INSTITUTO DE ENSEÑANZA SECUNDARIA CASTELAR

FAMILIA PROFESIONAL DE INFORMÁTICA

TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

MEMORIA DEL MÓDULO PROFESIONAL DE PROYECTO

Tattoo3D

ALUMNO: Francisco José de los Placeres Diaz

ALUMNO: José Javier Acosta Blasco

TUTOR: Antonio Paniagua

CURSO ACADÉMICO: 2º DAM

1. Análisis de contexto:

a) Conclusión estudio tras estudio y analisis de las paginas web de los estudios de tatuajes locales (Badajoz).

-Páginas muy simples y sin funciones, todas se basan en un simple "Contacta" y como mucho una galeria mostrando imagenes de algunos de sus trabajos

-Casi ningún estudio cuenta con pagina.

-El potencial cliente de dicho servico no se siente atraído por una página así.

b) Estudio de mercado y potenciales clientes a los que les generaría valor añadido en su empresa.

-Tipo de clientes o Empresas en donde se implantará el proyecto.

Este proyecto podrá venderse y podrá ser implantado en los siguientes tipos de empresas:

1. Estudio de tatuajes

2- Estudio de piercings

3.-Cualquier tipo de negocio en el que se quiera mostrar en directo la simulación del resultado del producto tras aplicar algún tipo de pintura o dibujo

-Utilidad o beneficio que encontrarán los clientes con la implantación y explotación del proyecto.

Los estudios o negocios que adquieran este software, podrán beneficiarse de que ganaría una gran cantidad de clientes ya que dispondría de una página con funciones que la competencia no tiene, como por ejemplo un simulador 3d en el que se visualiza el tatuaje elegido, un sistema de para pedir cita etc. Al no tener casi ningún estudio una pagina web hay muchos clientes potenciales.

2. Diseño del proyecto:

a) Coste de implantación del proyecto para una empresa o cliente.

Para saber el coste del proyecto habría que calcular las horas trabajadas para asi poder tener un coste más real pero la página tendría un coste aproximado de unos 1000 euros por el trabajo realizado. Luego por el resto de la aplicación como hemos usado firebase si supera un limite: 50.000 operaciones de lectura dia, escritura 20.000 dia, eliminacion 20.000 dia y salida en red 10GiB mes de peticiones google aplicará unos cargos es decir el plan Blaze este es el plan pago de Firebase que usa el modelo de precios de pago a medida que crece. Los costos de uso del servidor son de $0.18 por GB para el almacenamiento en base de datos y de $0.026 por GB de almacenamiento. La plataforma también cotiza las operaciones de bases de datos y la transferencia de datos.

b) Si es un producto software, plataforma donde se ejecutará el proyecto y otros requisitos que deba tener la plataforma.

Hardware:

- **2.00 GB de RAM.**
- **3.00 GB de espacio libre en disco duro.**

Compatible con dispositivos Android, IOS, Windows..

Compatible con todos los navegadores

c) Aplicación de la ley de protección de datos en el proyecto.

Investigar el reglamento de protección de datos y cómo va a afectar a la implantación del proyecto.

Entre los cambios más notables que introduce el RGPD respecto a la LOPD que afectan a una web, están los siguientes 10 puntos básicos:

1. **La necesidad de obtener un consentimiento explícito, inequívoco, informado y verificable de los usuarios o clientes para poder gestionar sus datos, con nuevos mecanismos para obtenerlo y acreditarlo.**

2. **La necesidad de ofrecer a usuarios y clientes información completa de manera previa, sobre el uso de su propia información personal con nuevos requisitos informativos respecto a la LOPD, entre otros, el de la claridad, la transparencia y la accesibilidad a esta información.**

3. **La necesidad de concretar y especificar con mucho más rigor todo lo que incluya la información personal de las personas a quienes les requieras sus datos (en una suscripción, comentario, registro, etc.).**

4. **La necesidad de garantizar un acceso fácil de todos los que te aportan sus datos personales a la información que les concierne.**

5. **El derecho al olvido y al de oponerse incluso al uso de datos personales, a efectos de establecimiento de perfiles.**

6. **La necesidad de llevar un registro de actividades tratamientos interno (RAT).**

7. **La necesidad de acreditar el cumplimiento de todos los colaboradores con quienes se compartas información.**

8. **La necesidad de aplicar medidas de seguridad adecuadas al nivel de riesgo.**

9. **La necesidad de llevar a cabo controles periódicos para garantizar todas las medidas y obligaciones recogidas en el RGPD.**

10. **La necesidad de disponer de un protocolo de detección y notificación de brechas de seguridad.**

    Gracias a Firebase no tenemos problemas de seguridad este encarga de encriptar las contraseñas etc.

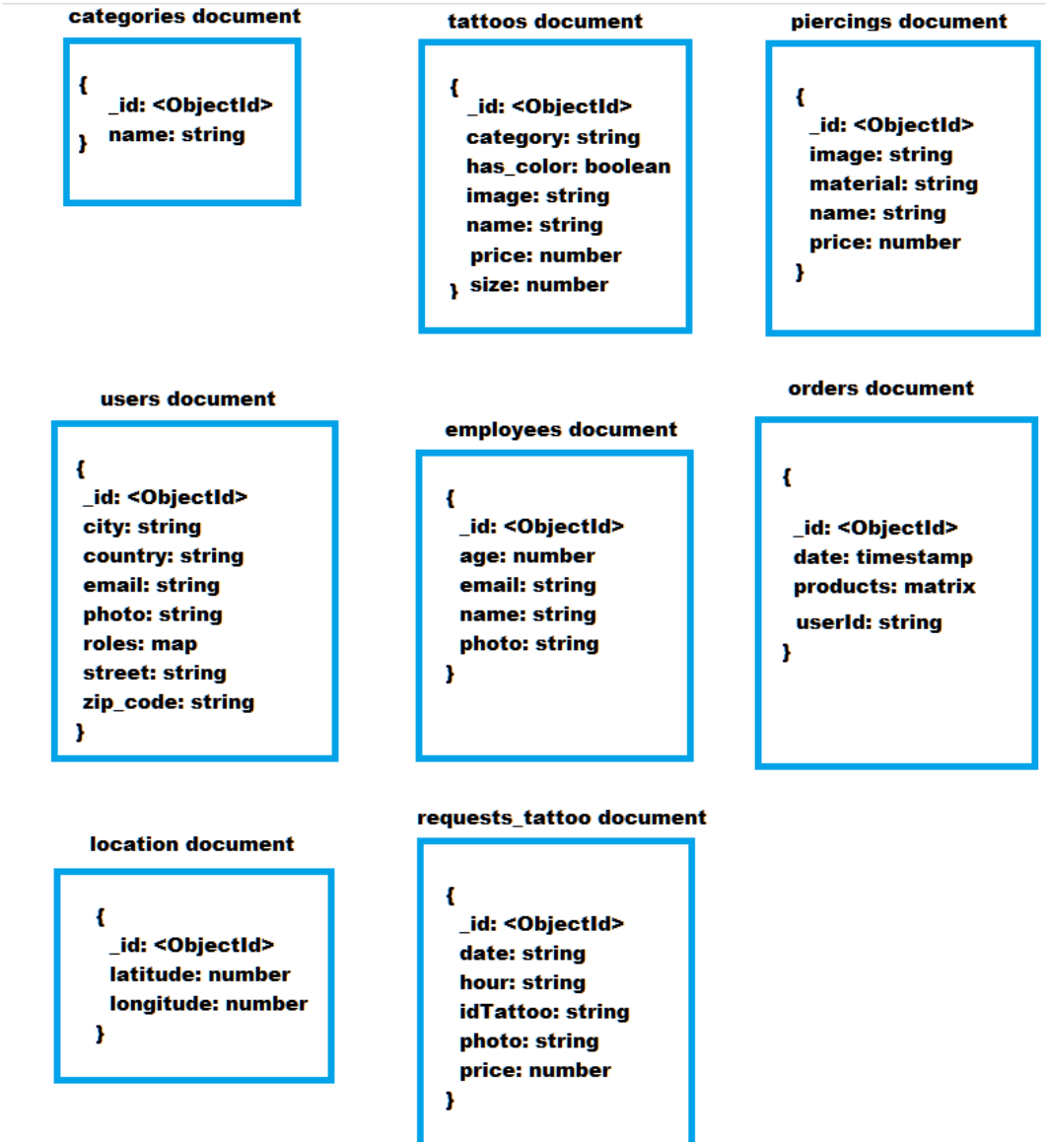
d) Metodología que se ha seguido para el desarrollo del proyecto

Qué pasos has seguido para la realización del proyecto y cómo has abordado su desarrollo

Lo primero que hicimos fue investigar la librería threejs buscamos documentación para hacer algo parecido a lo que teníamos en mente poder acoplar tatuajes a un modelo 3D por lo que investigamos y nos parecía bastante compleja pero conseguimos encontrar un ejemplo algo similar a lo que buscábamos. A partir de aquí lo cargamos en local en Visual Code con el plugin Live Server. Nuestra idea era un cuerpo entero y el que teníamos era medio cuerpo, por tanto, Vimos un generador de modelos personalizados llamado Character Creator lo conseguimos crear pero al añadirlo no pudimos acoplar tatuajes al modelo por tanto a partir de aquí continuamos con el modelo de medio cuerpo con vistas a un futuro cambiarlo por

otro modelo. A partir de aquí comenzamos a desarrollar la página web decidimos usar firebase y angular y comenzamos creando los crud básicos. Nos compartimos el trabajo, quedando cada semana para explicar que hacer durante la siguiente semana. Así fuimos creando la diferentes vistas de nuestras web app.

e) Esquema de Bases de datos sobre el que se va a basar el proyecto.



