

PROYECTO FINAL DE CICLO

ANTONIO JOSÉ LOJO OJEDA



Ilustración 1 Logo SurfBetter

SURF BETTER

INDICE

1.	<i>Introducción.....</i>	5
1.1.	<i>Antecedentes</i>	5
1.2.	<i>Objetivo</i>	5
1.3.	<i>Tecnologías</i>	5
•	<i>Back-end</i>	5
•	<i>Front-end</i>	5
•	<i>Diseño.....</i>	6
•	<i>Despliegue</i>	6
•	<i>Seguridad.....</i>	6
1.4.	<i>Aplicaciones similares.....</i>	6
1.5.	<i>Expectativas.....</i>	7
2.1.	<i>Organización.</i>	7
	Back-end	8
	Extensions:.....	8
	Models:.....	9
	Seeders.....	10
	Services.....	10
	Punto de encuentro.....	11
	Front-end	12
	PUBLIC	30
	<i>Contiene el contenido público de la aplicación. Este directorio está prácticamente vacío incluye el siguiente contenido:</i>	30
2.2	<i>Descripción de la funcionalidad y manual de uso</i>	31
	1º STEP Inicio.....	31
	2º STEP Registro	32
	3º STEP Cierre de sesión.....	35
	4º STEP Inicio de sesión.	37
	5º STEP Actualización de los datos de usuario.....	38
	<i>¿Dónde encuentro la app?</i>	55
	<i>Tipos de Instalación</i>	56
	1º Desarrollo	56
	1º Preparación del entorno virtual	56
	<i>Para instalar los requerimientos se debe ejecutar en /api.....</i>	56

<i>pip3 install -r apirequirements.txt.....</i>	56
2º Producción Local	59
Arranque.....	59
1. <i>Guía de estilos.....</i>	63
2. <i>Fuentes.....</i>	64
3. <i>Tecnologías.....</i>	65
5. <i>Sketch.....</i>	66
6. <i>Mockups</i>	67
7. <i>Wireframe</i>	68
<i>Diseño de la aplicación.</i>	68
1. <i>Entidad-Relación</i>	69
2. <i>Modelos en UML</i>	70
3. <i>Flujos típicos</i>	71
6.1. <i>Front:.....</i>	72
6.2 <i>Servidor:</i>	73
7.1 <i>Control de versiones.</i>	74
Herramientas.	74
Desarrollo.	74
Dificultades encontradas:	75
7.2 <i>Despliegue.</i>	75
Herramientas.	75
7.3 <i>Diseño.....</i>	76
Herramientas.	76
Desarrollo.	76
Problemas encontrados.	76
7.4 <i>Front</i>	76
Herramientas.	76
Ides → Inicialmente se comenzó trabajando con Visual Studio Code. Cuando el proceso de desarrollo se volvió más complejo y fue adquiriendo más componentes y dependencias se migró de Visual Studio Code a WebStorm.	76
Revisiones de código →	76
Módulos o paquetes →	77
Problemas encontrados.	77
7.5 <i>Back End</i>	77
Herramientas:	77
Ides →	77
Revisiones de código →	77

Paquetes → S	77
8.1 <i>Front</i>	78
Ordenación de los métodos en los componentes	78
8.2 <i>Back-end</i>	81
<i>React</i>	89
<i>Flask</i>	89
<i>Design</i>	89
<i>Deploy</i>	89

1. Introducción

1.1. 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 intuitiva**, 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**.

1.2. 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.

1.3. 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 largo 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 usará 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 aleatoria 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.eyJpYXQiOjE2MjMwMDgwNTksImV4cCI6MTYyMzA5NDQ1OSwanRpljoiMjRlY2U0M2YtMTViZC00NjlILWi4ZGltMWViZTkwZDZjNGE0liwiaWQiOjEslnJscyl6ImFkbWluliwicmZfZXhwIjoxNjI1NjAwMDU5fQ.G_YzZp4CNdGEguvNwT_p3eqzvcuAsY0G04P52L9NyjU"
```

1.4. 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.

1.5. 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, responsive 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 modularizados 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.

2. 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.

2.1. 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

api	APP FINISHED !!!	yesterday
docs	great changes	5 days ago
front	fixed local storage with localstorage	23 hours ago

Ilustración 2 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 dividido 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 elija el **deploy** en modo desarrollo. En **/api** se incluye un archivo **apirequirements.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.

```
from flask_sqlalchemy import SQLAlchemy
from flask_praetorian import Praetorian

db = SQLAlchemy()
desc = db.desc
guard = Praetorian()
```

Ilustración 3 extensions.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.

```
d_beach.py × models.py × main.py × image_service.py × gmail_service.py ×
from extensions import db
from flask import jsonify
import datetime

class User(db.Model):
    """[User model]
    Relationships:
        1:1 with likesOfComment
        1:M with Comments
        1:M With Likes
    Args:
        db.Model (SQLAlchemy model): [SQLAlchemy ORM database]
    """

    __tablename__ = "user"
    __table_args__ = {'extend_existing': True}

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.Text, unique=True, nullable=False)
    name = db.Column(db.String(63), unique=False, nullable=False)
    surname = db.Column(db.String(63), unique=False, nullable=False)
    avatar = db.Column(db.Text, unique=False, nullable=True, server
    nick = db.Column(db.String(30), unique=True, nullable=False)
    password = db.Column(db.String(63), unique=False, nullable=Fals
    description = db.Column(db.Text, unique=False, nullable=True, s
    roles = db.Column(db.String(10))
```

Ilustración 4 : 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**.

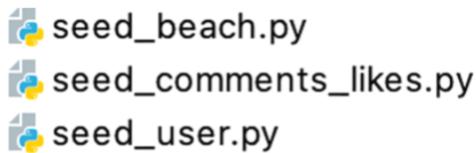


Ilustración 5 Seeders.py

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 correo 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.

```
gmail_service.py
1 from flask import Blueprint, request
2 from flask_praetorian import auth_required
3 import smtplib
4
5 # Gmail service module
6 gmail_service = Blueprint('gm
7
8
9 @gmail_service.route("/api/send_email", methods=['POST'])
10 @auth_required
11 def send_email():
```

Ilustración 6 image_service.py

```
from extensions import Blueprint, request, current_app, current_user
from models import User, Beach
from extensions import db
from flask_praetorian import auth_required, current_user
import flask_praetorian
import os

api_images = Blueprint('api_images', __name__)

@api_images.route("/api/images")
def pruebla():
    return "Image service by flask running", 200

@api_images.route("/api/avatar", methods=['PUT', 'GET'])
@auth_required
```

Ilustración 7 gmail_service.py

Routes

Se compone de 3 archivos de rutas divididos en **módulos Blueprint**. Estos antes estaban divididos 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 pues este punto se divide en

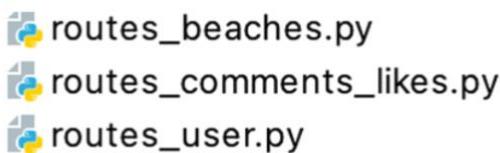


Ilustración 8 rutas.py

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.

```
from flask import Flask
import flask_cors
from extensions import db, guard
from models import User
from routes_user import routes_user
from routes_beaches import routes_beaches
from routes_comments_likes import routes_comments_likes
from image_service import api_images
from gmail_service import gmail_service
from seed_user import seed_user
from seed_beach import seed_beaches, seed_description_points
from seed_comments_likes import seed_likes, seed_comments, seed_likes_of_comment


def create_app():
    """
    return instance of the api
    """
|
```

Ilustración 9 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 aplicación interactúa 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.

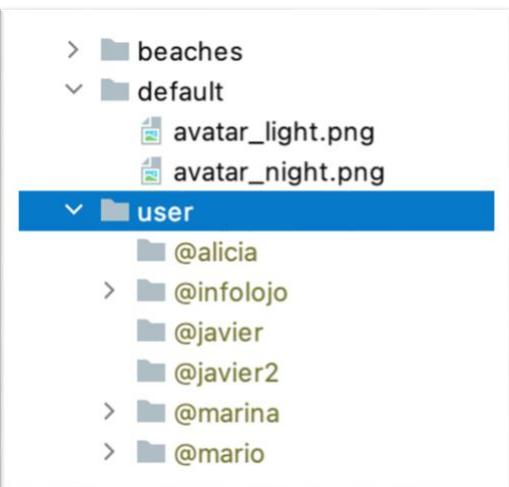


Ilustración 10 usuarios servidor

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 instancia un directorio para almacenar la imagen de perfil. Esta imagen 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**.

Front-end

Reiterando en el modelo de arquitectura que se ha tratado seguir este punto es más complejo. La aplicación está dividida de manera similar a un proyecto en Java. Dentro de

SRC

src se divide el código en diferentes archivos que en lugar de clases son componentes. Estos componentes **en lugar de código js están compuestos de código jsx**. Dichos componentes se han dividido por funcionalidad dedicando cada uno de ellos a una acción en concreto.

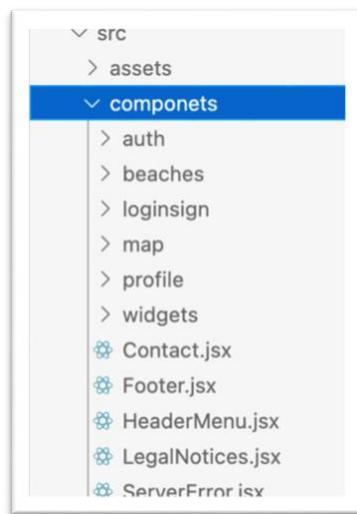


Ilustración 11 Componentes

La manera de actuar fue la siguiente. Inicialmente existían los componentes justos para cada vista de la aplicación. Muchas de estas (la mayoría) se han hecho más complejas por lo que se ha requerido encapsular en directorios (que llamaré paquetes).

La distribución final fue la que se muestra en la imagen de la derecha.

Si se desglosan estos componentes todos coinciden en lo mismo. Se trata **de uno principal** que hace de **host** para el resto de componentes de la aplicación.

Auth -> En este se encapsula el componente dedicado a la autenticación de la aplicación. Este hace uso del paquete **react-token-auth** y se surte del token proporcionado por **flask**



Ilustración 12 Paquete auth

```

front > src > components > auth > auth.jsx > ...
1 import {createAuthProvider} from 'react-token-auth';
2
3 export const [useAuth, authFetch, login, logout] =
4   createAuthProvider({
5     accessTokenKey: 'access_token',
6     onUpdateToken: (token) => fetch('/api/refresh', {
7       method: 'POST',
8       body: token.access_token
9     })
10    .then(response => response.json)
11  });
12

```

Ilustración 13 Componente autenticación

Beaches -> Este paquete se compone de 5 componentes. **BeachCard**. De este componente se hace uso tanto en el **Perfil del usuario** para cargar las playas comentadas y favoritas de cada usuario así como en el componente **Beaches**. Este componente se carga en **BeachHost** y carga todas las playas o las playas buscadas por el propio usuario. Dependiendo de la acción realizada por el usuario. Por último el paquete cuenta con un componente extra **BeachStar** que se encarga de cargar las puntuaciones mokeadas de cada playa en el componente **BeachInfo**. Este componente se carga tanto desde el paquete **Beaches** en cada **BeachCard**, como en el perfil del usuario como en el **mapa de playas** (paquete maps). Es posible visualizar la información de las playas y dejar o ver comentarios. Eliminar los propios, darle a favoritos a la propia playa o a un comentario en concreto así como eliminar los comentarios dejados por el usuario autor. Las playas que sean comentadas o favoritas aparecerán en el perfil de cada usuario. BeachInfo.jsx incluye entre sus variables **props el nombre del componente llamado** incluyendo un botón para volver al componente anterior.

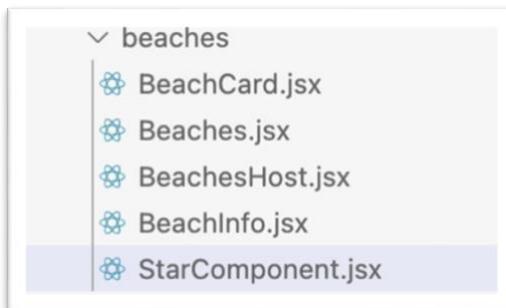


Ilustración 14 Paquete beaches

Login Sign in -> Este paquete se compone de 4 componentes. Componente **Header**. Exclusivo para el login (única sección pública de la aplicación).

Con pública me quiero referir a que no está restringida por token de autenticación

Más adelante se comentarán los componentes que no dependen de paquetes. Entre ellos está Header Común.

Continuado con **Login** la distribución del contenido es la siguiente:

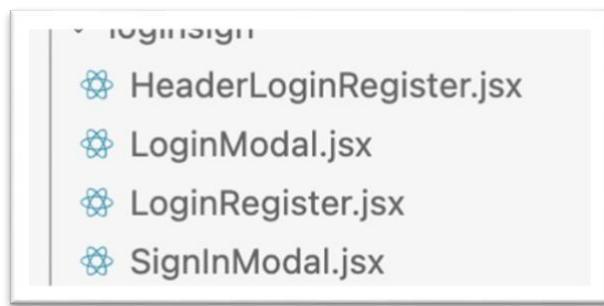


Ilustración 15 Paquete loginsigin

El componente principal es **LoginRegister**. Este incluye el resto de componentes. El **Header** contenedor de dos botones que hacen de trigger para abrir o bien el componente **LoginModal** o el componente **SignInModal**. (Destacar que ambos están codificados en SASS).

Map -> Este Paquete se dedica a cargar un componente final **Map.jsx** el cual muestra un fragmento de googleMaps con todas las playas. Al pasar el ratón por encima de cada uno de sus marques muestra un **diálogo de información** donde se muestra un la media de puntuación mockeada junto al nombre de cada playa así como un botón de información para cargar la información de cada playa. Llamando a **BeachInfo.jsx**.

```

import React, { useState, useEffect } from "react";
import Map from "./Map";
import HeaderMenu from "../HeaderMenu";
import mapsKey from "../../credentials/credentials";
import { mapLight, mapDark } from "./mapStyle";
import swal from "sweetalert";

const GOOGLE_MAP_URL = "https://www.google.com/maps/api/js?key=${mapsKey.mapsKey}";

/**
 * MapHost component
 * @returns {JSX.Element}
 * @constructor
 */
const MapHost = (props) => {
  const [beaches, setBeaches] = useState([]);

  useEffect(() => {
    getBeaches().then(() => {
      //NO-LOOP
    }, []);
  });

  const getBeaches = async () => {
    fetch("/api/beaches")
      .then(response => response.json())
      .catch(() => async () => {
        await swal("Error", "Markers not found", { icon: "warning" }).then(() => {
          //NO-LOOP
        });
      })
      .then(response => setBeaches(response));
  };
};

```

Ilustración 16 BeachHost.jsx

En este componente se carga el **mapa**. Variando el tema en función de si la preferencia del usuario con respecto al tema es **lightMode** o **DarkMode**.

```
const setMap = () => {
  if (props.darkMode){
    return(
      <Map
        mapStyle={mapDark}
        markers={beaches}
        googleMapURL={GOOGLE_MAP_URL}
        containerElement=<div style={{height: '90vh'}}/>
        mapElement=<div style={{height: '100%'}}/>
        loadingElement=<i className="fas fa-spinner fa-4x fa-spin">/>
    );
  } else {
    return(
      <Map
        mapStyle={[mapLight]}
        markers={beaches}
        googleMapURL={GOOGLE_MAP_URL}
        containerElement=<div style={{height: '90vh'}}/>
        mapElement=<div style={{height: '100%'}}/>
        loadingElement=<i className="fas fa-spinner fa-4x fa-spin">/>
    );
  }
};

return (
  <div>
    <HeaderMenu/>
    {
      (beaches[0] !==undefined&&beaches[0]!==null)&&(setMap())
    }
  </div>
);
};

export default MapHost;
```

Ilustración 17 BeachHost.jsx (2)

El tema cargado se almacena en formato en Json en el archivo **mapStyle**, Este exportados **Json**. Uno para El mapa en **modo light** y un segundo para **modo dark**

```
1 export const mapLight = [
2   {
3     "featureType": "administrative",
4     "elementType": "geometry",
5     "stylers": [
6       {
7         "visibility": "off"
8       }
9     ],
10    },
11    {
12      "featureType": "administrative",
13      "elementType": "labels",
14      "stylers": [
15        {
16          "visibility": "off"
17        }
18      ],
19    },
20  ],
```

Ilustración 18 mapLight const (1)

Por último el mapa cargado incluye todos las playas y el tema y renderiza todo esto.

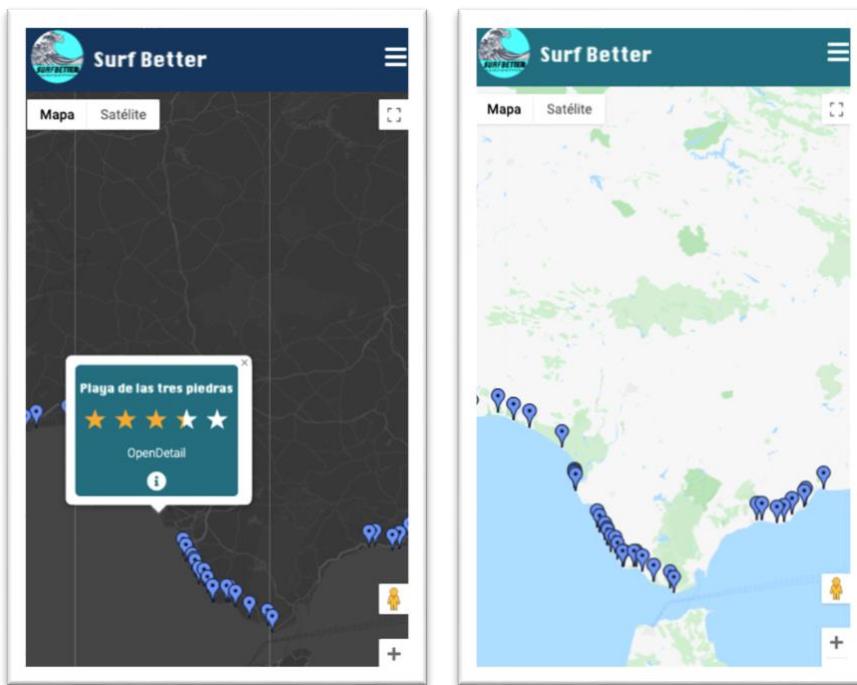


Ilustración 19 mapHost

Ilustración 20 lightMode

Profile -> Este paquete es parecido al anterior incluye un componente principal **Profile.jsx** que incluye **3 modales extras** para la configuración de los datos personales del usuario dentro de la aplicación.

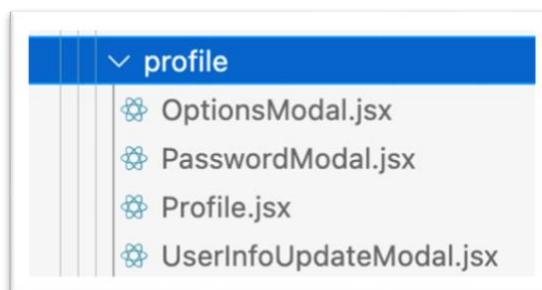


Ilustración 21 Paquete perfil

Además de estos modales el usuario puede haciendo click sobre su imagen de usuario actualizar la misma a través de un formulario oculto en la imagen. Además de eso en **Profile.jsx**. Se cargan **las playas favoritas y comentadas**. Pudiendo a través de ellas comentar, darle a favoritos o acceder al detalle de cada una de ellas.

```
    // ...scroll logic...
    <section className="ProfileBeaches">
      <h2>Favorite Beaches</h2>
      <section className="contentBeaches">
        { /*Loop by map to set beaches*/
          favorites==undefined&&
            favorites.map(it => {
              return(
                <BeachBox beach={it} from={"profile"} />
              )
            })
        }
      </section>
      <h2>Commented Beaches</h2>
      <section className="contentBeaches">
        {
          beachComments==undefined&&
            beachComments.map(it => {
              return (
                <BeachBox beach={it} key={it.id} from={"profile"} />
              )
            })
        }
      </section>
    </section>
```

Ilustración 22 Profile.jsx

Widgets -> Actualmente incluye un solo componente para el botón flotante de scroll. A través del evento **scroll** se carga cuando el scroll inferior supera 300. Este se carga de manera global en **App.jsx** para toda la aplicación. Por añadir destacar. Que dado al tiempo de desarrollo no se ha podido finalizar la aplicación de la manera más deseada. A futuro la idea es continuar modularizando esta e incluir de manera común más widgets como este que se instancian en otros componentes. De ahí la idea de instanciar un paquete para este propósito.

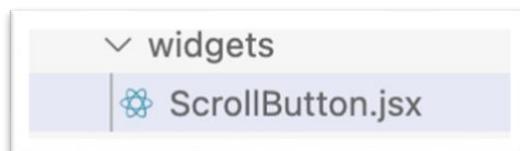


Ilustración 23 Paquete widgets

```

import React, {useState} from "react";

const ScrollButton = () => {
  const [visible, setVisible] = useState(false);

  const toggleVisible = () => {

    const scrollTop = document.documentElement.scrollTop;
    if (scrollTop > 300){
      setVisible(true);
    } else if (scrollTop <= 300){
      setVisible(false);
    }
  };

  /**
   * Do scroll
   */
  const scrollToTop = () =>{
    window.scrollTo({
      top: 0,
      behavior: 'smooth'
    });
  };

  window.addEventListener('scroll', toggleVisible);

  return (
    /*show if its visible*/
    visible&&
      /*Set event listener on toggleVisible*/
      <span id={"scroll-button"} onClick={() => scrollToTop()}>
        <i className="fas fa-arrow-alt-circle-up fa-2x"/>
      </span>
  );
}

```

Ilustración 24 ScrollButton.jsx

Componentes sin paquete -> Estos componentes son componentes son : **Contact.jsx**, **LegalNotices.jsx** , **HeaderMenu.jsx** y **ServerError.jsx**

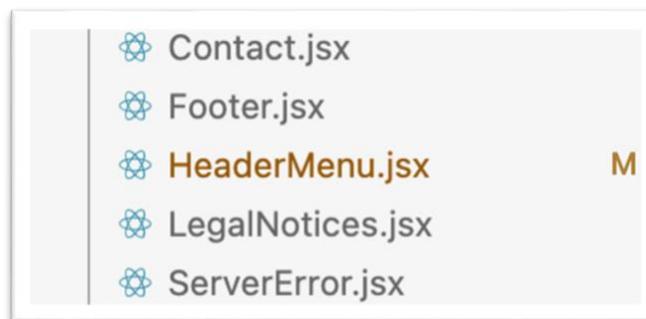


Ilustración 25 Componentes sin paquete

Estos componentes se excluyen de paquetes ya que están destinados a un contenido más simple y específico.

Contact.jsx -> Incluye el contenido de contacto. Información acerca de **infoLojo** (**proyecto personal**). Integrantes del grupo **infoLojo** y un formulario de contacto funcional que atraves del servicio **gmail_service.py** envía a mí correo personal un email del usuario (en el caso de que el email sea real).

