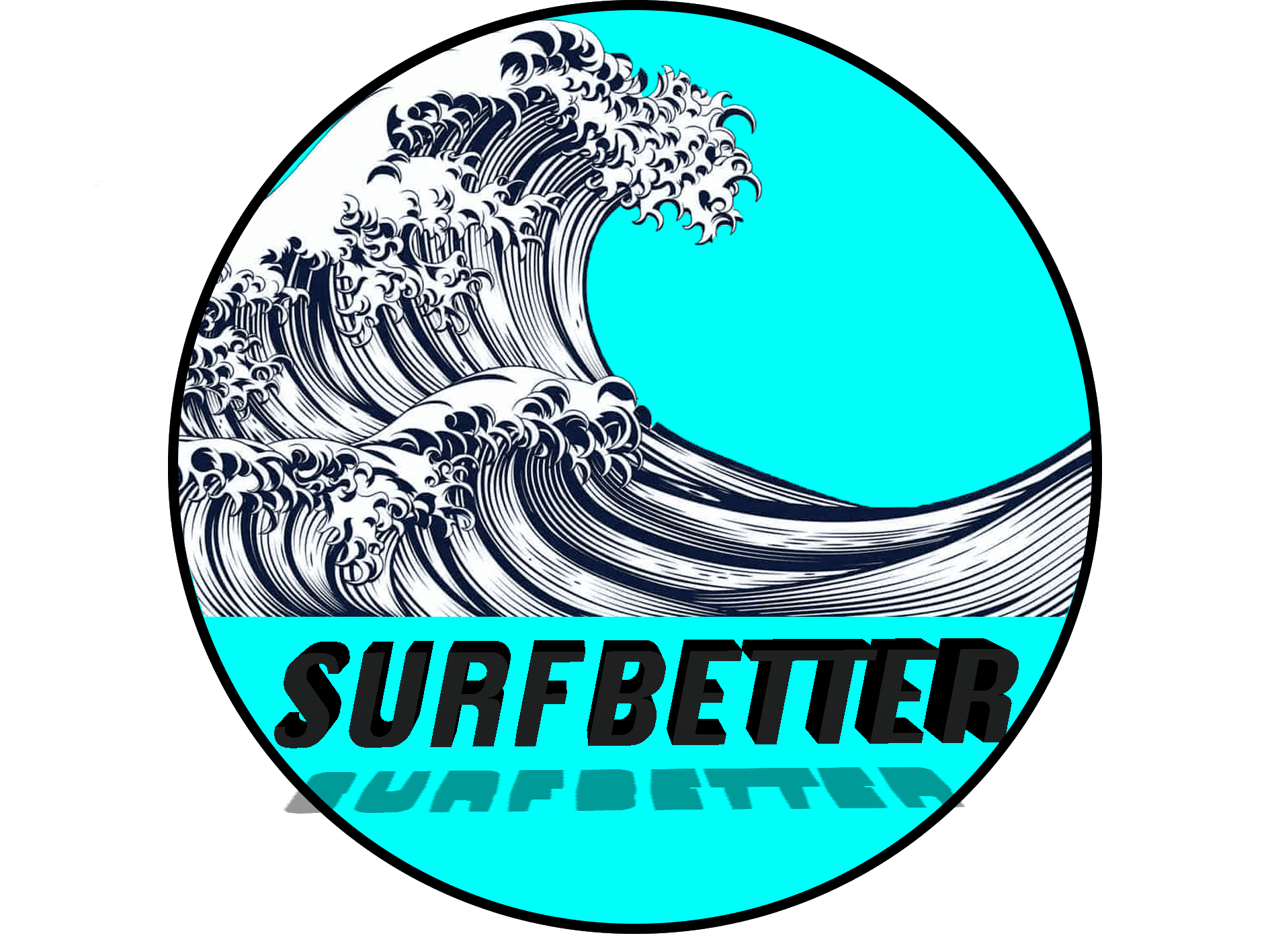
PROYECTO FINAL DE CICLO

# ANTONIO JOSÉ LOJO OJEDA



SURF BETTER

ÍNDICE

Introducción: Expectativas/objetivos y antecedentes: Desarrollo de la idea inicial, de las tecnologías a utilizar y de antecedentes (aplicaciones similares).