3. Organización de la ejecución:

a) **Planing de ejecución del proyecto a lo largo del periodo Marzo-Junio 2020.**

MARZO-

El primer mes ha sido principalmente dedicado a investigación, teníamos una idea pero no sabíamos como ejecutarla.

La idea era simular un tatuaje elegido en un muñeco 3d.

Estuvimos gran parte del mes investigando sobre en qué tecnología podríamos lograr esto, tras una ardua e intensiva investigación descubrimos una librería de JavaScript llamada Three.js, vimos que se podía manipular diseños 3d en directo, a partir de aquí la investigación se enfocó en buscar ejemplos realizados con esta librería.

Encontramos un ejemplo en el cual tras presionar cualquier parte de la figura 3d se manchaba de pintura el punto seleccionado, pensamos es modificar este mismo ejemplo para que hiciese lo que buscábamos, en vez de manchas de pintura, imágenes PNG.

Adaptarlo fué fácil, cuestión de una tarde.

En las siguientes semanas nos dedicamos a intentar cambiar la figura 3d por una figura de una persona, tras intentos, descargas de multitud de software distintos y muchas horas, decidimos dejar la figura del ejemplo ya que no logramos crear algo mejor que esto y el tiempo se nos acababa.

Tras esto nos tocaba elegir con que tecnología hacer el proyecto (Angular), ya que esto de la figura 3d es simplemente una implementación de este.

Los siguientes días discutimos la estructura de la base de datos de nuestro proyecto, además de decidir qué funciones habrá.

Nuestro proyecto ha sido desarrollado en Angular y firebase para el uso de firebase hemos tenido que incluir en el environment de Angular el siguiente JSON de configuración de Firebase.

```
// This file can be replaced during build by using the `fileReplacements` array.
// `ng build --prod` replaces `environment.ts` with `environment.prod.ts`.
// The list of file replacements can be found in `angular.json`.

export const environment = {
  production: false,
  // For Firebase JS SDK v7.20.0 and later, measurementId is optional
  firebase: {
    apiKey: 'AIzaSyDATXR3gnEzAlcWy4v6HVHpeyOEeVMu0cU',
    authDomain: 'tatto3d-2c482.firebaseapp.com',
    projectId: 'tatto3d-2c482',
    storageBucket: 'tatto3d-2c482.appspot.com',
    messagingSenderId: '589217802759',
    appId: '1:589217802759:web:8b547c38eb989023c058a2',
    measurementId: 'G-N2640Q1RCM'
  }
};

/*
 * For easier debugging in development mode, you can import the following file
 * to ignore zone related error stack frames such as `zone.run`, `zoneDelegate.invokeTask`.
 *
 * This import should be commented out in production mode because it will have a negative impac
 * on performance if an error is thrown.
 */
// import 'zone.js/dist/zone-error';  // Included with Angular CLI.
```

Y hay que incluirlo en el app.module

```
imports: [
  BrowserModule,
  AppRoutingModule,
  AngularFireModule.initializeApp(environment.firebase),
  AngularFirestoreModule,
  AngularFireAuthModule,
  ReactiveFormsModule,
  FormsModule,
  BrowserAnimationsModule,
  NgbModule,
  IvyCarouselModule,
  AgmCoreModule.forRoot( lazyMapsAPILoaderConfig: {
    apiKey: 'AIzaSyAYJj4QvIrs-Qax20xxYccDRSmwS4bDJxQ'
  })
]
```

Para poder incluirlo obviamente hay que instalar los paquetes pertinentes de Firebase con npm i firebase en la raíz del proyecto

Servicios

A partir de aquí los servicios a medida que los íbamos necesitando:

Necesitamos importar AngularFirestore para poder crear, leer, borrar y actualizar datos en Firebase.

- **Servicio de tatuajes**

```
export class TattosService {

  constructor(private firestore: AngularFirestore) { }

  addTattoo(tatto: TattooInterface): Promise<any> {
    return this.firestore.collection( path: 'tattoos').add(tatto);
  }

  getTattoos(): Observable<any> {
    return this.firestore.collection( path: 'tattoos',  queryFn: ref => ref.orderBy( fieldPath: 'name',  directionStr: 'asc')).snapshotChanges();
  }

  deleteTattoo(id: string): Promise<any> {
    return this.firestore.collection( path: 'tattoos').doc(id).delete();
  }

  getTattoo(id: string): Promise<TattooInterface> {
    let tattoo: any = null;
    const docRef =  this.firestore.collection( path: 'tattoos').doc(id);
    tattoo = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return tattoo;
  }

  updateTattoo(id: string, data: TattooInterface): Promise<any> {
    return this.firestore.collection( path: 'tattoos').doc(id).update(data);
  }
}
```

Funciones del servicio de Tatuajes:

- addTattoo(tatto: TattooInterface): **recibe como parámetros la interfaz de tattoo y va devolver una promesa. Básicamente va añadir el tatuaje en caso de que vaya todo bien.**
- getTattoos(): **Obtiene el observable de la colección de tattoos ordenándolos por nombre ascendentemente.**
- deleteTattoo(id: string): **recibe por parámetros el id del tattoo a eliminar, esta función devolverá una promesa; si todo va bien se eliminará el tatuaje.**
- getTattoo(id: string): **recibe por parámetros el id del tattoo a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicho tatuaje.**
- updateTattoo(id: string, data:TattooInterface): **recibe por parámetros el id del tatuaje a actualizar y el tatuaje nuevo (data). Esta función devolverá una promesa, es decir, actualizara el tattoo en caso de que vaya todo bien.**

- **Servicio de Piercings**

```typescript
export class PiercingsService {

  constructor(private firestore: AngularFirestore) { }

  addPiercing(piercing: PiercingInterface): Promise<any> {
    return this.firestore.collection( path: 'piercings').add(piercing);
  }

  getPiercings(): Observable<any> {
    return this.firestore.collection( path: 'piercings',  queryFn: ref => ref.orderBy( fieldPath: 'name',  directionStr: 'asc')).snapshotChanges();
  }

  deletePiercing(id: string): Promise<any> {
    return this.firestore.collection( path: 'piercings').doc(id).delete();
  }

  getPiercing(id: string): Promise<PiercingInterface> {
    let piercing: any = null;
    const docRef =  this.firestore.collection( path: 'piercings').doc(id);
    piercing = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return piercing;
  }

  updatePiercing(id: string, data: PiercingInterface): Promise<any> {
    return this.firestore.collection( path: 'piercings').doc(id).update(data);
  }
```

Funciones del servicio de Piercings:

- **addPiercing(tatto: PiercingInterface): recibe como parámetros la interfaz de piercing y va devolver una promesa. Básicamente va añadir el piercing en caso de que vaya todo bien.**
- **getPiercings(): Obtiene el observable de la colección de piercings ordenándolos por nombre ascendentemente.**
- **deletePiercing(id: string): recibe por parámetros el id del piercing a eliminar, esta función devolverá una promesa; si todo va bien se eliminará el piercing.**
- **getPiercing(id: string): recibe por parámetros el id del piercing a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicho piercing.**
- **updatePiercing(id: string, data:PiercingInterface): recibe por parámetros el id del piercing a actualizar y el piercing nuevo (data). Esta función devolverá una promesa, es decir, actualizará el piercing en caso de que vaya todo bien.**