```

const getUser = async () => {
  authFetch("/api/current_user")
    .then(response => response.json())
    .catch(error => console.log(error))
    .then(userInfo => setUser(userInfo))
}

/**States for email and subject*/
const [text, setText] = useState("")
const [subject, setSubject] = useState("")
const [user, setUser] = useState({})

const checkAndSendEmail = (e) => {
  e.preventDefault()
  const opts = {
    "user_email": user.email,
    "subject": subject,
    "message": text
  }
  authFetch("/api/send_email", {
    method: "POST",
    body: JSON.stringify(opts)
  }).then(response => response.json())
    .then(async () => await swal("Message has been send success", {icon: "success"}))
    .catch(async () => swal("Error", "Something was wrong with email", {icon: "warning"}))
}

```

Ilustración 26 Contact.jsx

```

<Router ru="contact-me">
  <section className="footer-content">
    <form action="/" onSubmit={(e) => checkAndSendEmail(e)}>
      <fieldset>
        <legend title="Send us an email">Email ?</legend>
        <input type="text" name="subject" id="subject" aria-label="subject"
          onChange={(e) => setSubject(e.target.value)}
          placeholder="Subject" required={true}/>
        <textarea name="message" id="message" title="Your message"
          aria-label="message"
          onChange={(e) => setText(e.target.value)}
          placeholder="Hello SurfBetter 🙌" required={true}/>
        <input className="buttonBlue" type="submit" value="Send"/>
      </fieldset>
    </form>
  </section>
</Router>

```

Ilustración 27 Form.jsx

LegalNotices.jsx

Contenido de información legal. Este componente incluye de manera estática toda la información referente al uso de datos de usuario. Es de obligado cumplimiento su aceptación en el momento de registro. Pudiendo hacer click sobre el ícono de información se puede acceder a su contenido durante el registro.

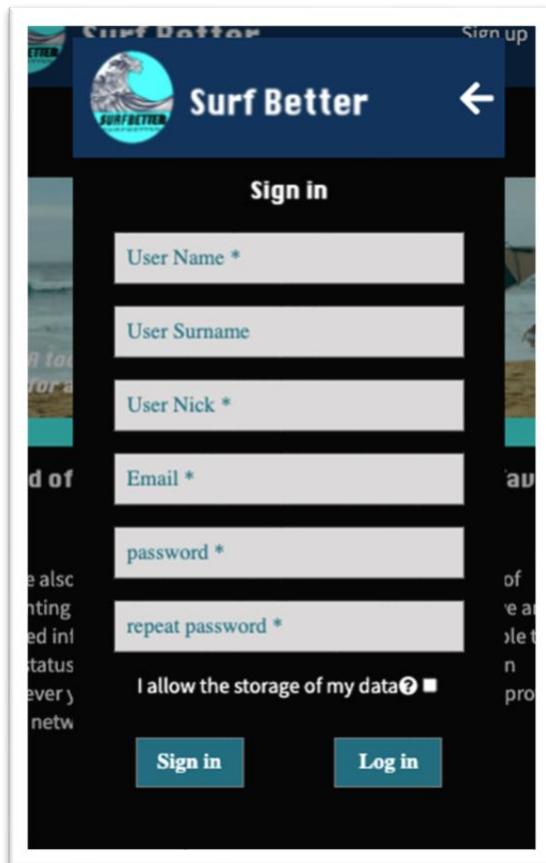


Ilustración 28 : Acceso a LegalNotices

El usuario puede consultar la información referente a su información personal desde cualquier punto de la aplicación una vez aceptado el alta desde el **footer** componente de la aplicación **Footer.jsx**.



Ilustración 29 : Acceso a LegalNotices (2)

Terms and conditions

SurfBetter informs users of the website of its policy of protection of personal data. The use of the SurfBetter web app space and any of the services that are incorporated presupposes the full acceptance of the conditions that are manifested in the privacy policy that is exposed.

Data Collect

In compliance with Law 15/1999, of December 13, on the protection of personal data, it is reported that the personal data requested in our forms will be included in a personal data file the person in charge and owner of which is SurfBetter. Likewise, when a person fills in any of the forms with the personal data requested and accepts the shipment, expressly authorizes SurfBetter to treat or incorporate into the automated file of their property the personal data provided in the aforementioned form and all data that are generated in relation to your participation or use of the different events that are offered on this website.

Unless specifically stated otherwise, it will be considered necessary to fill in all the fields of each form, for which the user must fill out the forms with true, accurate, complete and updated data. The user is solely responsible for any loss or damage, direct or indirect, caused to SurfBetter or any third party by filling out the forms with false, inaccurate, incomplete or outdated information or with data from third parties. Through the different areas that are part of this web space, users can obtain information, make inquiries and participate in the different events that SurfBetter offers through its web space.

As soon as you stop using the SurfBetter application, it deletes all your data regarding the application.

Purpose of the treatment

The data provided will never be used for a purpose other than that for which they have been assigned, and will be canceled immediately after ceasing to be necessary for this purpose, except when a law establishes otherwise.

The real intention of SurfBetter is to offer a real iteration of the users apart from consulting the data of their favorite beaches. In the form of comments and I like you

Security

All information regarding personal data received by SurfBetter is treated with the utmost confidentiality, and only the communications strictly necessary for the provision of the service of interest in each case and the communications and assignments authorized by the user through this policy are used. Of privacy.

SurfBetter saves a random token regarding your session when you start the application that is stored locally. This token changes with each login and identifies you as a user throughout the application

Ilustración 30 Contenido legal

HeaderMenu.jsx

A nivel de funcionalidad este componente si que se puede considerar complejo. Junto con el componente **Route** de App.jsx comprende las rutas completas de la aplicación. En él se instancian los menús superiores tanto del modo escritorio como móvil que con SASS se reorganizan y escalan para los 6 tipos de pantalla diferentes. Haciendo uso del componente de React **Link** se llama a la ruta correspondiente al elemento seleccionado.

```
return (
  <div>
    {/*div or section required because all must be under any parent target*/}
    <header id="header-common-app">
      <img src={logoSurfBetterHeader} alt="surfbetter logo" title="surf better logo" width="330" height="120"/>
      <nav className="navDesk">
        <ul>
          <li>
            <Link to="/contact" title="Go to contact" alt="Link to contact">
              Contact
            </Link>
          </li>
          <li>
            <Link to="/" title="go to Beaches" alt="link to beaches">
              Beaches
            </Link>
          </li>
          <li>
            <Link to="profile" title="go to Profile" alt="link to profile">
              Profile
            </Link>
          </li>
          <li>
            <Link to="/map" title="go to resources" alt="link to resources">
              Map
            </Link>
          </li>
        </ul>
      </nav>
    </header>
  </div>
)
```

Ilustración 31 HeaderMenu.jsx

Además de las navegación entre rutas se incluye la lógica de **log-out**. Haciendo uso de la dependencia instalada **swal alert**. Se muestra un modal preguntando por el cierre de sesión para que en caso afirmativo eliminé el token de sesión y tras esto React envía al usuario a **/login** ya que no puede estar en otra ruta sin autenticar.

```
/** search token on localStorage. Delete it and push history to home */
const logOut = () => {
  swal({
    title: "Log out",
    text: "Are you shure you want to log out?",
    icon: "warning",
    buttons: true,
    dangerMode: true,
  })
  .then((logout) => {
    if (logout) {
      swal("See you next time surfer", {
        icon: "success",
      })
      .then(async () => {
        localStorage.removeItem('REACT_TOKEN_AUTH_KEY')
        window.location.replace("/login")
      })
    } else {
      swal("You will not regret 😊")
    }
  })
}
```

Ilustración 32 const logOut

Además de las funcionalidades anteriores se incluye el trigger de cambio de token. Este hace uso de un método global que en función de un **estado reactivo y una variable en localStorage** hace el cambio de tema agregando o suprimiendo la clase **darkMode** a HTML.

```
//Note: State used to display and hidde menu
const [menuNavState, setMenuNavState] = useState(false);
//Note: DarkMode val TODO: SAVE ON MY API SERVICE
const [darkModeState, setDarkModeState] = useState(null);

/**
 * Change between darkMode and light mode
 * @param {event} e: Event
 */
const changeToDark = () => {
  const htmlTarget = document.querySelector('html');

  //ChangeTheme
  changeDarkMode(htmlTarget,darkModeState)
  //Set State
  setDarkModeState(!darkModeState)
  //Set localStorageItem
  darkModeState?localStorage.setItem("theme","darkMode"):localStorage.setItem("theme","lightMode")
```

Ilustración 33 Lógica de cambio de tema

Key	Value
theme	lightMode
REACT_TOKEN_AUTH_KEY	{"access_token": "eyJ0eXAiOi...
1	lightMode

Ilustración 34 Variable de tema

```
/**  
 * SetUnset darkMode  
 */  
export const changeDarkMode = (htmlTarger, darkPressed) => {  
    darkPressed?htmlTarger.classList.add("darkMode"):  
    htmlTarger.classList.remove("darkMode")  
}  
:  
:
```

Ilustración 35 Lógica del cambio de tema

Para terminar la lógica del cambio de tema es necesario acudir a App.jsx donde en cada carga se comprueba la existencia de este token para de nuevo llamar al método común de Util y cambiar el tema. Por defecto se utiliza siempre el tema light.

```
/*
 *
 * @returns {jsx component with routes and user}
 */
function App() {
  useEffect(() => {
    const htmlTarget = document.querySelector('html');
    if (!localStorage.getItem("theme")) {
      localStorage.setItem("theme", "lightMode")
      setDark(false)
    } else {
      if (localStorage.getItem("theme") === "darkMode") {
        htmlTarget.classList.add('darkMode')
        changeDarkMode(htmlTarget, true)
        setDark(true)
      } else {
        htmlTarget.classList.remove('darkMode')
        changeDarkMode(htmlTarget, false)
        setDark(false)
      }
    }
  }, []);
}
```

Ilustración 36 Comprobación del tema

Footer -> Componente estático . muestra información relevante para el usuario. Acceso a contacto. Mapa de la web, acceso a la información legal y acceso a **infoarlo** así como al github de infoarlo.

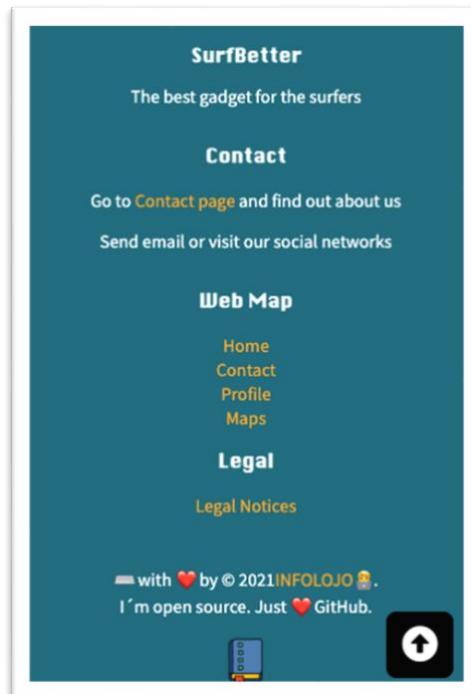


Ilustración 37 Footer.jsx

Fuera de los paquetes y referente al código se encuentran **ubicados 3 archivos. Index.js Util.js y App.jsx.**



Ilustración 38 Archivos generales

App.jsx

El último de ellos es el **último componente jsx que conforma la aplicación**. Este hace de unión del resto de componentes. La funcionalidad del mismo es unir toda la aplicación, cargar las rutas y cargar el tema correspondiente en función de la preferencia del usuario.

A screenshot of the App.jsx file content. The code defines a return function that wraps a Router component with history. It uses a Switch component to handle different routes. The routes include public routes for login and legal pages, and protected routes for beaches and contact pages. Each route includes components like LoginRegister, Footer, and ScrollButton.

```
return (
  <Router history>
    <Switch>
      {/*Public routes*/}
      <Route path="/login" exact>
        <LoginRegister history={history} />
        <ScrollIndicator/>
      </Route>
      <Route path="/legal" exact>
        <LegalNotices/>
        <Footer/>
        <ScrollIndicator/>
      </Route>
      {
        !logged &&
        | <Redirect to="login"/>
      }
      {/*Protected routes*/}
      <Route path="/" exact>
        <BeachesHost/>
        <Footer/>
        <ScrollIndicator/>
      </Route>
      <Route path="/beaches" exact>
        <BeachesHost/>
        <Footer/>
        <ScrollIndicator/>
      </Route>
      <Route path="/contact" exact>
        <Contact/>
        <ScrollIndicator/>
      </Route>
    </Switch>
)
```

Ilustración 39 App.jsx

Util.jsx

Carga las constantes comunes de la aplicación. Me hubiese gustado modularizar algo más la aplicación y crear más archivos como este que provengan de dependencias al resto de dependencias. Por falta de tiempo se queda como punto a mejorar a futuro.

```

25  export const checkString = (cadena, min, max) => {
26    let flag = false
27    if (cadena.length > min && cadena.length < max){
28      cadena.split(" ").forEach(element => {
29        if (element !== "" && !isNaN(element)) {
30          console.log("I have found a number");
31          flag = true;
32        }
33      })
34    } else {
35      flag = true
36    }
37    return !flag;
38  }
39
40  export const isLikeFromUser = (likes, user_id) => {
41    let result = ""
42    likes.forEach(it => {
43      if (it.user_id === user_id) && (result = "red-flag")
44    })
45    return result
46  }
47
48 /**
49 * set the error
50 * @param {string} state
51 * @param {target} input
52 * @param {min_max} min and max
53 */
54 export const setError = (state, input, min_max) => {

```

Ilustración 40 Util.jsx

Index.jsx

Renderiza la aplicación en `public/index.html` -> `<div id="root"/>`. Más adelante comentaré **esta sección de mi front**.

```

src > index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import App from './App.jsx';
4
5  ReactDOM.render(
6    <React.StrictMode>
7      <App />
8    </React.StrictMode>,
9    document.getElementById('root')
0  );
1

```

Ilustración 41 Index.js

El último directorio de **SRC** es **asssets**. Aquí se encuentra el contenido estático que se sirve desde el front. Pero que por seguridad no quiero servir desde **public**. Además de este contenido estático se encuentran los estilos en SASS.

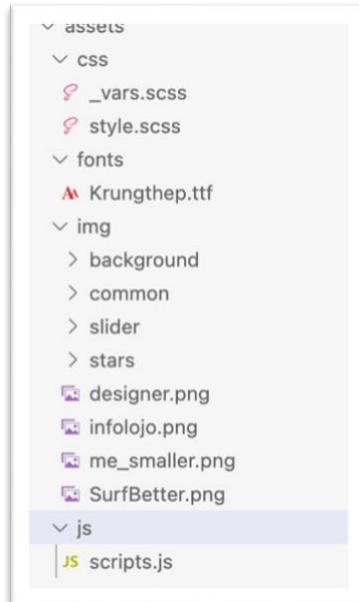


Ilustración 42 statics

css-> En **css** se incluyen dos archivos. Uno dedicado a variables globales **_vars.scss** y un segundo donde se importa este con el contenido de los estilos **style.scss**.

_vars.scss -> Se incluye los imports, fuentes, tamaños de letras, colores , temas, transiciones

```
/*Style Vars file*/
@import url('https://fonts.googleapis.com/css2?family=Noto+Sans+JP');

@font-face{
    font-family: krugthep;
    src: url('../fonts/Krungthep.ttf');
}

/*Fonts*/
$font-common : 'Noto Sans JP';
//$font-common : 'Noto Sans, sans-serif';
$font-special : krugthep, sans-serif;
$font-inputs: 'Playfair Display', serif;
$font-common-big-desk: 16pt;
$font-common-big-title-desk: 18pt;

$size-menu-desk: 14pt; /*fontsize menu desktop*/
$size-common-desk: 14pt;
$size-common-desk-title: 16pt;
$size-modal-label: 10pt; /*font-size label*/
$size-common-mobile-title: 14pt;
$size-common-mobile: 12pt;

/*Colors*/
$color-header: #16697A;
$color-footer: #0f3057;
$color-yellow: #FFA62B;
$color-font: dark;
$color-white: white;
$color-black: black;
$color-red: red;
$color-little-red: #a03535;
$color-blue-content: #3EFDF6;
```

Ilustración 43: _vars.scss

En **styles.scss** Se incluye el import del módulo de **font-awesome**, lógica de los modales, métodos de extensiones comunes, funciones o mixins, clase modo oscuro y breakpoints para cubrir todas las pantallas posibles

```

/*Modules*/
@import "vars";
@import "~@fortawesome/fontawesome-free/css/all.css";

/*Extends*/
%common-transition-hover-menu {
    transition: $mouse-is-hover-menu;
    font-weight: bolder;
    text-decoration: none;
}

/*Extends for all kind of buttons*/
%buttons {
    font-weight: bolder;
    padding: .5em;
    transition: $mouse-is-over-menu;
    width: 90px;
    display: block;
    border-style: solid;
}

```

Ilustración 44 style.scss

Img -> Imágenes estáticas del front.

Font -> Fuente Especial para títulos de la aplicación

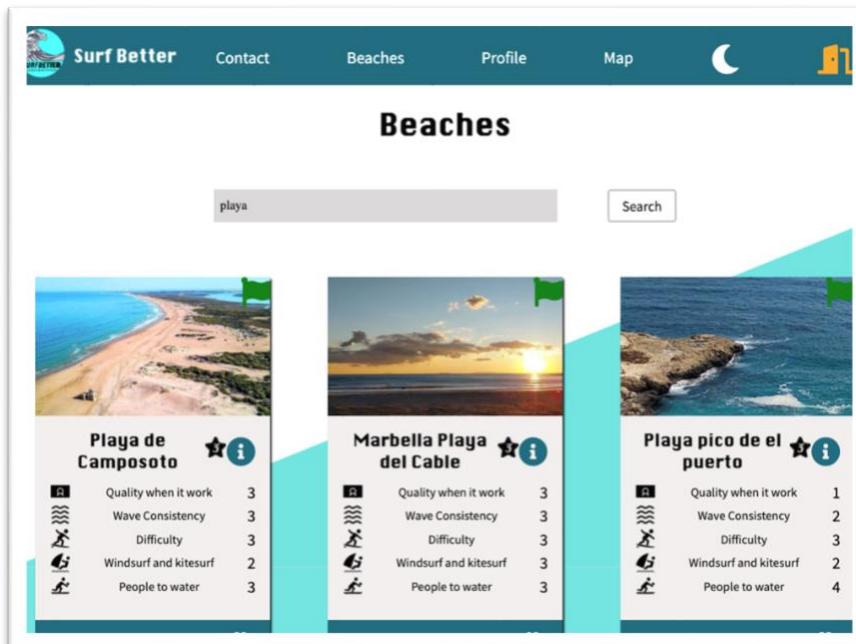


Ilustración 45 Ejemplo de fuente de título

Para finalizar el contenido de **SRC**. Se incluye una paquete extra **credentials**. Con la clave de GoogleMaps.



Ilustración 46 credentials.js

TODO EL CONTENIDO DESCRIBIDO DURANTE ESTE PUNTO SE COMPILA AL LLEVAR LA
APLICACIÓN A PRODUCCIÓN

PUBLIC

Contiene el contenido público de la aplicación. Este directorio está prácticamente vacío incluye el siguiente contenido:

favico.ico -> ícono de **SurfBetter**

index.html -> HTML base. Carga el contenido renderizando por **index.js** de **src**. Que a su vez carga todos los componentes de la aplicación

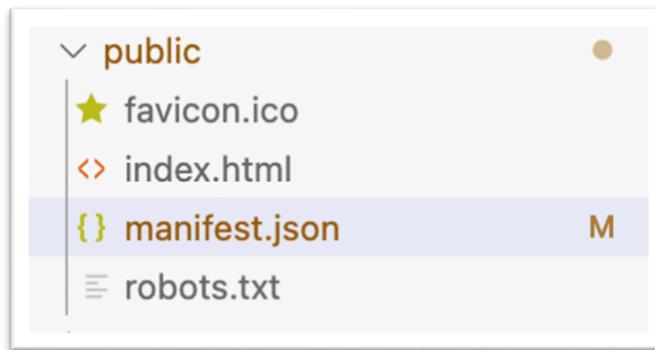


Ilustración 47 Public

Para finalizar con la organización del proyecto existen dos archivos más **package.json** y **package-lock.json**. (Es el gradle de Node). Incluye todas las dependencias y scripts de la aplicación.

```
  "name": "tronteria",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@fortawesome/fontawesome-free": "^5.15.3",
    "@testing-library/jest-dom": "^5.11.9",
    "@testing-library/react": "^11.2.5",
    "@testing-library/user-event": "^12.8.3",
    "html-react-parser": "^1.2.6",
    "node-sass": "^5.0.0",
    "nodemailer": "^6.6.1",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-google-maps": "^9.4.5",
    "react-router-dom": "^5.2.0",
    "react-scripts": "^4.0.3",
    "react-star-ratings": "^2.3.0",
    "react-token-auth": "^1.1.8",
    "sweetalert": "^2.1.2",
    "web-vitals": "^1.1.1"
  },
  "scripts": {
    "start-api": "cd api && env/bin/python3 main.py",
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```

Ilustración 48 package.json

2.2 Descripción de la funcionalidad y manual de uso

En este punto explicaré la funcionalidad de la aplicación en su estado actual junto con imágenes de que puede hacer en cada descripción. Estas imágenes varían entre tamaños de pantalla para mostrar así el responsive trabajado en la misma. **Dividiré cada acción de funcionalidad en steps (pasos).**

1º STEP Inicio

El usuario podrá en primera instancia y previo registro tener **una vista general de que podrá hacer en la aplicación a partir de un texto y un slider de imágenes.**

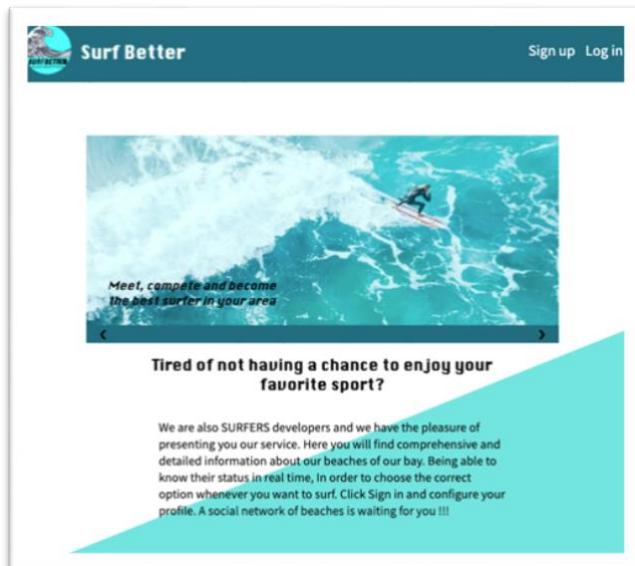


Ilustración 49 Vista inicial

2º STEP Registro

El usuario podrá acceder al registro desde el botón de la barra de navegación **Sign in**

TODOS los campos se comprueban usando componentes reactivos en cliente

Ilustración 50 Modal de registro

Ilustración 51 Mensajes a los usuarios



Ilustración 52 Mensajes a los usuarios (2)

El usuario podrá leer las condiciones previo a aceptar el registro haciendo click en el ícono ?.

