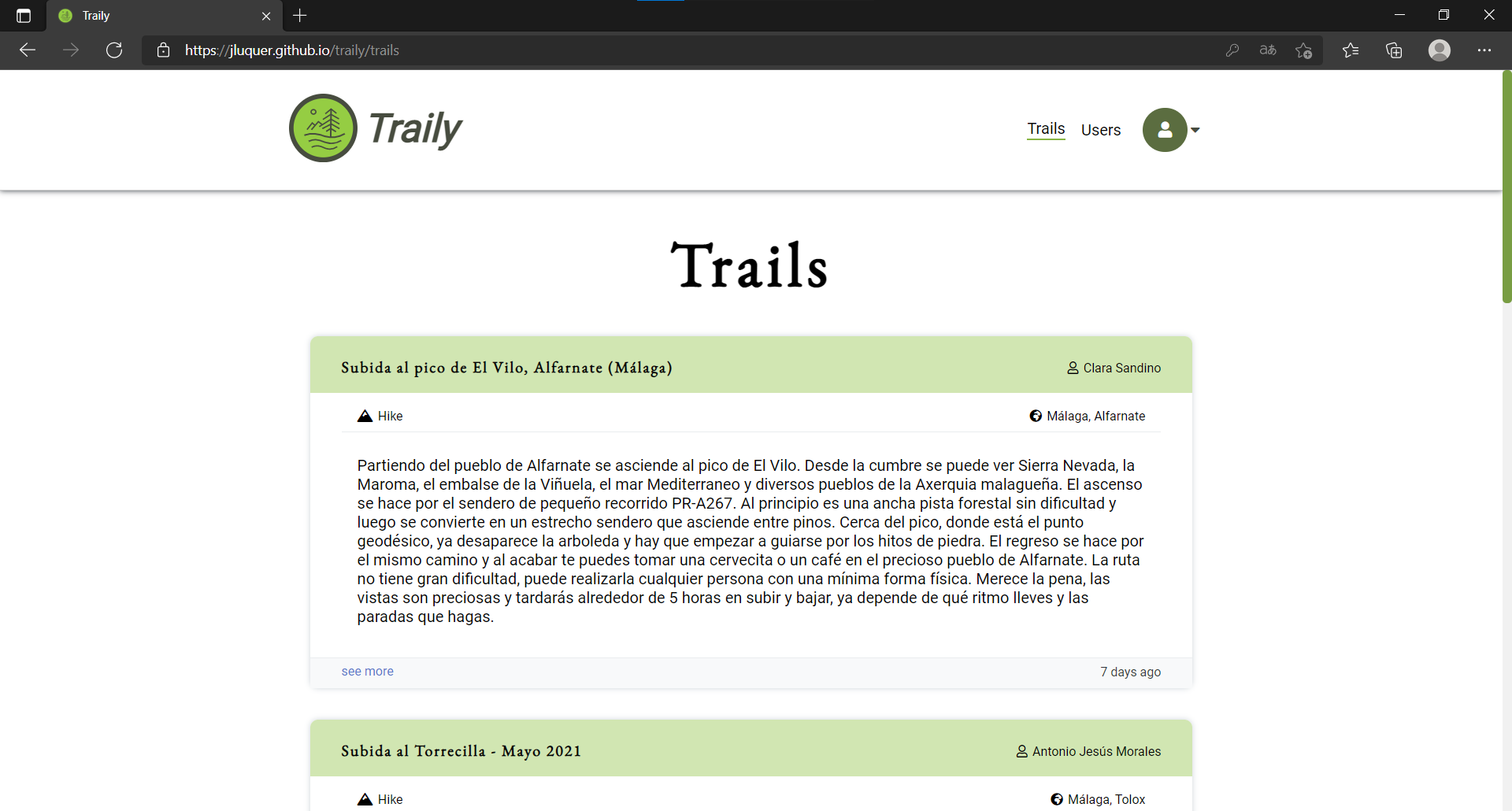
* Firefox



* Opera

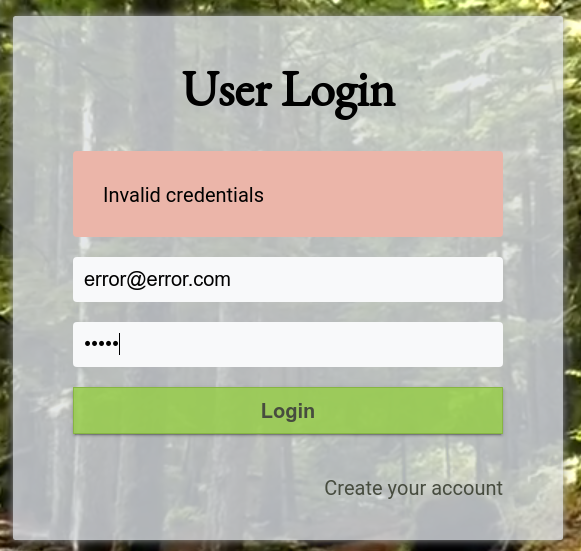


* Internet Explorer/Edge



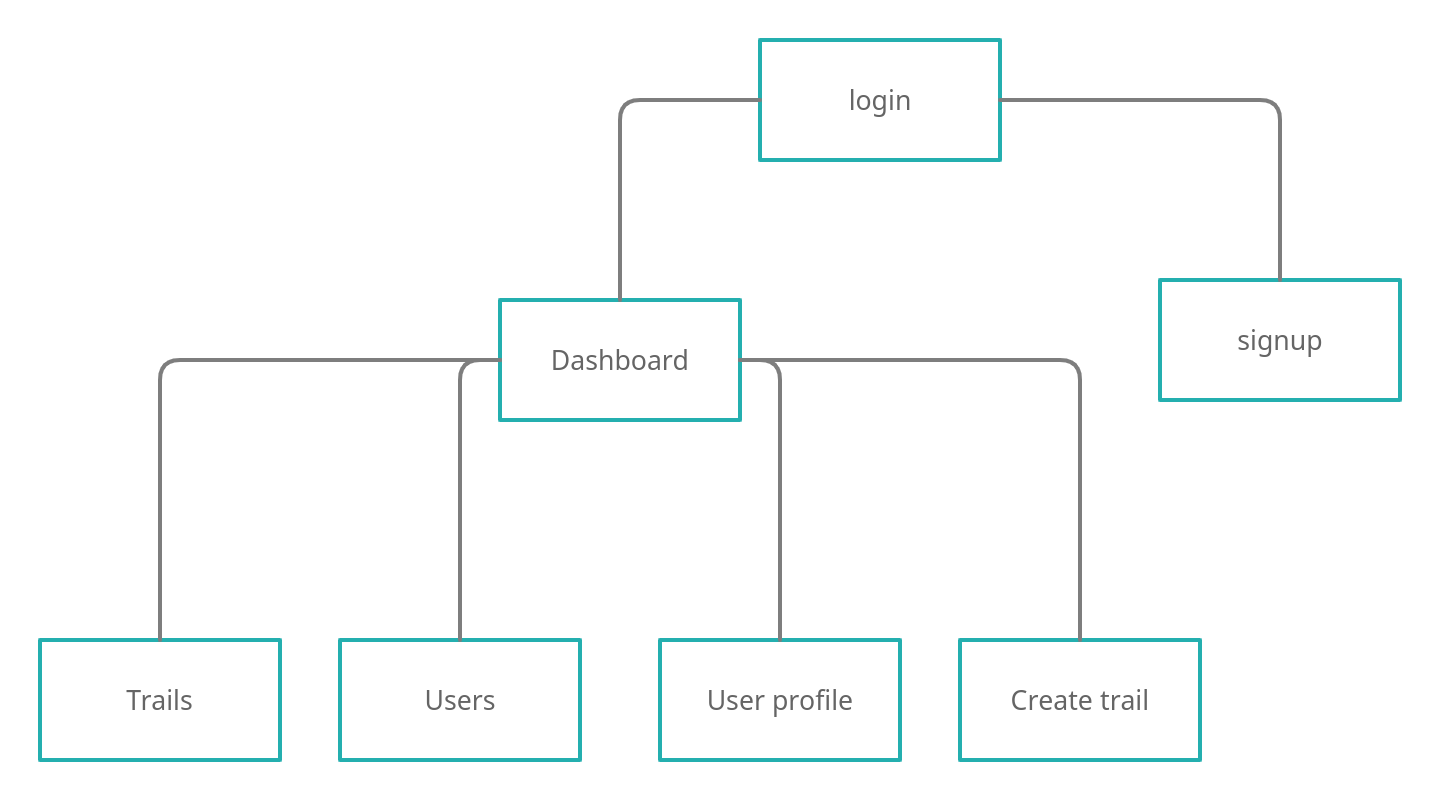
### Validación de la seguridad en el acceso a la zona privada

Para el control de acceso a las rutas de la aplicación he utilizado los guards de Angular. Por tanto, cuando intento acceder a una ruta que requiere estar logueado me redirecciona al login. Lo mismo ocurre si intento acceder al login y estoy logueado, Angular me redirecciona de nuevo al dashboard de la aplicación. En el caso de introducir unos datos erróneos en el login se muestra un error como el siguiente.