- **Servicio de categorías: Este servicio fue creado porque el tatuaje tiene un campo llamado categoria y asi con este servicio podemos manejar mejor estas categorías**

```
export class CategoriesService {
  constructor(private firestore: AngularFirestore) { }

  addCategory(category: CategoryInterface): Promise<any> {
    return this.firestore.collection( path: 'categories').add(category);
  }

  getCategories(): Observable<any> {
    return this.firestore.collection( path: 'categories',  queryFn: ref => ref.orderBy( fieldPath: 'name',  directionStr: 'asc')).snapshotChanges();
  }

  deleteCategory(id: string): Promise<any> {
    return this.firestore.collection( path: 'categories').doc(id).delete();
  }

  getCategory(id: string): Promise<CategoryInterface> {
    let category: any = null;
    const docRef =  this.firestore.collection( path: 'piercings').doc(id);
    category = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return category;
  }

  updateCategory(id: string, data: CategoryInterface): Promise<any> {
    return this.firestore.collection( path: 'categories').doc(id).update(data);
  }
```

Funciones del servicio de Categorias:

- addCategory(category: CategoryInterface): **recibe como parámetros la interfaz de categoría y va devolver una promesa. Básicamente va añadir la categoría en caso de que vaya todo bien.**
- getCategories(): **Obtiene el observable de la colección de categorías ordenándolos por nombre ascendentemente.**
- deleteCategory(id: string): **recibe por parámetros el id de la categoría a eliminar, esta función devolverá una promesa; si todo va bien se eliminará la categoría .**
- getCategory(id: string): **recibe por parámetros el id de la categoría a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicha categoría .**
- updateCategory(id: string, data:CategoryInterface): **recibe por parámetros el id de la categoría a actualizar y la categoría nueva (data). Esta función devolverá una promesa, es decir, actualizará la categoría en caso de que vaya todo bien.**

- **Servicio de usuarios**

```
export class UsersService {
  constructor(private firestore: AngularFirestore) { }

  addUser(user: UserInterface): Promise<any> {
    return this.firestore.collection( path: 'users').add(user);
  }

  getUsers(): Observable<any> {
    return this.firestore.collection( path: 'users', queryFn: ref => ref.orderBy( fieldPath: 'email', directionStr: 'asc')).snapshotChanges();
  }

  deleteUser(id: string): Promise<any> {
    return this.firestore.collection( path: 'users').doc(id).delete();
  }

  getUser(id: string): Promise<UserInterface> {
    let user: any = null;
    const docRef =  this.firestore.collection( path: 'users').doc(id);
    user = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return user;
  }
  updateUser(id: string, data: UserInterface): Promise<any> {
    return this.firestore.collection( path: 'users').doc(id).update(data);
  }
}
```

Funciones del servicio de Usuarios:

- **addUser(user: UserInterface): recibe como parámetros la interfaz de usuario y va devolver una promesa. Básicamente va añadir el usuario en caso de que vaya todo bien.**
- **getUsers(): Obtiene el observable de la colección de usuarios ordenándolos por nombre ascendentemente.**
- **deleteUser(id: string): recibe por parámetros el id del usuario a eliminar, esta función devolverá una promesa; si todo va bien se eliminará al usuario .**
- **getUser(id: string): recibe por parámetros el id del usuario a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicho usuario .**
- **updateUser(id: string, data:UserInterface): recibe por parámetros el id del usuario a actualizar y el usuario nuevo (data). Esta función devolverá una promesa, es decir, actualizará al usuario en caso de que vaya todo bien.**

- **Servicio de empleados**

```
export class EmployeeService {
  constructor(private firestore: AngularFirestore) { }

  addEmployee(employee: EmployeeInterface): Promise<any> {
    return this.firestore.collection( path: 'employees').add(employee);
  }

  getEmployees(): Observable<any> {
    return this.firestore.collection( path: 'employees', queryFn: ref => ref.orderBy( fieldPath: 'name', directionStr: 'asc')).snapshotChanges();
  }

  deleteEmployee(id: string): Promise<any> {
    return this.firestore.collection( path: 'employees').doc(id).delete();
  }

  getEmployee(id: string): Promise<EmployeeInterface> {
    let employee: any = null;
    const docRef =  this.firestore.collection( path: 'employees').doc(id);
    employee = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return employee;
  }

  updateEmployee(id: string, data: any): Promise<any> {
    return this.firestore.collection( path: 'employees').doc(id).update(data);
  }
}
```

Funciones del servicio de Empleados:

- **addEmployee(employee: EmployeeInterface): recibe como parámetros la interfaz de empleado y va devolver una promesa. Básicamente va añadir el empleado en caso de que vaya todo bien.**
- **getEmployees(): Obtiene el observable de la colección de empleados ordenándolos por nombre ascendentemente.**
- **deleteEmployee(id: string): recibe por parámetros el id del empleado a eliminar, esta función devolverá una promesa; si todo va bien se eliminará al empleado.**
- **getEmployee(id: string): recibe por parámetros el id del empleado a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicho empleado.**
- **updateEmployee(id: string, data:EmployeeInterface): recibe por parámetros el id del empleado a actualizar y el empleado nuevo (data). Esta función devolverá una promesa, es decir, actualizará al empleado en caso de que vaya todo bien.**

- **Servicio de citas para los tatuajes**

```typescript
export class RequestsTattoosService {
  constructor(private firestore: AngularFirestore) { }

  addRequestTattoo(requestTatto: RequestTattooInterface): Promise<any> {
    return this.firestore.collection( path: 'requests_tattoo').add(requestTatto);
  }

  getRequestsTattoos(): Observable<any> {
    return this.firestore.collection( path: 'requests_tattoo',  queryFn: ref => ref.orderBy( fieldPath: 'date',  directionStr: 'asc')).snapshotChanges();
  }

  deleteRequestTattoo(id: string): Promise<any> {
    return this.firestore.collection( path: 'requests_tattoo').doc(id).delete();
  }

  getRequestTattoo(id: string): Promise<RequestTattooInterface> {
    let requestTattoo: any = null;
    const docRef =  this.firestore.collection( path: 'requests_tattoo').doc(id);
    requestTattoo = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return requestTattoo;
  }

  updateRequestTattoo(id: string, data: any): Promise<any> {
    return this.firestore.collection( path: 'requests_tattoo').doc(id).update(data);
  }
}
```

Funciones del servicio de Citas de tatuajes:

- **addRequestTattoo(requestTattoo: RequestTattooInterface): recibe como parámetros la interfaz la cita y va devolver una promesa. Básicamente va añadir la cita en caso de que vaya todo bien.**
- **getRequestsTattoos(): Obtiene el observable de la colección de citas ordenándolos por nombre ascendentemente.**
- **deleteRequestTattooid: string): recibe por parámetros el id de la cita a eliminar, esta función devolverá una promesa; si todo va bien se eliminará la cita.**
- **getRequestTattoo(id: string): recibe por parámetros el id de la cita a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicha cita.**
- **updateRequestTattoo(id: string, data:RequestTattooInterface): recibe por parámetros el id de la cita a actualizar y la cita nueva (data). Esta función devolverá una promesa, es decir, actualizará la cita en caso de que vaya todo bien.**

- **Servicio de pedidos**

```
export class CartService {
  constructor(private firestore: AngularFirestore) { }
  addOrder(order: OrderInterface): Promise<any> {
    return this.firestore.collection( path: 'orders').add(order);
  }

  getOrders(): Observable<any> {
    return this.firestore.collection( path: 'orders',  queryFn: ref => ref.orderBy( fieldPath: 'date',  directionStr: 'asc')).snapshotChanges();
  }

  deleteOrder(id: string): Promise<any> {
    return this.firestore.collection( path: 'orders').doc(id).delete();
  }

  getOrder(id: string): Promise<OrderInterface> {
    let order: any = null;
    const docRef =  this.firestore.collection( path: 'orders').doc(id);
    order = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return order;
  }
```

Funciones del servicio de Pedidos:

- **addOrder(order: OrderInterface): recibe como parámetros la interfaz de pedido y va devolver una promesa. Básicamente va añadir el pedidoen caso de que vaya todo bien.**
- **getOrders(): Obtiene el observable de la colección de pedidos ordenándolos por nombre ascendentemente.**
- **deleteOrder(id: string): recibe por parámetros el id del pedido a eliminar, esta función devolverá una promesa; si todo va bien se eliminará al pedido.**
- **getOrder(id: string): recibe por parámetros el id del pedido a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicho pedido.**