Descripción: Descripción del resultado obtenido: funcionalidades, organización, etc.

Instalación y preparación: Todo lo necesario para hacer funcionar el proyecto y cómo se instala. Edición de los archivos de configuración, creación y puesta en funcionamiento del control de versiones en github, etc.

Guía de estilos y prototipado:Se incluye tanto la guía de estilos como los wireframe o mockups de la aplicación.

Diseño: Descripción y diagramas de diseño: entidad relación, casos de uso, diagramas de flujo...etc. Es preferible usar el estándar UML.

Desarrollo: Secuencia de desarrollo, dificultades encontradas y decisiones afrontadas. Descripción de las herramientas de control de versiones y revisión del código utilizadas.

Pruebas: Descripción de las pruebas realizadas y automatización de las pruebas unitarias.

Despliegue: Tecnología de despliegue y descripción del proceso.

Manual: Manual de uso de la aplicación.

Información Legal

Seguridad

Conclusiones: Comparación del resultado con la idea inicial y mejoras futuras.

índice de tablas e imágenes.

Bibliografía y referencias.

1. Introducción

# Antecedentes

La aplicación viene dada por una problemática personal. Me gusta bastante el surf y recientemente descubrí una web que me informaba de la situación de las playas en tiempo real <https://es.surf-forecast.com/>. Esta aplicación permite ver la situación de una playa antes de ir a hacer Surf. Antes de descubrirla sufría a menudo el problema con el que muchos compañeros de hobby se sentirán identificados **¡LEVANTARTE A LAS 5 PARA NADA!** No será la primera vez que uno de mostros madruga para coger olas y se encuentra con nada. Por ello mejoraré modernizaré y crearé una aplicación mejor que esta.

**Problema:** **SURF-forecast** **no** tiene una **interfaz de usuario amigable** y realmente la **información** que se muestra es demasiado **compleja** y **poco intuitiv**a, si a este punto se le agrega que poca gente que conozco es consciente de su existencia. Mi intención **hacer como idea base algo similar**, pero **ampliando la idea**.

# Objetivo

Un punto de encuentro para todos los surfistas que aman este deporte, un sitio donde conectar, donde poder tener su propio perfil, debatir sobre las playas comentarlas y ver la información de las mismas antes de ir. Todo ello con una interfaz bonita un diseño UX/UI moderno e intuitivo, una aplicación que de gusto usarla y verla.

# Tecnologías

Como idea inicial **las tecnologías a utilizar** son las siguientes. En este punto divide el contenido en dos 5 bloques. Cada uno de ellos corresponde a un bloque de tecnologías utilizadas, Estos son **Back-end, Front-end, Diseño, Despliegue, Seguridad**. Partiendo de la base de que el modelo de arquitectura cliente servidor es **REST-API SERVER-LESS:**

## Back-end

Para el Servidor utilizaré el framework de python **Flask**. Este almacenará en SQLITE únicamente los datos pertenecientes a las entidades. Dejando los datos estáticos de la aplicación a el cliente. Además, será meramente testimonial. Se encargará de la autenticación (JWT), y de conectar el cliente con las tablas a través de sus rutas. Prácticamente no lo hemos visto en clase, pero tenía muchas ganas de aprender esta tecnología.

## Front-end

Para el front utilizaré **React**. Es una librería de JavaScript, Con esta librería podré dividir mi aplicación en módulos o componentes, cada uno de ellos dedicado a una función específica, las posibilidades de JSX y el componente

reactivo con UseEffect() y useState() creo que me brindará las herramientas necesarias para llevar a cabo el proyecto. Al igual que en el caso anterior. No hemos visto demasiado a fondo esta tecnología en clase. Pero quiero aprender a trabajar con ella mas a fondo.

## Diseño

Partiendo de la base de que utilizaré React y divide la aplicación en **componentes JSX** , trataré de que el código sea lo más legible posible, **Intentaré usar HTML-5**. Con esto quiero decir todas las etiquetas descriptivas para hacer una aplicación limpia y accesible. **El esquema de colores y el diseño** de la aplicación será **el mismo que he estado trabajando a lo alargo de todo el curso**. Modificaré un poco el mismo durante “*el cambio de lenguaje de estilos”*, pero en esencia será el mismo. De esta manera usaré CSS, pero a través de SASS. Me gusto cuando lo vimos y creo que me puede dar la potencia para la ejecución de los estilos, lo usará para hacer vía código la lógica de apertura y cierre de los modales o ventanas flotantes. La aplicación contará también con un tema oscuro. El cual se cambiará según la preferencia del usuario. Todas las imágenes estarán optimizadas y re escaladas.