# Diseño Interfaz

## Mapa de navegación

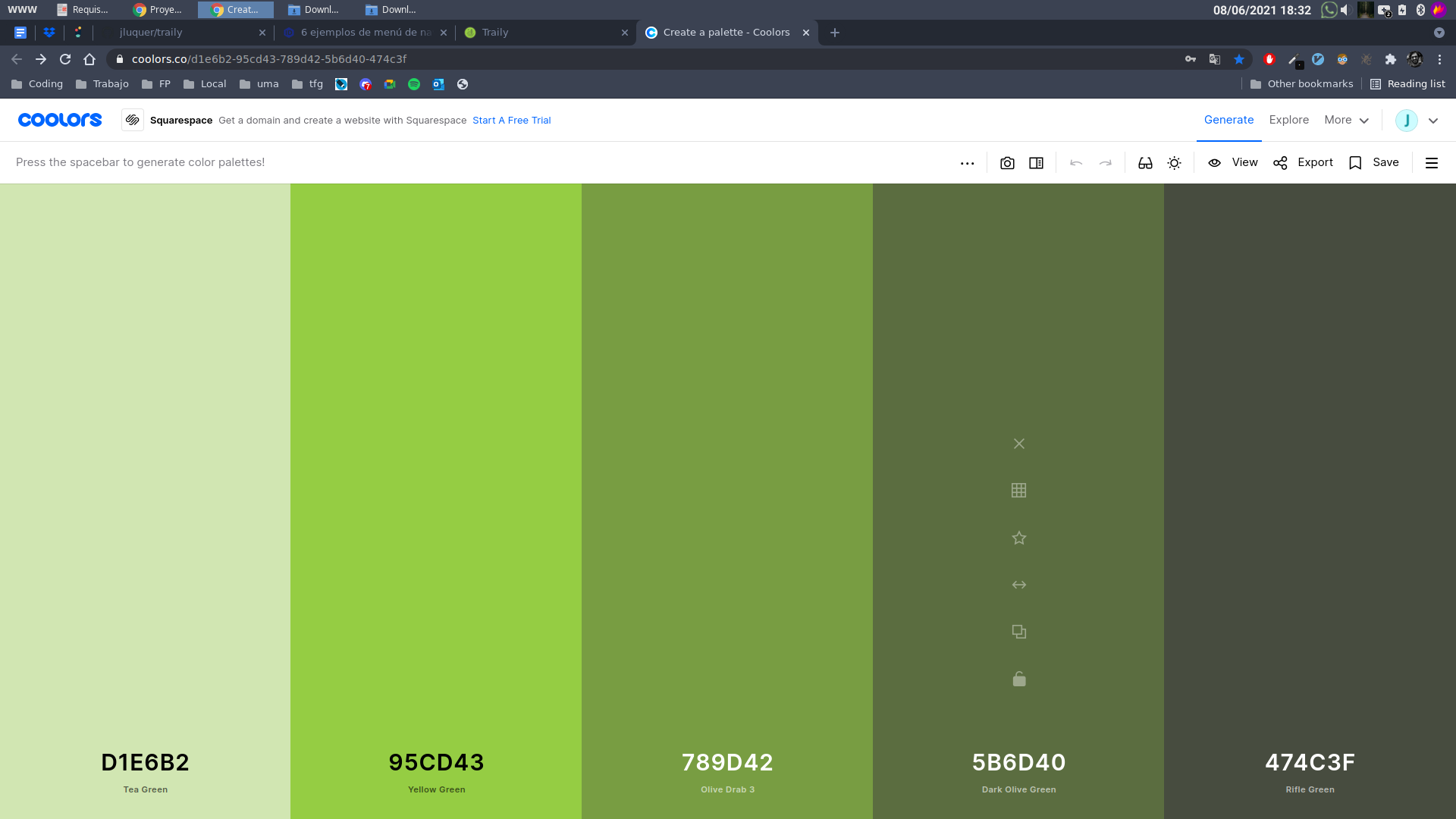


## Paleta de color

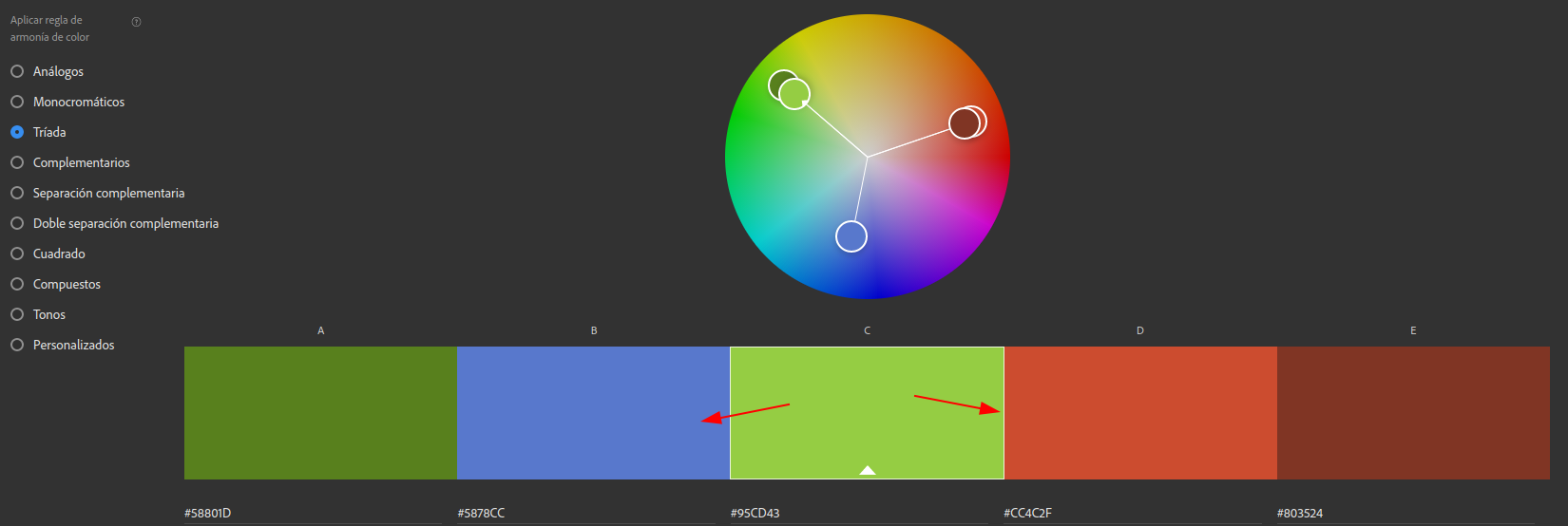
Para comenzar a realizar la paleta de color primero escogí el color principal de la paleta que fue el verde, ya que la aplicación está pensada para realizar rutas principalmente por la montaña y, por tanto, un color tierra como es el verde es perfecto para esta temática. El color verde es el color que predomina en el logo tal y como podemos ver:



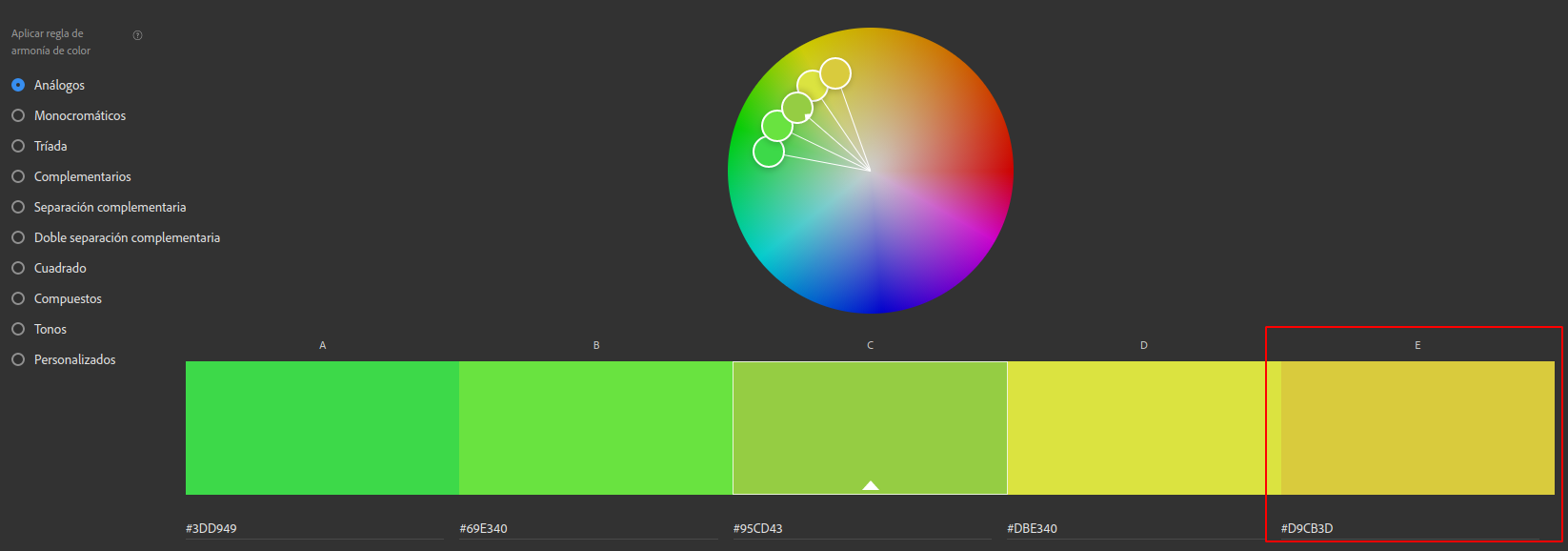
Una vez obtenido el color principal se ha obtenido la paleta de colores utilice la herramienta online [coolors](https://coolors.co/d1e6b2-95cd43-789d42-5b6d40-474c3f). Esta herramienta me permite generar una paleta de color monocromática con distintos tonos de verde, los cuales iba a necesitar para utilizar por toda la aplicación.



Aunque con esta paleta cubría prácticamente todas las necesidades de colores que tenía para la aplicación, los botones de alerta y algunas secciones necesitaba alguna diferenciación, de esta manera decidí utilizar [adobe colors](https://color.adobe.com/create/color-wheel) para extraer colores como azul, rojo o amarillo para ello. El azul y el rojo lo extraje de la tríada de color del color principal:



Sin embargo, seguía necesitando colores como el amarillo para poder mostrar mensajes de aviso, o simbolizar algo que supone un riesgo como es el boton de modificar.



Una vez obtenida la paleta de colores que cumplia con las necesidades de la aplicación, ya podía pasar a seleccionar las fuentes a utilizar en la aplicación.

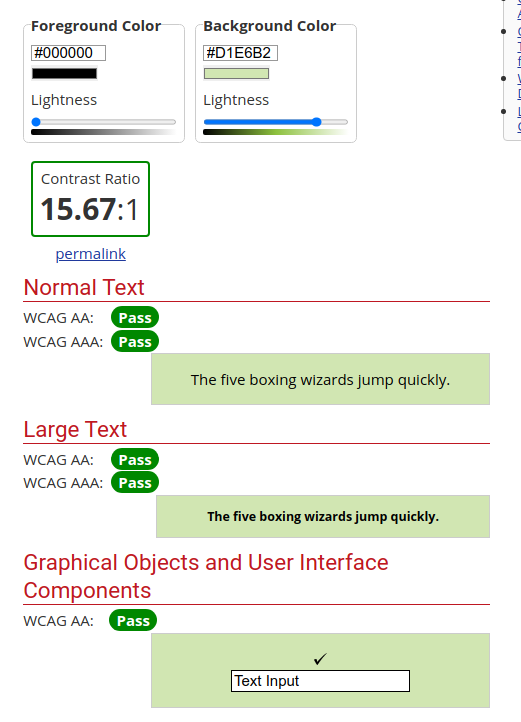
## Elección de la fuente

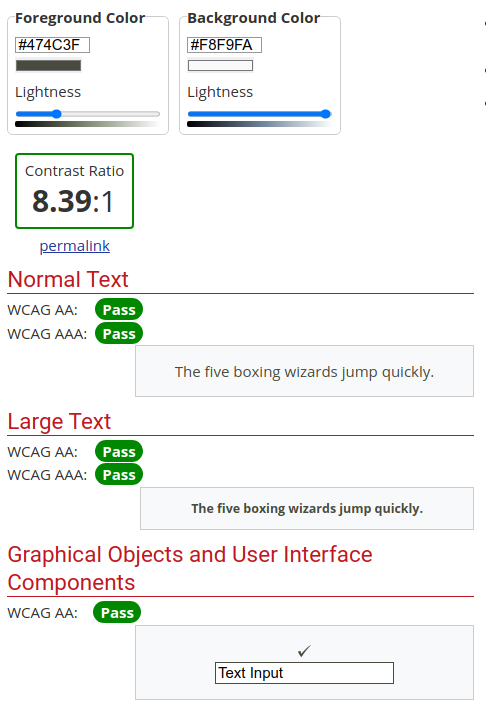
Para el cuerpo se ha seleccionado la fuente Roboto, se trata de una fuente de Google que se caracteriza por ser una letra sans serif humanista con un pequeño toque geométrico, la cual nos viene perfecta para nuestra página web ya que permite una buena lectura y se pretende hacer una labor educativa y divulgativa a través de las rutas que todos los usuarios que utilizan la aplicación.

Los títulos utilizan la fuente EB Garamond que se trata de una fuente serif humanística, la cual combina muy bien con Roboto que hemos utilizado para el cuerpo. Se ha elegido este tipo de fuente más clásica y tradicional también debido a la difusión de rutas que se pretende hacer con la web, es decir, una labor un tanto periodística e histórica al fin y al cabo.

## Contraste fondo con letra

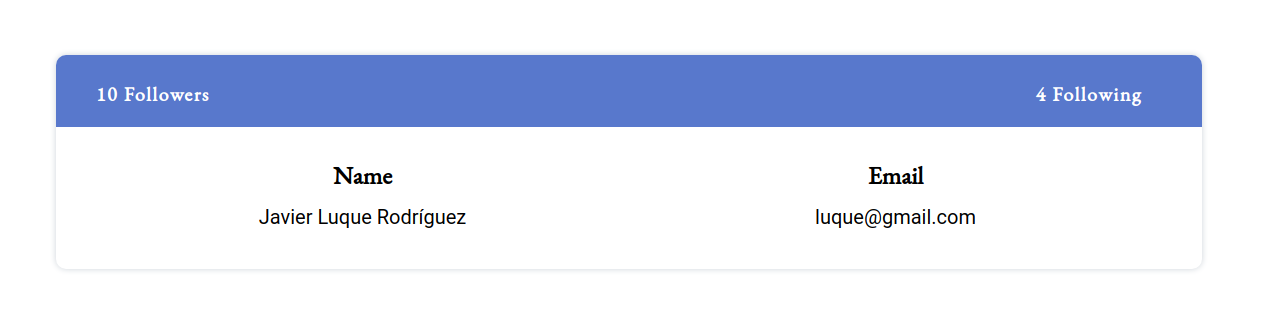
Aunque hayamos elegido los mejores y colores y fuentes para nuestra aplicación de nada nos sirve si el contraste del fondo con la letra es malo, ya que los usuarios no podrán leer bien la información. Para comprobar que el contraste fuera correcto utilice la siguiente aplicación web que analiza el contraste [webaim contrast checker](https://webaim.org/resources/contrastchecker/?fcolor=000000&bcolor=D1E6B2). Tal y como podemos ver en la captura de pantalla ha pasado sin problemas la prueba:





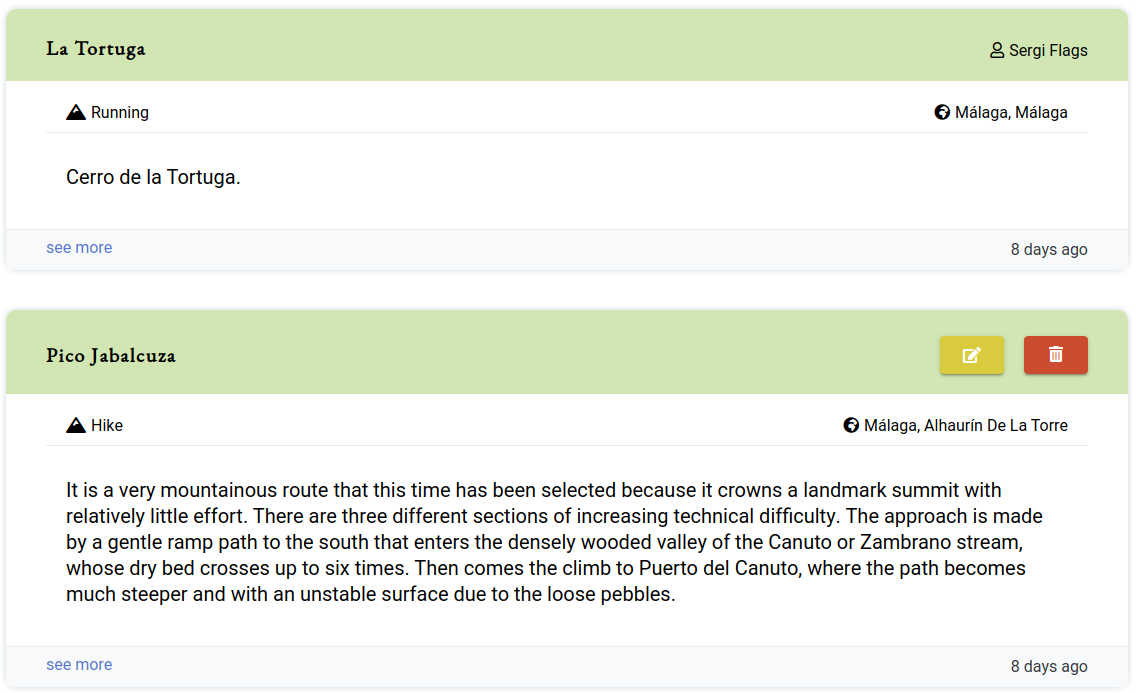
## Equilibrio visual

Uno de los ejemplos más claros de equilibrio visual que tenemos en la aplicación es en el componente del perfil de usuario, ya que no queremos distraer al usuario con una tensión compositiva de lo que es el contenido principal. En definitiva, se pretende que el usuario sepa cuáles son los datos del usuario a un golpe de vista sin que le distraigan otros factores.