- **Servicio de Ubicación de la empresa**

```
export class LocationService {

  constructor(private firestore: AngularFirestore) { }
  getLocations(): Observable<any> {
    return this.firestore.collection( path: 'location').snapshotChanges();
  }
  getLocation(id: string): Promise<LocationInterface> {
    let location: any = null;
    const docRef =  this.firestore.collection( path: 'location').doc(id);
    location = docRef.get().toPromise().then((doc : DocumentSnapshot<unknown> ) => {
      if (doc.exists) {
        console.log('Document data:', doc.data());
        return doc.data();
      } else {
        // doc.data() will be undefined in this case
        console.log('No such document!');
      }
    }).catch((error) => {
      console.log('Error getting document:', error);
    });
    return location;
  }
  updateLocation(id: string, data: LocationInterface): Promise<any> {
    return this.firestore.collection( path: 'location').doc(id).update(data);
  }
}
```

Funciones del servicio de Ubicación de la empresa:

- **getLocations(): Obtiene el observable de la colección de ubicaciones.**
- **getLocation(id: string): recibe por parámetros el id de la ubicación a obtener, esta función devolverá un promesa; si todo va bien obtendrá dicha ubicación.**
- **updateLocation(id: string, data:LocationInterface): recibe por parámetros el id de la ubicación a actualizar y la ubicación nueva (data). Esta función devolverá una promesa, es decir, actualizará la ubiación en caso de que vaya todo bien.**

- **Servicio de Authentication**

```
export class AuthService {
  constructor(public afAuth: AngularFireAuth, private afs: AngularFirestore) { }

  // tslint:disable-next-line:typedef
  async login(email: string, password: string){
    try{
      const result =  await this.afAuth.signInWithEmailAndPassword(email, password);
      return result;
    }
    catch (error){
      return error.message;
    }
  }

  // tslint:disable-next-line:typedef
  async register(email: string, password: string, extraData = null){
    try{
      const result =  await this.afAuth.createUserWithEmailAndPassword(email, password);
      await this.updateUserData(result.user, extraData);
      return result;
    }
    catch (error){
      return error.message;
    }
  }

  // tslint:disable-next-line:typedef
  async logout(){
    try{
      await this.afAuth.signOut();
    }
    catch (error) {
      console.log(error);
    }
  }
```

```
// tslint:disable-next-line:typedef
getCurrentUser(){
  return this.afAuth.authState.pipe(first()).toPromise();
}

// tslint:disable-next-line:typedef
 updateUserData(user, extraData = null){
  const userRef: AngularFirestoreDocument<any> = this.afs.doc( path: `users/${user.uid}`);
  const data: UserInterface = {
    id: user.uid,
    email: user.email,
    roles: {
      client: true
    },
    country: extraData.country,
    city: extraData.city,
    street: extraData.street,
    zip_code: extraData.zip_code,
    photo: extraData.photo
  };
  return userRef.set(data, {merge: true});
}
// tslint:disable-next-line:typedef
isAuth(){
  return this.afAuth.authState.pipe(map( project: auth => auth));
}

// tslint:disable-next-line:typedef
isUserAdmin(userUid){
  return this.afs.doc<UserInterface>( path: `users/${userUid}`).valueChanges();
}
```

Para este servicio necesitamos AngularFireAuth y AngularFirestore

Funciones del servicio:

- **async login(email: string, password): recibe por parámetros el email y el password. Esta función asíncrona permitirá loguear a un usuario mediante Firebase con el email y contraseña pasados por parámetros y devolverá dicho usuario si va todo bien, en caso contrario, devolverá el mensaje de error.**
- **async register(email: string, password: string, extraData = null): recibe por parámetros el email, password y una variable opcional que por defecto es nula. Esta función asíncrona permitirá registrar a un usuario mediante Firebase con el email y contraseña pasados por parámetros, además al usar la función updateUserData creará en firebase al mismo usuario para la gestión de roles y con los posibles datos extra. La función devolverá dicho usuario si va todo bien, en caso contrario, devolverá el mensaje de error.**
- **async logout(): Esta función asíncrona permitirá desloguearse la cuenta actual logueada con firebase si todo va bien, en caso contrario, saltará el catch y se mostrará por consola el mensaje de error.**
- **getCurrentUser(): Esta función retorna al usuario actual de Firebase.**
- **updateUserData(user, extraData = null): recibe por parámetros al user y el extraData que es opcional con valor por defecto nulo. Esta función crea en Firestore al usuario con roles por defecto cliente y con los datos extra. Retorna a dicho usuario.**
- **isAuth(): comprueba si hay un usuario autenticado actualmente en firebase y retorna el observable.**
- **isUserAdmin(userUid): recibe por parámetros el id de un usuario y retorna el observable de dicho usuario. Esta funcion se usa para comprobar si tiene el rol de admin.**

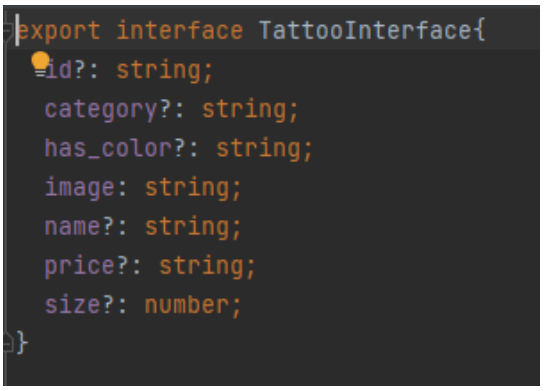
Intefaces / Modelos

Las Interfaces / Modelos que fuimos creando según lo íbamos necesitando.

Cada interfaz/modelo lo guardamos en una carpeta llamada models separados en distintos ficheros typescript. Básicamente es una representación de la colección de la base datos

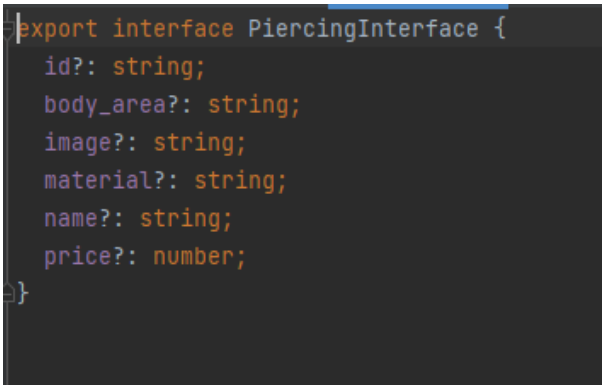
han sido los siguientes:

- **Interfaz Tattoo:**



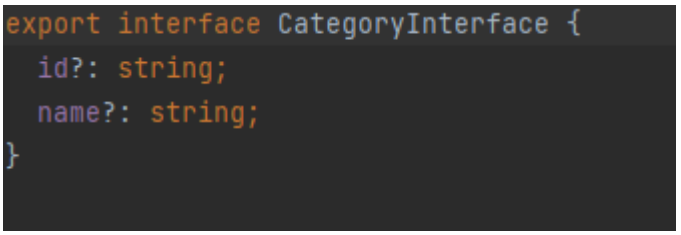
```
export interface TattooInterface{
  id?: string;
  category?: string;
  has_color?: string;
  image: string;
  name?: string;
  price?: string;
  size?: number;
}
```

- **Interfaz de Piercing**



```
export interface PiercingInterface {
  id?: string;
  body_area?: string;
  image?: string;
  material?: string;
  name?: string;
  price?: number;
}
```

- **Interfaz de Category**



```
export interface CategoryInterface {
  id?: string;
  name?: string;
}
```

- **Interfaz de User/Roles**

```
export interface Roles {
  client?: boolean;
  admin?: boolean;
}

export interface UserInterface{
  id?: string;
  name?: string;
  email: string;
  password?: string;
  roles: Roles;
  city?: string;
  country?: string;
  id_user?: string;
  photo?: string;
  street?: string;
  zip_code?: string;
}
```

- **Interfaz de Order**

```
export interface OrderInterface {
  id?: string;
  date?: string;
  userId: string;
  products?: [];
}
```

- **Interfaz de Employee**

```
export interface EmployeeInterface {
  id?: string;
  email?: string;
  age?: string;
  name?: string;
  photo?: string;
}
```

- **Interfaz de location**

```
export interface LocationInterface {
  latitude?: number;
  longitude?: number;
}
```

- **Interfaz de RequestTattoo**

```
export interface RequestTattooInterface {
  id?: string;
  date?: string;
  hour?: string;
  idTattoo?: string;
  idUser?: string;
  photo?: string;
  price?: string;
}
```