## Despliegue

comandos para instalar y arrancar el front y el back. O bien mediante **docker-compose.**

Para el despliegue trataré de usar Docker. La idea es un contenedor con **Nginx-Node**. En el contexto de que **Nginx** haga de servidor web publicando el código compilado de **React** a través **de Node**. Y un segundo contenedor con **Python-Gunicorn**, El primero lógicamente consumirá los recursos servidos por **Gunicorn** y **Flask.** Para el despliegue habrán dos opciones o bien descargar el proyecto del repositorio y ejecutar los

## Seguridad

Me gustaría destacar este punto porque me parece bastante importante. Utilizaré **JWT (Json Web Token)**. Este se genera y asocia de manera aleatorio cada vez que el usuario inicia sesión. Solo podrá acceder a las rutas en el caso de que este se autentique enviando dicho **Token** al servidor.

Ejemplo de uno de los Token generados por mi aplicación:

"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2MjMwMDgwNTksImV4cCI6MTYyMzA5NDQ1OSwianRpIjoiMjRlY2U0M2YtMTViZC00NjllLWI4ZGItMWViZTkwZDZjNGE0IiwiaWQiOjEsInJscyI6ImFkbWluIiwicmZfZXhwIjoxNjI1NjAwMDU5fQ.G\_YzZp4CNdGEguvNwT\_p3eqzvcuAsY0G04P52L9NyjU"

# Aplicaciones similares.

Realmente no existe ninguna aplicación similar. Una app dedicada a la idea base sí. **SURF-forecast.** Pero incluir una pequeña red social junto con la información que puede brindar de las playas SURF-forecast es algo nuevo. Destacar que el proceso de desarrollo no ha sido únicamente de 3 meses. El diseño de la aplicación se ha ido trabajando y perfeccionando a lo largo de todo el curso. Estando bajo mi punto de vista bastante logrado, tematizado y moderno.

# Expectativas

Las expectativas se centran en finalizar el desarrollo con una aplicación funcional y correctamente ejecutada que permita a los usuarios administrar su información personal, un panel donde poder buscar todas las playas de la bahía de Cádiz, poder hacer uso de un mapa donde localizar y ver los spots como si estuviesen allí, poder comentar acerca del estado y de la experiencia en cada una de las playas, ver en tiempo real la información meteorológica de las playas (viento, olas …). En definitiva, tener un lugar donde poder ver el estado de un spot para ir a surfear y comentar si este les ha gustado. Por supuesto a través de una interfaz moderna, responsiva y agradable. La idea es que el código siga una serie de reglas estrictas que se explicarán en el punto **CodeStyle**. Que tanto el servidor como el front estén lo más modula rizados posibles (teniendo en cuenta el tiempo con el que contamos para desarrollar la aplicación y que son tecnologías que apenas se han trabajado en clase pero que evidentemente son el futuro y el presente del mercado). Una aplicación testeada y funcional y **SEGURA**. Ya no solo a nivel de ataques si no que cuente con una información legal y un tratamiento de los datos del usuario de acorde con la categoría del resultado final.

1. Descripción

**Surf Better**. Es una aplicación desarrollada **con tecnologías modernas para llegar a un nicho muy concreto “los surfistas”.** Que no tienen demasiadas opciones para informarse de la situación de las playas antes de ir a Surfear con el añadido extra de poder leer comentarios de otros surfistas sobre las playas. Apoyar sus comentarios con un like y guardar sus playas favoritas en su perfil. Es una aplicación moderna y elegante que permite su visualización variable entre 6 tipos diferentes de pantallas (**350px-419px**, **520px-599px, 600px-767px, 768px-991px, 991-1290px**). Siendo esta última el ancho máximo variable permitido por la aplicación. Además, ofrece dos temas **light & dark**. Para que el usuario escoja a placer entre ambos.