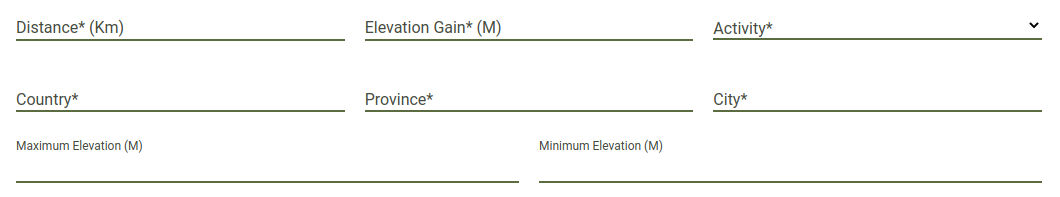
## Tensión compositiva

En el listado de rutas he creado una tensión compositiva con los botones para modificar y borrar poniéndolos en unos colores llamativos y mostrando un contenido distinto a las rutas subidas a la de otros usuarios. De esta manera los botones captan la atención del usuario y sabrá que puede utilizar los botones para editar o borrar la ruta.



## Accesibilidad y usabilidad

* **Percepción**: no tendría problema ya que no tiene anuncios ni tiene música ni sonidos de fondo. Sin embargo, en la sección de música determinados usuarios podrían tener problemas a la hora de escuchar la música si le dan al botón de reproducir debido a las limitaciones de accesibilidad al contenido multimedia HTML. Estas limitaciones requieren una mejora tal y como se remarca en la página [W3](https://www.w3.org/html/wg/wiki/MultimediaAccessibility), lo cual es un reto para mejorar la experiencia de usuario de usuarios con discapacidades.
* **Comprensión**: no utiliza términos modernos ni abreviaturas no explicadas. Sin embargo, las descripciones utilizadas por los usuarios no es algo que se pueda controlar y, por tanto, puede ocasionar algunos problemas a la hora de la comprensión. No obstante para ello los usuarios pueden comentar las rutas que ellos deseen y de esta manera pueden preguntar las dudas que ellos tengan y el autor u otros usuarios la pueden resolver.
* **Interacción y navegación**: tiene una interfaz sencilla y fácil de manejar con una jerarquía muy plana que no necesita muchos clics.
* **Aportación**: todas las rutas tienen una sección de comentarios donde los usuarios pueden dar feedback tal y como se ha comentado anteriormente. Además, para las personas con discapacidad visual se ha decidido marcar los campos obligatorios en el formulario para crear una nueva ruta con un asterísco tal y como podemos ver en la siguiente captura.



# Despliegue de la aplicaciones web

## Sistema de Control de Versiones (VCS)

### Introducción

Para el **control de versiones** he utilizado **GIT** y **GITHUB**, antes de explicar como lo he hecho vamos a ver que son. En resumen, GitHub es un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Esto permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso.

El control de versiones es un sistema que ayuda a rastrear y gestionar los cambios realizados en un archivo o conjunto de archivos. Utilizado principalmente por ingenieros de software para hacer un seguimiento de las modificaciones realizadas en el código fuente, el sistema de control de versiones les permite analizar todos los cambios y revertirlos sin repercusiones si se comete un error.

En otras palabras, el control de versiones permite a los desarrolladores trabajar en proyectos simultáneamente. Les permite hacer tantos cambios como necesite sin infringir o retrasar el trabajo de sus compañeros.

### Git

Git es un proyecto de código abierto que se inició en 2005 y creció hasta convertirse en uno de los VCS más populares del mercado. Se trata de un sistema de control de versiones distribuido. Esto significa que cualquier desarrollador del equipo que tenga acceso puede gestionar el código fuente y su historial de cambios utilizando las herramientas de línea de comandos de Git.

A diferencia de los sistemas de control de versiones centralizados, Git ofrece ramas de características. Esto significa que cada ingeniero de software en el equipo puede dividir una rama de características que proporcionará un repositorio local aislado para hacer cambios en el código.

Las ramas de características no afectan a la rama maestra, que es donde se encuentra el código original del proyecto. Una vez que se hayan realizado los cambios y el código actualizado esté listo, la rama de características puede fusionarse de nuevo con la rama maestra, que es la forma en que se harán efectivos los cambios en el proyecto.

### 

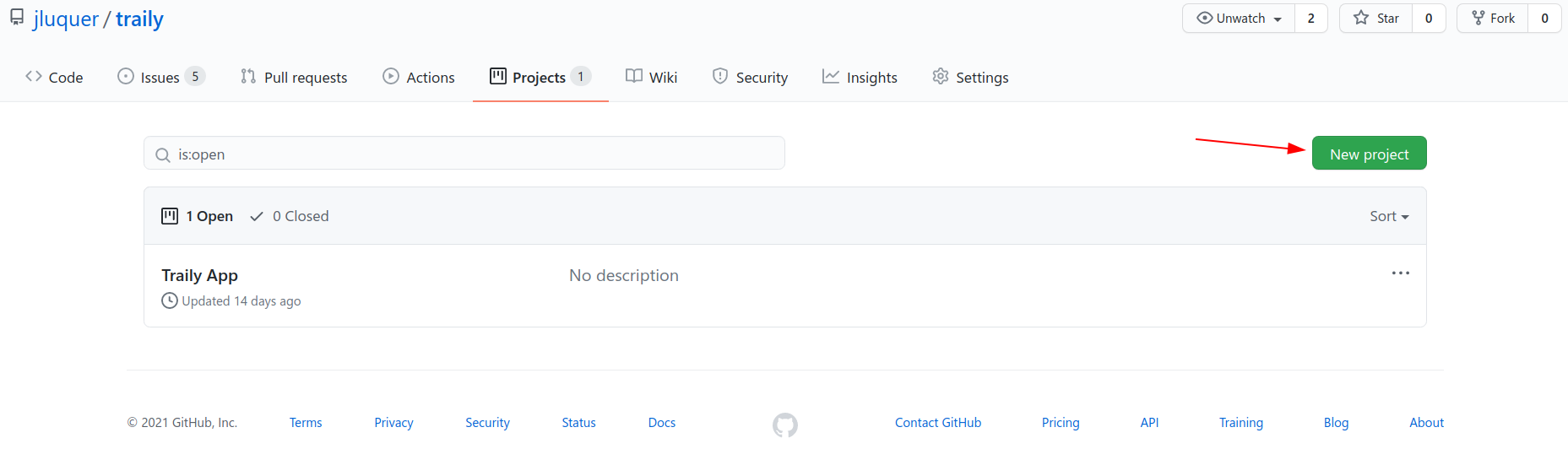
### Github

GitHub aloja más de 100 millones de repositorios, la mayoría de los cuales son proyectos de código abierto. Esta estadística revela que GitHub se encuentra entre los clientes Git GUI más populares. Esto se debe a que GitHub es una plataforma de gestión y organización de proyectos basada en la nube que incorpora las funciones de control de versiones de Git. Es decir que todos los usuarios de GitHub pueden rastrear y gestionar los cambios que se realizan en el código fuente en tiempo real, a la vez que tienen acceso a todas las demás funciones de Git disponibles en el mismo lugar.

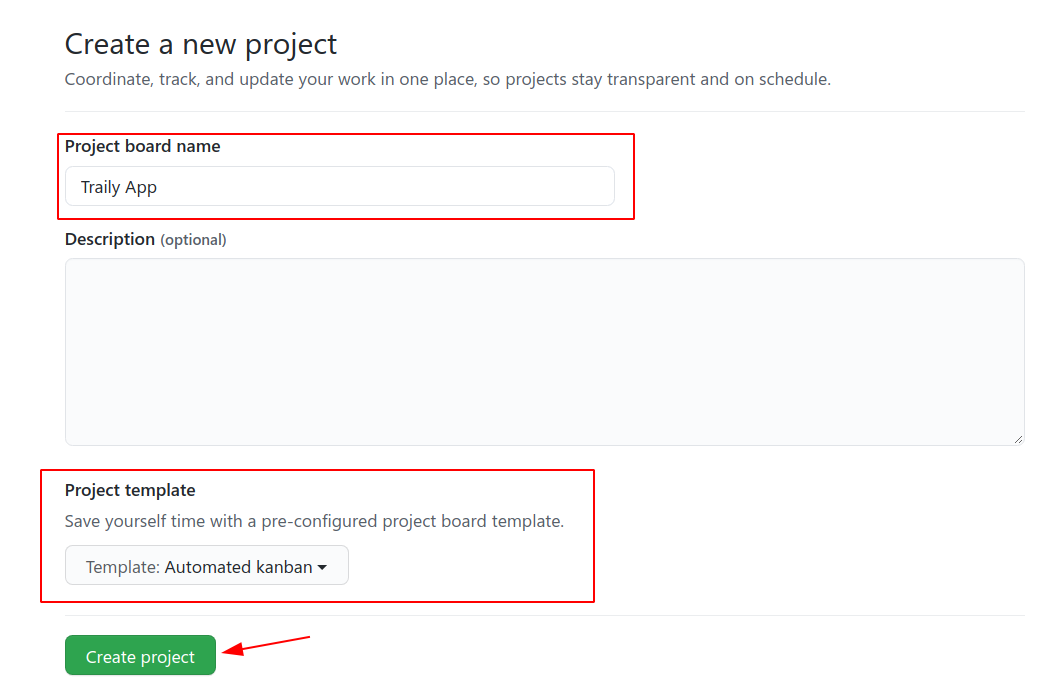
Además, la interfaz de usuario de GitHub es más fácil de usar que la de Git, lo que la hace accesible para personas con pocos o ningún conocimiento técnico. Esto significa que se puede incluir a más miembros del equipo en el progreso y la gestión de un proyecto, haciendo que el proceso de desarrollo sea más fluido.

### Organización del proyecto

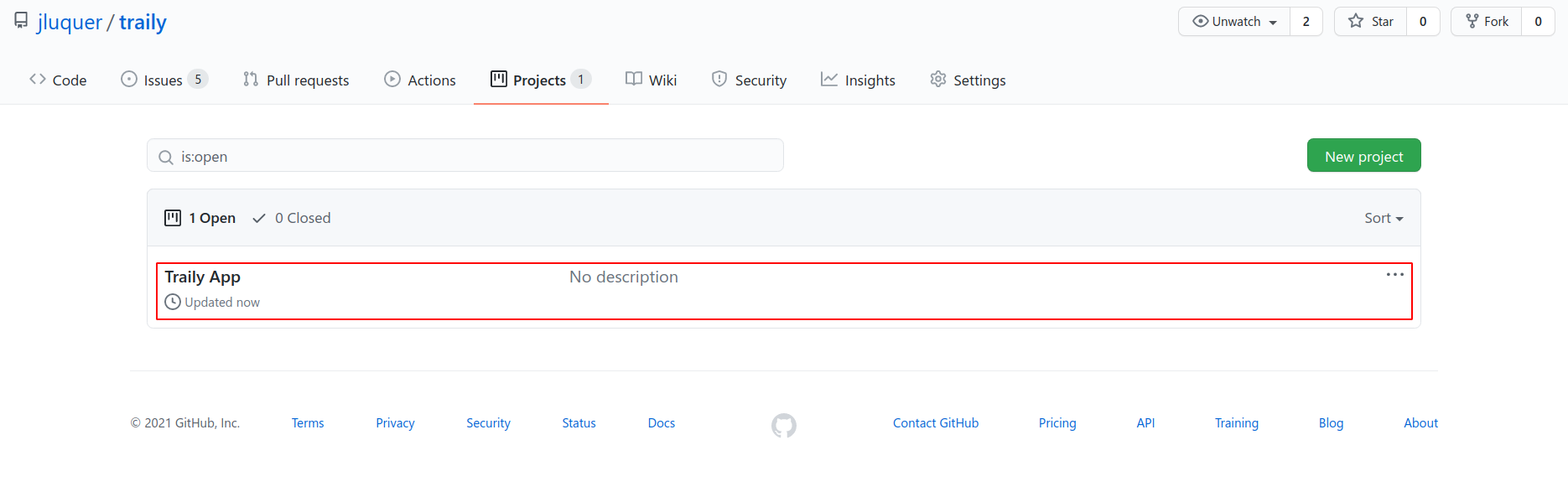
Para sacar el mayor rendimiento de las herramientas que estaba utilizando, decidí utilizar la herramienta que nos brinda Github en cuanto a organización de proyectos. Para ello cree un nuevo proyecto.