Vistas / Controladores

ejemplo list / create

Ahora vamos de los componentes que forman nuestro proyecto:

- **componente list-tattos:**

  list-tattos.component.ts

```
export class ListTattosComponent implements OnInit {

  tattoos: TattooInterface[] = [];
  isAdmin: any = null;
  userUid: string = null;
  filterTattoo = '';
  logged: any = null;
  // Pagination
  page = 1;
  pageSize = 4;

  // tslint:disable-next-line:variable-name
  constructor(private _tattoosService: TattosService,
              private authService: AuthService) {
  }

  ngOnInit(): void {
    this.getCurrentUser();
    this.getTattoos();
  }

  // tslint:disable-next-line:typedef
  getCurrentUser(){
    this.authService.isAuth().subscribe( next: auth => {
      if (auth){
        this.logged = true;
        this.userUid = auth.uid;
        this.authService.isUserAdmin(this.userUid).subscribe( next: userRole => {
          this.isAdmin = Object.assign( target: {}, userRole.roles).hasOwnProperty( v: 'admin');
        });
      }
    });
  }
```

```
getTattoos() {
  this._tattoosService.getTattoos().subscribe( next: data => {
    this.tattoos = [];
    data.forEach((element: any) => {
      this.tattoos.push({
        id: element.payload.doc.id,
        ...element.payload.doc.data()
      });
    });
  });
}

// tslint:disable-next-line:typedef
deleteTattoo(id: string) {
  Swal.fire( options: {
    title: 'Are you sure?',
    text: 'You won\'t be able to revert this!',
    icon: 'warning',
    showCancelButton: true,
    confirmButtonColor: '#3085d6',
    cancelButtonColor: '#d33',
    confirmButtonText: 'Yes, delete it!'
  }).then((result : SweetAlertResult<Awaited<any>> ) => {
    if (result.isConfirmed) {
      this._tattoosService.deleteTattoo(id).then(() => {
      }).catch(error => {
        console.log(error);
      });
      Swal.fire(
        title: 'Deleted!',
        html: 'Your file has been deleted.',
        icon: 'success'
      );
    }
  });
}
```

funciones del componente:

- **getCurrentUser(): Comprueba si está autenticado el usuario actual, obtiene su id y comprueba si es admin.**
- **getTattoos(): Lee los tattoos y los asigna al array de tattoos declarado.**
- **deleteTattoo(id: string): Borra el tattoo con el id pasado por parámetros si todo va bien, en caso contrario, mostrara el mensaje de error. Usa la libreria Swal que es al lanzar la funcion saltará una ventana emergente visualmente bonita que preguntará si estamos seguros de esta acción.**

list-tattos.component.html

```
<div class="container tattoos">
  <div class="card" *ngIf="isAdmin; else client" style="margin-top: 120px">
    <div class="card-body">
      <span class="h3">Listado de Tattoos</span>
      <button class="btn btn-primary btn-lg float-right" routerLink="/create-tattoo" *ngIf="isAdmin">Agregar</button>
      <h5 style="..." *ngIf="tattoos.length == 0">No hay datos para mostrar</h5>
      <table *ngIf="tattoos.length > 0" class="table table-striped mt-5">
        <thead>
          <tr>
            <th>Id</th>
            <th>Image</th>
            <th>Name</th>
            <th>Size</th>
            <th>Color</th>
            <th>Category</th>
            <th>Price</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let tattoo of tattoos | slice: start: (page-1) * pageSize : end: (page-1) * pageSize + pageSize">
            <td style="...">{{ tattoo.id }}</td>
            <td>
              <img [src]="tattoo.image ? tattoo.image : 'assets/bootstrap.png'" width="150" height="150">
            </td>
            <td style="...">{{ tattoo.name }}</td>
            <td style="...">{{ tattoo.size }}</td>
            <td style="...">{{ tattoo.has_color }}</td>
            <td style="...">{{ tattoo.category }}</td>
            <td style="...">{{ tattoo.price | currency}}</td>
            <td style="...">
              <i style="..." class="fas fa-edit fa-lg text-info mr-2" [routerLink]="['/editTattoo/', tattoo.id]"></i>
              <i style="..." (click)="deleteTattoo(tattoo.id)" class="fas fa-trash-alt fa-lg text-danger"></i>
            </td>
          </tr>
        </tbody>
      </table>
    <ngb-pagination style="..."
                    [collectionSize]="100" [(page)]="page" [maxSize]="5" [rotate]="true" [ellipses]="false" [boundaryLinks]="true"></ngb-pagination>
```

```
    </div>
  </div>
  <ng-template #client>
    <h1>Nuestros Tatuajes</h1>
    <h5 style="..." *ngIf="tattoos.length == 0">No hay datos para mostrar</h5>
    <div class="form-group">
      <input type="text" class="form-control" placeholder="Search..." name="filterTattoo" [(ngModel)]="filterTattoo">
    </div>
    <div class="card-columns">
    <div class="card" style="..." *ngFor="let tattoo of tattoos | filterName:filterTattoo">
      <img class="card-img-top" [src]="tattoo.image ? tattoo.image : 'assets/bootstrap.png'" width="150" height="250">
      <div class="card-body">
        <h4 class="card-title">{{tattoo.name}}</h4>
        <p class="card-text">{{tattoo.category}}</p>
        <p class="card-text">{{tattoo.price}}€</p>
        <a [routerLink]="['/request-tattoo/', tattoo.id]" class="btn btn-primary" *ngIf="logged">Encargar</a>
      </div>
    </div>
    </div>
  </ng-template>
</div>
```

Esta vista tiene dos partes:

- **vista admin: Mostrará una tabla con el listado de tattoos, la paginación y las opciones de crear, borrar, editar**
- **vista cliente:Mostrara en cards los tattoos con sus respectivos datos y si esta logueado la opción de encargar tattoo. También tendrá la posibilidad el cliente de filtrar los tattoos por el nombre.**

- **componente create-tatto**

  create-tatto.component.ts

```
export class CreateTattoComponent implements OnInit {
  categories: CategoryInterface[] = [];
  createTattoo: FormGroup;
  submitted = false;
  loading = false;
  id: string | null;
  titulo = 'Add Tattoo';

  constructor(private fb: FormBuilder,
              // tslint:disable-next-line:variable-name
              private _tattoosService: TattosService,
              // tslint:disable-next-line:variable-name
              private _categoriesService: CategoriesService,
              private router: Router,
              private aRoute: ActivatedRoute) {
    this.createTattoo = this.fb.group( controlsConfig: {
      image: ['', Validators.required],
      name: ['', Validators.required],
      size: ['', Validators.required],
      has_color: [''],
      category: ['', Validators.required],
      price: ['', Validators.required],
    });
    this.id = this.aRoute.snapshot.paramMap.get('id');
  }

  // tslint:disable-next-line:typedef
  getCategories() {
    this._categoriesService.getCategories().subscribe( next: data => {
      this.categories = [];
      data.forEach((element: any) => {
        this.categories.push({
          id: element.payload.doc.id,
          ...element.payload.doc.data()
        });
      });
    });
  }
```

```
ngOnInit(): void {
  this.getCategories();
  this.isEdit();
}

// tslint:disable-next-line:typedef
addEditTattoo() {
  this.submitted = true;
  if (this.createTattoo.invalid) {
    return;
  }

  if (this.id === null) {
    this.addTattoo();
  } else {
    this.editTattoo(this.id);
  }

}

// tslint:disable-next-line:typedef
addTattoo() {
  const tattoo: TattooInterface = {
    category: this.createTattoo.value.category,
    has_color: this.createTattoo.value.has_color ? this.createTattoo.value.has_color : false,
    image: this.createTattoo.value.image,
    name: this.createTattoo.value.name,
    price: this.createTattoo.value.price,
    size: this.createTattoo.value.size,
  };
  this.loading = true;
  this._tattoosService.addTattoo(tattoo).then(() => {
    this.loading = false;
    this.router.navigate( commands: ['/list-tattoos']);
  }).catch(error => {
    console.log(error);
    this.loading = false;
  });
}
```

```
editTattoo(id: string) {
  const tattoo: TattooInterface = {
    category: this.createTattoo.value.category,
    has_color: this.createTattoo.value.has_color ? this.createTattoo.value.has_color : false,
    image: this.createTattoo.value.image,
    name: this.createTattoo.value.name,
    price: this.createTattoo.value.price,
    size: this.createTattoo.value.size,
  };

  this.loading = true;

  this._tattoosService.updateTattoo(id, tattoo).then(() => {
    this.loading = false;
    this.router.navigate( commands: ['/list-tattoos']);
  });
}


// tslint:disable-next-line:typedef
isEdit() {
  if (this.id !== null) {
    this.titulo = 'Edit Tattoo';
    this.loading = true;
    this._tattoosService.getTattoo(this.id).then(data => {
      this.loading = false;
      this.createTattoo.setValue( value: {
        image: data.image,
        name: data.name,
        size: data.size,
        has_color: data.has_color ? data.has_color : false,
        category: data.category,
        price: data.price
      });
    });
  }
}
}
```

funciones del componente:

- **getCategories(): Obtiene las categorías y las asigna al array de categorías.**
- **addEditTattoo(): Si esta funcion se ejecuta submitted se iguala a true, si existe un id de un tattoo en la ruta que comprobamos en el constructor se ejecuta la función editTattoo, en caso contrario se ejecuta addTattoo().**
- **addTattoo(): Crea un tattoo con los datos recogidos del formulario, en caso de que vaya todo bien se añade y redirige la pagina al listado de tattoos, en caso contrario se mostrará por consola el error.**
- **editTattoo(id: string): actualiza el tattoo con el id pasado por parámetros por los datos del formulario.**
- **isEdit(): si el id es distinto de nulo setea el formulario con los datos obtenidos del tatto con ese id.**

create-tatto-component.html

```
<div class="container pt-4" style="margin-top: 120px">
  <div class="row">
      <div class="col-lg-6 offset-lg-3">
          <div class="card text-center">
              <div class="card-body">
                  <h3>{{ titulo }}
                      <div *ngIf="loading" class="spinner-border float-right" role="status">
                          <span class="sr-only">Loading...</span>
                      </div>
                  </h3>
                  <span *ngIf="submitted && createTattoo.invalid" class="badge badge-danger">INTRODUZCA LOS DATOS CORRECTAMENTE</span>
                  <form class="mt-4" [formGroup]="createTattoo" (ngSubmit)="addEditTattoo()">

                          <div class="form-group">
                            <label for="image">Image</label>
                            <input type="text" class="form-control form-control-lg" id="image" formControlName="image"
                                   placeholder="Image">
                          </div>
                          <div class="form-group">
                              <label for="name">Name</label>
                              <input type="text" class="form-control form-control-lg" id="name" formControlName="name"
                                  placeholder="Name">
                          </div>

                      <div class="form-group">
                        <label for="size">Size</label>
                        <input type="number" class="form-control form-control-lg mt-3" id="size" formControlName="size"
                           placeholder="Size">
                      </div>

                      <div class="form-group">
                        <label for="size">Price</label>
                        <input type="number" class="form-control form-control-lg mt-3" id="price" formControlName="price"
                        placeholder="Price">
                      </div>
```

```
                    <div class="form-group" wfd-id="356">
                      <label for="exampleSelect1" wfd-id="358">Category</label>
                      <select class="form-control" id="exampleSelect1" wfd-id="357" formControlName="category">
                        <option *ngFor="let category of categories" value="{{category.name}}">{{category.name}}</option>
                      </select>
                    </div>

                   <div class="form-check">
                      <input class="form-check-input" type="checkbox" value="" id="flexCheckDefault" formControlName="has_color">
                      <label class="form-check-label" for="flexCheckDefault">
                        Con color
                      </label>
                    </div>

                    <br><br>


                    <div class="mt-3">
                        <button type="text" class="btn btn-secondary btn-lg mr-3"
                            routerLink="/list-tattoos">Volver</button>
                        <button type="submit" class="btn btn-primary btn-lg">{{titulo}}</button>
                    </div>
                  </form>
              </div>
          </div>
      </div>
  </div>
</div>
```

Esta vista es un formulario con doble cara edicion o creaccion de tattoos. Como direrente al resto de creates tiene el combobox de categorías.

No voy mostrar el resto de list y creates porque son similares.

Component Location:

- **location.component.ts**

```
export class LocationComponent implements OnInit {
  userUid: string;
  isAdmin: boolean;
  lat: number;
  lng: number;
  zoom: number;
  mapTypeId: string;
  location: LocationInterface;

  locationForm: FormGroup;
  loading = false;
  id: string | null;

  constructor(private locationService: LocationService,
              private fb: FormBuilder,
              private userService: UsersService,
              private router: Router,
              private authService: AuthService,
              private aRoute: ActivatedRoute) {
    this.locationForm = this.fb.group( controlsConfig: {
      latitude: [0],
      longitude: [0],
    });
    this.id = this.aRoute.snapshot.paramMap.get('id');

  }

  ngOnInit(): void {
    this.getCurrentUser();
    this.getLocation();
    this.isEdit();
  }
```

```
editLocation(): void {

    const location: LocationInterface = {
      latitude:  this.locationForm.value.latitude ? this.locationForm.value.latitude : 0,
      longitude: this.locationForm.value.longitude ? this.locationForm.value.longitude : 0
    };

    this.loading = true;

    this.locationService.updateLocation(this.id, location).then(() => {
      this.loading = false;
      this.router.navigate( commands: ['/home']);
    });
}

isEdit(): void {
  this.loading = true;
  this.locationService.getLocation(this.id).then((data :LocationInterface ) => {
    this.loading = false;
    this.locationForm.setValue( value: {
      latitude: data.latitude ? data.latitude : 0,
      longitude: data.longitude ? data.longitude : 0
    });
  });
}
getCurrentUser(): void{
  this.authService.isAuth().subscribe( next: auth => {
    if (auth){
      this.userUid = auth.uid;
      this.authService.isUserAdmin(this.userUid).subscribe( next: userRole => {
        this.isAdmin = Object.assign( target: {}, userRole.roles).hasOwnProperty( v: 'admin');
      });
    }
  });
}
```

```
getLocation(): void {
  if (this.id === null) {
    this.locationService.getLocations().subscribe( next: data => {
      this.location = data[0].payload.doc.data();
      this.lat = this.location.latitude;
      this.lng = this.location.longitude;
      this.zoom = 16;
      this.mapTypeId = 'hybrid';
    });
  } else {
    this.locationService.getLocation(this.id).then((data :LocationInterface ) => {
      this.location = data;
      this.lat = data.latitude;
      this.lng = data.longitude;
      this.zoom = 16;
      this.mapTypeId = 'hybrid';
    });
  }
}
// tslint:disable-next-line:typedef
getCurrentPosition(){
  navigator.geolocation.getCurrentPosition( successCallback: position => {
    this.lat =  position.coords.latitude;
    this.lng = position.coords.longitude;
    this.zoom = 57;
    this.mapTypeId = 'hybrid';
  });
}
}
```

funciones del componente:

- **getCurrentUser(): Comprueba si está autenticado el usuario actual, obtiene su id y comprueba si es admin.**
- **editLocation(): actualiza la ubicación con el id obtenido de la ruta por los datos del formulario.**
- **isEdit(): si hay una ubicación con el id de la ruta setea el formulario con los datos obtenidos de la ubicación con ese id.**
- **getLocation(): si el id es nulo obtiene las ubicaciones de la firestore y coge el 1, en caso contrario obtiene la ubicación con el id pasado por ruta.**
- **getCurrentPosition(): Es una función que al final no ha tenido uso pero la hemos dejado porque es interesante te calcula la ubicación tuya y la setea en el mapa.**

- **location.component.html**

  Esta vista se compone de dos partes:

  - **vista cliente:**

```
<div class="container location">
  <h1> Nuestra ubicación</h1><br><br>

  <agm-map [latitude]="lat"
           [longitude]="lng"
           [zoom]="zoom"
           [mapTypeId]="mapTypeId" *ngIf="!isAdmin; else Admin">
    <agm-marker [latitude]="lat"
                [longitude]="lng">
      <agm-info-window>
        Mi posición
      </agm-info-window>
    </agm-marker>
  </agm-map>
```

Utilizamos la api de google para mostrar el mapa.

vista admin:

```
<ng-template #Admin style="margin-top: 120px">
  <div class="container pt-4">
    <div class="row">
      <div class="col-lg-6 offset-lg-3">
        <div class="card text-center">
          <div class="card-body">
            <h3>Location company
              <div *ngIf="loading" class="spinner-border float-right" role="status">
                <span class="sr-only">Loading...</span>
              </div>
            </h3>
            <span *ngIf="locationForm.invalid" class="badge badge-danger">INTRODUZCA LOS DATOS CORRECTAMENTE</span>
            <form class="mt-4" [formGroup]="locationForm" (ngSubmit)="editLocation()">

              <div class="form-group">
                <label for="latitude">Latitude</label>
                <input type="number" class="form-control form-control-lg" id="latitude" formControlName="latitude"
                       placeholder="Latitude">
              </div>
              <div class="form-group">
                <label for="longitude">Longitude</label>
                <input type="number" class="form-control form-control-lg" id="longitude" formControlName="longitude"
                       placeholder="Longitude">
              </div>

              <br><br>


              <div class="mt-3">
                <button type="text" class="btn btn-secondary btn-lg mr-3"
                        routerLink="/home">Volver</button>
                <button type="submit" class="btn btn-primary btn-lg">Actualizar datos</button>
              </div>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>

</ng-template>
```

En la vista admin mostramos un formulario con la latitud y la longitud que se mostrará en el mapa, donde podremos editar dichos valores.