# Organización.

Se ha tratado de trabajar un diseño de arquitectura **REST-API** **SERVER-LESS**. Teniendo esto en cuenta el Servidor en comparación con el cliente dista bastante. En el siguiente punto a este se hablará sobre **CodeStyle**. Punto que completará este sub-apartado. Dividiré la explicación en dos puntos **back-end & front-end**. Esta misma organización es la dada en el repositorio

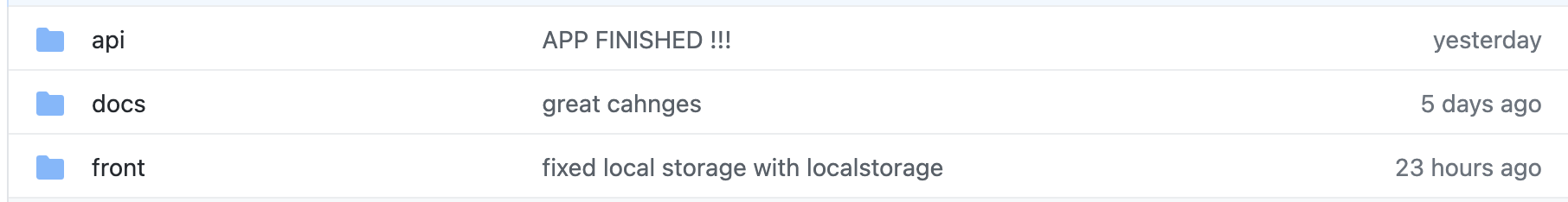


Figura 1: Git files

## Back-end

Comenzaré comentando este punto dado que es más corta por la arquitectura planteada. Se ha intentado seguir un modelo de diseño basado en componentes. Con esto en mente La aplicación se ha divido en 5 bloques (**Extensions, Models, Seeder, Services, Routes**)y un punto de encuentro **main.py**.

**Nota:** **El entorno virtual no viene incluido con la aplicación**. Este debe ser instalado por el usuario en caso de que escoja el **deploy** en modo desarrollo. En **/api** se incluye un archivo **apirequiremets.txt** con todas las dependencias a instalar con **python3.X**

**Comando:** (env) pip3 install -r apirequirements.txt

Siguiendo el orden propuesto (orden basado en el flujo de funcionamiento) en la línea anterior comentó:

### Extensions:

Se compone de un solo archivo **extensions.py** A modo de inyección de dependencia se incluyen las instancias principales de las que se van a hacer uso en el resto del servidor. Estas son principalmente **Praetorian()** instancia de flask-praetorian para llevar a cabo la autenticación y **SQLAlchemy** para trabajar en ORM.

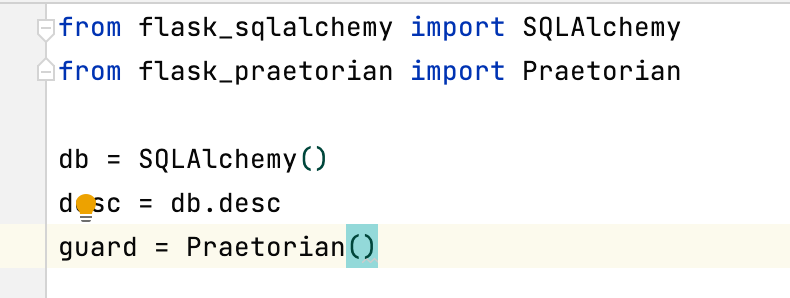


Figura 2: exrensions.py

### Models:

Se compone de un solo archivo **models.py**. En este se incluyen la definición de las **entidades y relaciones de la aplicación** (Se compone de 5 Entidades que se describirán en el apartado de diseño), estas hacen a su vez de clases de datos en la aplicación.