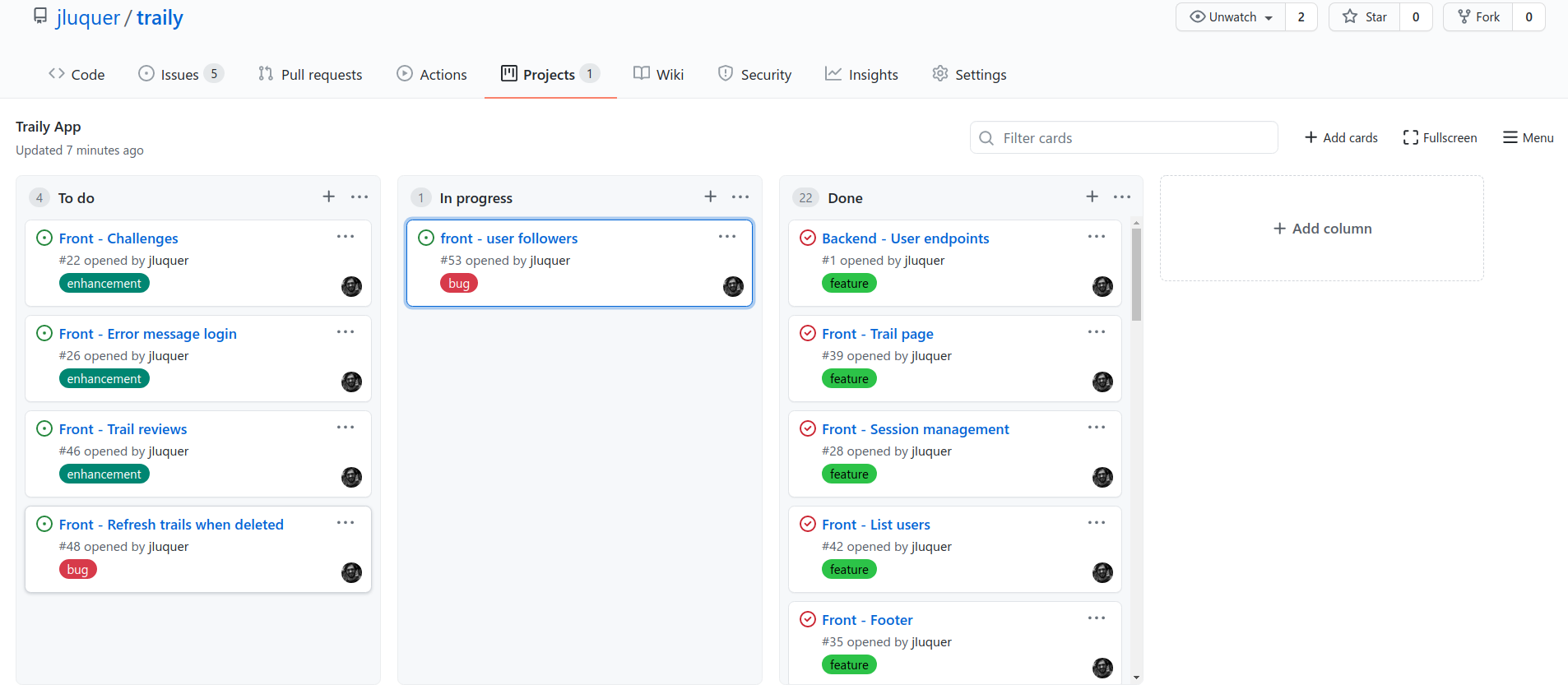
Una vez le hemos dado a crear un nuevo proyecto, nos redirige a un formulario para crear un nuevo proyecto igual que el que podemos ver en la siguiente captura. Una vez aquí introducimos el nombre del proyecto que queramos y seleccionamos una plantilla para el tablero de trabajo. Aunque hay varias plantillas decidí utilizar la plantilla de kanban automatizado porque automáticamente irá moviendo las tarjetas entre los distintos tableros cuando creamos o cerramos issues. Los issues nos permiten llevar un buen seguimiento de las mejoras, tareas y bugs que tenemos que hacer en nuestros proyectos.



Una vez he creado el proyecto aparecerá listado en los proyectos que tengo.

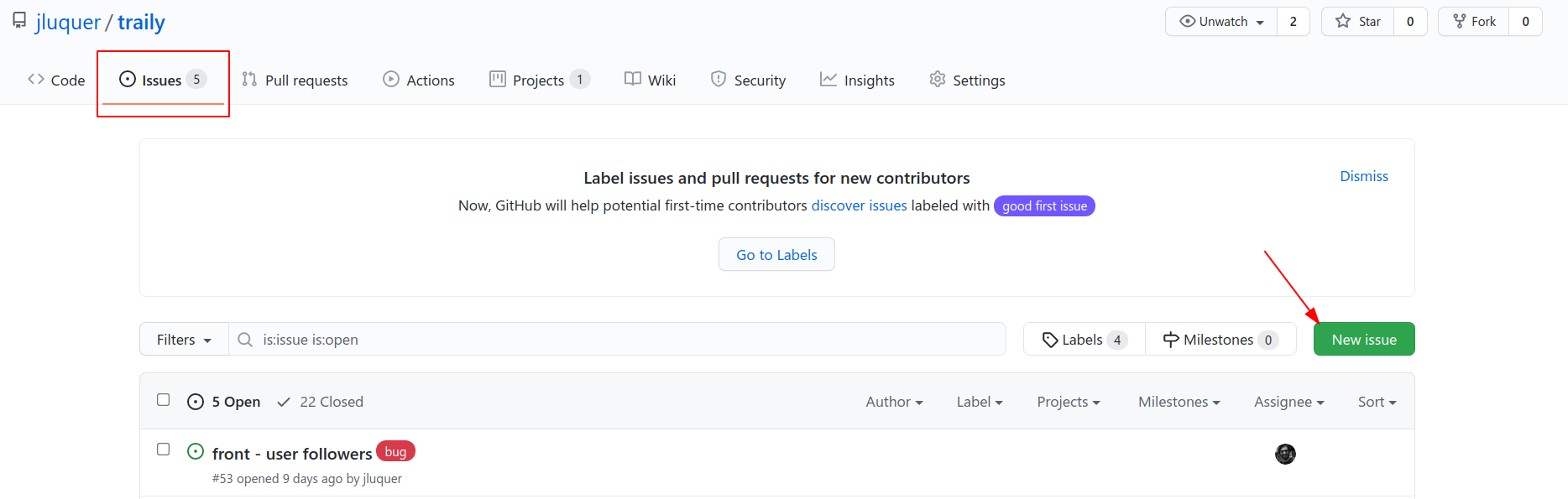


Una vez hecho esto empecé a crear issues para ir organizando el trabajo y asignando al proyecto los distintos issues que iba creando, los cuales explico más adelante cómo crearlos. Por tanto, cada vez que creaba un issue se añadía en “To do”, y yo los iba moviendo a “In progress” cuando comenzaba una tarea. Una vez había finalizado la tarea cerraba el issue y este se movía a “Done”.

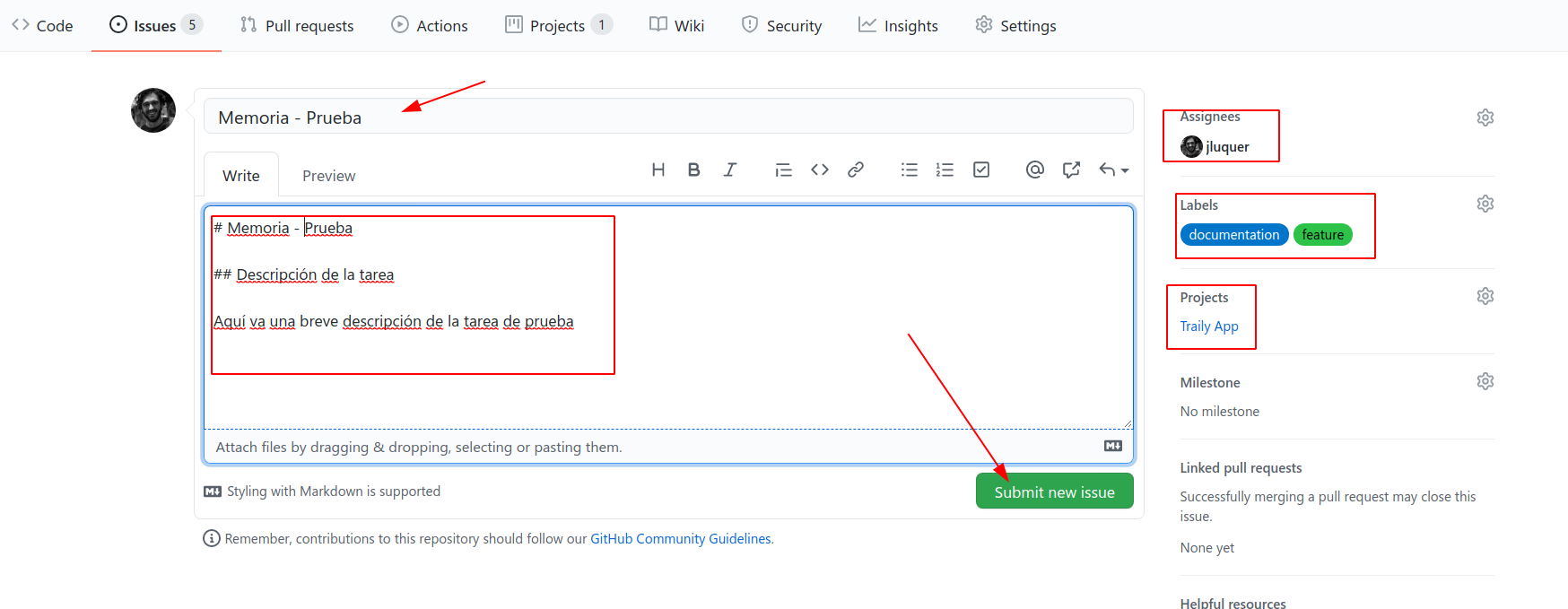


### Issues

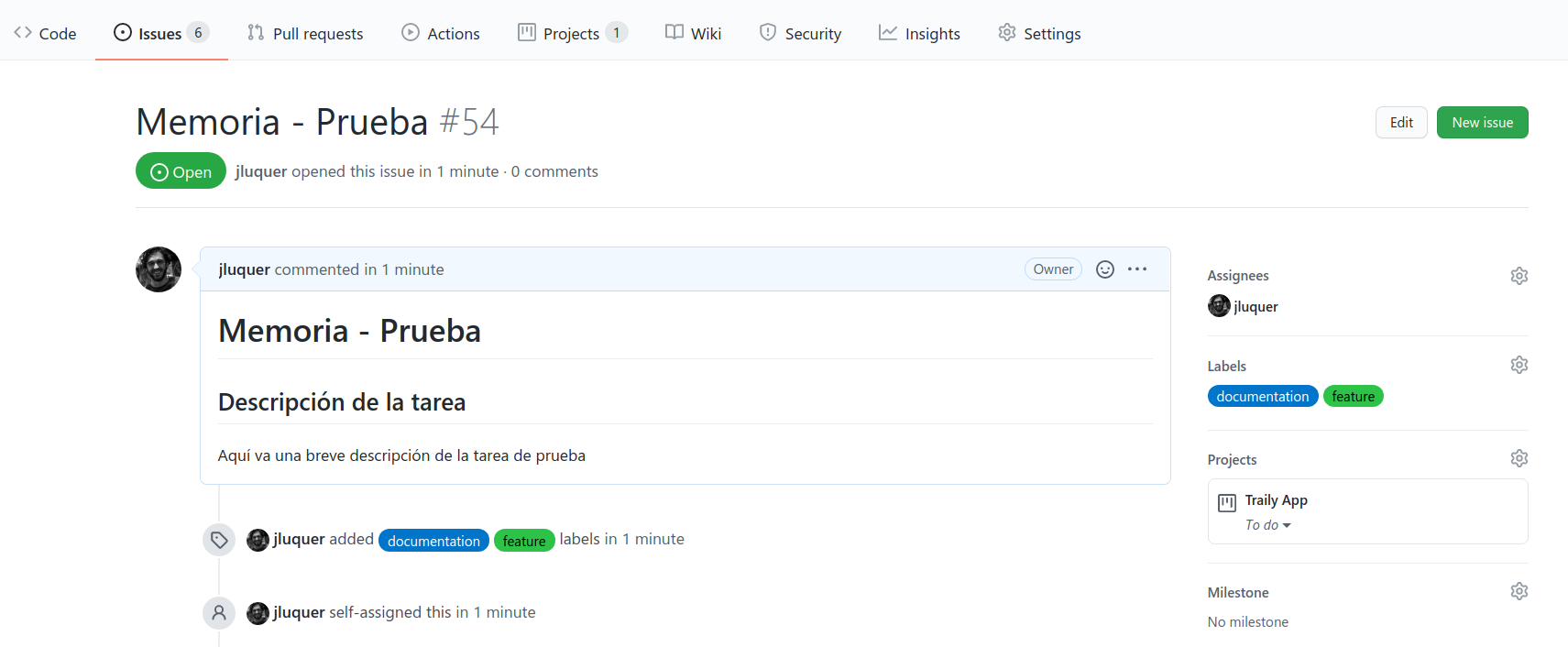
En este apartado se presente explicar cómo utilice los issues para crear las tareas, fallos o mejoras que veía que podía ir haciendo en el proyecto para organizar mejor mi trabajo. Para crear un nuevo issue sólo tengo que irme al apartado de issues y hacer click en crear uno nuevo.



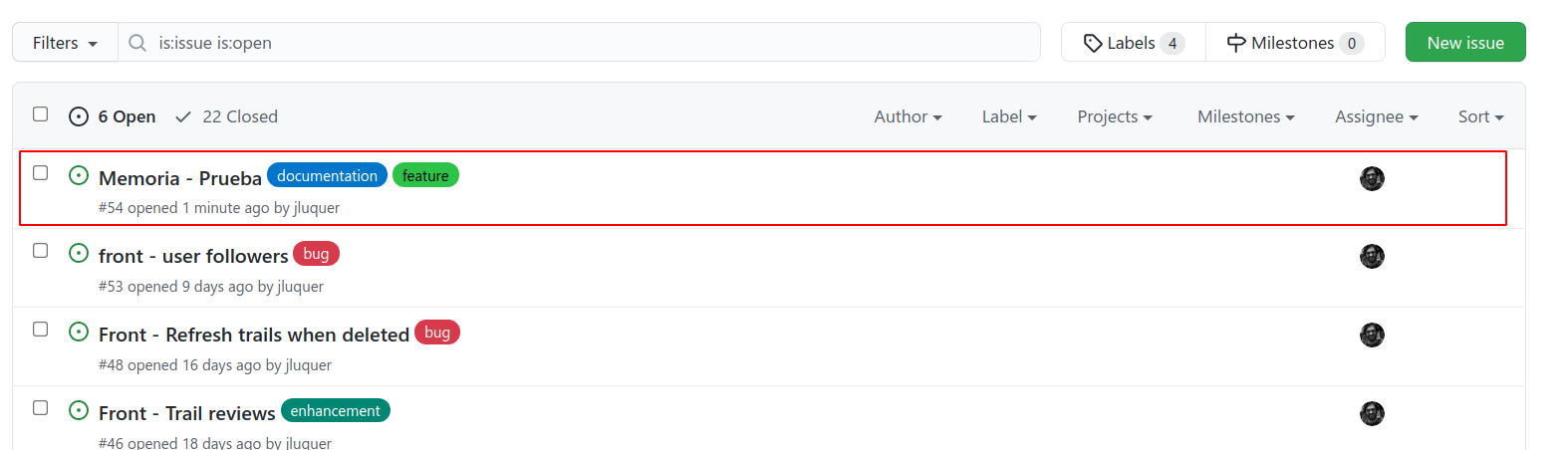
Esto nos redirige al formulario para crear un nuevo issue. En este formulario pongo el título, descripción, encargados de la tarea, etiquetas de la tarea que sean necesarias y el proyecto al que lo queremos asignar. Una cosa que es muy útil a la hora de redactar la descripción de la tarea es que nos permite utilizar el lenguaje de markdown, lo cual es muy útil ya que nos permite crear títulos, listas, listas de tareas, etc, dentro de la descripción permitiendo hacer una descripción mucho más detallada.