Component Cart:

- **cart.component.ts**

```
export class CartComponent implements OnInit {
  constructor(private cartService: CartService,
              private userService: UsersService,
              private authService: AuthService) { }
  piercingsCart: OrderInterface[] = [];
  total = 0;
  userUid: string;
  emailUser = '';
  user: UserInterface = null;
  logoDataUrl: string;

  ngOnInit(): void {
    this.getCurrentUser();
    this.getOrders();
    Utils.getImageDataUrlFromLocalPath1( localPath: 'assets/descarga.png').then(
      result => this.logoDataUrl = result
    );
  }
  // tslint:disable-next-line:typedef
  getCurrentUser() {
    this.authService.isAuth().subscribe( next: auth => {
      if (auth) {
        this.userUid = auth.uid;
        this.emailUser =  auth.email;
        this.userService.getUser(this.userUid).then((data : UserInterface ) => {
          this.user = data;
        });
      }
    });
  }
```

```
  getOrders(): void {
    this.cartService.getOrders().subscribe( next: data => {
      this.piercingsCart = [];
      data.forEach((element: any) => {
        if (element.payload.doc.data().userId === this.userUid) {
          this.piercingsCart.push({
            id: element.payload.doc.id,
            ...element.payload.doc.data()
          });
        }
      });
    });
  }
  calcularTotalOrder(order: OrderInterface): number{
    let total = 0;
    for (const piercing of order.products){
      total += piercing.price;
    }
    return total;
  }
```

```
confirmOrder(order: OrderInterface){
  this.generarPDF(order);
  const Toast = Swal.mixin( options: {
    toast: true,
    position: 'bottom-left',
    showConfirmButton: false,
    timer: 3000,
    width: 600,
    timerProgressBar: true,
    didOpen: (toast : HTMLElement ) => {
      toast.addEventListener( type: 'mouseenter', Swal.stopTimer);
      toast.addEventListener( type: 'mouseleave', Swal.resumeTimer);
    }
  });

  Toast.fire( options: {
    icon: 'success',
    title: 'Order confirmed'
  });
}
deleteOrder(id: string): void{
  Swal.fire( options: {
    title: 'Are you sure?',
    text: 'You won\'t be able to revert this!',
    icon: 'warning',
    showCancelButton: true,
    confirmButtonColor: '#3085d6',
    cancelButtonColor: '#d33',
    confirmButtonText: 'Yes, delete it!'
  }).then((result : SweetAlertResult<Awaited<any>> ) => {
    if (result.isConfirmed) {
      this.cartService.deleteOrder(id).then(() => {
      }).catch((error) => {
        console.log(error);
      });
      Swal.fire(
         title: 'Deleted!',
         html: 'Your file has been deleted.',
         icon: 'success'
```

```
generarPDF(order: OrderInterface) {
  let products = [['Name', 'Material', 'Body area', 'Price']];
  for (let piercing of order.products){
    products.push([piercing.name, piercing.material, piercing.body_area, String(piercing.price) + '$']);
  }
  const documentDefinition = {
    content: [
      {
       image: this.logoDataUrl,
       alignment: 'center',
      },
      'User: ' + this.emailUser,
      'Date: ' + new Date(order.date).toLocaleDateString(),
      {
        table: {
          // headers are automatically repeated if the table spans over multiple pages
          // you can declare how many rows should be treated as headers
          headerRows: 1,
          widths: ['*', 'auto', 100, '*'],

          body: products
        }
      },
      { text: 'Total Price: ' + this.calcularTotalOrder(order) + '$', alignment: 'right' },
      { text: 'Country: ' + this.user.country},
      { text: 'City: ' + this.user.city},
      { text: 'Street: ' + this.user.street},
      { text: 'Zip code: ' + this.user.zip_code},
    ],
  };
  pdfMake.createPdf(documentDefinition).open();
}
mostrarTabla(idOrder: string): void{
  const table = document.getElementById(idOrder);
  if (table.style.display === 'none') {
    table.style.display = '';
  } else {
    table.style.display = 'none';
  }
```

funciones del componente:

- **getCurrentUser(): Comprueba si está autenticado el usuario actual y en caso de que lo esté lo obtiene.**
- **getOrders(): Obtiene todos los pedidos que tiene el usuario actual.**
- **calcularTotalOrder(order: OrderInterface): Obtiene la suma del precio de todos los productos del pedido.**
- **confirmOrder(order: OrderInterface): imprime el pedido y confirma mediante un toast el pedido**
- **deleteOrder(id: string): Elimina el pedido con el id pasado por parámetros aunque antes pide confirmación mediante una alerta de la librería Swal.**
- **generarPdf(order: OrderInterface): Imprime el pedido con la libreria pdfMake.**
- **mostrarTabla(idOrder: string): muestra la tabla del pedido con id al pasado por parametros.**

- **cart.component.html**

```
<div class="container" style="margin-top: 200px" id="htmlData">
  <div class="card">
    <div class="card-body">
      <span class="h1">Pedidos</span><br>
      <h5 style="..." *ngIf="piercingsCart.length == 0">No hay pedidos para mostrar</h5>
      <br><br>
      <div *ngFor="let order of piercingsCart; let i=index">
        <div (click)="mostrarTabla(order.id)" style="cursor: pointer; background: goldenrod"><h3>{{order.date | date: format: 'dd/MM/YYYY'}}</h3></div
        <table *ngIf="piercingsCart.length > 0" class="table table-striped mt-5" id="{{order.id}}" style="...">
          <thead>
            <tr>
              <th>Image</th>
              <th>Name</th>
              <th>Material</th>
              <th>Area</th>
              <th>Price</th>
            </tr>
          </thead>
          <tbody id="contenido">
            <tr *ngFor="let piercing of order.products">
              <td>
                <img [src]="piercing.image ? piercing.image : 'assets/bootstrap.png'" width="150" height="150">
              </td>
              <td style="...">{{ piercing.name }}</td>
              <td style="...">{{ piercing.material }}</td>
              <td style="...">{{ piercing.body_area }}</td>
              <td style="...">{{ piercing.price | currency}}</td>
            </tr>
            <tr>
              <th class="text-right" style="...">Precio total: {{calcularTotalOrder(order)|currency}}</th>
              <td>
                <button class="btn btn-primary" *ngIf="piercingsCart.length" (click)="confirmOrder(order)">Imprimir pedido</button>
                <button class="btn btn-danger" style="..." *ngIf="piercingsCart.length" (click)="deleteOrder(order.id)">Eliminar pedido</button>
              </td>
            </tr>
          </tbody>
        </table>
        <br><br>
      </div>
    </div>
```

Básicamente esta vista recorre los pedidos y muestra en distintas tablas los productos separados por fechas cuando haces click a la fecha se expande y ves los productos y se puede Imprimir o eliminar pedido en los respectivos botones.