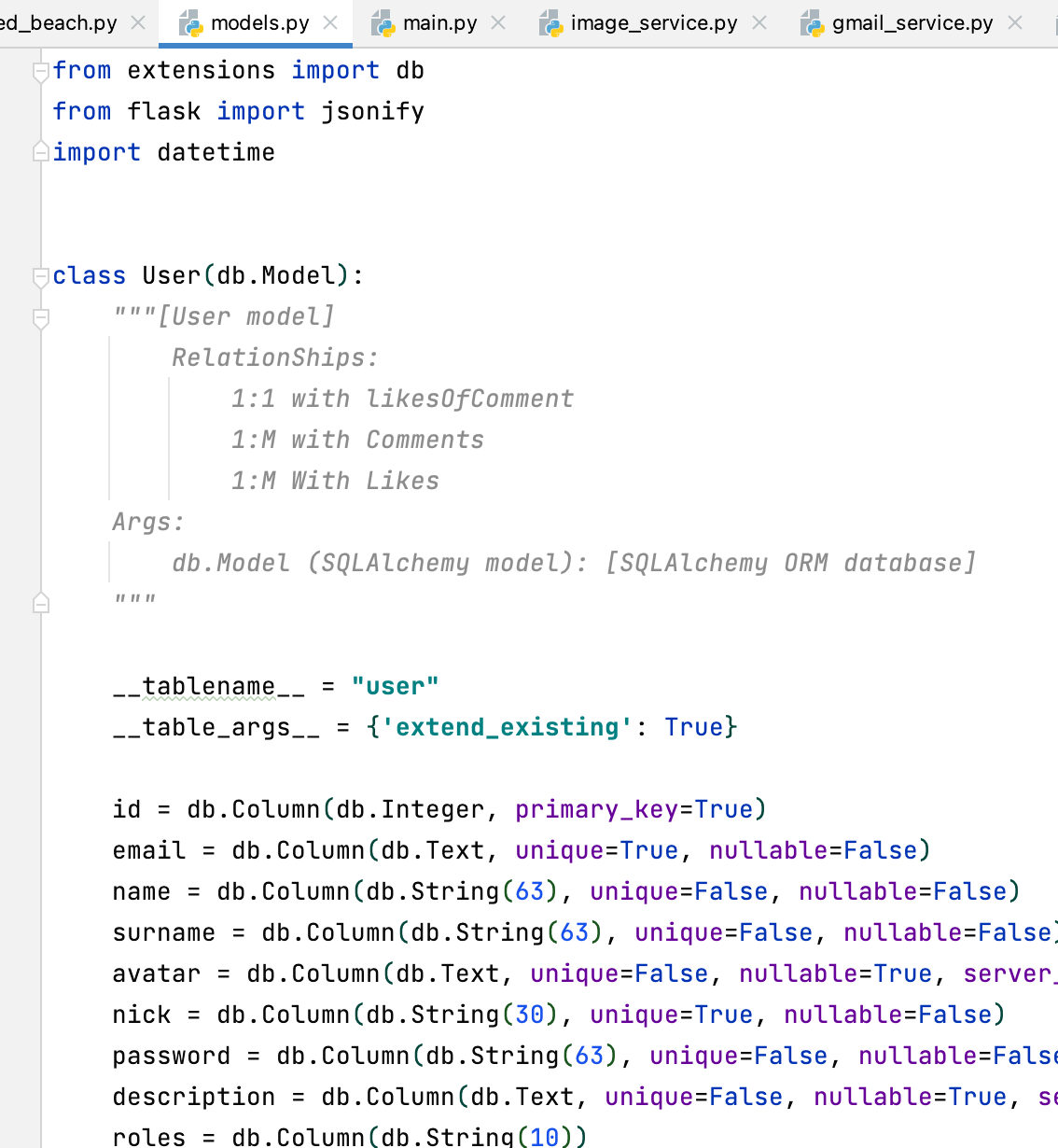


Figura 3: models.py

### Seeders.

Se compone de 3 clases cada destinadas a crear tanto la base de datos como realizar los **seeders** iniciales, así como de crear los distintos directorios de cada usuario. En el caso de que no existan datos en la aplicación. Estas clases son **seed\_user.py, seed\_comments.py, seed\_beaches.py.**

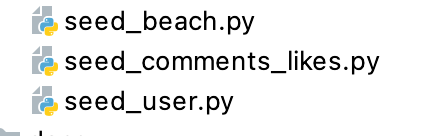
****

Figura 4: Seeders.py

A partir de este punto los diferentes componentes se dividen en módulos **blueprint**. Estos se inyectan directamente en el punto de encuentro de la aplicación **main.py**

### Services

Se compone de dos servicios. El primero de ellos destinado al servicio de coreo electrónico (este se usa para el formulario de contacto), el segundo al servicio de subida de imágenes de los usuarios, la carga del avatar y la carga de la imagen de cada playa.