Una vez creado, lo podemos previsualizar:



Y además, podemos verlo en el listado de issues:

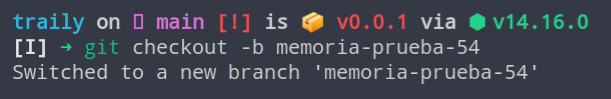


Por último, es importante señalar que dividí en dos categorías los issues, los que eran de front y los que eran de back siguiendo de esta manera la siguiente lógica para crear un nuevo issue:

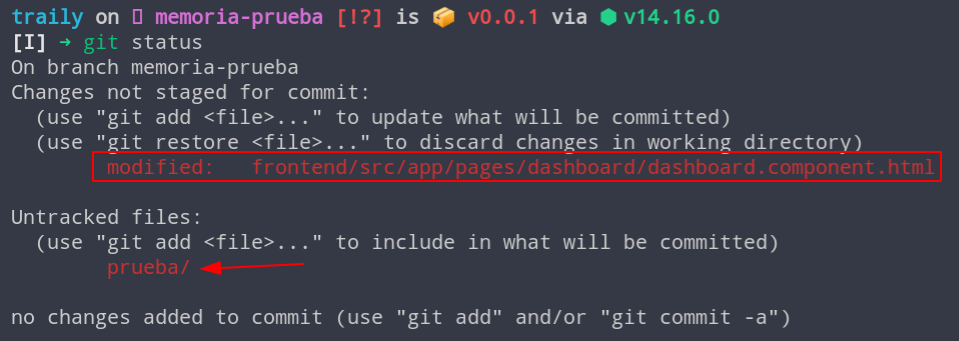
* front - tarea - número del issue: tareas correspondientes al frontend de la aplicación.
* back - tarea - número del issue: tareas correspondientes al backend de la aplicación.

### Ramas

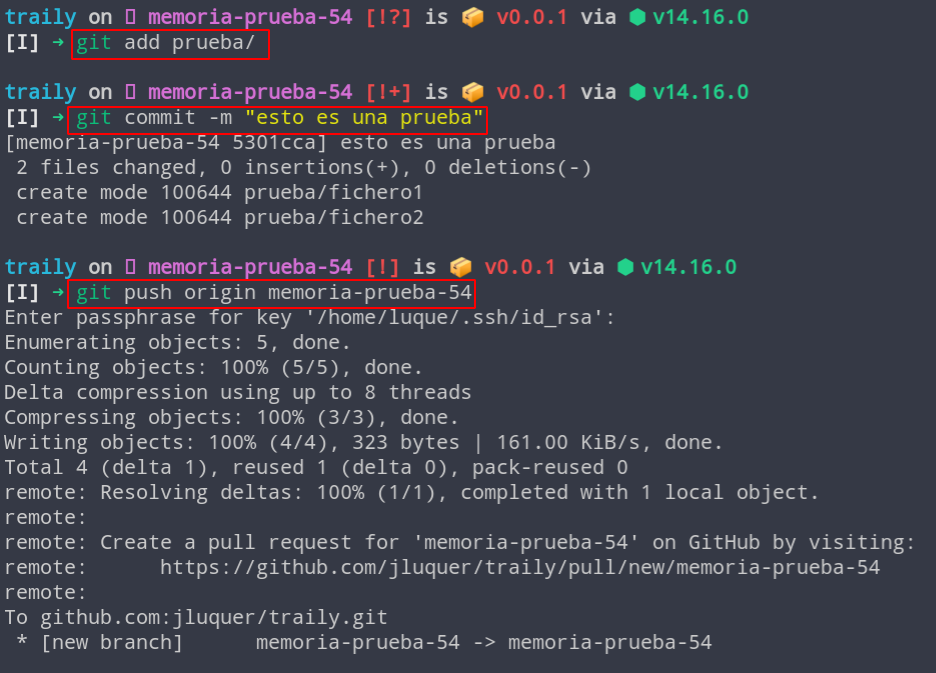
Una vez había creado un nuevo issue para la tarea que tenía que realizar ya podía crear una nueva rama para la tarea. Para ello en Visual Studio Code (VSCode) abrí una consola nueva y cree una rama nueva con el comando **git checkout -b**, con el parámetro -b estoy indicando que me cree una rama nueva y cambie a esa nueva rama.



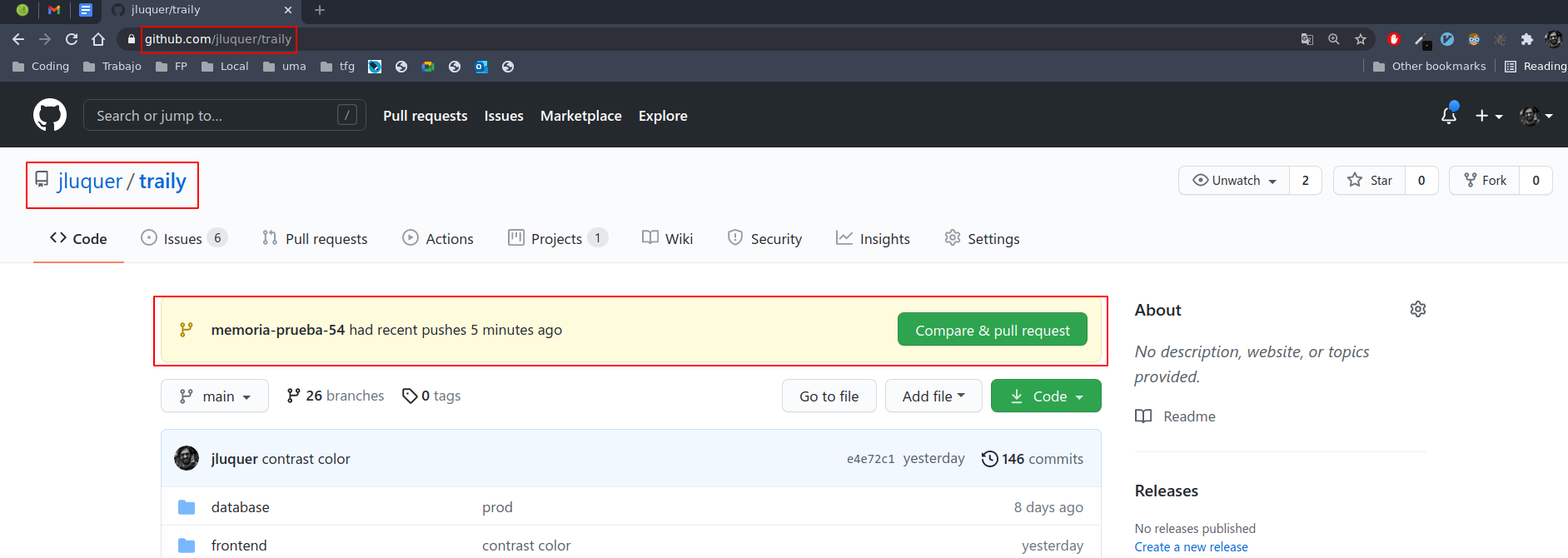
En esta nueva rama realizó los cambios que sean necesarios para la tarea que he creado. Una vez he terminado la tarea, compruebo que los ficheros que se han editado o creado son los que quería con el comando **git status**:



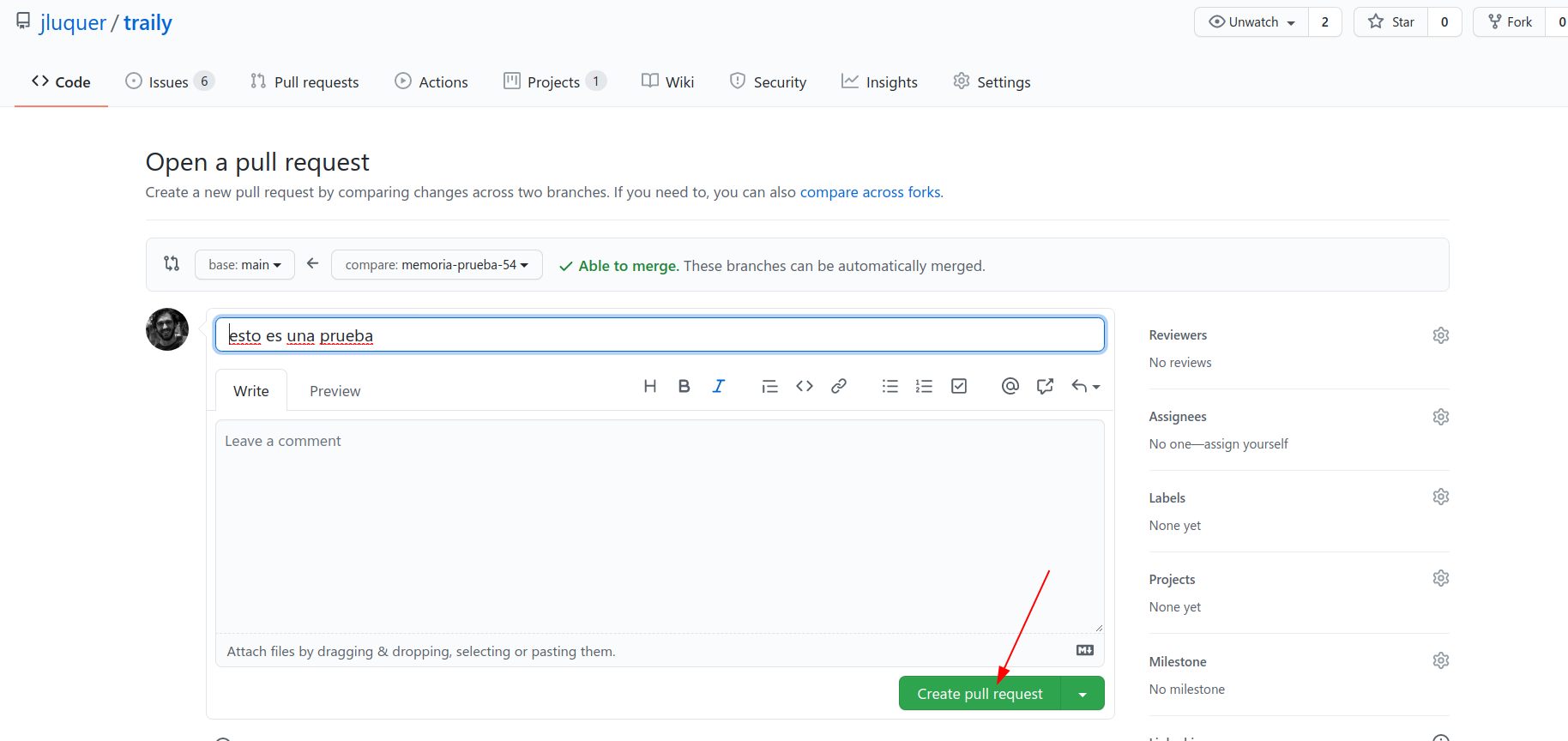
En la captura podemos ver que se han modificado más ficheros de los deseados, ya que nosotros solo queremos subir la carpeta de pruebas con los ficheros que hay dentro. Normalmente subo todos los ficheros con **git add .**, sin embargo, como no quiero que se suba el fichero de dashboard component añadiré sólo los ficheros que deseo. Luego, hago un **git commit -m “mensaje para el commit”** y **git push origin rama** para subir los cambios a remoto en github.