Una vez se finaliza el registro la aplicación muestra una alerta dándole la bienvenida a la comunidad de SurfBetter.

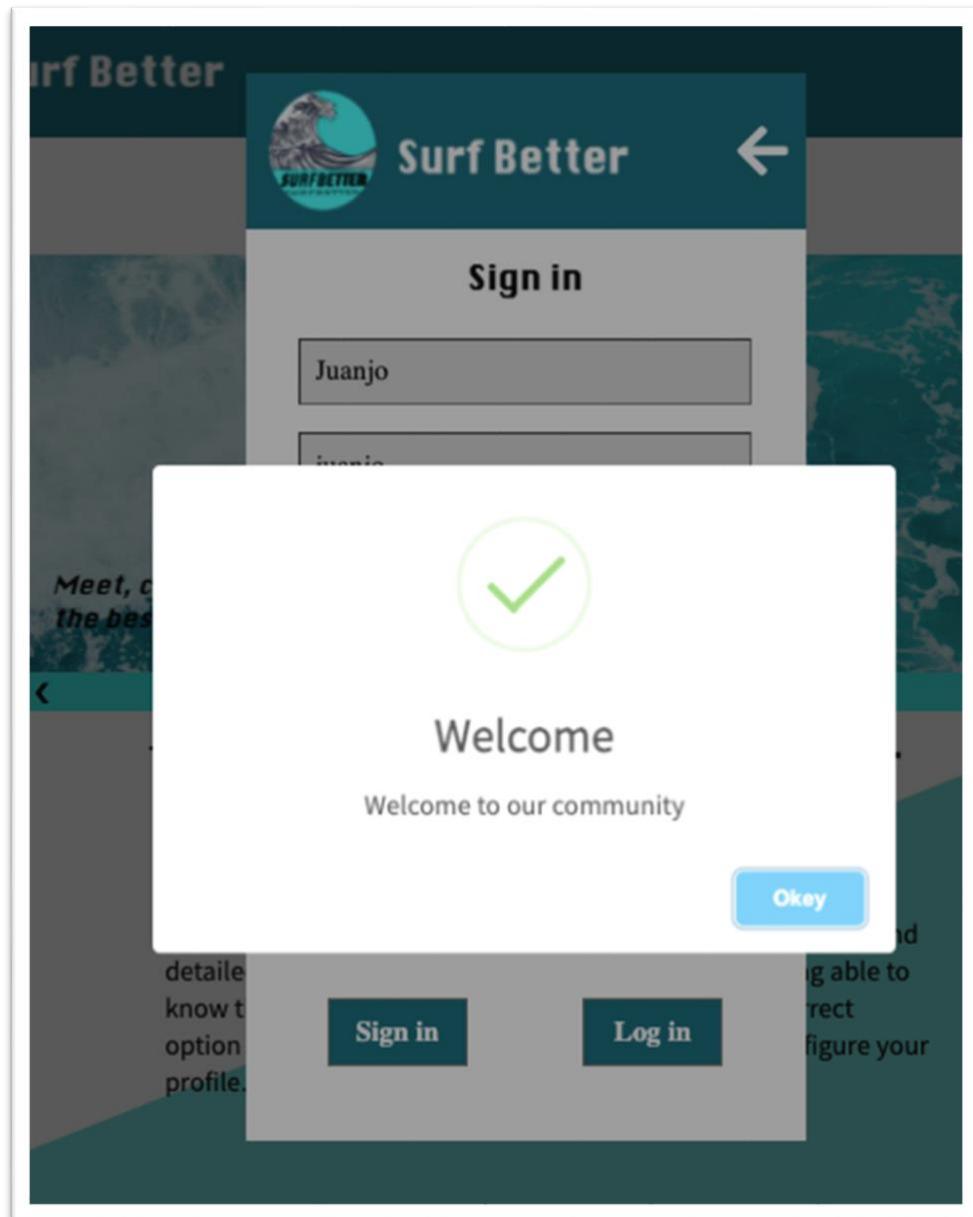


Ilustración 53 Mensaje de registro

Tras el registro la aplicación lleva al usuario a la pantalla de perfil donde tendrá una descripción e imagen de perfil por defectos. Evidentemente aún no tendrá playas asociadas a su perfil.

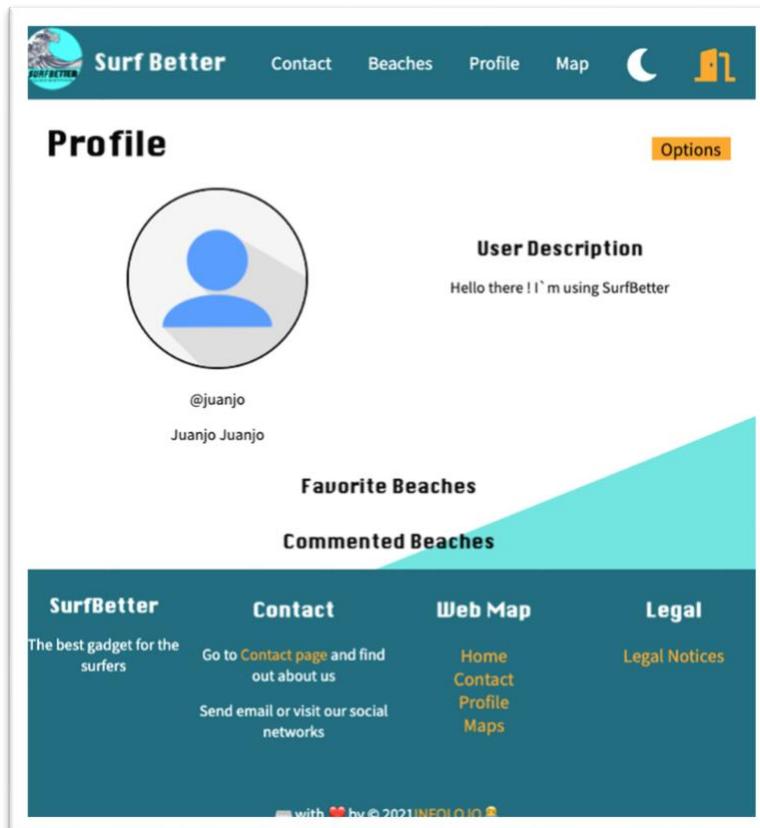


Ilustración 54 Perfil del usuario

Tras hacer el registro el servidor ha creado ya un directorio para almacenar la información del usuario con su **Nick**. El cual es único



Ilustración 55 Directorio del usuario

Al cargar esta pantalla el servidor devuelve la información del usuario a partir de la ruta /api/user, tras comprobar que este accede con su Token aleatorio devuelve en Json la información completa del usuario.

3º STEP Cierre de sesión.

Una vez dado de alta el usuario puede abandonar la sesión actual para entrar en otro momento a través del botón superior derecho de la puerta.

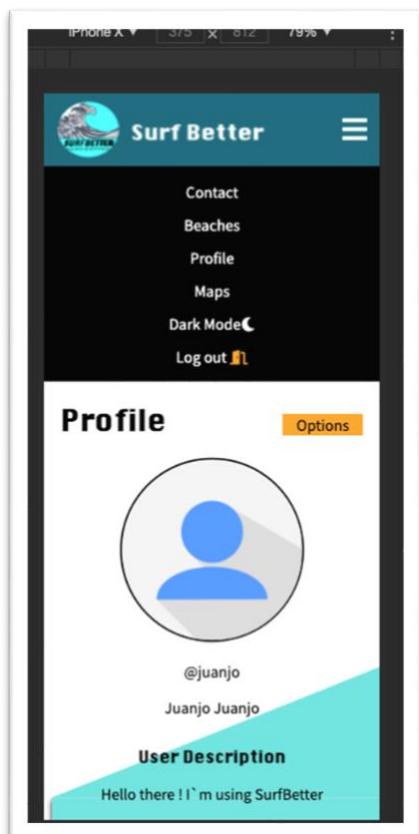


Ilustración 56 Cierre de sesión

Al tocar o hacer click la aplicación pregunta si realmente quiere abandonar en ese momento la aplicación.

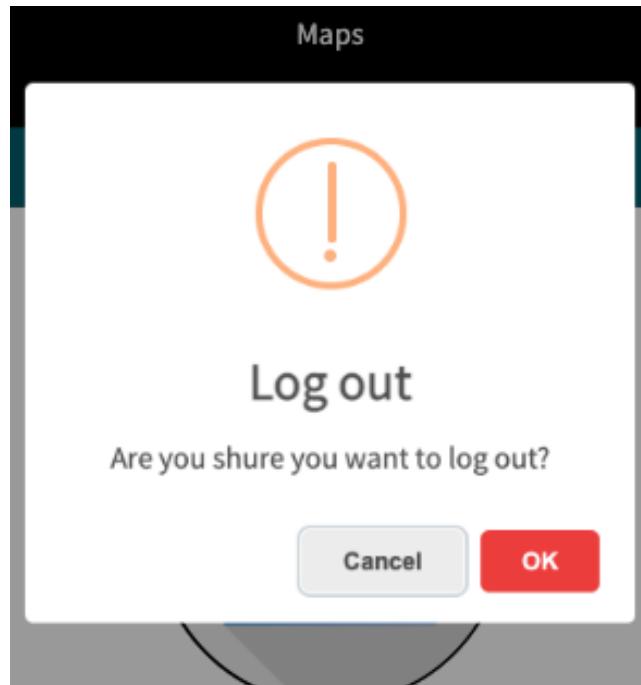


Ilustración 57 Alerta cierre de sesión

Si el usuario rechaza el cierre de sesión la aplicación muestra un mensaje agradable dándole las gracias por permanecer en la aplicación.

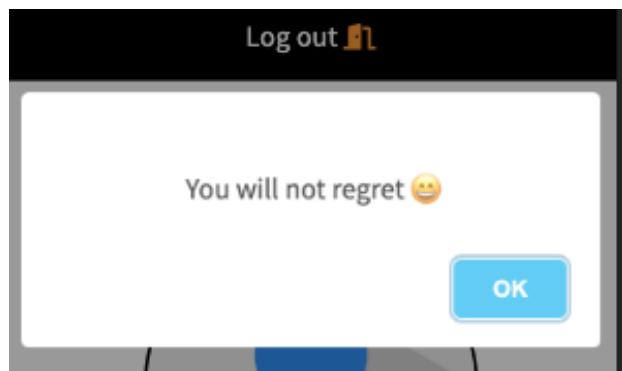


Ilustración 58 Cancelación del cierre de sesión

Si por el contrario el usuario decide abandonar la aplicación. Esta muestra un mensaje dándole las gracias por usar **SurfBetter** y elimina el Token de sesión. Al ser esta una ruta protegida se reenvía automáticamente por **Routes** a la vista principal.

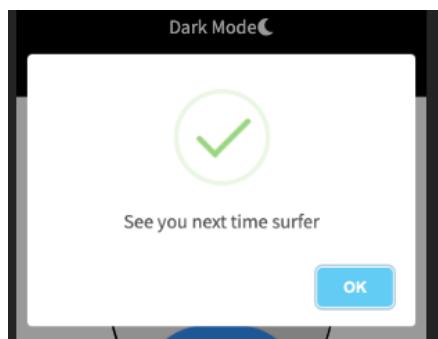


Ilustración 59 Cierre de sesión

4º STEP Inicio de sesión.

El usuario puede iniciar ahora sesión directamente sin pasar por el registro. En el caso de equivocarse el servidor podrá enviar un error que la aplicación capturará y mostrará el error correspondiente

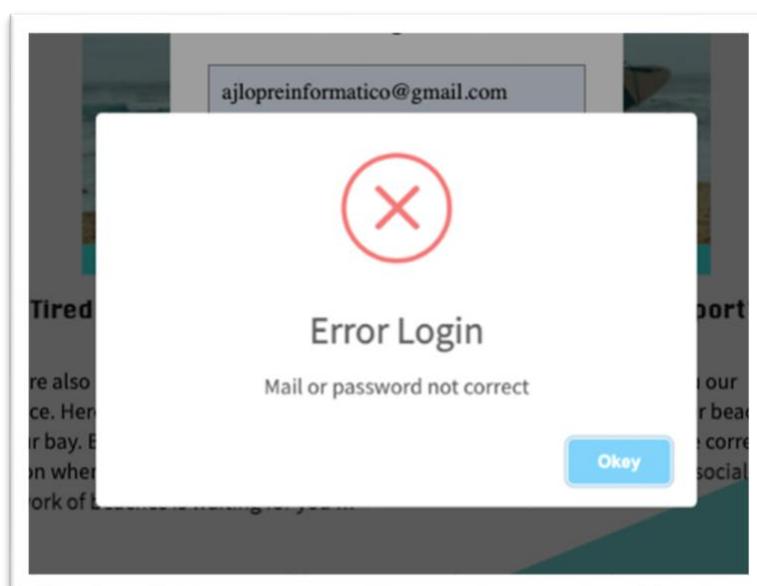


Ilustración 60 Error en el login

Tras iniciar sesión la aplicación muestra un mensaje de bienvenida al usuario

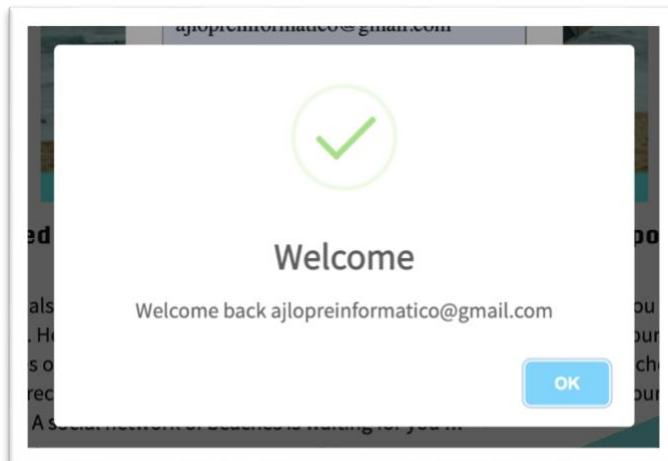


Figura 59: Mensaje de bienvenida

5º STEP Actualización de los datos de usuario

Una vez dentro de la aplicación, **podrá en primera instancia cambiar su imagen de perfil pinchando o tocando sobre la imagen**. En el caso de que esté en un teléfono podrá sacar directamente una fotografía.

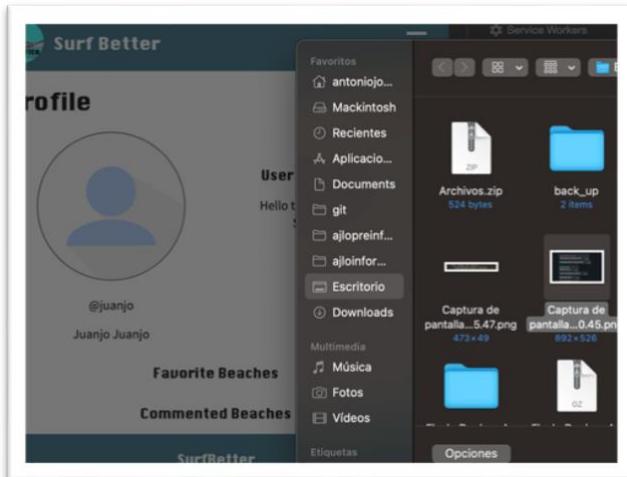


Figura 60: Actualización de la imagen de perfil

Tras actualizar la imagen de perfil del usuario la aplicación vuelve a mostrar un modal indicando que se ha actualizado correctamente y carga la foto a través de un

componente **useState** en la imagen. Internamente tras enviar la nueva imagen y almacenarla se destruye la anterior y la aplicación envía una petición al servidor para que le devuelva la imagen.

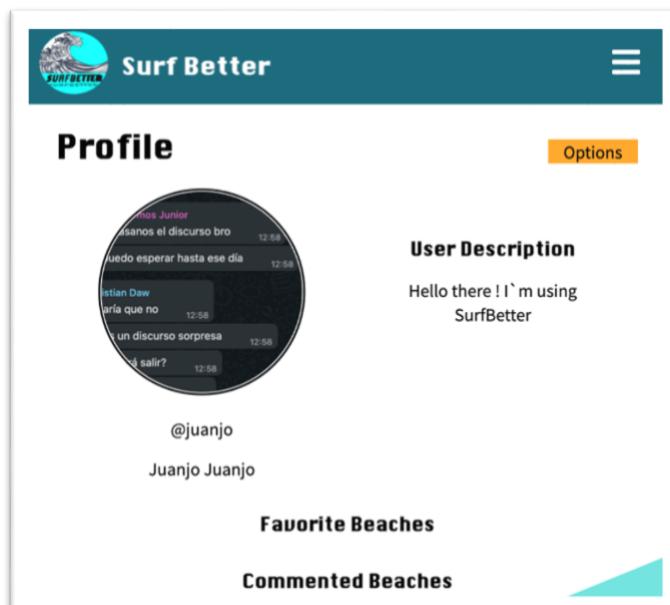


Ilustración 61 imagen de usuario actualizada

Además de esta acción el usuario puede acceder a la configuración de sus datos personales. Sea la información personal o el cambio de contraseña.

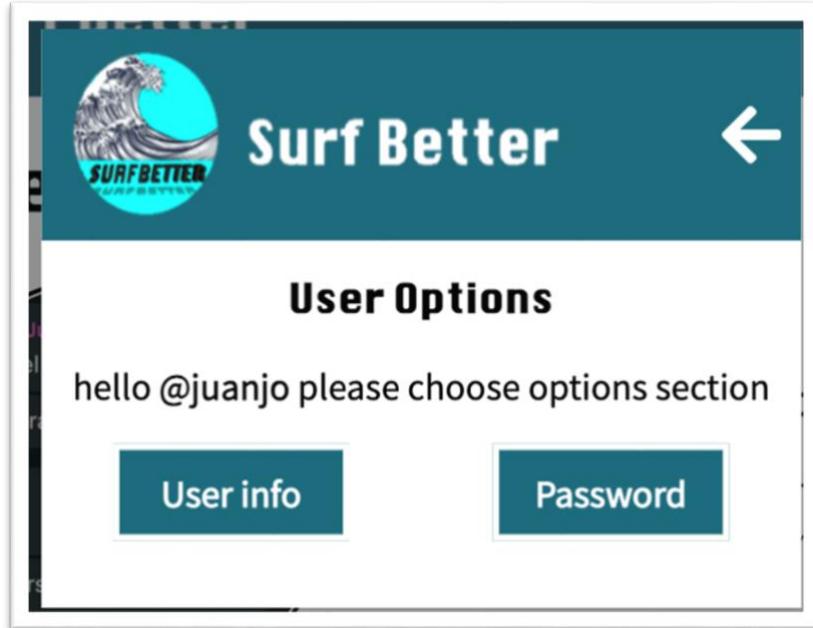


Ilustración 62 Modal de opciones

En el caso de que desee actualizar su contraseña deberá introducir la antigua. Al realizar esta acción el servidor recibe ambas contraseñas para a continuación simular un login con la antigua contraseña y si este es válido sustituye la antigua contraseña por la nueva.

POR SUPUESTO TODAS ESTAS RUTAS ESTÁN PROTEGIDAS POR JWT

Ejemplo de muestra de ruta protegida en mi servidor.

```
@routes_user.route('/api/passwordreset', methods=['PUT'])
@auth_required
def password_reset():
    """Reset password by flask_praetorian"""

    req = request.get_json(force=True)
    user = flask_praetorian.current_user()
    if req["old-password"] != req["new-password"]:
        try:
            guard.authenticate(user.email, req['old-password'])
            user.password = guard.hash_password(req["new-password"])
            db.session.commit()
            # ret = {'access_token': guard.encode_jwt_token(user)}
            return jsonify("password updated"), 200
        except Exception:
            return {"Error", "password incorrect"}, 401
    else:
        return {"Error", "password is the same"}, 401
```

Ilustración 63 Ruta flask passwordReset

Tras realizar la actualización flask envía un código 200 indicando que todo está correcto y entonces desde cliente muestro una alerta indicando que la actualización se ha realizado correctamente.

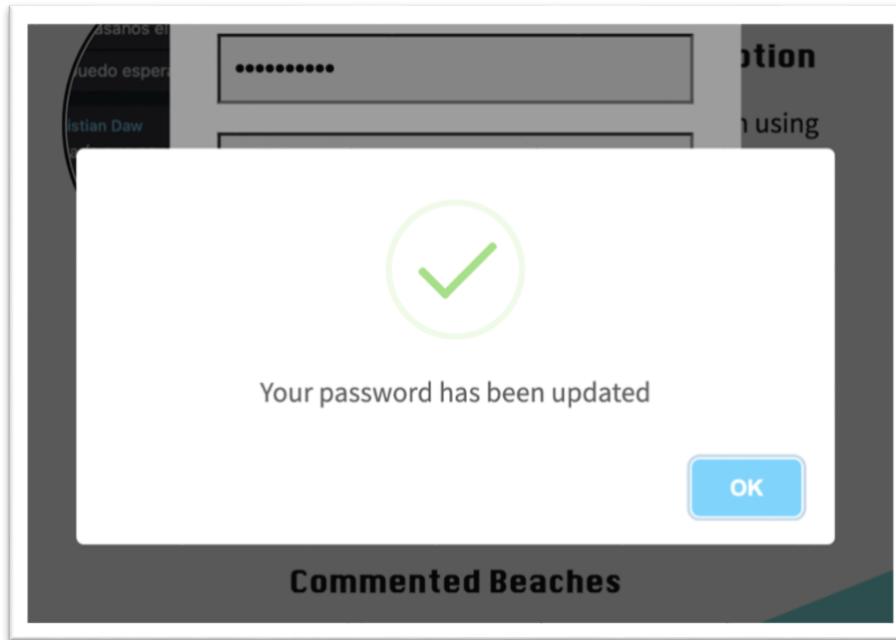


Ilustración 64 Alerta contraseña actualizada

Si el usuario se confundiese al introducir su contraseña actual la aplicación será capaz de capturar el error desde servidor y mostrar el siguiente error:



Ilustración 65 Contraseña incorrecta

A parte de actualizar su contraseña el usuario podrá también modificar sus datos personales. Incluyendo aquellos que no son obligatorios. Como su descripción. Este modal funciona exactamente igual que el del registro. Con la diferencia de que tras actualizar la aplicación pide al servidor los datos del usuario para cargarlos en su estado reactivo correspondiente. A modo de extra cuando el usuario no introduce descripción la aplicación le setera uno de ejemplo similar al estado por defecto de WhatsApp. A modo de broma indica que no sabe usar la aplicación. Por supuesto la aplicación comprueba en cliente todos los inputs. Haciendo uso de estados y de eventos onChange y onBlur.