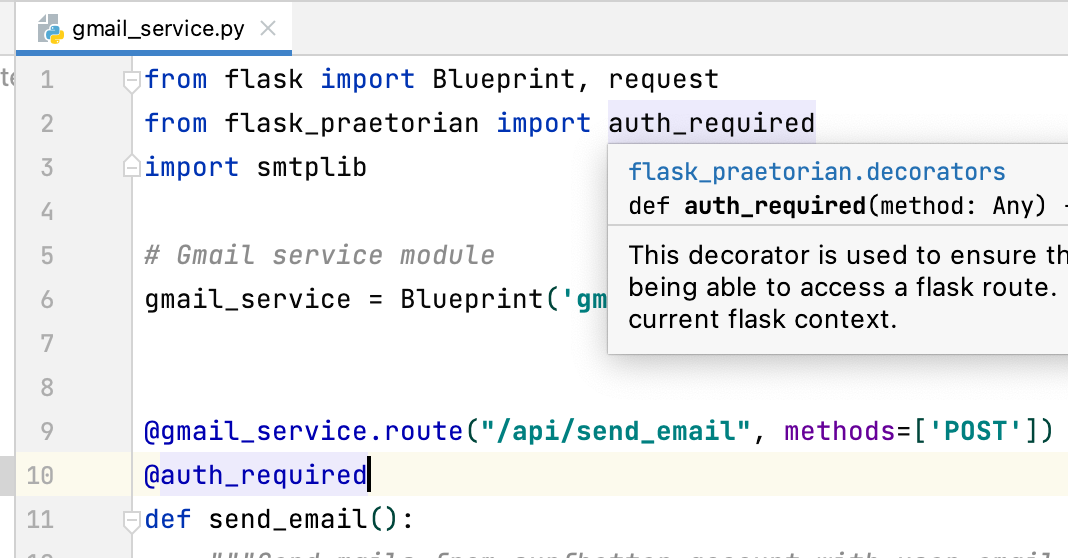
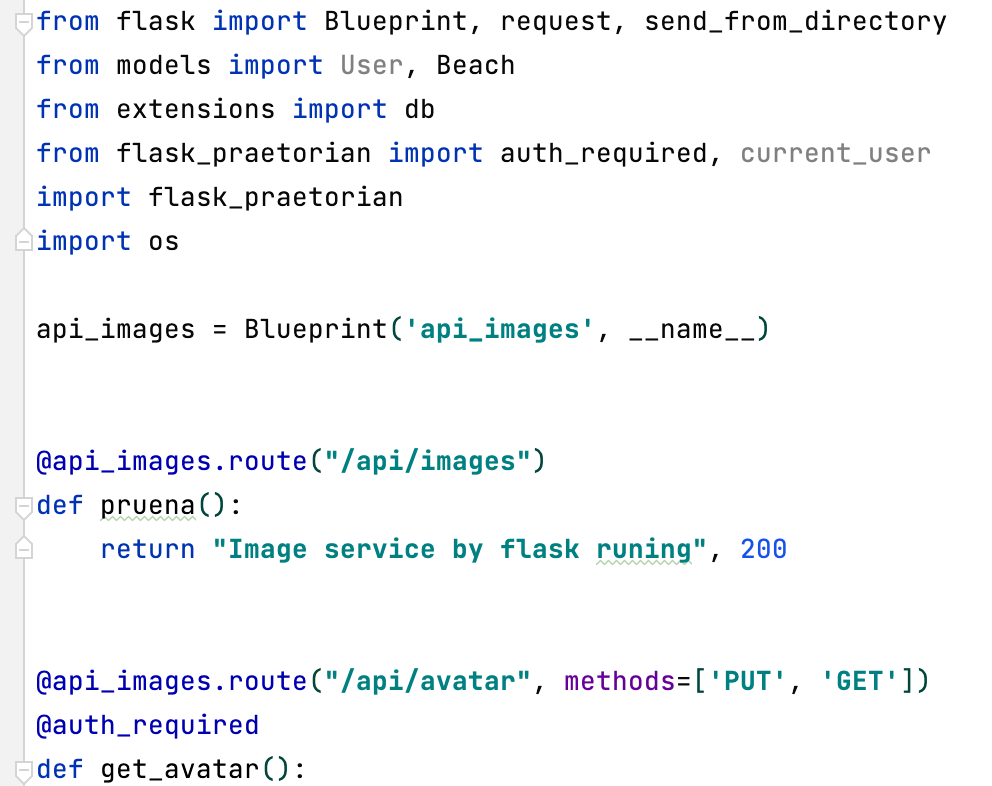
 