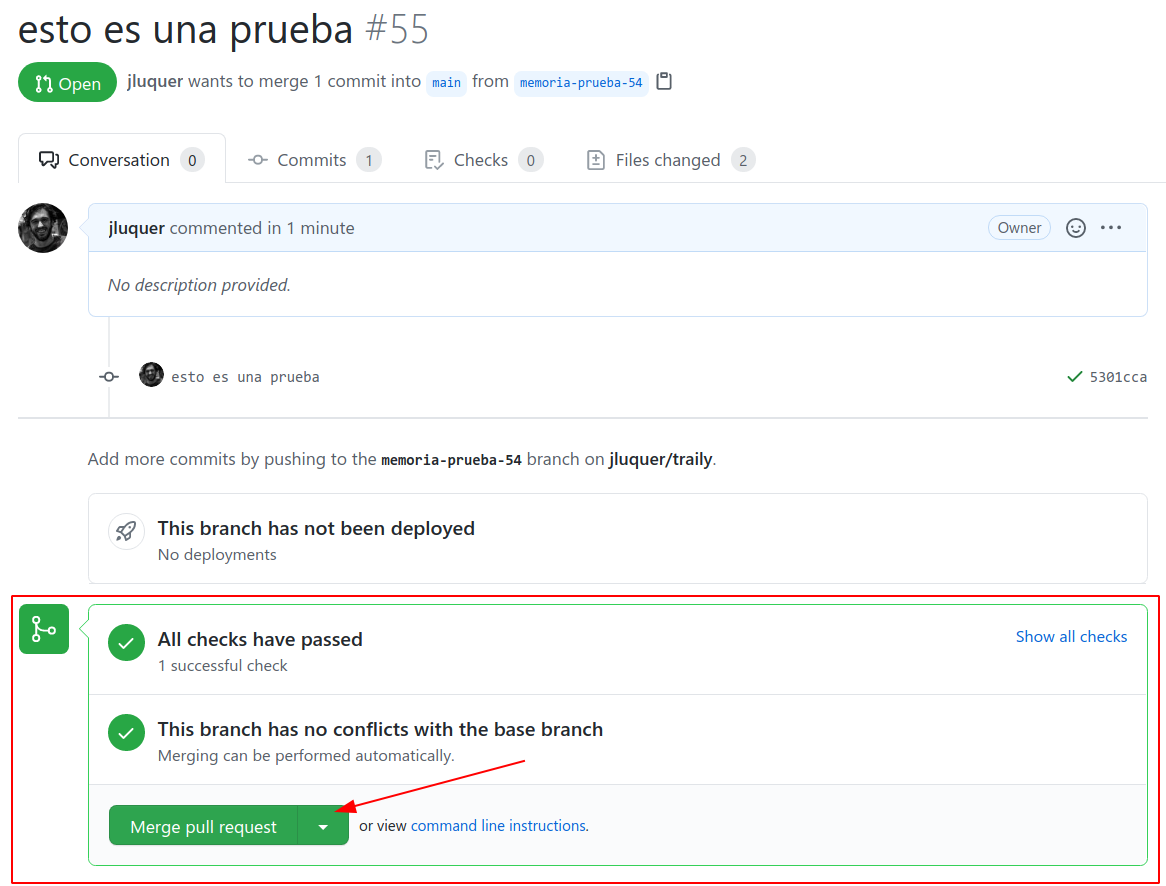
Ahora ya tengo subido los cambios en remoto, ahora faltaría añadir los cambios a la rama principal. De esta manera cuando cree una rama partiré de la rama principal que es donde se encuentran todos los cambios realizados. Por tanto, para poder hacer esta fusión con la rama principal tendré que mergear los cambios. Para ello me voy a github de nuevo. Al entrar en mi repositorio ya github ha detectado los cambios y solo tendré que crear la pull request para que se traiga los cambios de la otra rama. Hago click en “compare & pull request”.



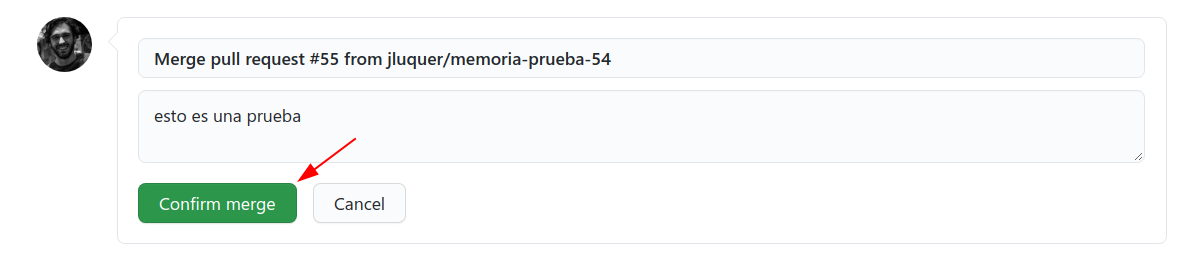
Le doy a crear la pull request.



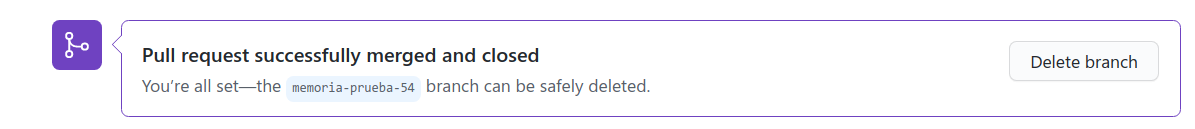
Si después de que github compare las ramas no hay ningún conflicto a la hora de mergear con la rama principal, entonces habrá pasado los checks y github nos permitirá mergear ambas ramas



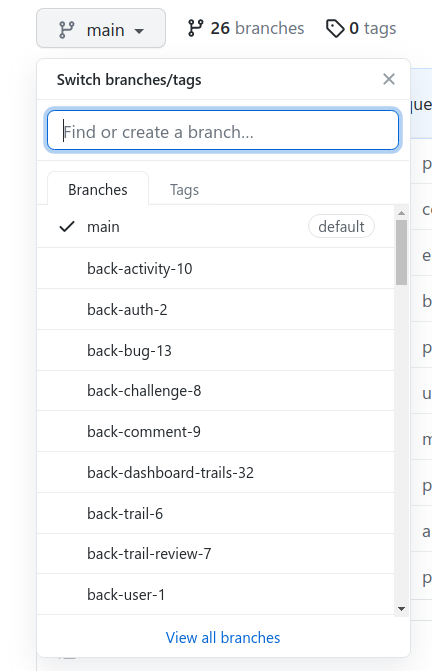
Confirmo el merge:



Y si se ha mergeado correctamente nos aparecerá lo siguiente:



Si queremos podemos borrar las ramas, pero yo las he dejado para que quede constancia de las ramas que he ido creando para las tareas del proyecto.



## 

## Despliegue en remoto

Aunque mi aplicación esté funcionando de manera local no sirve de nada si otros usuarios no la pueden utilizar como ocurre hasta ahora. Para que todos los usuarios puedan utilizar mi aplicación web hay que desplegarla de manera remota o en una intranet de manera que los usuarios de esa intranet puedan utilizarla. Yo he decidido desplegarla de manera remota. Para ello se ha utilizado github pages para desplegar el frontend y heroku para el backend.

### Heroku

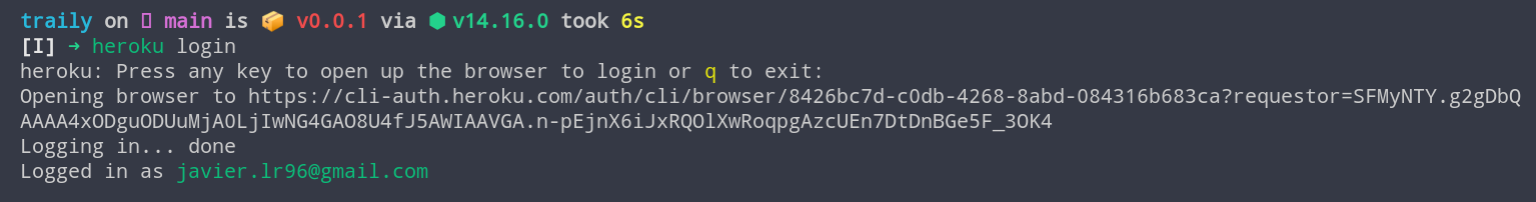
Heroku es una plataforma como servicio (PaaS) de computación en la Nube que soporta distintos lenguajes de programación.

Heroku es propiedad de [salesforce](https://www.salesforce.com/es/).​ Heroku, es una de las primeras plataformas de computación en la nube, que fue desarrollada desde junio de 2007, con el objetivo de soportar solamente el lenguaje de programación Ruby, pero posteriormente se ha extendido el soporte a Java, Node.js, Scala, Clojure, Python, PHP, Go y Clojure. La base del sistema operativo es **Debian** o, en la nueva plataforma, el sistema basado en Debian Ubuntu.

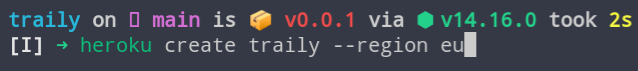
Dado que Heroku da soporte a aplicaciones hechas con Node.js y para bases de datos con mysql es perfecta para nuestra api.

Antes de empezar me cree una cuenta en Heroku, con esto ya me permitía desplegar la aplicación, sin embargo, para poder hacer uso del addon de la base de datos también hay que introducir una tarjeta de crédito como método de pago. Aunque introduzca la tarjeta de crédito no cobran nada ya que todo de lo que hago uso forma parte de la capa gratuita de heroku. Una vez realizado esto empecé a desplegar la aplicación.

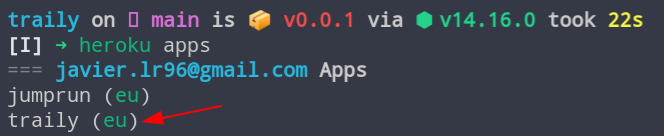
Primero instale heroku cli y me loguee desde la consola.



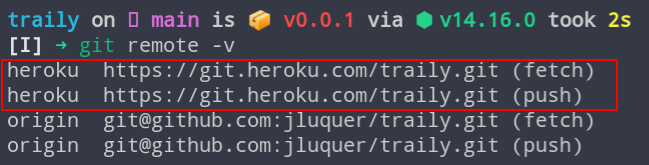
Después cree una aplicación con el comando **heroku create nombre de mi aplicación** y con el parámetro --region indique la región como europa.



Para comprobar que se haya creado correctamente utilizo el comando **heroku apps**:



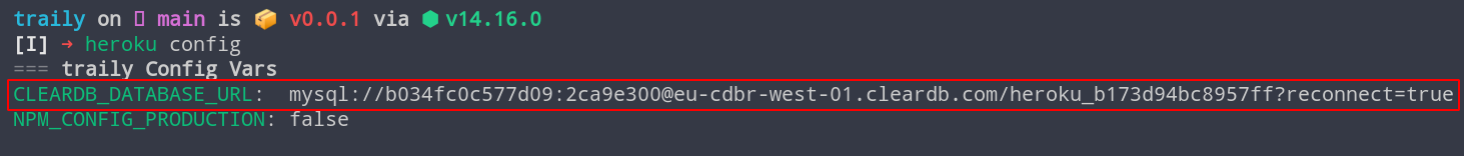
Al crear la aplicación heroku me generó un nuevo remote, que es el de heroku donde subiré los cambios de mi aplicación y se despliegue mi api.



Una vez creada la aplicación configuré el addon para la base de datos mysql en heroku. Para ello utilice el comando **heroku addons:create**.



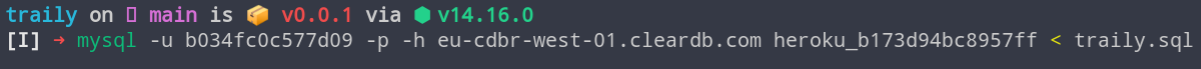
Con esto ya he creado la base de datos mysql en heroku. Para poder acceder necesito la url de la base de datos y esto lo obtuve con el comando **heroku config**. CLEAR\_DATABASE\_URL es la url de mi base de datos y aquí me vienen todos los datos para poder acceder a ella como el usuario, contraseña, host y nombre de la base de datos.



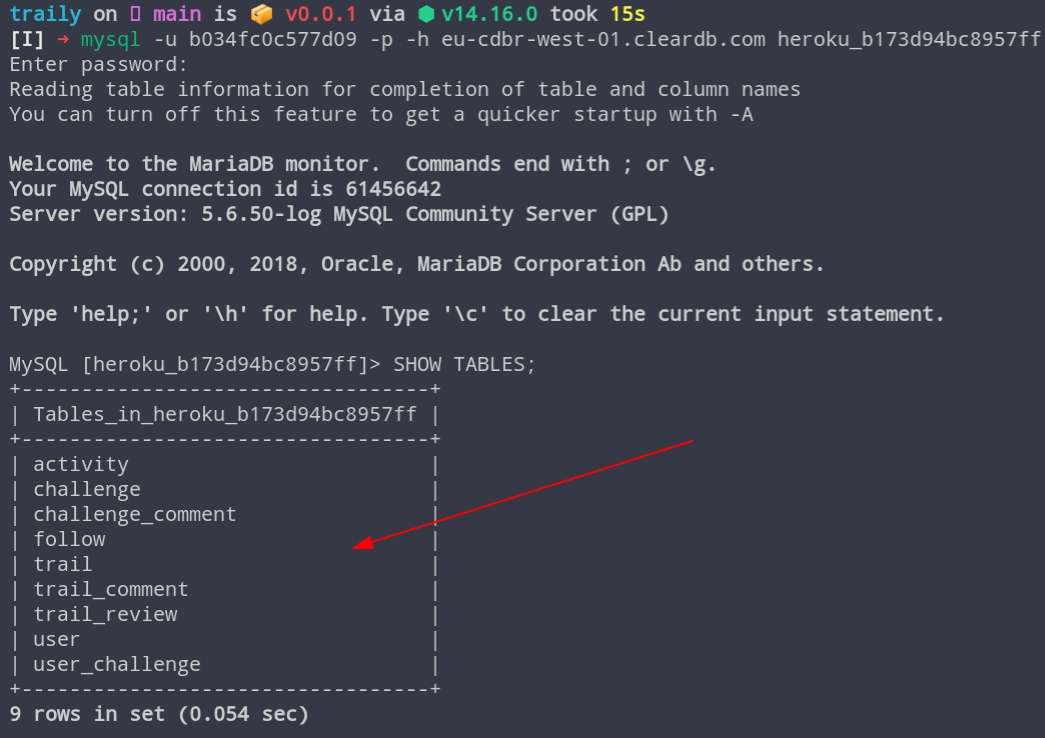
Veamos los datos de acceso de nuestra base de datos extraídos de la url:

* **usuario**: b034fc0c577d09
* **contraseña**: 2ca9e300
* **host**: eu-cdbr-west-01.cleardb.com
* **nombre de la base de datos**: heroku\_b173d94bc8957ff

Una vez conseguido los datos de acceso de la aplicación importe las tablas y datos de la Base de Datos. Para ello utilice el comando mysql con los siguientes parametros -u para el usuario, -p para que me pida contraseña, -h para el host, nombre de la base de datos y por último, “<” y el fichero que quería importar.