Ejemplo:

```
const checkInputs = (e) => {
  const input = e.target; //Note: react say that its an error but its okey
  input.classList.remove("errors");
  const mailReg = new RegExp(/^(?=[\w.%+-]{1,64}@(?:[A-Z0-9-]{1,63}\.){1,125}[A-Z]{2,63}$)/i);
  switch (input.id) {
    case ("name") :
      setError(name, input, [2, 64])&&setName("");
      break;
    case ("surname") :
      setError(surname, input, [2,64])&&setSurname("");
      break;
    case ("nick") :
      setError(nick, input, [2, 30])&&setNick("");
      break;
    case ("email") :
      if (!email.trim()) {
        input.classList.add("errors");
        input.value = "";
        input.placeholder = input.id + " is empty";
        setEmail("");
      }
      else if (email.length > 130 || email.length < 4 || !mailReg.test(email)) {
        input.classList.add("errors");
        input.value = "";
        input.placeholder = input.id + " isn't valid";
        setEmail("");
      }
      break;
    case ("description") :
      if (description.length > 64) {
        input.classList.add("errors");
        input.value = "";
        input.placeholder = input.id + " isn't valid";
        setDescription("");
      }
  }
}
```

Ilustración 66 Comprobación de los datos de usuario previo insert

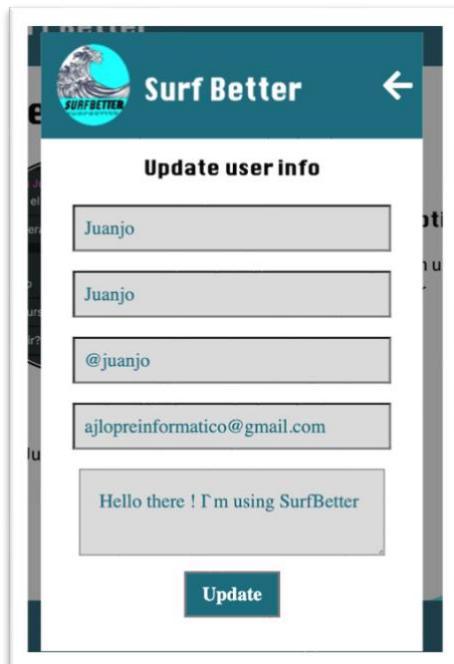


Ilustración 67 Actualización de datos del usuario.

5º STEP Búsqueda de playa

La siguiente acción inmediata que el usuario puede hacer es entrar a **Beaches**. Esto puede hacerlo desde el menú superior o inferior.

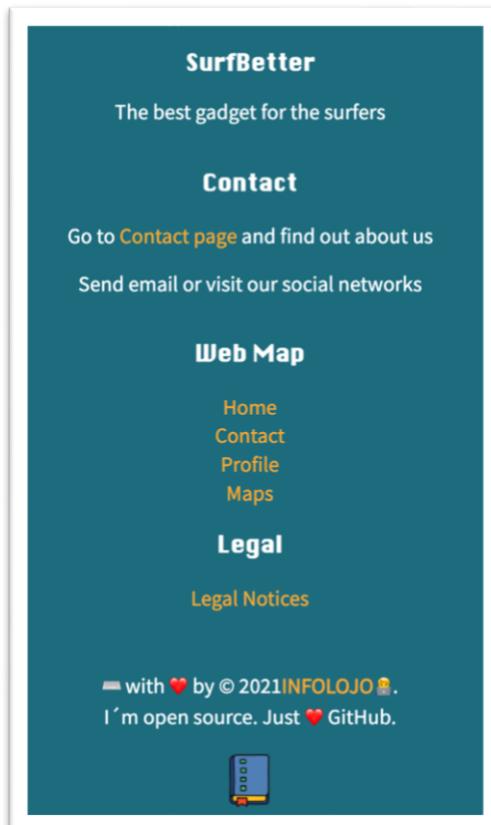


Ilustración 68 Accediendo a beaches

Figura 68: Accediendo a beaches

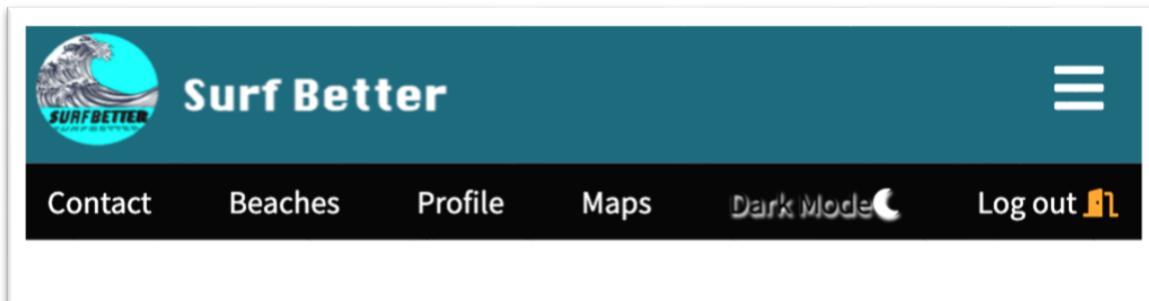


Ilustración 69 Accediendo a beaches (2)

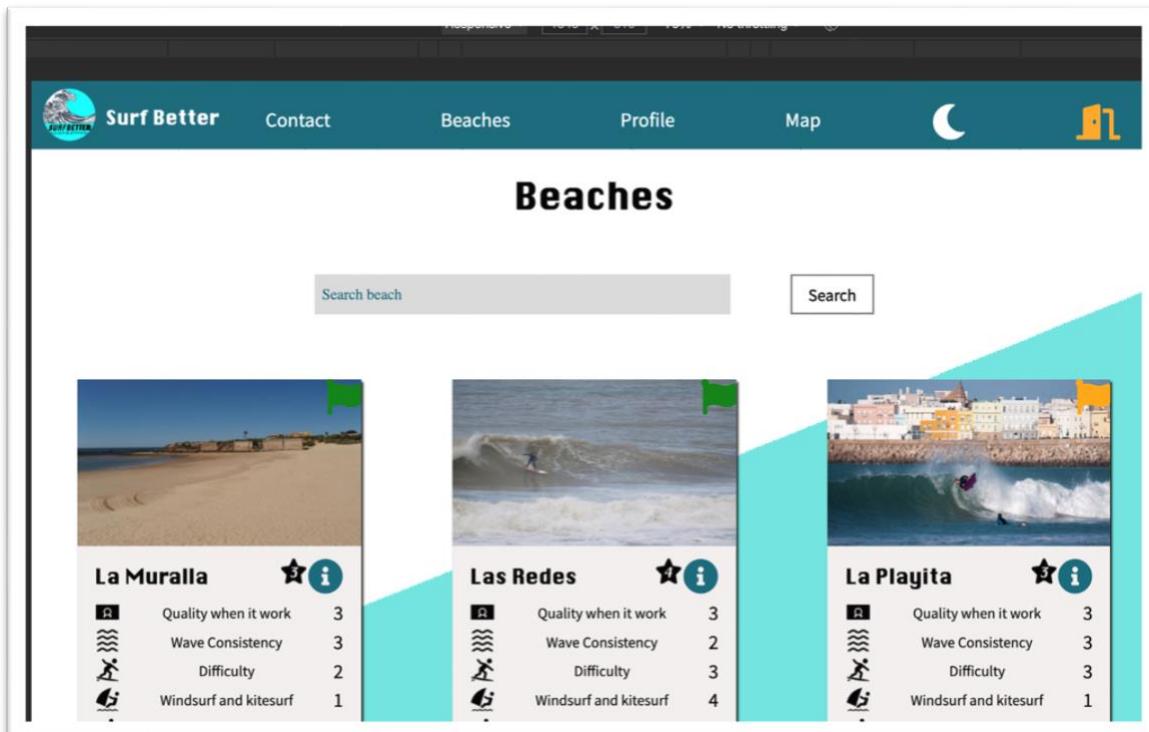


Ilustración 70 Vista de playas

Tras acceder dispondrá de un botón de búsqueda para ahorrarse el tener que mirar entre todas las playas.

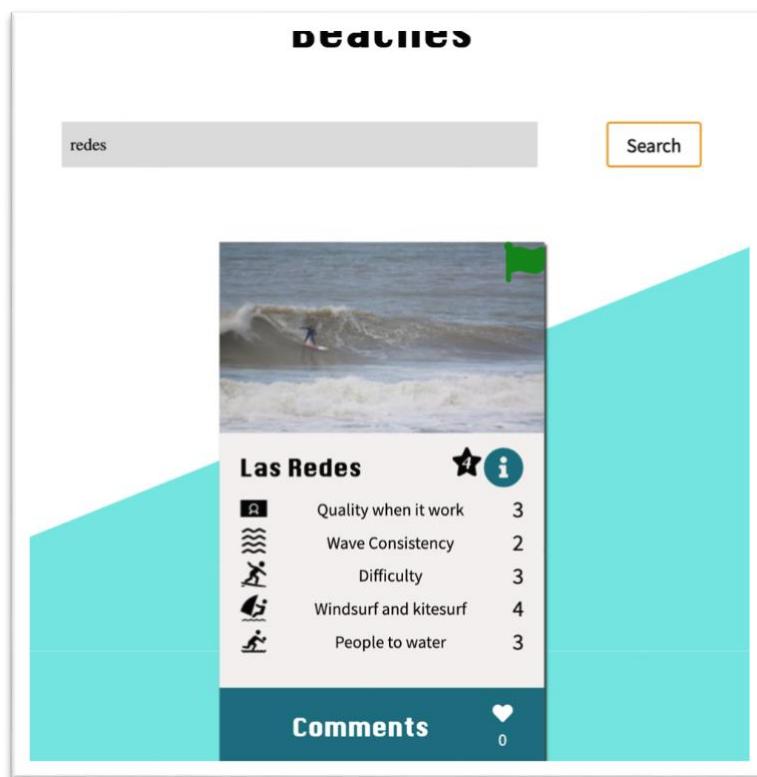


Ilustración 71 Beach Carta de la playa

6º STEP Comprobar los comentarios.

Una vez encontrada la playa la aplicación permite al usuario comprobar los comentarios de dicha playa. Para ello puede clicar en el botón **comment** situado en la parte inferior de la tarjeta.

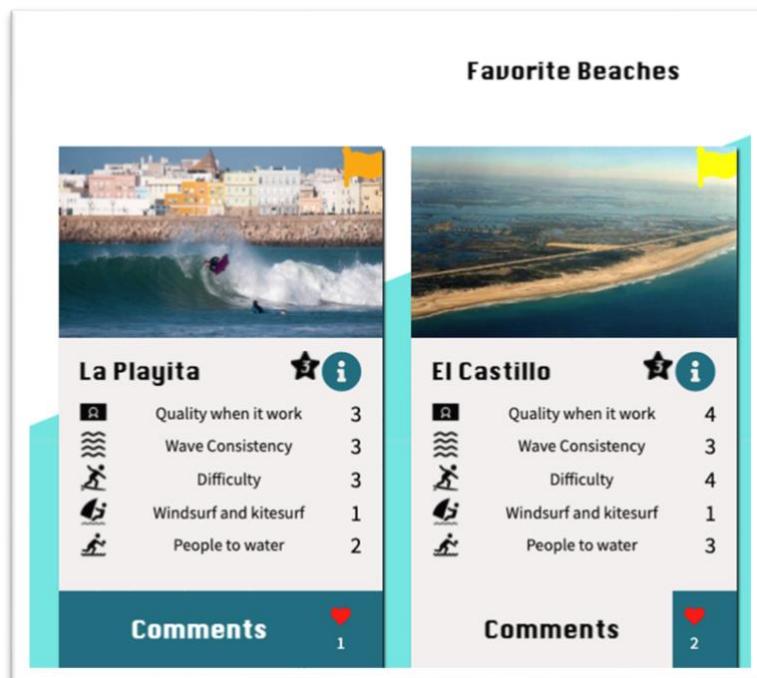


Ilustración 72 Botón comment

Tras pulsar el botón se muestran los comentarios junto al botón de like sobre un comentario, así como un botón para eliminar comentario en el caso de que sea el usuario el **propietario del comentario**

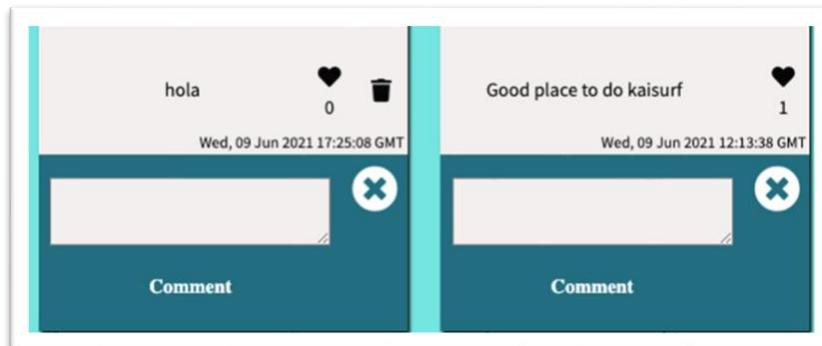
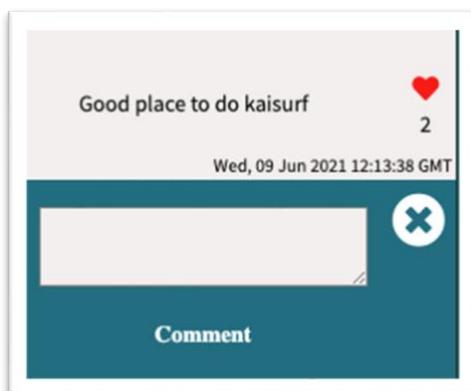


Ilustración 73 Imágenes de comentarios

7º STEP: Actuar sobre los comentarios

El usuario puede interactuar con los comentarios directamente. Bien dejando un like, quitándolo o eliminándolo en el caso de que suyo.



En el caso **de que el like sea de la propiedad del usuario el color del like será rojo**

Si por el contrario no le ha dado a **like**. O hace click de nuevo en el **like** el color será rojo y el contador variará

Ilustración 74 Like

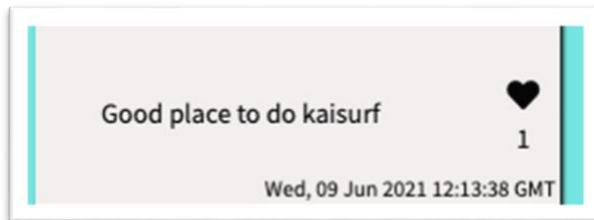


Ilustración 75 dislike

Si el comentario fuese del propietario podrá hacer click sobre la **papelera**. Al hacer **esto se mostrará una alerta** preguntando al usuario **si está seguro de que desea eliminar** dicho comentario.



Ilustración 76 Eliminar comentario.



Ilustración 77 comentario eliminado

Para **cerrar un comentario y volver a la vista anterior** de la parte inferior de la carta basta **con hacer click sobre el icono de la x**.

8º STEP: Añadir like a una playa

En la vista previa a los comentarios el usuario podrá darle a like a una de las playas. Al hacerlo cambiará el color del corazón a rojo

Beach	Rating	Quality when it work	Wave Consistency	Difficulty	Windsurf and kitesurf	People to water	Comments
La Muralla	★ ⓘ	3	3	2	1	4	1
Las Redes	★ ⓘ	3	2	3	4	3	0
La Playita	★ ⓘ	3	3	3	1	2	1

Ilustración 78 Likes en las playas

9º STEP Contenido del Perfil

Además de poder actualizar su información personal el usuario tiene una vista de sus playas favoritas como de sus playas comentadas.

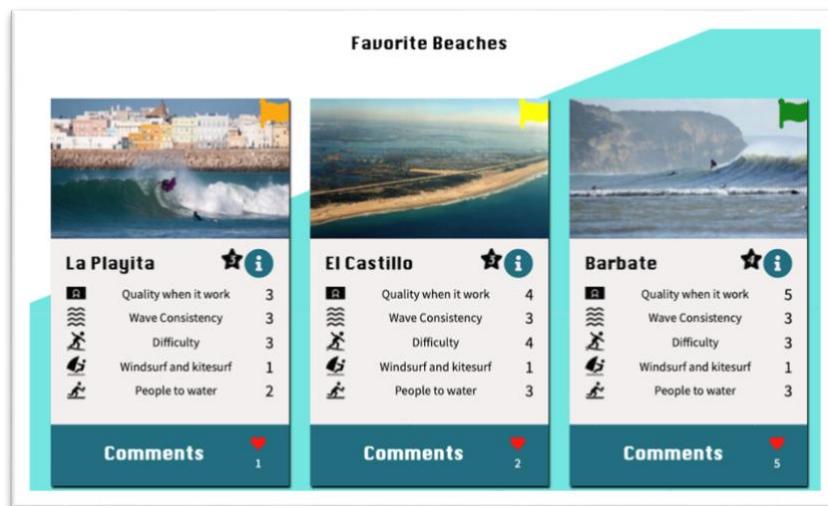


Ilustración 79 Playas favoritas

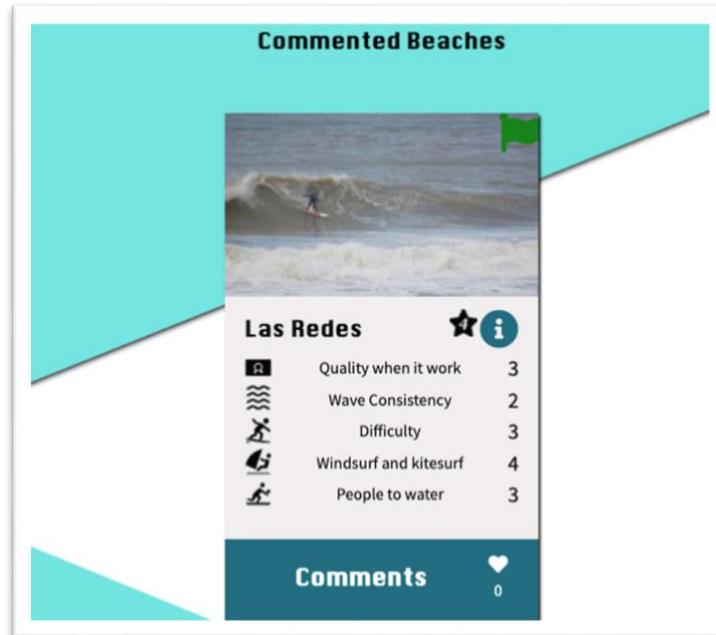


Ilustración 80 playas comentadas

10º STEP Acceso al mapa de playa

El usuario puede acceder a el **mapa de playas** desde la barra de navegación superior o desde el footer. (**Desde cualquiera de estos dos componentes puede acceder a cualquier parte de la aplicación**).



Ilustración 81 Footer

En el mapa el usuario tiene una vista **de todas las playas de la bahía de Cádiz**

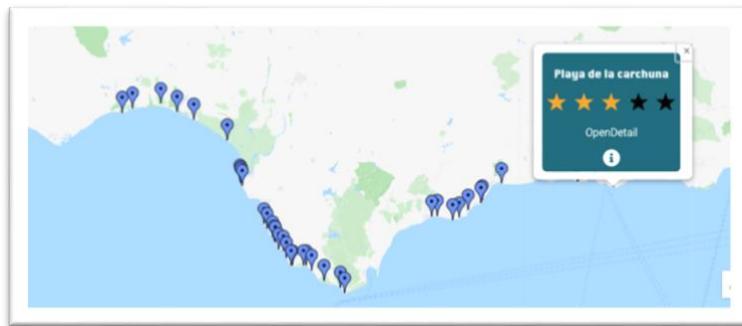


Ilustración 82 Mapa

11º STEP Acciones sobre el mapa

El usuario puede interactuar de varias maneras sobre el mapa. Puede en primera instancia **cambiar el estilo del mapa**



Ilustración 83 Mapa satélite

Puede **ampliar la vista a pantalla completa**. Haciendo click sobre el ícono de la esquina superior derecha.



Ilustración 84 Mapa pantalla completa

Haciendo uso de los botones laterales + - o bien con el ratón o dedo puede moverse libremente por el mapa. Y por último tiene disponible la opción de Street View. Opción muy interesante que presta google con su api para ver las playas in situ.

Aparte de la propia funcionalidad dada por la propia **api**. Al pasar el ratón sobre cualquiera de los **markers** se muestra un dialogo de información. Haciendo click sobre el se accede al detalle de la playa.

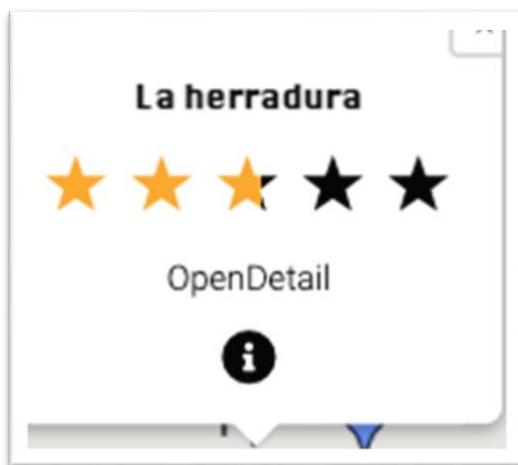


Ilustración 85 Dialogo de información.

11º STEP Acciones sobre el mapa

En el detalle de la playa el usuario puede visualizar una imagen de la playa, descripción y los datos moqueados (**más adelante por scraping**). En la parte final del detalle el usuario puede dejar un comentario visualizar todos los comentarios de la misma manera que en las tarjetas pero con una vista diferente.

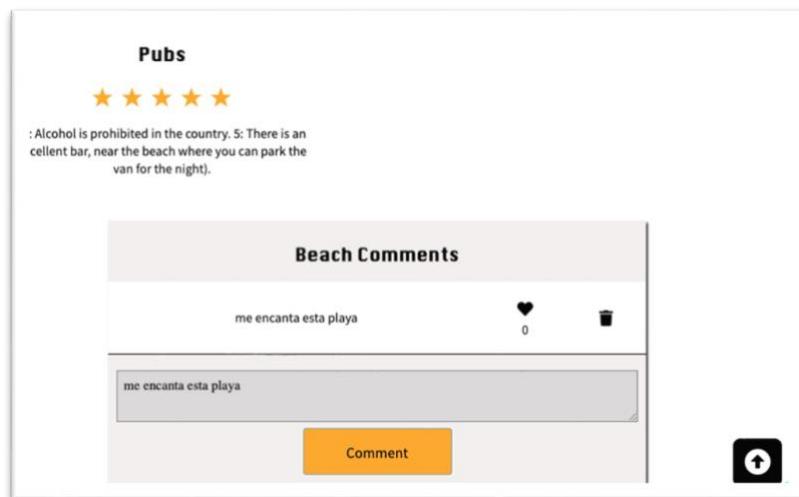


Ilustración 86 Comentarios del detalle

En la zona superior se muestra un botón **back**. Este cambia su funcionalidad dependiendo del comente desde el que se llame.