Component Home:

home.component.ts:

```
export class HomeComponent implements OnInit {
  userUid: string;
  isAdmin: boolean;
  constructor(private authService: AuthService,
              private router: Router) { }

  ngOnInit(): void {
    this.getCurrentUser();
    Scrollbar.init(document.querySelector( selectors: '#my-scrollbar'));
  }
  getCurrentUser(): void {
    this.authService.isAuth().subscribe( next: auth => {
      if (auth) {
        this.userUid = auth.uid;
        this.authService.isUserAdmin(this.userUid).subscribe( next: userRole => {
          this.isAdmin = Object.assign( target: {}, userRole.roles).hasOwnProperty( v: 'admin');
          if (this.isAdmin) {
            this.router.navigate( commands: ['/list-tattoos']);
          }
        });
      }
    });
  }
}
```

Este componente usa una librería `smooth-scrollbar` para el uso de un Scroll Smooth, que se inicializa en el ngOnInit. Luego como en varios componentes usamos el getCurrentUser() para obtener datos del usuario actual.

home.component.html:

```
<div class="home" id="my-scrollbar">
<app-slider></app-slider>
    <section class="section-list" id="tattoos">
      <app-list-tattos></app-list-tattos>
    </section>

    <section class="section-list" id="piercings">
      <app-list-piercings></app-list-piercings>
    </section>

    <section class="section-location" id="location">
      <app-location></app-location>
    </section>

    <section class="section-about" id="about">
      <app-about></app-about>
    </section>
</div>
```

En el html tenemos el slider y el resto listados separados en secciones

Component Navbar:

navbar.component.ts:

```
export class NavbarComponent implements OnInit {

  public isLogged =  false;
  public user$: Observable<any> = this.authService.afAuth.user;
  private userUid: string;
  photo: string;
  userS: UserInterface;
  isAdmin: any = null;
  idLocation: string;

  constructor(private authService: AuthService,
              private usersService: UsersService,
              private cartService: CartService,
              private locationService: LocationService,
              private router: Router) { }

  // tslint:disable-next-line:typedef
  async ngOnInit(){
    this.getLocation();
    this.getCurrentUser();
  }
```

```
  getCurrentUser(){
    this.authService.isAuth().subscribe( next: auth => {
     if (auth){
       this.userUid = auth.uid;
       this.usersService.getUser(this.userUid).then((data : UserInterface ) => {
         this.userS = data;
         this.photo = this.userS.photo;
       });
       this.authService.isUserAdmin(this.userUid).subscribe( next: userRole => {
         this.isAdmin = Object.assign( target: {}, userRole.roles).hasOwnProperty( v: 'admin');
       });
     }
    });
  }

  getLocation(): void {
    this.locationService.getLocations().subscribe( next: data => {
      this.idLocation = data[0].payload.doc.id;
    });
  }
```

```
async onLogout(){
  try{

    await this.authService.logout();
    // tslint:disable-next-line:no-unused-expression
    this.router.navigate['/login'];

  } catch (error){console.log(error); }
}

toTattos(): void {
  const dom  = document.getElementById( elementId: 'my-scrollbar');
  if (dom === null){
    this.router.navigate( commands: ['/home']);
  }
  const scrollbar = Scrollbar.init(dom);
  scrollbar.scrollTo( x: 0,  y: 850,  duration: 600);
}

toPiercings(): void{
  const dom  = document.getElementById( elementId: 'my-scrollbar');
  if (dom === null){
    this.router.navigate( commands: ['/home']);
  }
  const scrollbar = Scrollbar.init(dom);
  scrollbar.scrollTo( x: 0,  y: 2080,  duration: 600);
}
```

```
toLocation(): void {
  const dom  = document.getElementById( elementId: 'my-scrollbar');
  if (dom === null){
    this.router.navigate( commands: ['/home']);
  }
  const scrollbar = Scrollbar.init(dom);
  scrollbar.scrollTo( x: 0,  y: 3780,  duration: 600);
}

toAbout(): void{
  const dom  = document.getElementById( elementId: 'my-scrollbar');
  if (dom === null){
    this.router.navigate( commands: ['/home']);
  }
  const scrollbar = Scrollbar.init(dom);
  scrollbar.scrollTo( x: 0,  y: 4900,  duration: 600);
}
```

Utilizamos el getCurrentUser para mostrar o no en el navbar lo que es de cliente y lo que es de admin. La función getLocation() nos sirve para obtener el id de la ubicación que necesitamos para luego poder editarla. La funcion onLogout() como el propio nombre indica nos sirve para desloguearnos y las funciones toTattoos(), toPiercings(),toLocation() y toAbout utilizamos la librería anteriormente dicha para desplazarnos en el componente a esa sección.

navbar.component.html

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <a class="navbar-brand" href="#"><img src="assets/logo2.png" height="85" width="100"></a>
  <button class="navbar-toggler collapsed" type="button" data-toggle="collapse" data-target="#navbarColor01" aria-c
    <span class="navbar-toggler-icon" wfd-id="445"></span>
  </button>

  <div class="navbar-collapse collapse" id="navbarColor01" wfd-id="444">
    <ul class="navbar-nav mr-auto" wfd-id="624">
        <li class="nav-item" wfd-id="629" *ngIf="!isAdmin; else noAdmin2">
        <a class="nav-link" style="cursor: pointer" (click)="toTattos()">Tattoos</a>
      </li>
      <ng-template #noAdmin2>
        <li class="nav-item" wfd-id="629">
          <a class="nav-link" style="cursor: pointer" routerLink="/list-tattoos">Tattoos</a>
        </li>
      </ng-template>
      <li class="nav-item" wfd-id="629" *ngIf="!isAdmin; else noAdmin">
        <a class="nav-link" style="cursor: pointer" (click)="toPiercings()">Piercings</a>
      </li>
      <ng-template #noAdmin>
        <li class="nav-item" wfd-id="629">
          <a class="nav-link" style="cursor: pointer" routerLink="/list-piercings">Piercings</a>
        </li>
      </ng-template>
      <li class="nav-item" wfd-id="628">
        <a class="nav-link" *ngIf="!isAdmin" style="cursor: pointer" (click)="toLocation()">Location</a>
      </li>
      <li class="nav-item" wfd-id="628">
        <a class="nav-link" *ngIf="isAdmin" style="..." [routerLink]="['/location/', idLocation]">Location</a>
      </li>
      <li class="nav-item" wfd-id="628">
        <a class="nav-link" *ngIf="!isAdmin" style="..." (click)="toAbout()">About</a>
      </li>

      <li class="nav-item" wfd-id="628">
        <a class="nav-link" *ngIf="isAdmin" routerLink="/list-categories">Categories</a>
      </li>
      <li class="nav-item" wfd-id="628">
        <a class="nav-link" *ngIf="isAdmin" routerLink="/list-users">Users</a>
      </li>
```

```
  <li class="nav-item" wfd-id="628">
    <a class="nav-link" routerLink="/threejs">Tattoo Perry</a>
  </li>
  <li>
  <li class="nav-item" wfd-id="628">
    <a class="nav-link" *ngIf="isAdmin" routerLink="/list-employees">Employees</a>
  </li>
</ul>

<ul class="navbar-nav ml-auto" *ngIf="user$ | async as user; else login">

  <li class="nav-item" wfd-id="628" *ngIf="user">
    <a class="nav-link" href="#" (click)="onLogout()">Logout</a>
  </li>

  <li class="nav-item">
    <a  style="cursor: pointer" [routerLink]="['/user-profile/', userS.id]">
      <img *ngIf="userS.photo; else noPhoto" src="{{userS.photo}}" width="50" height="40">
      <ng-template #noPhoto>
        <img src="assets/user.png" width="40">
      </ng-template>
    </a>
  </li>
  <li class="nav-item">
    <a class="nav-link" *ngIf="!isAdmin" routerLink="/list-requests-tattoos">
      <i class="far fa-calendar-check">
      </i>
    </a>
  </li>
  <li class="nav-item">
    <a class="nav-link" routerLink="/cart" *ngIf="!isAdmin" >
      <i class="fas fa-shopping-cart">
      </i>
    </a>
  </li>
</ul>
```

```
    <ng-template #login>
      <ul class="navbar-nav ml-auto">
        <li class="nav-item" wfd-id="628">
          <a class="nav-link" routerLink="/login">Login</a>
        </li>
        <li class="nav-item" wfd-id="628">
          <a class="nav-link" routerLink="/register">Sign Up</a>
        </li>
      </ul>
    </ng-template>

  </div>
</nav>
```

Básicamente esta vista muestra las opciones que tiene el usuario ya sea cliente, no logueado, admin.

Component Slider:

Este es sin duda el componente más básico

slider.component.ts:

```
export class SliderComponent  {
  images = [
    {path: 'assets/tattobanner01.jpg'},
    {path: 'assets/tattobanner02.jpg'},
    {path: 'assets/tattobanner03.jpg'}
  ];

}
```

Usa un array de 3 imagenes para cargarlas en el carousel

slider.component.html:

```
<carousel
  [images]="images"
  [objectFit]="'cover'"
  [cellWidth]="'100%'"
  [height]="700"
  [autoplay]="true"
  [dots]="true"
>
</carousel>
```

El carousel es sacado de la libreria `angular-responsive-carousel`

Recursos utilizados para su desarrollo.

Ordenador sobremesa, portátil, alojamiento en la nube con firebase y librerías libres que hemos ido añadiendo en angular.