Figura 5: gmail\_service.py. Figura 6: image\_service.py

### Routes

Se compone de 3 archivos de rutas dividos en **módulos Blueprint**. Estos antes estaban dividios entre rutas protegidas y rutas públicas. Dado el tamaño del archivo de las rutas privada. Tome la decisión de refactorizar y dividir los módulos por funcionalidad.

Así pués este punto se divide en

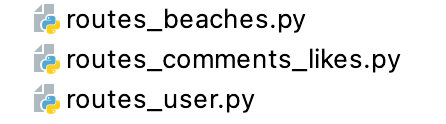


Figura 7: rutas.py

### Punto de encuentro

Este se compone de un único archivo **main.py** que hace de anclaje de entre el archivo de datos **models.py** y los diferentes módulos de rutas.

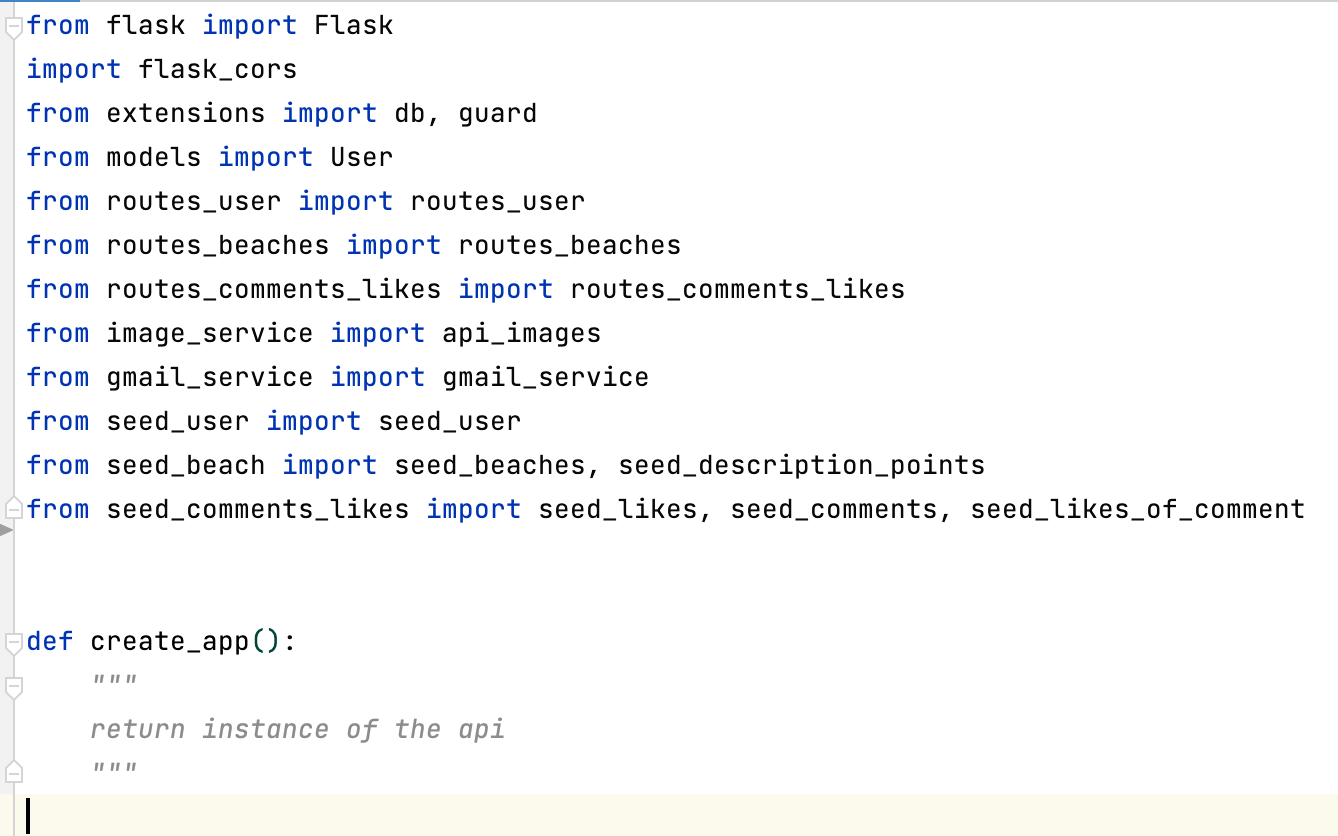
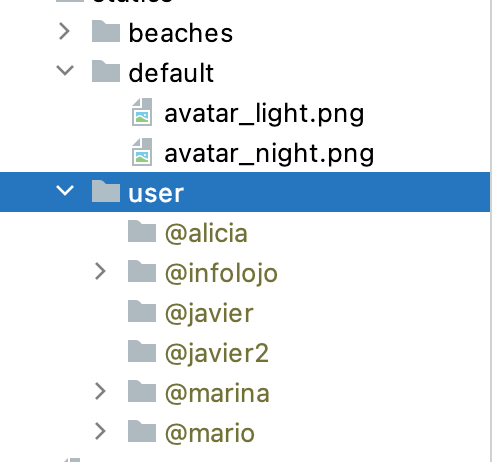