Ejemplo: Si se llama desde el mapa vuelve al mapa, si es desde el componente de playas vuelve a dicho componente.

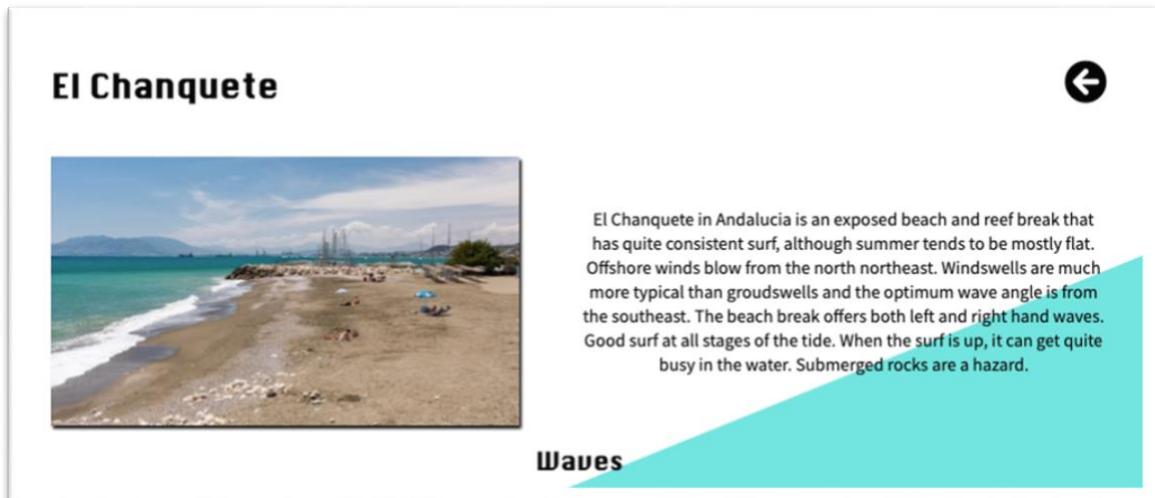


Ilustración 87 Botón de vuelta

12º STEP Acceso a contacto

El usuario puede acceder a **contacto** de la misma forma que con el resto de componentes. En este puede ver una vista en la que en primera instancia se muestra información de mi proyecto personal **infolojo**



Ilustración 88 Contacto (1)

Pulsando sobre este botón se puede acceder a la web de [infoarlo](http://infoarlo.es)

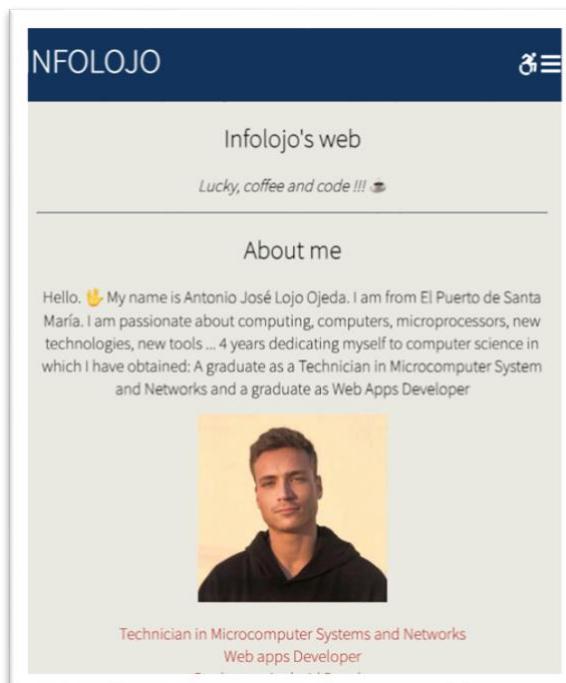


Ilustración 89 Infoarlo

En la zona inferior el usuario puede ver y acceder a los enlaces de interés de los integrantes del grupo **infoarlo**.

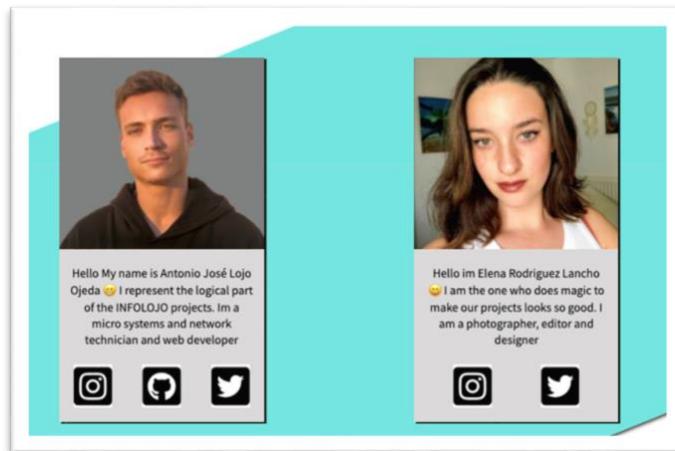


Ilustración 90 Integrantes

Por último, en la zona inferior y solo en este componente se incluye un formulario de correo electrónico donde a través del servicio **Gmail_service.py** implementado en **flask**. El usuario puede enviar un correo electrónico gestionado a través del correo "empresarial" surfbetter@gmail.com.

Al enviar el correo la aplicación muestra una alerta indicando que el envío se ha realizado correctamente. Si esto no fuese así la aplicación mostraría un mensaje de error.



Ilustración 91 Contacto Footer

13º STEP Temas

El usuario además puede cambiar el tema a gusto propio desde el botón **darkMode/lightMode**. Este está disponible en todo momento en el comente del menú.

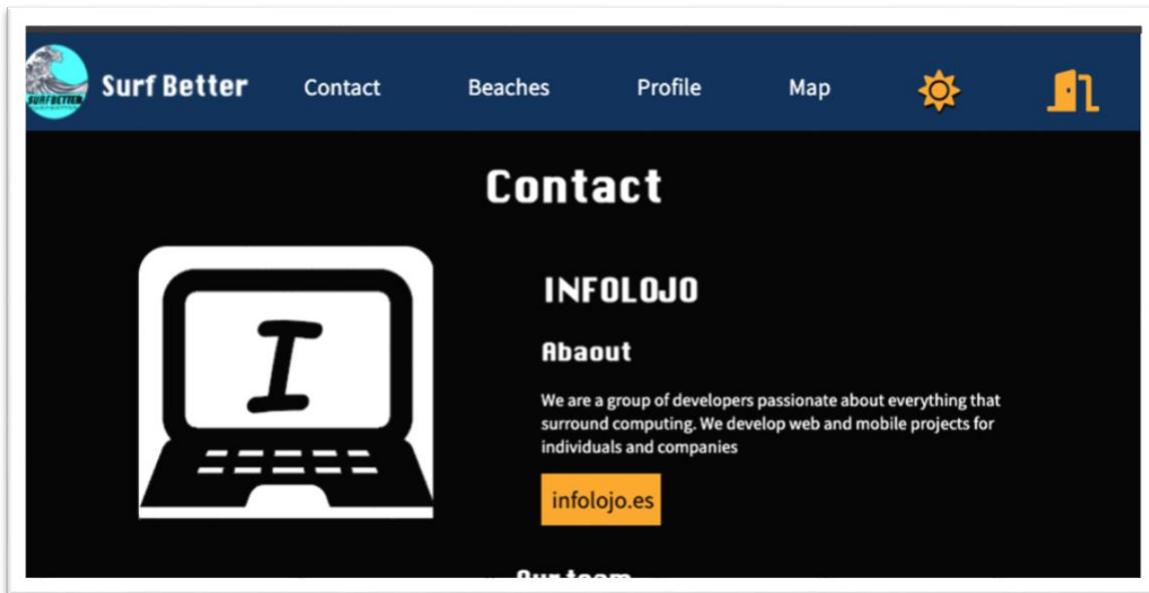


Ilustración 92 Modo oscuro activado

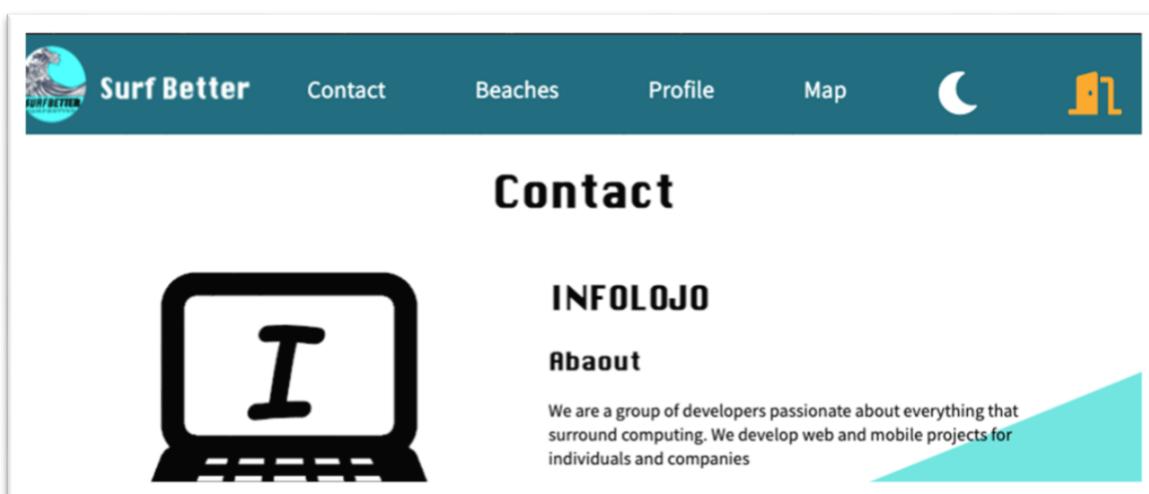


Ilustración 93 Modo light activado

El tema se actualiza de manera dinámica. Usando una variable del navegador. Por supuesto destacar el responsive en toda la aplicación.

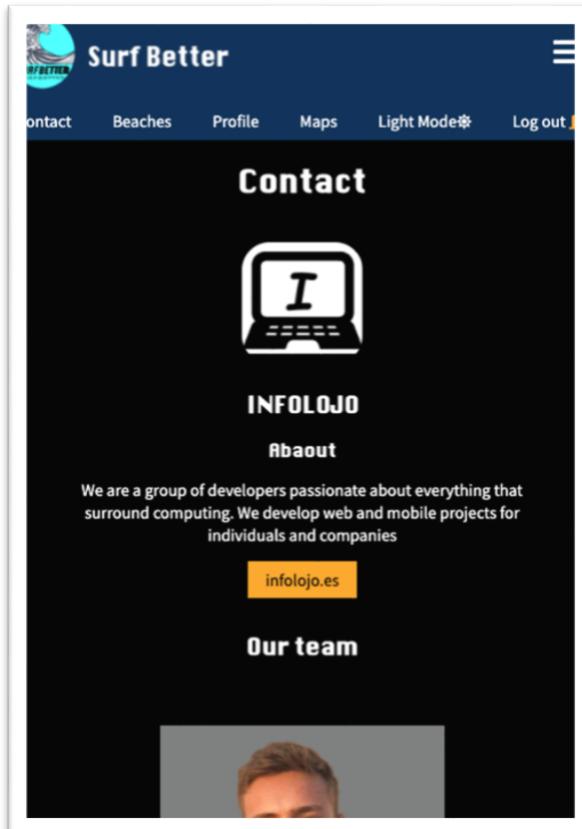


Ilustración 94 Modo oscuro activado (2)

14º STEP Errores.

Como se ha comentado anteriormente si hubiese un fallo en alguna de las peticiones la aplicación captura dicho error y muestra un mensaje de información. **Ejemplo**

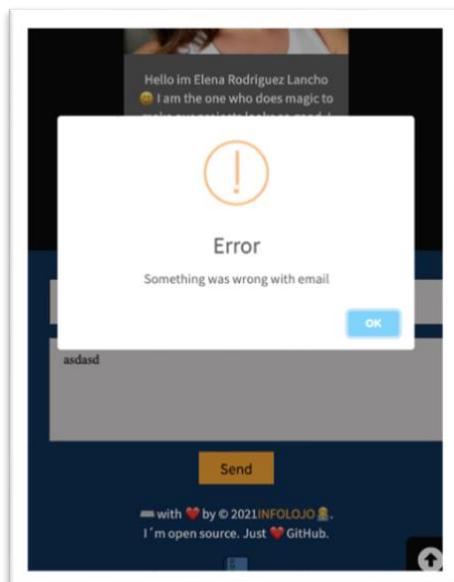


Ilustración 95 Error específico

Si el **error fuese global** la aplicación (**Cliente no encuentra el servidor**) mostraría al usuario un dialogo y no cargaría el contenido. Esto se da en todas las rutas protegidas.

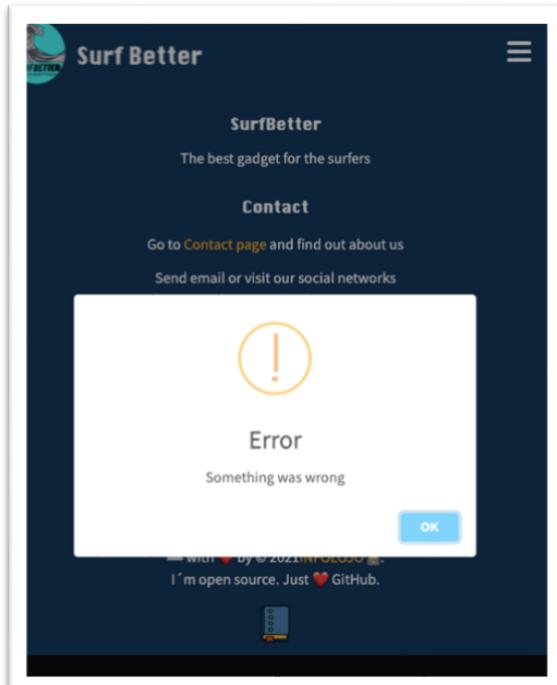


Ilustración 96 Error global

3. Instalación y preparación.

La intención inicial era tener preparados 3 deploy que varían según el estado del proyecto. (**React-server**, **Docker-compose**, **Heroku**). Por razones de tiempo el último estará listo en futuras versiones de la aplicación.

¿Dónde encuentro la app?

La aplicación está alojada en un [repositorio de GitHub](#), es abierta y está accesible para todo el mundo. Para descargar el contenido **basta con clonar el repositorio**

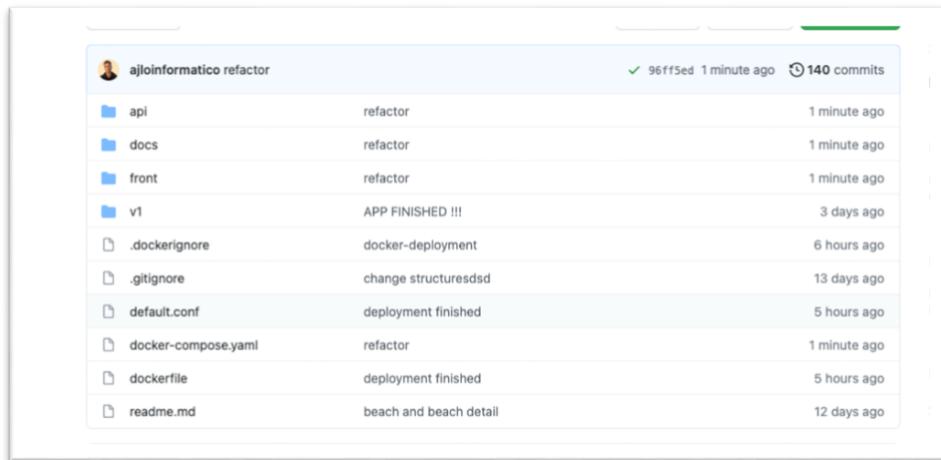


Ilustración 97 Repositorio

Link del repositorio: <https://github.com/ajloinformatico/SurfBetter>

Una vez descargado el usuario puede elegir entre dos tipos de instalaciones.

Tipos de Instalación

1º Desarrollo

Este punto requiere algunos pasos extras pero son bastante simples.

1º Preparación del entorno virtual

El primer paso será acceder a **/api** aquí se aloja el contenido del servidor. A continuación, se deben instalar los requisitos del archivo **apirequirements.py**.

Nota: Se recomienda utilizar un entorno virtual (venv)

En **/api** :

Instalación: **python3 -m venv env**

Activación: **source activate env/bin/activate**

Para instalar los requerimientos se debe ejecutar en **/api**

pip3 install -r apirequirements.txt

Una vez listo. Vamos a **/front**. Y desde aquí ejecutamos los comandos configurados por mí en **npm** para trabajar en el desarrollo

```
(env) MBP-de-Antonio:surfBetter antoniojoselojoojeda$ ls
api           docker-compose.yaml   docs          README.md
default.conf   dockerfile         front        v1
(env) MBP-de-Antonio:surfBetter antoniojoselojoojeda$ cd front
```

Ilustración 98 Deploy (1)

npm run start & npm run start-api

Ilustración 99 Deploy React

```
MBP-de-ANTONIO:front antoniojoseloojeda$ npm run start-api
> frontend@0.1.0 start-api
> cd ../api && env/bin/python3 main.py

* Tip: There are .env or .flaskenv files present. Do "pip install python-dotenv" to use them.
* Serving Flask app "main" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://192.168.43.130:5000/ (Press CTRL+C to quit)
```

Ilustración 100 Deploy Flask

Mediante estos dos comandos definidos en **Package.json** Se arranca React en el puerto **3000** y **Flask** arranca desde **main.py** en **/api** desde el puerto **5000**

Nota: **Flask solo arranca desde main en modo desarrollo.** En producción (**docker**) El servidor se arranca con **Gunicorn** desde **wsgy.py** Donde se guarda una instancia de la api que es usada por **Gunicorn** para hacer de pasarela

Podrás acceder a la aplicación desde :

<http://localhost:3000/>

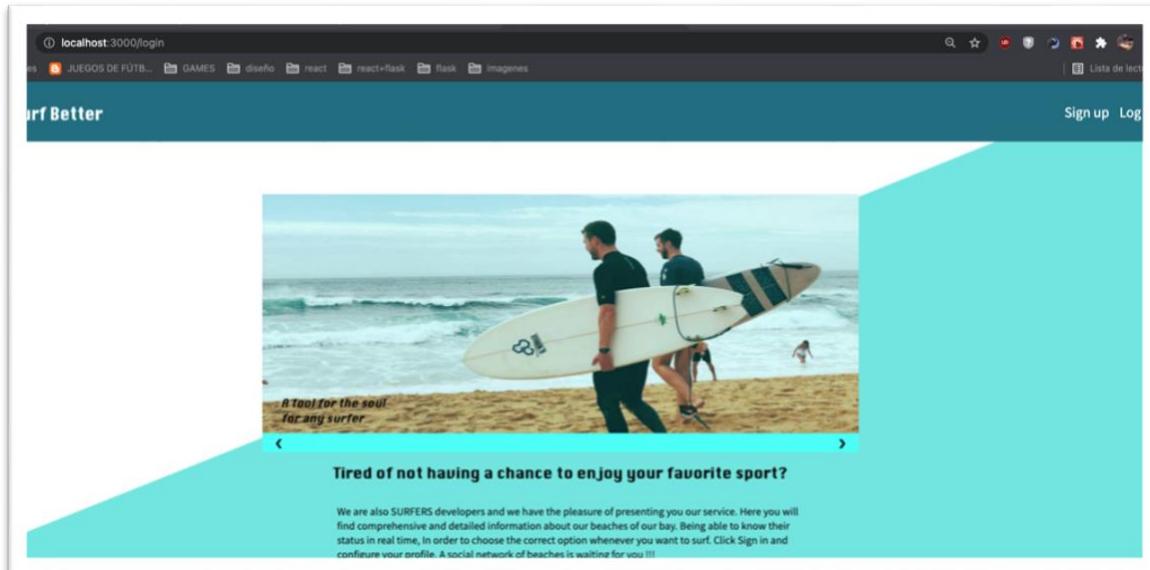


Ilustración 101 Deploy desarrollo

El servidor entra en funcionamiento desde <http://localhost:5000/api>

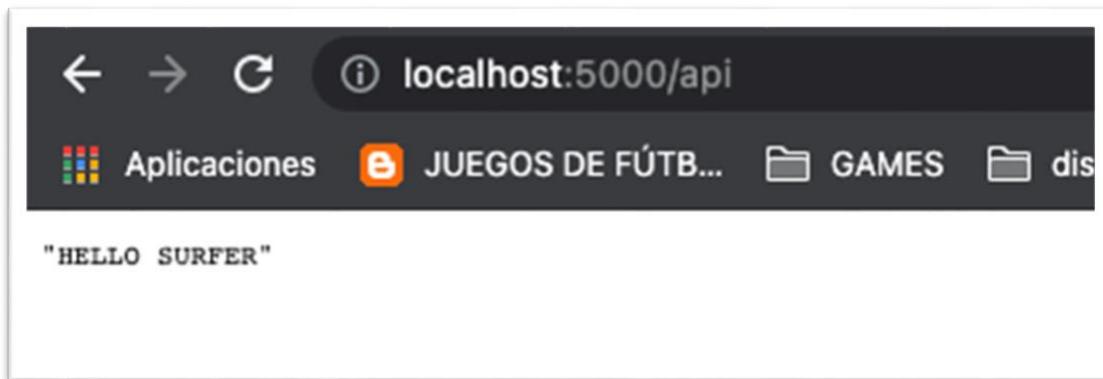


Ilustración 102 Api en desarrollo

En el repo dentro de /doc se encuentra un Json con todas los entrypoints

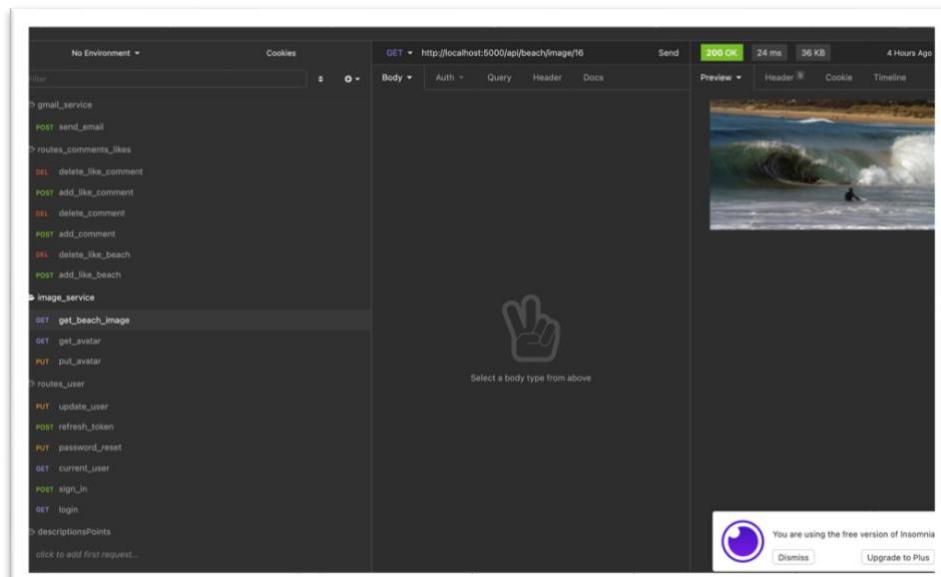


Ilustración 103 api Insomniac

2º Producción Local

Además de la puesta en escena en modo desarrollo. Es posible poner el proyecto en producción haciendo uso de docker y con tan solo dos comandos