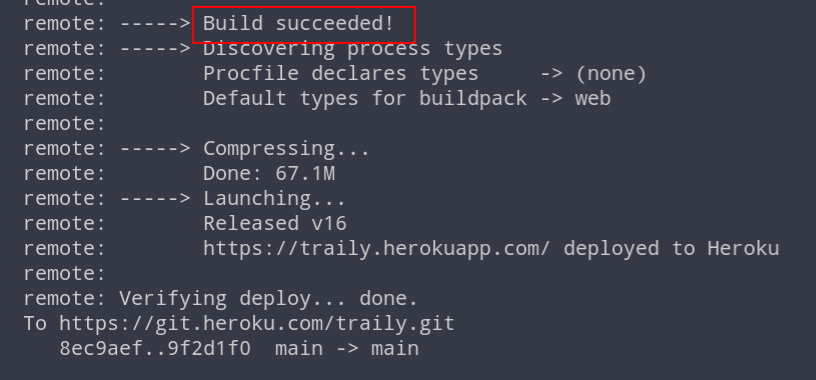


Al acceder a la base de datos ya podía ver que las tablas y datos se habían creado correctamente:



Después configure el fichero **ormconfig.ts** que es el fichero donde se encuentra la configuración del ORM y subí los cambios a github. Una vez subido los cambios a github solo había que hacer un push a heroku para que la aplicación se deplegará.

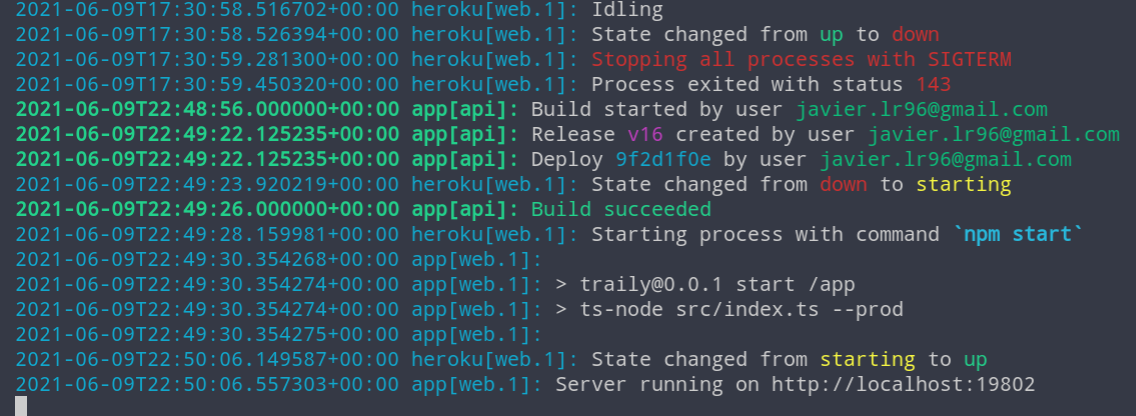




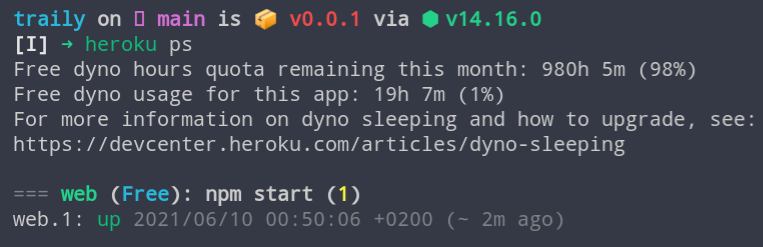
Una vez terminó de hacer el build con éxito comprobé que la api estuviera corriendo correctamente. Para ello utilice dos comandos:

* **heroku logs --tail**: muestra los logs más recientes.



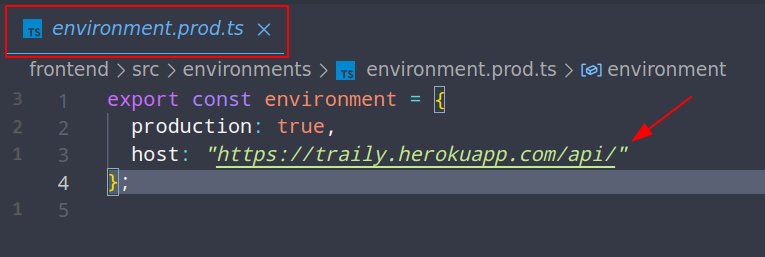


* **heroku ps**: me permite ver el estado de las aplicaciones que están corriendo. Yo utilice este comando con este propósito, sin embargo, son muchas más la utilidades que tiene como por ejemplo reiniciar la aplicación, entre otros tantos.

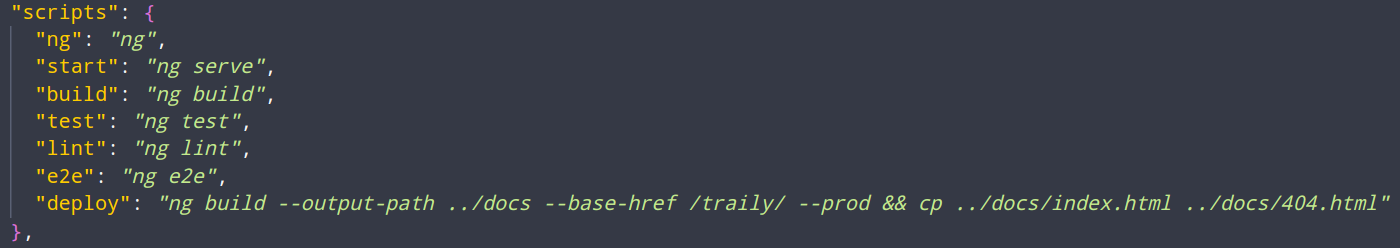


### Github Pages

Una vez desplegada mi api configure Angular para que utilizara la api que estaba desplegada en Heroku y no la que se ejecuta en local. Para ello configure el environment de producción para que cada vez que hiciera ng build o ng serve y añadiera el parametro --prod cogiera el fichero de configuración de producción y no el de desarrollo.



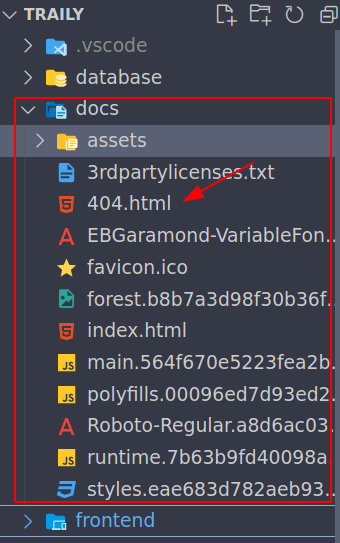
Ahora al ejecutar Angular en producción las peticiones se mandaran a la api que tenemos desplegada en Heroku. Luego configuré un script en el package.json para hacer el build de la aplicación y lo llamé deploy, que luego explicaré mejor su utilidad.



Una vez configurado el environment de producción, comencé a desplegar la aplicación a github pages. Primero cree una nueva rama llamada gh-pages.



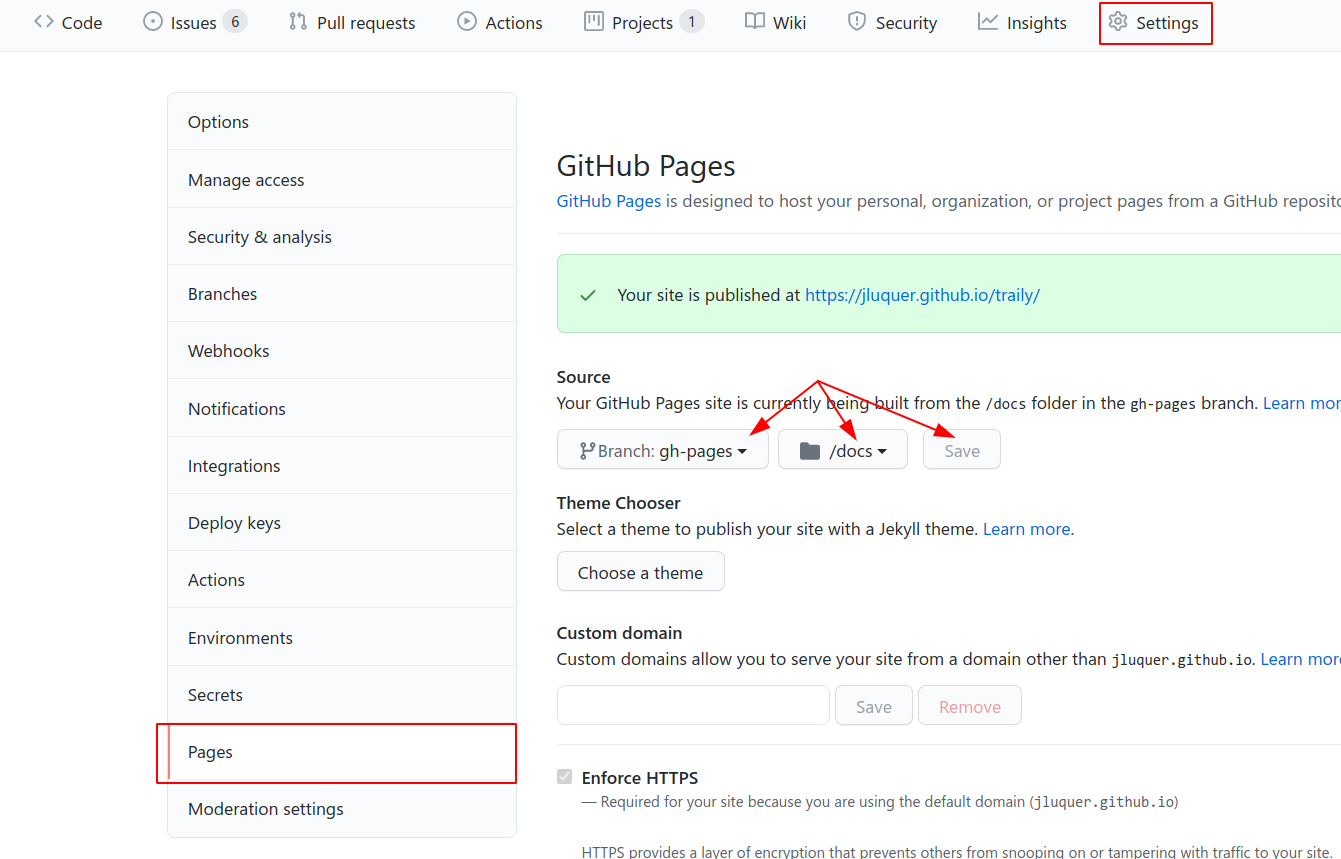
Luego utilice el comando **npm run deploy**, que es el comando que configuré en los scripts para que me construyera la aplicación utilizando el environment de producción (--prod). El build estaba configurado para que generará todos los archivos en la carpeta docs sobre la ruta base del proyecto. Además copia el fichero index.html a 404.html. Es muy importante este último paso, el de copiar el fichero ya que si no las rutas que hayamos generado en el proyecto no van a funcionar bien. Esto se debe a que si nosotros tratamos acceder a una ruta que no sea la principal donde se encuentra el index.html, github buscará la ruta que estamos buscando y en esa ruta no existe ningún fichero index.html que vaya a cargar. Por tanto, cuando trate de acceder a alguna ruta desconocida para github nos llevará al fichero 404.html que es una copia del index.html del build que ha hecho Angular y de esta manera tendremos todas las rutas que hay configuradas en Angular disponibles.



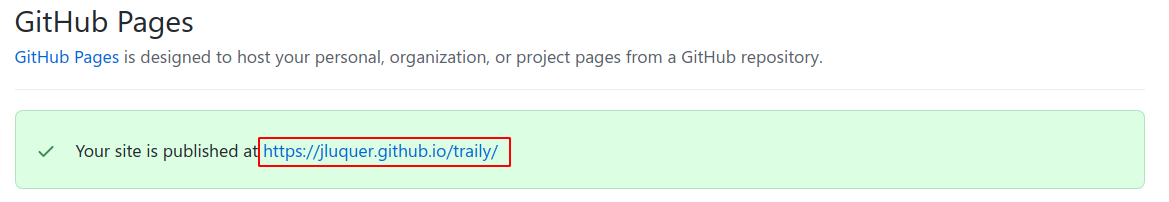
Luego, subí los cambios a git haciendo:

* git add .
* git commit -m “deploying frontend”
* git push origin gh-pages

Después de haber subido los cambios configure github pages. Para ello, me fui a settings>pages. En pages seleccione la rama y la carpeta donde se encuentra subido el codigo de haber hecho el build con angular. Y por último guarde los cambios.



Una vez he guardado los cambios, ya se ha desplegado a la página y por tanto puedo acceder a ella con el enlace que me genera.



Cada vez que hacía un cambio cambiaba a la rama gh-pages y mergeaba la rama principal con **git pull origin main**, volvía a hacer **npm run deploy** y subía los cambios. Es algo tedioso y lo ideal hubiera sido automatizar este proceso haciendo uso de GitHub Actions que nos permite automatizar esta clase de acciones.

## 

## Generación automática de la documentación

Para la generación automatica de la documentación he utilizado [TypeDoc](https://typedoc.org/). TypeDoc convierte los comentarios en el código TypeScript en documentación HTML o en formato JSON. Además, permite una gran variedad de configuraciones, admite plugins, temas y se puede utilizar como herramienta CLI.

Lo primero que hice fue configurar los ficheros json donde se incluye la configuración de TypeScript. Para la documentación del backend utilice la siguiente configuración: **typedoc --out docs-back --tsconfig tsconfig.json src**. Con el parámetro out indico el directorio donde quiero que se genere la documentación, con el parámetro tsconfig indico el fichero donde se encuentran las configuraciones del compilador de TypeScript y por último indique la carpeta de la que quiero que me genere la documentación, en este caso la correspondiente al backend.