Figura 8: main.py

La base de datos se almacena en ./**api/database/database.db** Esta se crea junto a los seeders en el caso de que no existan datos.

**Archivos de la aplicación**. Además del código del que se compone el servidor. Este cuenta con una serie de archivos con los que a través de las rutas y los distintos servicios la apliación interactua con ellos.

Estos se encuentran **./api/statics**. Se dividen entre **beaches default user**

**Beaches :** En este directorio se almacena una imagen editada, reescalada y ajustada(reducir la carga de la aplicación) de cada playa.

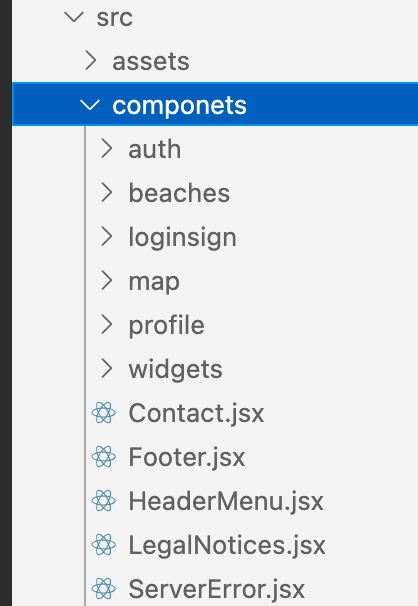
**User:** En este directorio se almacena un directorio con el nick de cada usuario. Este es único por lo que nunca existirán dos directorios iguales. En el momento en el que se registra un usuario y se acepta el acuerdo de uso se isntancia un directorio para almacenar la imágen de perfil. Esta imágen se inserta en el momento en el que el usuario decide actualizar su avatar, mientras que esto ocurre su imagen por defecto será la incluida dentro de **default.**

Figura 9: directorios servidor

## Front-end

Reiterando en el modelo de arquitectura que se ha tratado seguir este punto es más complejo. La aplicación está divida de manera similar a un proyecto en Java. Dentro de **src** se divide el código en diferentes archivos que en lugar de clases son componentes.

Estos componentes **en lugar de código js tienen código jsx.** Dichos componentes se han divido por funcionalidad dedicando cada uno de ellos a una funcionalidad en concreto. Si esta funcionalidad se hacía demasiado compleja se incluía dentro de un paquete con varios componentes para llevar a acabo dicha tarea. Los distintos componentes base son los siguientes:



# Descripción de la funcionalidad

Con esta aplicación los usuarios que se registren podrán en primera instancia acceder a una pantalla de **bienvenida** que corresponde a los componentes del paquete **loginsigin**