Arranque

El único requisito es tener instalado docker en la máquina. A continuación ejecutar **Docker-compose & docker-build**.

```
MBP-de-Antonio:front antoniojoseloojeda$ docker-compose build
Building api
[+] Building 4.6s (8/9)
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 37B
 => [internal] load .dockerignore
 => => transferring context: 348
 => [internal] load metadata for docker.io/library/python:3
 => [internal] load build context
 => => transferring context: 14.67kB
 => [1/5] FROM docker.io/library/python:3@sha256:e8e000f3c551f93d7b03d241e38c1206eb8c8a1f1a6179902f74e068fc98ee59
 => CACHED [2/5] RUN mkdir /api
 => CACHED [3/5] WORKDIR /api
 => CACHED [4/5] COPY * .
=> [5/5] RUN pip3 install -r apirequirements.txt
=> => # Collecting flask
=> => #   Downloading Flask-2.0.1-py3-none-any.whl (94 kB)
=> => # Collecting flask_sqlAlchemy
```

Ilustración 104 docker-compose build

```
MBP-de-Antonio:front antoniojoseloojeda$ docker-compose up
Docker Compose is now in the Docker CLI, try `docker compose up`

Recreating surfbetter_api_1 ... done
Recreating surfbetter_front_1 ... done
Attaching to surfbetter_api_1, surfbetter_front_1
api_1  | [2021-06-09 21:22:03 +0000] [1]  [INFO] Starting gunicorn 20.1.0
api_1  | [2021-06-09 21:22:03 +0000] [1]  [INFO] Listening at: http://0.0.0.0:5000 (1)
api_1  | [2021-06-09 21:22:03 +0000] [1]  [INFO] Using worker: sync
api_1  | [2021-06-09 21:22:03 +0000] [8]  [INFO] Booting worker with pid: 8
front_1 | 2021/06/09 21:22:06 [notice] 1#1: using the "epoll" event method
front_1 | 2021/06/09 21:22:06 [notice] 1#1: nginx/1.21.0
front_1 | 2021/06/09 21:22:06 [notice] 1#1: built by gcc 10.2.1 20201203 (Alpine 10.2.1_pre1)
front_1 | 2021/06/09 21:22:06 [notice] 1#1: OS: Linux 5.10.25-linuxkit
front_1 | 2021/06/09 21:22:06 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
front_1 | 2021/06/09 21:22:06 [notice] 1#1: start worker processes
front_1 | 2021/06/09 21:22:06 [notice] 1#1: start worker process 8
front_1 | 2021/06/09 21:22:06 [notice] 1#1: start worker process 9
front_1 | 2021/06/09 21:22:06 [notice] 1#1: start worker process 10
front_1 | 2021/06/09 21:22:06 [notice] 1#1: start worker process 11
```

Ilustración 105 docker-compose up

Listo puedes acceder desde: <http://localhost:80> o directamente localhost

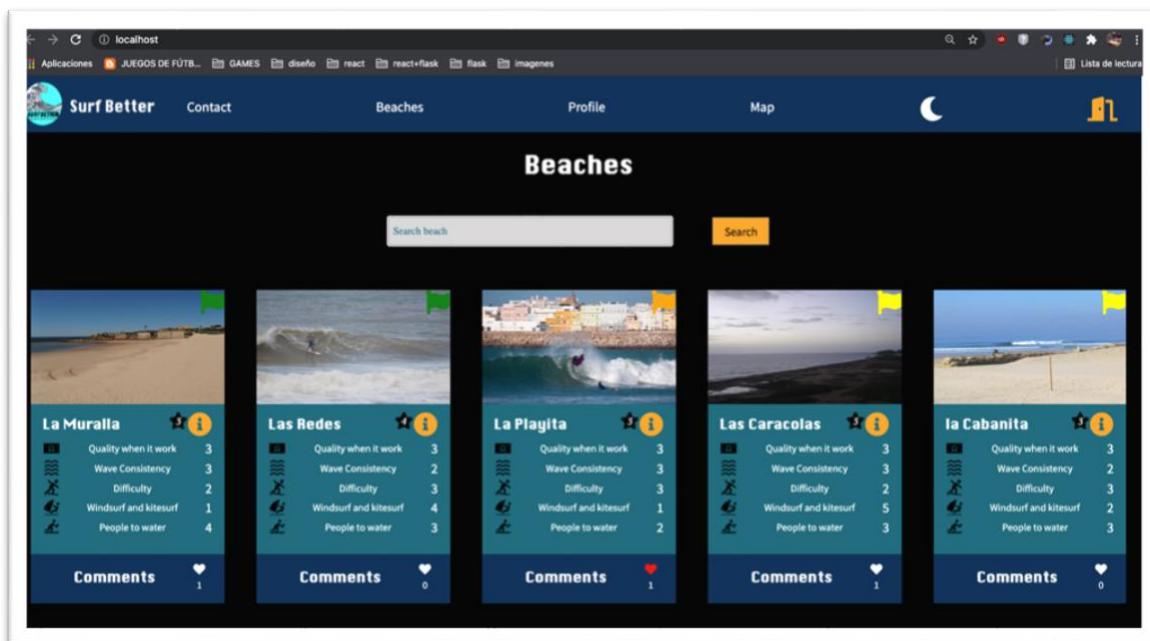
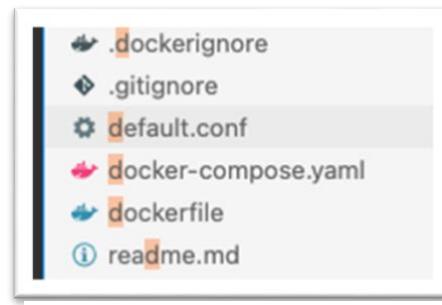


Ilustración 106 docker-compose build

¿Cómo está configurado docker?

El deploy está preparado para funcionar con dos contenedores. Uno para front con **node** y **nginx** y un segundo para **flask** y **Gunicorn**. La configuración incluye

4 archivos en la raíz.



107 docker (1)

.dockerfile → Se incluye el contenido que no quiero omitir

docker_compose.yaml → Se incluyen los dos contenedores con los que se hace el build

```

1  version: "3.9"
2  services:
3    api:
4      build: ./api
5      ports:
6        - "5000:5000"
7      volumes:
8        - ./api:/api
9      environment:
10        FLASK_ENV: build
11
12    front:
13      build: .
14      ports:
15        - "80:80"
16      volumes:
17        - ./front:/app
18      depends_on:
19        - api

```

Ilustración 108 docker (1)

Default.conf → Se incluye la configuración del servidor **Nginx** para el cliente. Este necesario ya que se debe definir mediante location root que el encargado de enrutar es **React**. Esta referencia la encontré en **stack overflow**. Por ello indico el enlace

```

server {
  listen      80;
  server_name localhost;

  location / {
    root   /usr/share/nginx/html;
    try_files $uri /index.html; # https://stackoverflow.com/questions/43951720/react-router-and-nginx#answer-43954597
  }

  location /api {
    proxy_pass http://localhost:5000;
  }

  error_page  500 502 503 504  /50x.html;
  location = /50x.html {
    root   /usr/share/nginx/html;
  }
}

```

Ilustración 109 default.conf

Dockerfile → Este incluye el **build** del cliente: En el se utilizan dos imágenes una de **Node** donde se ejecuta un **install** y un **build** para instalar las dependencias del proyecto y compilar el código y **Nginx** que copia la configuración de **default.conf** y sustituye su contenido por defecto por el compilado por **Node** (básicamente elimina y copia)

```

👉 dockerfile > ...
1  # Node image on builder
2  FROM node:10 AS builder
3  # Set work directory
4  WORKDIR /app
5  # Copy project into ../front
6  COPY /front .
7  # Install and build project
8  RUN yarn install && yarn add react-token-auth
9  # Dont know why it continue fail
10 RUN yarn build
11
12 # Nginx
13 FROM nginx:alpine
14 # SET WORKING DIRECTORY for Nginx
15 WORKDIR /usr/share/nginx/html
16 # Remove default nginx statics
17 RUN rm -rf /*
18 # Copy static assets from builder
19 COPY --from=builder /app/build .
20 # Copy conf to nginx
21 COPY default.conf /etc/nginx/conf.d/default.conf
22 # Containers run nginx with global directives
23 ENTRYPOINT ["nginx", "-g", "daemon off;"]
24
25
26

```

Ilustración 110 dockerfile cliente

Y en **/api** Se incluyen dos 3 archivos extras (incluyendo apirequirements.txt). Importantísimo comentarlo para mí ya que hace de **package.json** de **python**. En el se incluyen todas las dependencias que se han utilizado en la api.

.dockerignore → archivos a ignorar

apirequirements.txt → dependencias del servidor

```

api >  apirequirements.txt
1  flask
2  flask_sqlAlchemy
3  flask_cors
4  flask-praetorian
5  gunicorn
6  python-dotenv
7  python-decouple

```

Ilustración 111 apirequirements.txt

dockerfile → build de python. En el se establece un directorio base sobre el que trabajar **api** por mantener la coherencia con mi distribución. A continuación, se copia todo el código de **/api** a el directorio predeterminado. A continuación, se instala el contenido de **apirequirements.txt** y se llama a **wsgy.py** con **Gunicorn** para arrancar el servidor por el puerto **5000**. Puerto escogido también para mantener la coherencia.

```
api > 🐳 Dockerfile > ...
1  # Flask
2  FROM python:3
3  # Create api directory
4  RUN mkdir /api
5  # Set work directory
6  WORKDIR /api
7  # Copy API data
8  COPY * .
9  # Run Api dependency
10 RUN pip3 install -r apirequirements.txt
11 # Expose port
12 EXPOSE 5000
13 # Run flask by gunicorn server
14 CMD ["gunicorn" , "-b", "0.0.0.0:5000", "wsgi:app"]
15
```

Ilustración 112 Dockerfile servidor

4. Guía de estilos y prototipado

1. Guía de estilos

Se ha tratado de seguir una guía de estilos propia inspirada en la intencionalidad de la aplicación. Dado que esta está dedicada a **surfistas**. Se ha tratado de escoger una paleta de colores relacionados con el mar y la playa. Por ello predominan los azules así como los amarillos suaves. Ambos colores relacionados con el agua y la arena respectivamente. Los colores de la aplicación fueron los siguientes:

Para el **cambio de tema** se buscó de manera directa el cambio de contraste respetando la idea base de usar azules y amarillos

```

t
; /*Colors*/
; $color-header: #16697A;
; $color-footer: #0f3057;
; $color-yellow: #FFA62B;
; $color-font: dark;
; $color-white: white;
; $color-black: black;
; $color-red: red;
; $color-little-red: #a03535;
; $color-blue-content: #3EFDF6;
; $color-back-beaches: #EFEDEC;
; $color-gray-comment-form: #d7d7d7;
; $color-background-beach-form: #6FE2DD;
; $color-background-modal-opacity: RGBA(0,0,0,0.45);
; $color-background-button-modal: #633c05;
)
;
/*DARK*/
; $dark-header: #0f3057;
; $dark-back: $color-black;
; $dark-color-content: #5e5e5e;
; $dark-font: $color-white;
; $dark-hover: $color-yellow;
;

```

Ilustración 113 Colores

2. Fuentes

Tipografía. Para la tipografía siguiendo los estándares de buenas prácticas se han utilizado tan solo 2. Estas fueron escogidas meticulosamente y creo que encaja muy bien con el tema de la aplicación. Ambas están importadas en el proyecto de manera externa y ajena a las fuentes por defecto.

Krungthep → Es una fuente display que proporciona un estilo “**Hippie y desenfadado**” a las distintas vistas de la App. Además de ser muy vistosa y llamativa.

```

5 @font-face{
6   font-family: krungthep;
7   src: url('../fonts/Krungthep.ttf');
8 }

```

Ilustración 114 krungthep import

NotoSnas JP → Es una tipografía tipo sans, esta se entiende bien, es elegante y proporciona un buen contraste con los títulos.

```

/*Imports*/
@import url('https://fonts.googleapis.com/css2?family=Noto+Sans+JP');

```

Ilustración 115 Noto Sans import

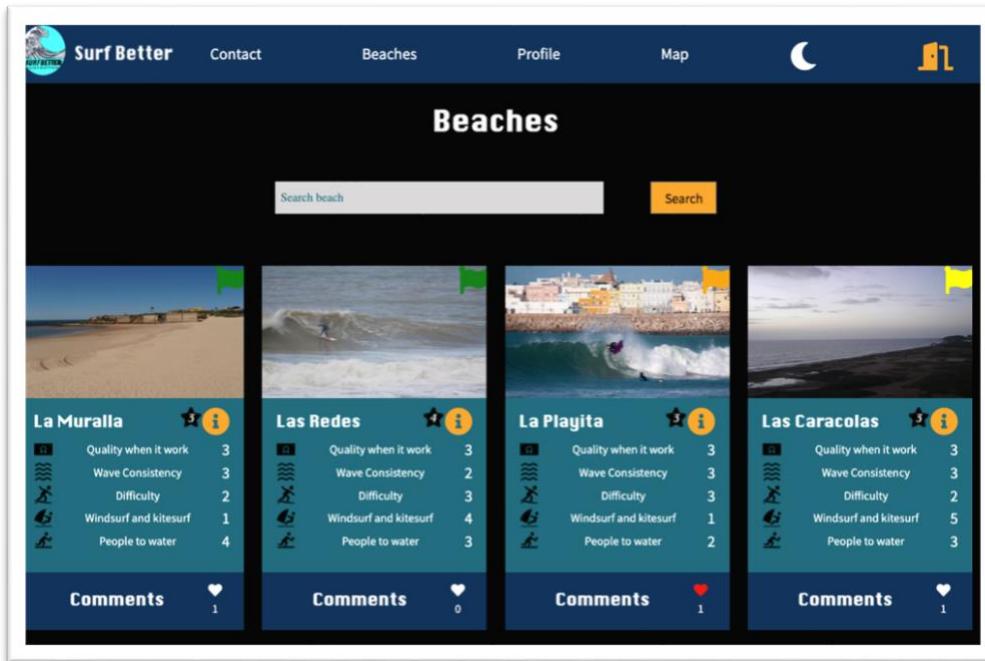
3. Tecnologías

Como tecnologías en el diseño se ha utilizado **adobe XD**. En el diseño de los wireframe previos al desarrollo, **CSS 3 HTML5**. Durante el desarrollo de la versión 1.0 <https://ajloinformatico.github.io/SurfBetter/v1/beaches>

Durante el desarrollo de las versión final **SASS** y componentes **JSX** con código **HTML 5**.

4. Resultado final

El resultado final es el siguiente:



116 Ejemplo darkMode

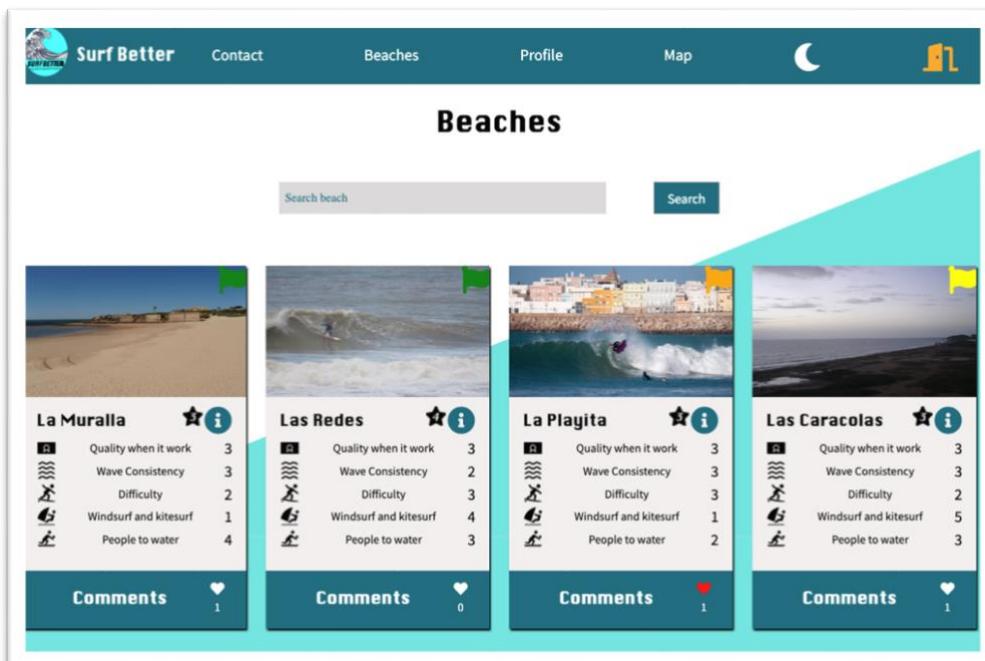


Figura 117 Ejemplo lightMode

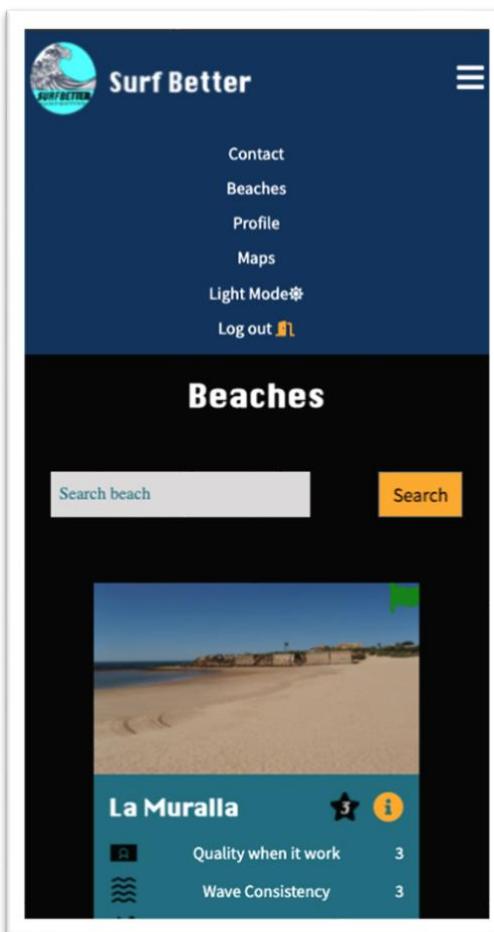


Ilustración 118 Ejemplo darkMode mobile

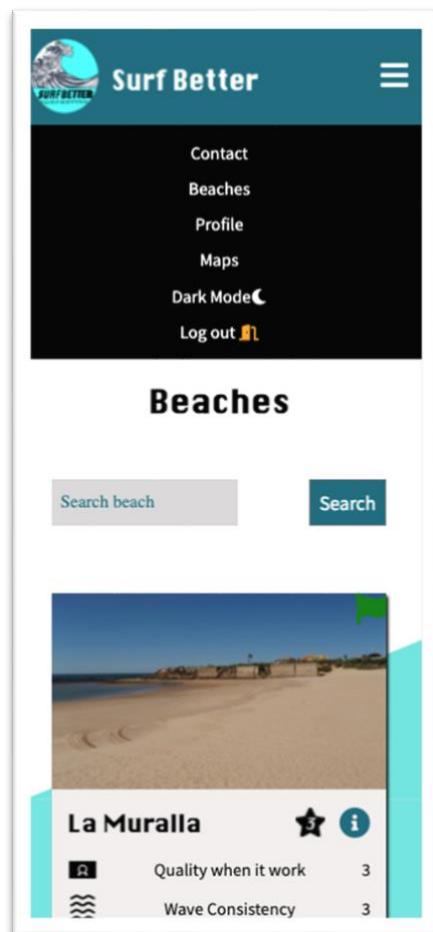


Ilustración 119 lightMode mobile

Nota: Los Sketch Mockups y Wireframe han ido evolucionando en el orden que los nombre durante el desarrollo del proyecto. El último **Wireframe** es el estilo que se ha tratado de seguir durante el desarrollo de **SurfBetter** en su versión final. Hay puntos que han cambiado debido a decisiones de diseño pero conforman el paso previo al desarrollo de las vistas de la aplicación.