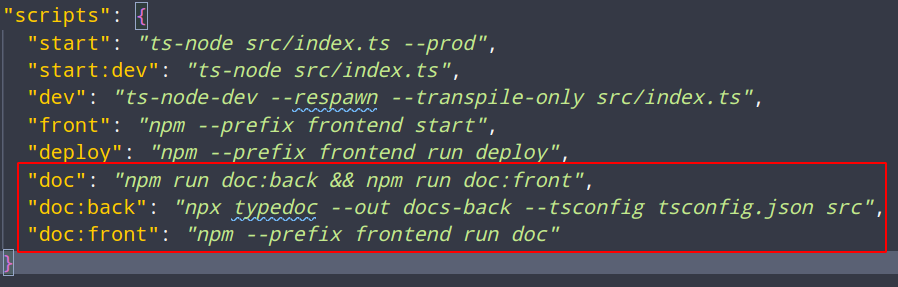
Para el front utilice el siguiente comando: **typedoc --options typedoc.json**. Dado que Angular requería de más configuraciones decidí crear un fichero de configuración donde incluir la configuración de TypeDoc, el cual lo indico con el parámetro options. Veamos la configuración que realice en este fichero:



Tanto out como tsconfig ya hemos visto para que sirven, veamos qué utilidad tienen los otras dos opciones:

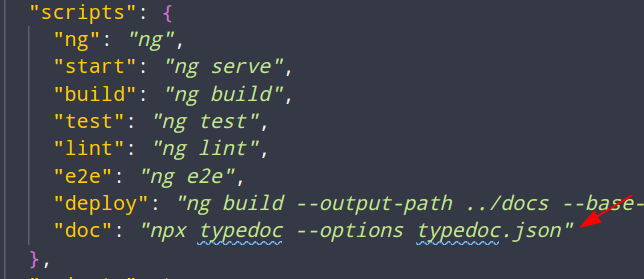
* **entryPoints**: aquí pongo la carpeta de la que quiero que me genere la documentación.
* **exclude**: ficheros que se excluyen de la documentación.

Aunque introduciendo estos comando ya podía generar la documentación, decidí automatizar el proceso creando algunos scripts en el package.json para que cada vez que quisiera generar la documentación solo tuviera que utilizar los comandos que configure en los scripts. En el fichero package.json del backend, que es el principal del proyecto, cree tres scrips:

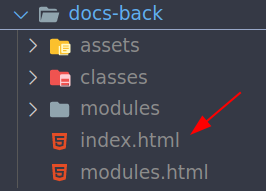
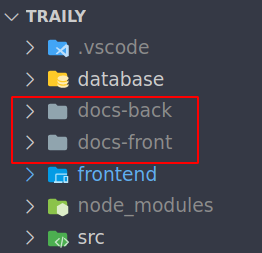


El script doc lo puedo utilizar **npm run doc**, con este comando estoy generando la documentación tanto del backend como del frontend. Veamos ambos comandos que se están ejecutando:

* **npm run dock:back**: es el comando que hemos visto, pero con la peculiaridad de que utiliza npx. Npx es un paquete de npm que permite ejecutar cualquier paquete de npm sin necesidad de instalarlo, de esta manera no es necesario añadirlo como dependencia del proyecto.
* **npm run dock:front**: aquí estoy utilizando el parámetro --prefix para indicar la carpeta donde se encuentra el ejecutable de node, que puede ser o un package.json o la carpeta de node\_modules. En mi caso, lo utilizo para poder utilizar los scripts que están configurados en el package.json en el frontend. Por tanto, en el front también configure el script **npm run doc** para generar la documentación del frontend, el cual utiliza el comando que hemos visto anteriormente.

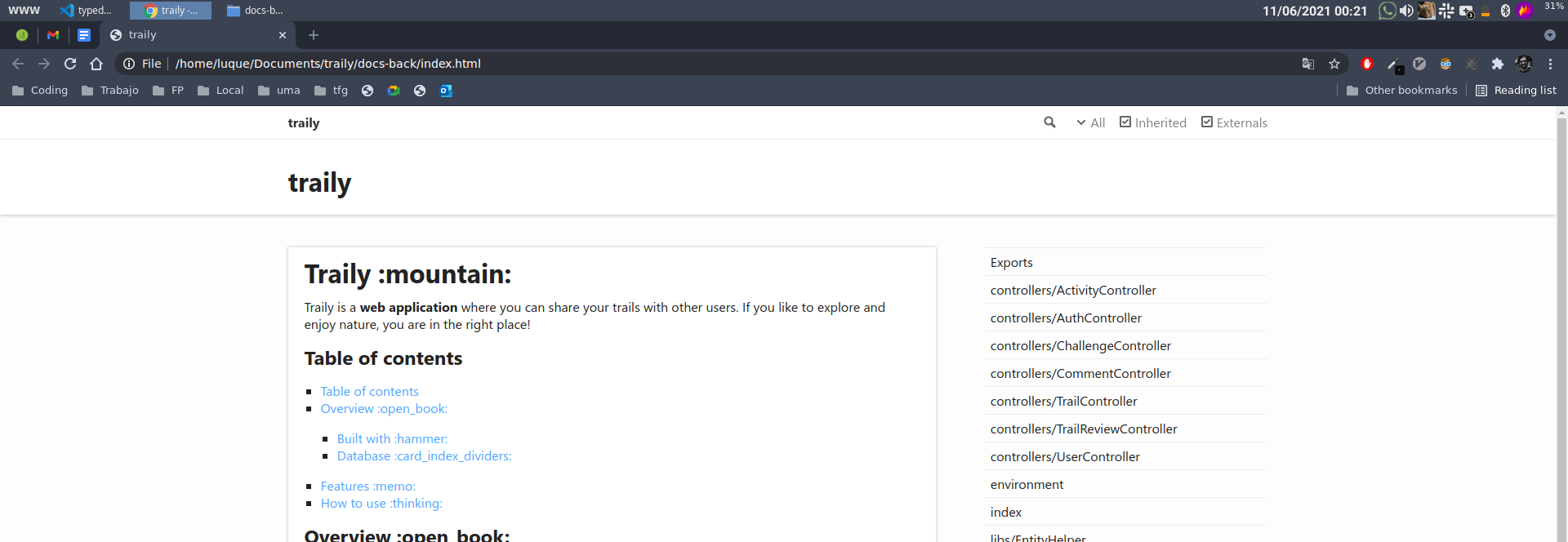


Si utilizo el comando npm run doc en el directorio principal genero estas dos carpetas:

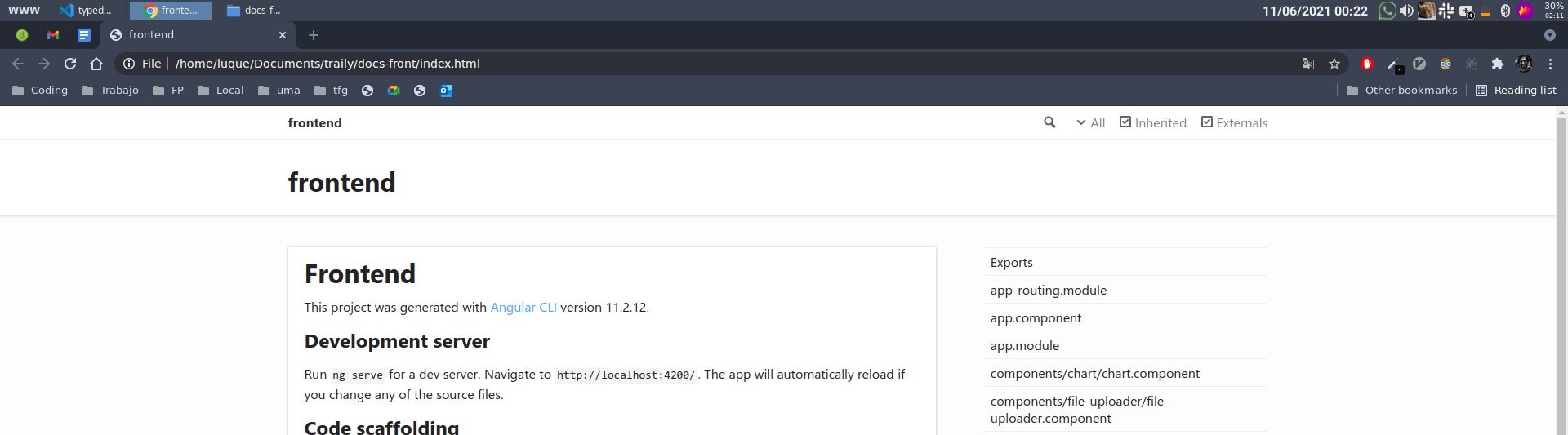


Si abro cualquiera de las dos carpetas dentro se encuentra un fichero index.html, si abro el fichero en el navegador puedo visualizar la documentación de cada uno.

* Documentación del backend



* Documentación del frontend



# Conclusión

Para ir concluyendo voy a hacer un breve resumen de todo el trabajo. Lo primero a realizar fue especificar los requisitos que quería que cumpliera la aplicación. En segundo lugar, realice los modelos de la base de datos, clase y UML necesarios. Tercero, elaboré el diseño de las distintas capas que componen la arquitectura de la aplicación, que es una arquitectura de tres capas. Una vez elaborado los diseños pasé a realizar la implementación de la aplicación que sin duda fue lo que más tiempo me llevó. Realice pruebas para comprobar que el funcionamiento de la aplicación fuera el correcto. Por último, una vez realizado el diseño de la web, su implementación y sus pruebas, realicé el despliegue en remoto de la aplicación utilizando GitHub Pages y Heroku para ello, y su correspondiente comprobación de que todo estuviera funcionando correctamente.

## Validación personal del trabajo realizado

La realización del proyecto me ha gustado mucho ya que considero que integra muy bien todos los conocimientos que hemos ido adquiriendo a lo largo del Grado Superior en Desarrollo de Aplicaciones Web. En este proyecto integramos nuestros conocimientos en Base de Datos, programación, diseño, despliegue de aplicaciones, etc.

Dado que la aplicación que se ha presentado es un prototipo, no tiene toda la funcionalidad que tendría la versión final. A pesar de ello, nos sirve para acercarnos a una primera experiencia de lo que sería trabajar como desarrollador web, ya que hemos contado con un cierta presión debido al limitado tiempo que hemos tenido para su realización. Aunque no se haya implementado toda la funcionalidad en el frontend que se pretendía, si que se ha integrado las funcionalidad más importante de la aplicación, lo que sería el núcleo de ella. Por tanto, estoy muy contento con el trabajo realizado.

## Posibles ampliaciones

A continuación se listan una serie de ampliaciones que pudieran implementarse sobre el prototipo:

- Gestión de la biblioteca.

- Tablón de anuncios y sección de noticias.

- Calendario de exámenes y excursiones.

- Hacer segura la transmisión de datos sensibles de la aplicación como pueden ser las contraseñas, mediante técnicas de encriptación de dato

# Bibliografía

* <https://rockcontent.com/es/blog/api-rest/>
* <https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/>
* [https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/#:~:text=En%20el%20campo%20de%20las,Transfer%2D%20Transferencia%20de%20Estado%20Representacional&text=Buscando%20una%20definici%C3%B3n%20sencilla%2C%20REST,posibles%2C%20como%20XML%20y%20JSªON.](https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/#:~:text=En%20el%20campo%20de%20las,Transfer%2D%20Transferencia%20de%20Estado%20Representacional&text=Buscando%20una%20definici%C3%B3n%20sencilla%2C%20REST,posibles%2C%20como%20XML%20y%20JSON.)
* <https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html#:~:text=Modelo%20Vista%20Controlador%20(MVC)%20es,control%20en%20tres%20componentes%20distintos.&text=La%20Vista%2C%20o%20interfaz%20de,los%20mecanismos%20interacci%C3%B3n%20con%20%C3%A9ste.>
* <https://www.redhat.com/es/topics/middleware/what-is-middleware#:~:text=El%20t%C3%A9rmino%20middleware%20se%20refiere,autenticaci%C3%B3n%20y%20gesti%C3%B3n%20de%20API.>
* <https://www.w3counter.com/globalstats.php>
* <https://developer.mozilla.org/es/docs/Learn/Accessibility/HTML>
* <https://www.w3.org/html/wg/wiki/MultimediaAccessibility>
* <https://www.hostinger.es/tutoriales/que-es-github#:~:text=En%20resumen%2C%20GitHub%20es%20un,seguimiento%20detallado%20de%20su%20progreso.>
* <https://guides.github.com/features/issues/#:~:text=Issues%20are%20a%20great%20way,own%20section%20in%20every%20repository.>
* <https://www.heroku.com/>
* <https://www.youtube.com/watch?v=dw1y7qwNb4E&ab_channel=DominiCode>
* <https://levelup.gitconnected.com/deploy-your-next-js-app-to-heroku-in-5-minutes-255e829a9966>
* <https://pages.github.com/>
* <https://angular.io/guide/deployment>
* <https://medium.com/dottech/build-deploy-angular-apps-en-github-pages-con-github-actions-8213466ef8dc>
* <https://docs.github.com/en/pages/getting-started-with-github-pages/configuring-a-publishing-source-for-your-github-pages-site#publishing-your-github-pages-site-from-a-docs-folder-on-your-master-branch>
* <https://www.vojtechruzicka.com/documenting-angular-apps-with-typedoc-compodoc-and-angulardoc/>
* <https://typedoc.org/guides/options/>
* <https://www.geeksforgeeks.org/what-are-the-differences-between-npm-and-npx/#:~:text=NPX%3A%20The%20npx%20stands%20for,without%20even%20installing%20that%20package.>