5. Sketch

Los sketches estarán disponibles en el directorio **/docs** del repositorio

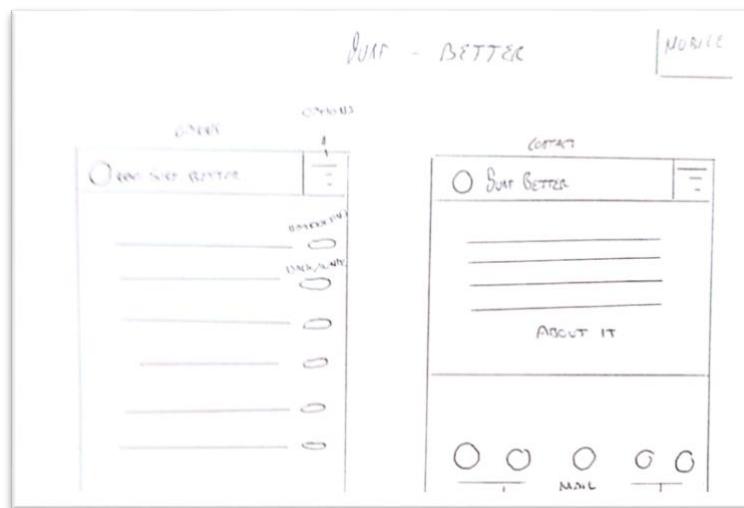


Ilustración 120 Ejemplo del Sketch

6. Mockups

Los mockups se agregarán en [/docs](#) del repositorio



Ilustración 121 Ejemplo del Mockups

7. Wireframe

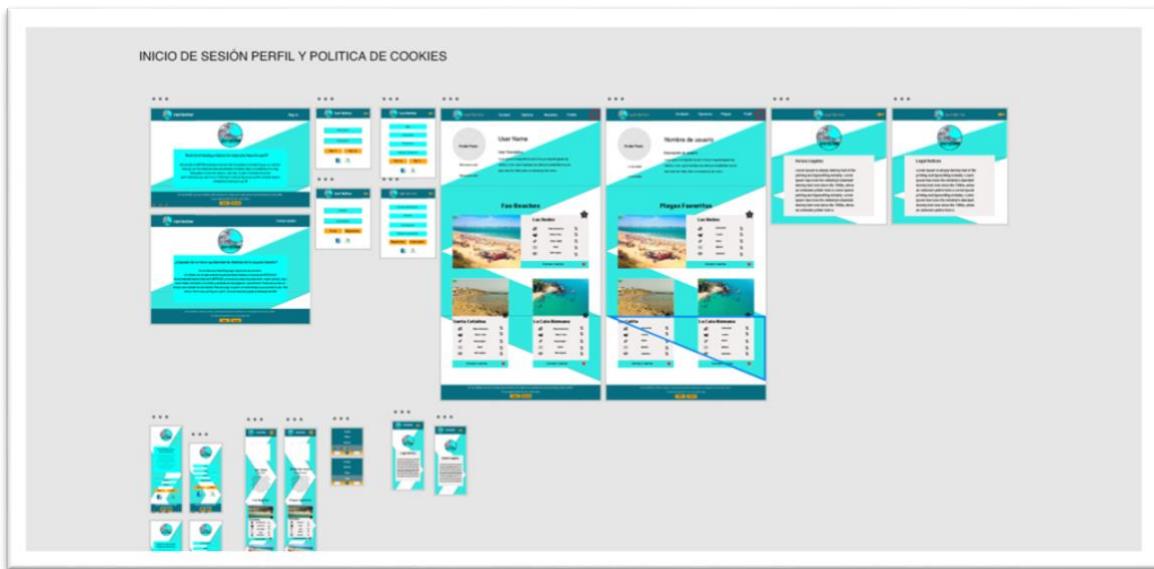


Ilustración 122 Ejemplo del Wireframe

5. Diseño

Diseño de la aplicación.

A continuación, incluiré el esquema entidad – relación que he utilizado. Su posterior transformación a modelos en UML y finalmente dos ejemplos de diagramas de flujo típicos que se dan en **SurfBetter**.

Nota: Como añadido y sabiendo que no forma parte del estándar m e-r. He añadido una línea discontinua indicando la procedencia de las claves foráneas. Lo he creído necesario pues debido a la densidad del diagrama pensé que aclararía un poco el contenido

1. Entidad-Relación

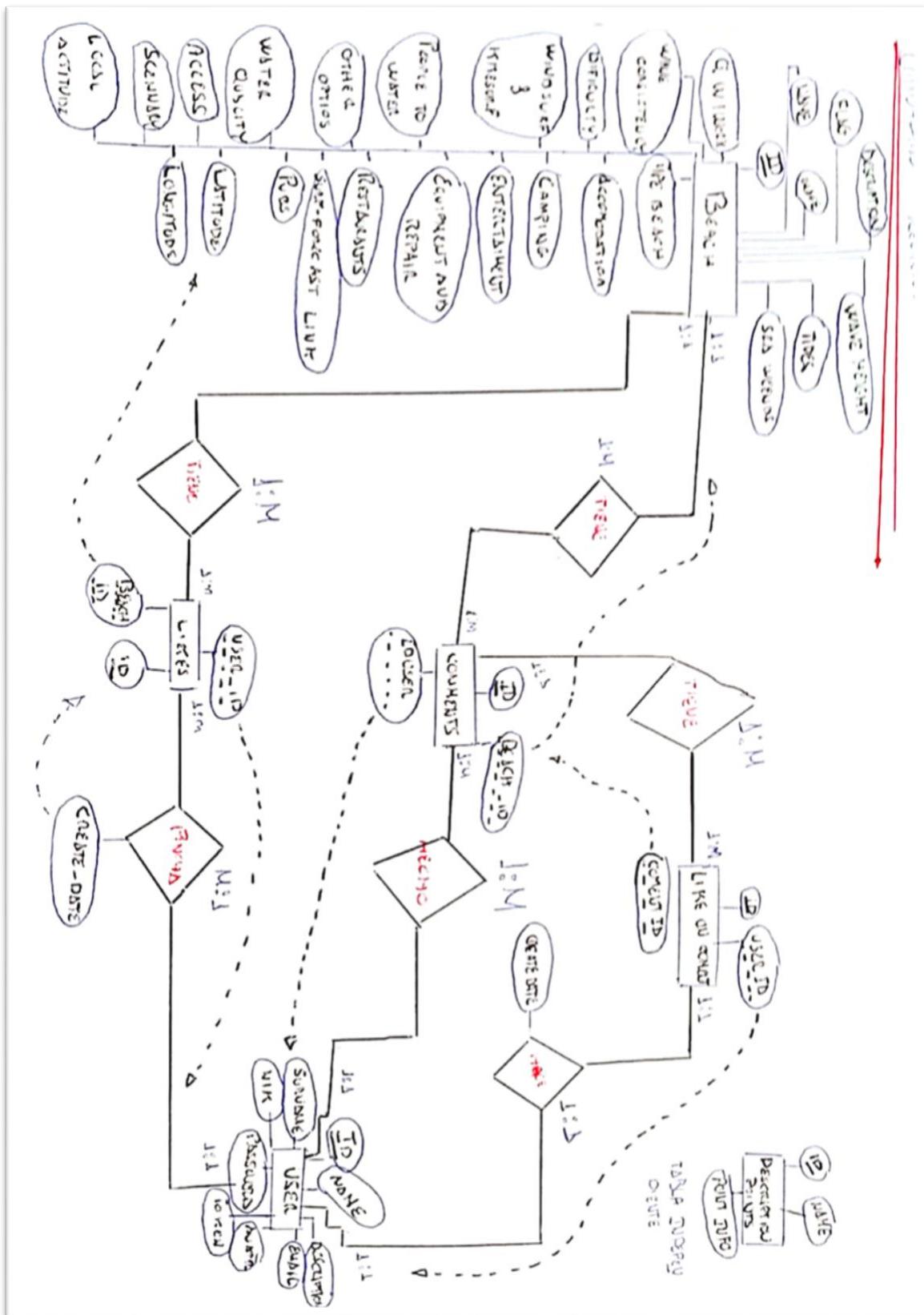


Ilustración 123 Entidad-Relación

2. Modelos en UML

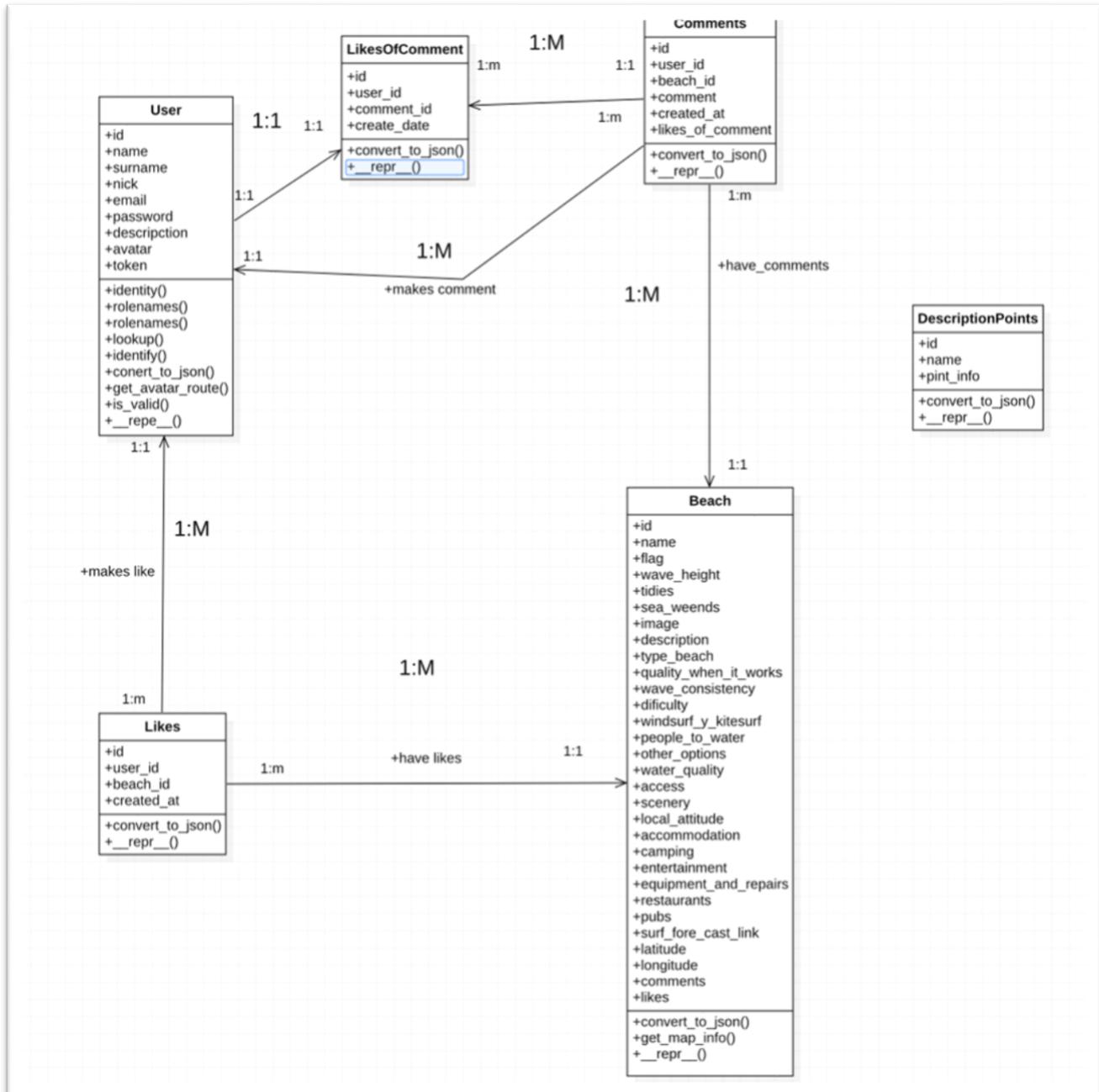


Ilustración 124 Modelos de la aplicación UML

3. Flujos típicos

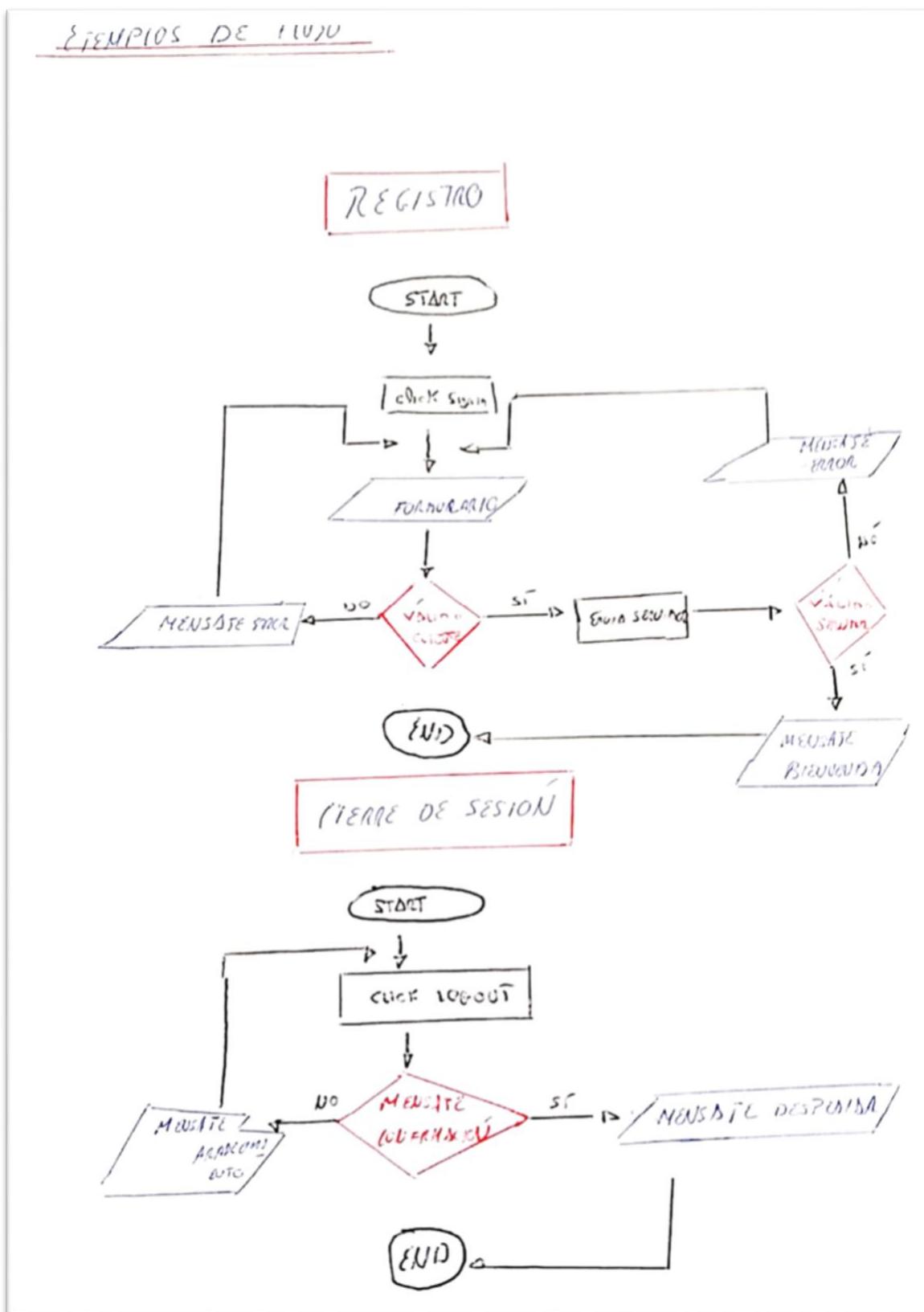


Ilustración 125 Ejemplo de diagrama de flujo

6. Pruebas

Por razones de tiempo no se ha podido realizar los tests que me hubiesen gustado. Las pruebas sobre la aplicación se dividen en dos bloques de la misma manera que se hizo con los puntos anteriores. **Front Back-end**

6.1. Front:

Siguiendo las pautas que pudimos ver en **cliente**. Las pruebas de la aplicación se han realizado utilizando **debugger**. Primero comprobando que información se recibe de servidor mediante **Insomniac**. Para a continuación desarrollar los distintos componentes reactivos (**useEffect** o **useState**) y comprobar mediante el **debugger que todo va bien**. Además de esto la consola de **React** informaba de cualquier incidencia. Tras la cual me encargaba de tratar de solucionar.

Además, he utilizado la herramienta **es-lint** de **VisualStudioCode**. Que me permitía localizar errores en el código. Hay ciertos warning que usando **es-lint** he deshabilitado ya que realmente no lo eran como **la ausencia de href** en las etiquetas `<a/>` que no tenían un destino tradicional sino que hacían las veces de botón o invocaban a un **useHistory** para cambiar entre las rutas de mi aplicación.

```
1  /* eslint-disable jsx-a11y/anchor-is-valid */
2  /* eslint-disable array-callback-return */
3  /* eslint-disable react-hooks/exhaustive-deps */
4  import React, {useEffect, useState} from "react";
5  import {useHistory} from "react-router-dom":
```

Ilustración 126 Ejemplo del uso de eslint

6.2 Servidor:

La parte de servidor se ha testeado utilizando **Insomniac**. Se han dividido los endpoints por directorios y testeado todas las peticiones, así como sus respuestas.

```
↳ gmail_service
    POST send_email

↳ routes_comments_likes
    DEL delete_like_comment
    POST add_like_comment
    DEL delete_comment
    POST add_comment
    DEL delete_like_beach
    POST add_like_beach

↳ image_service
    GET get_beach_image
    GET get_avatar
    PUT put_avatar

↳ routes_user
    PUT update_user
    POST refresh_token
```

Ilustración 127 Endpoints

Junto al proyecto en docs. Se ha construido un **yaml** en open Api con la información de la api.

```
openApi: "3.0.1"
info:
  title: "SurfBetter Api"
  description: "Api de mi proyecto de fin de ciclo"
  version: "1.0.0"
servers:
  - url: "http://localhost:5000/api"
paths:
  /:
    get:
```

Ilustración 128 Open Api

Agregando a la documentación de los test. Se ha agregado la documentación construida con **swagger** de la api. En </doc/api/index.html>

The screenshot shows the SurfBetter API documentation. On the left, there's a sidebar with 'API SUMMARY' and a list of 'API METHODS - DEFAULT' including addComment, addLikeBeach, addLikeComment, deleteComment, deleteLike, deleteLikeComment, getAvatar, getBeachCords, getBeachFiel, getBeachId, getBeachInfo, getBeaches, getBeachesPoints, getCurrentUser, getLogin, getOneBeachCords, getProfileCommonFavBeaches, postSignIn, putAvatar, putAvatar_1, putAvatar_2, refresh, rootGet, sendEmail. The main content area has a title 'SurfBetter Api' and 'API and SDK Documentation'. It shows 'Version: 1.0.0' and a note 'Api de mi proyecto de fin de ciclo'. Below this is a 'Default' section with a 'addComment' method. It shows a 'POST' request to '/beach/comment'. There are sections for 'Usage and SDK Samples' with links to Curl, Java, Android, Obj-C, JavaScript, C#, PHP, Perl, and Python. A code sample for 'curl -X POST "https://localhost/beach/comment"' is shown. At the bottom, there's a 'Parameters' section.

Ilustración 129 Api docs

Es posible acceder a esta desde <http://localhost:5000/docs>. Desde cualquier de los deployments

7. Desarrollo

Este punto lo dividiré por ámbito de desarrollo. Con ello me refiero a que comentaré dividiendo por bloques (**Control de versiones, Back-end, front-end, diseño y despliegue**) comentando los puntos **herramientas, desarrollo y problemas** de cada bloque

7.1 Control de versiones.

Herramientas.

He utilizado como herramienta de control de versiones **GitHub**. El repo se encuentra alojado en: <https://github.com/ailoinformatico/SurfBetter>. Dicho control de versiones lo he utilizado desde **terminal, Visual Studio Code, Pycharm y WebStorm**. Dependiendo del momento de desarrollo y el equipo donde me encontraba trabajando

Desarrollo.

El funcionamiento durante el proyecto ha sido bastante simple. Se ha ido trabajando directamente sobre el mismo **repositorio y la misma rama** tratando de hacer cada commit **atómico** con el objetivo de que estos tengan sentido y diesen una funcionalidad nueva y clara a la aplicación, o bien se commiteaba un fix como parches de los commits atómicos previos en el caso de que se encontrase un error a posteriori.

Dificultades encontradas:

No se han encontrado dificultades durante el desarrollo con el control de versiones. El único momento en el que **se produjo un fallo con este fue a mitad del desarrollo** preparando el estilo de las playas. El problema fue que no se porque perdí el diseño de las playas. Creo que esto fue debido a que estuve trabajando de la api en otro pc. Y al hacer commit actualicé el diseño con un antiguo diseño (el cargado en este repositorio local). La solución fue hacer rollback y recuperar el trabajo anterior.

7.2 Despliegue.

Herramientas.

La herramienta utilizada ha sido **docker**. Si concreto las tecnologías utilizadas dentro de los contenedores serían (**Node, Nginx y Flask con Gunicorn**).

Desarrollo.

La preparación del despliegue con docker se ha realizado al final del proceso de desarrollo. Previo al despliegue se ha estado trabajando con el propio servidor de **desarrollo de React** y un **entorno virtual** con **python**. Dicho servidor se integró dentro de la configuración de mi proyecto **React** configurando su arranque en **package.json**.

Una vez finalizado el desarrollo se montó el despliegue con un **docker-compose inicial** Que buildea dos contenedores en red. El primero de ellos con **Node** y **nginx**. El cual compila con **Node** el contenido de **src** en **front**, cambia la configuración de **nginx** para que este permita a **React** trabajar con sus rutas y monta el contenido compilado en el directorio de publicación de nginx.

El segundo contenedor comprende una imagen de **python** en la cual se instalan las dependencias python y lanza con **Gunicorn** la api en desarrollada con flask.

Problemas encontrados.

Se encontró un error durante el proceso de despliegue con las rutas de la aplicación. Al cargar la aplicación e iniciar un cambio de ruta la aplicación re direccionaba a un sitio de **nginx** que no existía. Es decir, no estaban funcionando las rutas de **React**. La solución la encontré en **stack Overflow**. Este error se daba debido a que había que indicarle a **nginx** en su configuración que dejase a **React** enrutar él el proyecto.

```
location / {
    root /usr/share/nginx/html;
    try_files $uri /index.html; # https://stackoverflow.com/questions/43951720/react-router-and-nginx#answer-43954597
}
```

Ilustración 130 Solución despliegue

7.3 Diseño

La mayor parte de este punto se ha comentado ya previamente en el punto 4 Guías de estilo y prototipado. Por lo que comentaré este punto de forma breve

Herramientas.

Se han utilizado para preparar el diseño previo al desarrollo **Adobe XD** y **zeplin**.

Desarrollo.

El desarrollo ha sido bastante simple. Se ha cogido el estilo de la versión previa. Este estilo se a desglosado, analizado y porteados a **SASS**. Finalmente se ha refinado y modularizado el código de SASS de la mejor manera posible. Incluyendo un archivo único para las variables. Donde se definen colores, fuentes, imports y transiciones. Este archivo a su vez es importado en **style.scss**. En este archivo se ha incluido todo el código de estilos.

El estilo se ha llevado siguiendo el patrón de trabajo mobile First.

Una vez definidos colores, fuentes y animaciones se establecieron los distintos breakPoints (6 en total comentados en el punto Guía de estilos y prototipado). A continuación se ha ido desarrollando cada pantalla tratando en primera instancia modularizar el contenido usando funciones y extensiones de SASS y en segunda instancia siguiendo el orden de pantallas. Haciendo el diseño previo en el más pequeño y escalando vista a vista conforme estas se iban finalizando.

Ejemplo: Se realiza la vista de detalle usando el método `crear_boton()` y extendiendo de el estilo común de formularios. Esta se hace pensando en las dimensiones más pequeñas y una vez que la vista está lista se adapta breakPoints a breakPoints tratando de mantener una coherencia en el estilo y de que los distintos elementos realicen una transformación correcta.

Problemas encontrados.

Ha día de hoy tengo un error con un widget moqueado que carga solo una porción del mismo. Este error fue dividido recientemente y no comprendo porque se produce.

7.4 Front

Herramientas.

El front está trabajado con **React**. Se han utilizado las siguientes herramientas.

Ides → Inicialmente se comenzó trabajando con **Visual Studio Code**. Cuando el **proceso de desarrollo se volvió más complejo** y fue adquiriendo más componentes y dependencias se migró de **Visual Studio Code** a **WebStorm**.

Revisiones de código → Se ha seguido las pautas propuestas en el punto 8 CodeStyle y se ha usado **eslint** de los ides para garantizar que el código esta bien estructurado.

Módulos o paquetes → Se han utilizado los siguientes módulos definidos en **Package.json**

```
"dependencies": {  
    "@fortawesome/fontawesome-free": "^5.15.3",  
    "@testing-library/jest-dom": "^5.11.9",  
    "@testing-library/react": "^11.2.5",  
    "@testing-library/user-event": "^12.8.3",  
    "html-react-parser": "^1.2.6",  
    "node-sass": "^5.0.0",  
    "nodemailer": "^6.6.1",  
    "react": "^17.0.2",  
    "react-dom": "^17.0.2",  
    "react-google-maps": "^9.4.5",  
    "react-router-dom": "^5.2.0",  
    "react-scripts": "^4.0.3",  
    "react-star-ratings": "^2.3.0",  
    "react-token-auth": "^1.1.8",  
    "sweetalert": "^2.1.2",  
    "web-vitals": "^1.1.1"
```

Ilustración 131 Package.json

Problemas encontrados.

Dado que no hemos visto demasiado tiempo **React** los conocimientos con los que partí no eran los mejores. He tratado de modularizar la aplicación, pero no me encanta el resultado. Me gustaría a futuro reunir y dividir las peticiones **Fetch** en componentes que hiciesen las veces de Provider. Esto se aplicará en el futuro en la aplicación.

7.5 Back End

Herramientas:

Ides → De igual manera que en el front. Inicialmente se comenzó trabajando con Visual Studio Code. Cuando el proceso de desarrollo se volvió más complejo y fue adquiriendo más componentes y dependencias se migró de Visual Studio Code a Pycharm

Revisiones de código → De igual manera que en el front. Se ha seguido las pautas propuestas en el punto 8 CodeStyle y se ha usado **eslint** de los Ides para garantizar que el código está bien estructurado.

Paquetes → Se ha utilizado un entorno virtual donde se han instalado los siguientes módulos:

```
i > apirequirements.txt
1 flask
2 flask_sqlAlchemy
3 flask_cors
4 flask-praetorian
5 gunicorn
```

Ilustración 132 dependencias de la api

Extra: La aplicación se ha organizado en módulos Blueprint. Estos se ingresan en **main.py**. Este main arranca en modo desarrollo el servidor de pruebas. Dentro de **/api** se incluye **wsgy.py**. Este archivo instancia **app ()** y sirve de entrypoints para **Gunicorn** en el despliegue.

8. CodeStyle

Previo a desglosar el contenido destacar que se ha tratado de utilizar en todo momento nombres de variables y métodos lo más aclarativos posibles.

8.1 Front

Las reglas propuestas han sido las siguientes.

[Ordenación de los métodos en los componentes.](#)

El orden de los métodos y componentes ha de ser el siguiente:

1º Declaración de componentes reactivos en el orden de useState () UseEffect ()

2º Peticiones Fetch en el caso de que estas no dependan de una acción específica del código.

3º Métodos del componente

4º Return Código JSX

```

    ...
    const BeachBox = (props) => {
      const history = useHistory()
      const [it, setIt] = useState(props.beach)
      const [user, setUser] = useState({})
      const [isComment, setIsComment] = useState(true)
      const [comment, setComment] = useState("")
    }

    /**
     * UsseEffect to get User Name
     */
    useEffect(() => {
      getUser().then(/*NO-LOOP*/)
      getBeachData().then(/*NO-LOOP*/)
    }, [])

    const getUser = async () => {
      authFetch("http://localhost:5000/api/current_user")
        .then(response => response.json())
        .catch(error => console.log(error))
        .then(userInfo => setUser(userInfo))
    }

    const getBeachData = async () => {
      fetch("http://localhost:5000/api/beach/" + it.id)
        .then(res => res.json())
        ...
    }
}

```

Ilustración 133 Ejemplo de ordenación de métodos

Nombrados.

Paquetes:

Para el nombrado de los paquetes he usado las reglas de java y se han nombrado **Snake case** y dedicando cada uno de ellos a una funcionalidad en concreto dentro de la aplicación.



Ilustración 134 Paquetes

Componentes JSX:

Siguiendo las recomendaciones vistas en clase; el nombrado de los componentes JSX es en **UpperCamelCase**

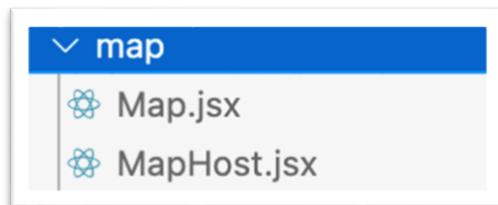


Ilustración 135 Componentes

Variables:

Se ha tratado de trabajar un código constante y que solo se use este tipo de variables durante el desarrollo. Excepto en los casos en los que no fuese posible otra opción. Antes de usar **let** se ha tratado así pues de utilizar componentes reactivos **useState()**.

```
const openCommentsOnTarget = (id) => {
  const target = document.getElementById("comments-container"+id)
  if (isComment) {
    target.classList.remove('beachBoxUnShow')
    target.classList.add('beachBoxShow')
    setIsComment(!isComment)
  } else {
    target.classList.remove('beachBoxShow')
    target.classList.add('beachBoxUnShow')
    setIsComment(!isComment)
  }
}
```

Ilustración 136 Ejemplo de variable

Nueva sintaxis:

Siguiendo las guías de buenas prácticas se ha intentado dentro de lo posible utilizar la nueva sintaxis de **JavaScript**. No usar **vars**, la utilización de **constants** como métodos y el uso **?:** como operadores ternarios en línea.

```
/** 
 * SetUnset darkMode
 */
export const changeDarkMode = (htmlTarger, darkPressed) => {
  darkPressed?htmlTarger.classList.add("darkMode"):
  htmlTarger.classList.remove("darkMode")
}
```

Ilustración 137 Uso de nueva sintaxis de JavaScript

Clases e id

Las clases que se incluyen en los **targets** se han nombrado en **LowerCamelCase**. Y las **ids** usando una sintaxis personalizada “**“nombre-descriptivo”**”. Aunque no existe un consenso sobre esto. He decidido usar estos tipos de nombrado para tener una diferenciación clara entre el resto de nombrados.

```
return <div id={comment.id} key={comment.id} className={"comment"}>
    <p>{comment.comment}</p>
    <div className={"commentLike"}>
        <span onClick={async () => {await unsetCommentLike(comment)}} className={isLikeFrom
            <i className={"fas fa-heart fa"}/>
```

Ilustración 138 Nombrado de clases

Nota: Por supuesto se ha tratado de respetar los espacios después de cada bloque y mantener una identación siempre correcta.

8.2 Back-end

En el caso del backend las reglas propuestas por mí han sido más genéricas. Nombrado de variables y archivos y módulos Blueprint, Snack case.

La ordenación no es necesaria ya que cada archivo tiene una función muy concreta y estos se encuentran modularizados.

Por ejemplo: en **Extensions.py** Se instancia la base de datos y la autenticación que es la que se va a utilizar en el resto de la aplicación. Entonces solo contiene esto respetando las normas básicas de **python** de nombrado de variables

```
from flask_sqlalchemy import SQLAlchemy
from flask_praetorian import Praetorian

db = SQLAlchemy()
desc = db.desc
guard = Praetorian()
```

Ilustración 139 Dependencias globales

En el caso de los seeders simplemente son los métodos para crear los directorios de los usuarios o bien inyectar SQL. Entonces es el mismo caso

```

from models import Likes, Comments, LikesOfComment
from extensions import db

def seed_likes(app):
    with app.app_context():
        if db.session.query(Likes).count() < 1:
            db.session.add_all([
                [
                    Likes(
                        id=1,
                        user_id=1,
                        beach_id=12
                    ),
                    Likes(
                        id=2,
                        user_id=1,
                        beach_id=9
                    ),
                    Likes(
                        ...
```

```

Figura 128: Seeder

Y en el caso de los módulos blueprint tan solo tiene la declaración del módulo y las posteriores rutas asociadas a dicho blueprint.

```

1 from flask import Blueprint, jsonify, request
2 from flask_praetorian import auth_required
3 import flask_praetorian
4 from extensions import guard, db
5 from models import User
6 import os
7
8 routes_user = Blueprint('routes_user', __name__)
9
10
11 @routes_user.route("/api/login", methods=['GET', 'POST'])
12 def login():
13 """
14 Logs an user in by parsing POST request contains user credentials and issuing a JWT token response
15 Fields valid by user
16 """
17 req = request.get_json(force=True)
18 email = req["email"]
19 password = req["password"]
20 try:
21 if db.session.query(User).filter_by(email=email).count() == 1:
22 user = guard.authenticate(email, password)
23 ret = {'access_token': guard.encode_jwt_token(user)}
24 return ret, 200
25 else:
26 return {'AuthenticationError': 'Email and/or password incorrect'}, 401
27 except Exception as e:
28 return {"Error": str(e)}, 500
30
```

```

Figura 129: Blueprint

9. Seguridad

Destacar el punto de seguridad de mi aplicación. Este es un tema muy importante y que en muchos desarrollos no se tiene demasiado en cuenta. Creo que en el caso de SurfBetter si que se ha tenido bastante en cuenta. Ya no solo en la propia validación de los formularios (Punto de entrada de muchos ataques), si no que también en todo el ciclo de vida de la aplicación. Integrando un Token único por sesión junto a las rutas de React. Y por supuesto a los entrypoints que el cliente tiene con mi api.

A modo de introducción **SurfBetter** utiliza **JWT (Json web Token)**. Esto está montado junto a React de manera que solo se puede acceder a las rutas de la aplicación si el usuario dispone de un **Token** autogenerado aleatorio y único por **sesión**. Este Token es gestionado por **flask** gracias al módulo **de python flask praetorian**.

guard = Praetorian()

Ilustración 140 Instancia e Praetorian

Praetorian se instancia en **Extensions.py** haciendo de “inyector de dependencias” se utiliza en todo la **Api**. Asociando el usuario con este módulo y instanciando varias propiedades para que **flask praetorian** funcione

```
@property
def identity(self):
    """[Get user id]
    Returns:
        [int]: [User id]
    """
    return self.id

@property
def rolenames(self):
    """[Generate property to return the roles of a user separated by commas]
    Returns:
        [List]: [User comma separated roles]
    """
    try:
        return self.roles.split(",")
    except Exception:
        return []

@classmethod
def lookup(cls, email):
    """[Get a player by email]
```

141 Flask praetorian Config on model

Una vez el modelo de usuario tiene sus propiedades definidas. Desde las rutas podemos en el **login** asociar un **Token** único a dicho usuario que es generado de manera aleatoria en el momento en que el usuario inicia sesión.

```

@routes_user.route("/api/login", methods=['GET', 'POST'])
def login():
    """
    Logs an user in by parsing POST request contains user credentials and issuing a JWT token response
    Fields valid by user
    """
    req = request.get_json(force=True)
    email = req["email"]
    password = req["password"]
    try:
        if db.session.query(User).filter_by(email=email).count() == 1:
            user = guard.authenticate(email, password)
            ret = {'access_token': guard.encode_jwt_token(user)}
    return ret, 200

```

Ilustración 142 Flask login

Este método devuelve un Token único para el usuario que es indispensable para acceder a cualquier ruta que no sea **/login** del cliente y dos **acceder** a la mayoría de **endpoints** de mi api. Los endpoints protegidos se definen mediante el decorador **@auth_required**

```

@routes_user.route('/api/userupdate', methods=['PUT'])
@auth_required
def update_user_profile():
    """[Update user information]

    Returns:
        [httpRseponse, 409]: ["email or nick is already in use"]
        [httpResponse Json Object, 200]: ["User object"]
    """
    req = request.get_json(force=True)
    nick = req["nick"]
    email = req["email"]
    description = req["description"].capitalize()

    if nick[0] != "@":
        nick = "@" + nick

    if description == "":
        description = "I dont like descriptions"

```

Ilustración 143 Flask protected endpoints

En **cliente**. El Token es gestionado por **react-token-auth**. Este está montado en un componente específico para la seguridad.

```

1 import {createAuthProvider} from 'react-token-auth';

2
3 export const [useAuth, authFetch, login, logout] =
4   createAuthProvider({
5     accessTokenKey: 'access_token',
6     onUpdateToken: (token) => fetch('http://localhost:5000/api/refresh', {
7       method: 'POST',
8       body: token.access_token
9     })
10    .then(response => response.json)
11  });
12

```

144 ken de seguridad React

Con este componente uso el método **login** y guardo el **Token** en el **localStorage** al iniciar sesión. React al leer el Token re direcciona a las rutas protegidas, en el momento de que dicho Token no exista el usuario vuelve al login.

```

return (
  <Router history>
    <Switch>
      {/*Public routes*/}
      <Route path="/login" exact>
        <LoginRegister history={history} />
        <ScrollButton/>
      </Route>
      <Route path="/legal" exact>
        <LegalNotices/>
        <Footer/>
        <ScrollButton/>
      </Route>
      {
        !logged &&
        <Redirect to="login"/>
      }
      {/*Protected routes*/}
      <Route path="/" exact>
        <BeachesHost/>
        <Footer/>
        <ScrollButton/>
      </Route>
      <Route path="/beaches" exact>
        <BeachesHost/>
        <Footer/>
        <ScrollButton/>
      </Route>
    </Switch>
  );

```

Ilustración 145 Rutas públicas y protegidas

10. Conclusiones

El resultado final a mi parecer como desarrollador es bastante bueno. La aplicación ofrece toda la funcionalidad requerida en su inicio con el añadido de un mapa y un modo noche mejorado.

Como aspectos a mejorar en el futuro me gustaría en primer lugar re factorizar el cliente con todo lo que he aprendido y modularizarlo un poco más, Instanciar métodos Provider para cumplir con los principios de responsabilidad única e incluir más componentes de widgets en dicho paquete. Por supuesto estaría bien poder incluir scraping en un futuro. La decisión de hacerlo moqueado fue por tiempo y porque la empresa que me iba a proveer el Json me dio el negativo porque no podían aceptar más aplicaciones.

Aún con estos inconvenientes creo que mi aplicación cubre todos los puntos necesarios y requeridos por cada modulo y además es muy funcional, ofrece una buena experiencia de usuario y está bastante bien montada en el sentido de la organización y la estructura del código.

11. Índice de imágenes

Ilustración 1 Logo SurfBetter	1
Ilustración 2 Git files	8
Ilustración 3 exrensions.py	8
Ilustración 4 : models.py	9
Ilustración 5 Seeders.py	10
Ilustración 6 image_service.py	10
Ilustración 7 gmail_service.py	10
Ilustración 8 rutas.py	10
Ilustración 9 main.py	11
Ilustración 10 usuarios servidor	11
Ilustración 11 Componentes	12
Ilustración 12 Paquete auth	12
Ilustración 13 Componente autenticación	13
Ilustración 14 Paquete beaches	13
Ilustración 15 Paquete loginsigin	14
Ilustración 16 BeachHost.jsx	14
Ilustración 17 BeachHost.jsx (2)	15
Ilustración 18 mapLight const (1)	15
Ilustración 19 mapHost	16
Ilustración 20 lightMode	16
Ilustración 21 Paquete perfil	16
Ilustración 22 Profile.jsx	17
Ilustración 23 Paquete widgets	17
Ilustración 24 ScrollButton.jsx	18
Ilustración 25 Componentes sin paquete	18
Ilustración 26 Contact.jsx	19
Ilustración 27 Form.jsx	19
Ilustración 28 : Acceso a LegalNotices	20
Ilustración 29 : Acceso a LegalNotices (2)	20

Ilustración 30 Contenido legal	21
Ilustración 31 HeaderMenu.jsx	21
Ilustración 32 const logOut	22
Ilustración 33 Lógica de cambio de tema	22
Ilustración 34 Variable de tema	23
Ilustración 35 Lógica del cambio de tema	23
Ilustración 36 Comprobación del tema	24
Ilustración 37 Footer.jsx	24
Ilustración 38 Archivos generales	25
Ilustración 39 App.jsx	25
Ilustración 40 Util.jsx	26
Ilustración 41 Index.js	26
Ilustración 42 statics	27
Ilustración 43: _vars.scss	28
Ilustración 44 style.scss	29
Ilustración 45 Ejemplo de fuente de título	29
Ilustración 46 credentials.js	30
Ilustración 47 Public	30
Ilustración 48 package.json	31
Ilustración 49 Vista inicial	32
Ilustración 50 Modal de registro	32
Ilustración 51 Mensajes a los usuarios	32
Ilustración 52 Mensajes a los usuarios (2)	33
Ilustración 53 Mensaje de registro	34
Ilustración 54 Perfil del usuario	35
Ilustración 55 Directorio del usuario	35
Ilustración 56 Cierre de sesión	36
Ilustración 57 Alera cierre de sesión	36
Ilustración 58 Cancelación del cierre de sesión	37
Ilustración 59 Cierre de sesión	37
Ilustración 60 Error en el login	37
Ilustración 61 imagen de usuario actualizada	39
Ilustración 62 Modal de opciones	39
Ilustración 63 Ruta flask passwordReset	40
Ilustración 64 Alerta contraseña actualizada	40
Ilustración 65 Contraseña incorrecta	41
Ilustración 66 Comprobación de los datos de usuario previo insert	41
Ilustración 67 Actualización de datos del usuario.	42
Ilustración 68 Accediendo a beaches	42
Ilustración 69 Accediendo a beaches (2)	43
Ilustración 70 Vista de playas	43
Ilustración 71 Beach Carta de la playa	44
Ilustración 72 Botón comment	44
Ilustración 73 Imágenes de comentarios	45
Ilustración 74 Like	45
Ilustración 75 dislike	45
Ilustración 76 Eliminar comentario.	46
Ilustración 77 comentario eliminado	46
Ilustración 78 Likes en las playas	46
Ilustración 79 Playas favoritas	47
Ilustración 80 playas comentadas	47
Ilustración 81 Footer	47
Ilustración 82 Mapa	48
Ilustración 83 Mapa satélite	48
Ilustración 84 Mapa pantalla completa	49
Ilustración 85 Dialogo de información.	49
Ilustración 86 Comentarios del detalle	50

<i>Ilustración 87 Botón de vuelta</i>	50
<i>Ilustración 88 Contacto (1)</i>	51
<i>Ilustración 89 Infojolo</i>	51
<i>Ilustración 90 Integrantes</i>	52
<i>Ilustración 91 Contacto Footer</i>	52
<i>Ilustración 92 Modo oscuro activado</i>	53
<i>Ilustración 93 Modo light activado</i>	53
<i>Ilustración 94 Modo oscuro activado (2)</i>	54
<i>Ilustración 95 Error específico</i>	54
<i>Ilustración 96 Error global</i>	55
<i>Ilustración 97 Repositorio</i>	56
<i>Ilustración 98 Deploy (1)</i>	56
<i>Ilustración 99 Deploy React</i>	57
<i>Ilustración 100 Deploy Flask</i>	57
<i>Ilustración 101 Deploy desarrollo</i>	58
<i>Ilustración 102 Api en desarrollo</i>	58
<i>Ilustración 103 api Insomniac</i>	58
<i>Ilustración 104 docker-compose build</i>	59
<i>Ilustración 105 docker-compose up</i>	59
<i>Ilustración 106 docker-compose build</i>	60
<i>107 docker (1)</i>	60
<i>Ilustración 108 docker (1)</i>	61
Ilustración 109 default.conf	61
<i>Ilustración 110 dockerfile cliente</i>	62
<i>Ilustración 111 apirequirements.txt</i>	62
<i>Ilustración 112 Dockerfile servidor</i>	63
<i>Ilustración 113 Colores</i>	64
<i>Ilustración 114 krugthep import</i>	64
<i>Ilustración 115 Noto Sans import</i>	64
<i>116 Ejemplo darkMode</i>	65
<i>Figura 117 Ejemplo lightMode</i>	65
<i>Ilustración 118 Ejemplo darkMode mobile</i>	66
<i>Ilustración 119 lightMode mobile</i>	66
<i>Ilustración 120 Ejemplo del Sketch</i>	67
<i>Ilustración 121 Ejemplo del Mockups</i>	67
<i>Ilustración 122 Ejemplo del Wireframe</i>	68
<i>Ilustración 123 Entidad-Relación</i>	69
<i>Ilustración 124 Modelos de la aplicación UML</i>	70
<i>Ilustración 125 Ejemplo de diagrama de flujo</i>	71
<i>Ilustración 126 Ejemplo del uso de eslint</i>	72
<i>Ilustración 127 Endpoints</i>	73
<i>Ilustración 128 Open Api</i>	73
Ilustración 129 Api docs	74
<i>Ilustración 130 Solución despliegue</i>	75
Ilustración 131 Package.json	77
<i>Ilustración 132 dependencias de la api</i>	78
Ilustración 133 Ejemplo de ordenación de métodos	79
<i>Ilustración 134 Paquetes</i>	79
<i>Ilustración 135 Componentes</i>	80
Ilustración 136 Ejemplo de variable	80
<i>Ilustración 137 Uso de nueva sintaxis de JavaScript</i>	80
<i>Ilustración 138 Nombrado de clases</i>	81
<i>Ilustración 139 Dependencias globales</i>	81
<i>Ilustración 140 Instancia e Praetorian</i>	83
<i>141 Flask praetorian Config on model</i>	83
<i>Ilustración 142 Flask login</i>	84
<i>Ilustración 143 Flask protected endpoints</i>	84

144 <i>ken de seguridad React</i>	85
Ilustración 145 Rutas públicas y protegidas	85

12. Referencias

Las referencias consultadas durante el desarrollo han sido las siguientes:

React.

<https://github.com/ajloinformatico/React-firebase-notes-app>
<https://javascript.plainenglish.io/redirecting-and-creating-protected-routes-with-react-router-1c0e2128cf3c>
<https://es.reactjs.org/docs/getting-started.html>

Flask.

<https://flask.palletsprojects.com/en/2.0.x/>
<https://flask-sqlalchemy.palletsprojects.com/en/2.x/>
<https://flask-praetorian.readthedocs.io/en/latest/index.html>

Design

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
<https://sass-lang.com/>
<https://colorhunt.co/>

Deploy

<https://yasoob.me/posts/how-to-setup-and-deploy-jwt-auth-using-react-and-flask/>