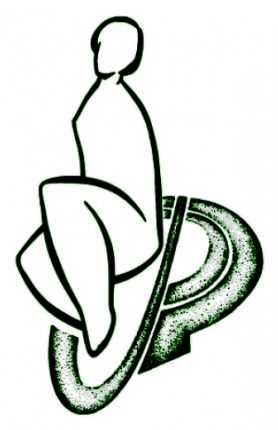


INSTITUTO DE EDUCACIÓN SECUNDARIA JOSÉ PLANES

Departamento de Informática y Comunicaciones

Técnico Superior en Desarrollo de aplicaciones Web

C/ Maestro Pérez Abadía, 2

30100 Espinardo – Murcia

T. 968 834 605

30010577@murciaeduca.es

www.iesjoseplanes.es

Memoria del proyecto

Desarrollo de aplicaciones web

Fotoorla

Autores/as:

Gonzalo Galiano Navarro

Sergio Quijada Cárceles

Sergio Martínez Ríos

Profesor/a-coordinador/a:

Fernando Ruiz Meseguer

Murcia, junio de 2024

Contenido

1 Resumen extendido

Fotoorla es una aplicación web que permite a los usuarios crear, gestionar y descargar sus propias orlas para sus centros educativos.

La aplicación está desarrollada con un backend en Spring Boot, un frontend en Angular y una base de datos en PostgreSQL. Para los estilos se ha usado Bootstrap, personalizado con Sass.

En Fotoorla, un usuario registrado puede crear cursos. A continuación, en el formulario de creación de personas, puede añadir alumnos a los cursos ya creados, indicando nombre y apellidos y una fotografía. En el mismo formulario también puede crear profesores, indicando los mismos datos. Los profesores no están asignados a ningún curso.

El usuario puede ver la lista de cursos que ha creado, así como los alumnos asignados a cada curso y los profesores que ha creado. El usuario puede gestionar esta información, borrando cualquiera de los elementos mencionados anteriormente.

En la pantalla de creación de orla, el usuario puede establecer el nombre del centro educativo y seleccionar un curso. A continuación podrá añadir alumnos de ese curso a la orla, así como los profesores que desee, y elegir un fondo a su gusto. Entonces se generará una previsualización de la orla. El usuario puede comprobar si todo es correcto y editar cada profesor para añadir la asignatura que imparte o el cargo que ocupa en el centro educativo. Finalmente, la orla se descarga en formato PDF.

La orla también se guarda en la aplicación, de forma que el usuario puede ver una lista de las orlas que ha creado anteriormente. Estas orlas pueden ser borradas o descargadas en cualquier momento.

2 Palabras clave

fotografía, orla, educación, javascript, angular, java, spring, spring boot, 39 bootstrap, sass, postgre

3 Introducción

Este proyecto surgió como una propuesta del departamento de informática del IES José Planes. La empresa Fotoorla, encargada de realizar las fotografías y las orlas para el centro, quería actualizar su página web.

El dueño de Fotoorla nos planteaba realizar una serie de mejoras a su web, entre las que se encontraban el aumento de profesores que se pueden añadir a una orla, la mejora del SEO y el posicionamiento en Google, la implementación de un sistema de correos automatizados para los clientes, la mejora de los paneles internos de administración o la creación de un videotutorial sobre el funcionamiento de la web

Los problemas aparecieron cuando accedimos al código fuente de la web y analizamos su contenido. Nos encontramos con un código complejo, desarrollado en React, un lenguaje que no conocemos, y prácticamente sin documentación. La situación nos desanimó; la falta de documentación hacía muy que el mantenimiento de la web fuese complejo, y necesitábamos aprender React para desenvolvernos con soltura y poder realizar las mejoras requeridas en la página.

Afortunadamente, fuimos capaces de reconducir el proyecto. Fernando, nuestro tutor, nos permitió cambiar el enfoque. Ahora íbamos a desarrollar de cero la aplicación web de Fotoorla, pero usando tecnologías que controlamos. Decidimos hacer el frontend en Angular, un framework que hemos aprendido en clase y que nos parece moderno y capaz de crear webs muy vistosas de forma sencilla. Para el back nos decantamos por Spring, un framework de Java. Aunque este lenguaje es nuevo para nosotros, uno de los integrantes del grupo lo estaba empleando en las prácticas de empresa. Por tanto, era una oportunidad ideal de conseguir soltura en este lenguaje y poner en práctica lo aprendido.

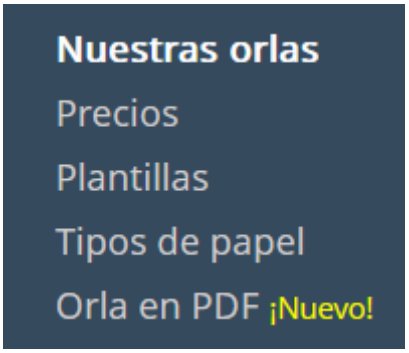
El objetivo del proyecto se convirtió entonces en desarrollar una aplicación que permitiese a un usuario crear sus propias orlas. Para ello era necesario que el usuario pudiese subir las fotografías e información de los profesores y alumnos. La aplicación debía almacenar todos estos datos y permitir su gestión. Es decir, había que ordenar los alumnos por curso y permitir el borrado de cursos,

alumnos y profesores. La creación de la orla debía ser un proceso sencillo e intuitivo para el usuario, y a la vez permitir cierto grado de personalización. Esto lo hemos conseguido permitiendo al usuario escoger qué personas añadir a la orla, añadir el nombre del centro y editar las asignaturas que imparten los profesores y los cargos que ocupan. Por último, era necesario almacenar las orlas creadas para que el usuario pudiese descargarlas cuando quisiese.

4 Estado del arte/trabajos relacionados

Como se ha mencionado anteriormente, nuestro proyecto surge como versión propia y mejora de una aplicación web ya existente, fotoorla.es. Por supuesto, esta no es la única web que permite crear orlas online. Otras páginas como orlaonline.es u orlainteractiva.com ofrecen servicios similares.

Nos ha parecido interesante que precisamente una de estas webs, orlaonline.es, muestra la creación de orlas en formato PDF como una novedad, cuando esta funcionalidad ha sido el objetivo principal de nuestro proyecto.



Esto nos muestra que hemos desarrollado una aplicación que no solo es funcional, sino que además tiene cabida en el mercado y es usada por empresas reales.

Por otra parte, es cierto que estas webs profesionales disponen de muchas más funcionalidades que nuestra web: videotutoriales, guías, más variedad de plantillas para las orlas, mensajería con los clientes, etc.

Creemos, sin embargo, que estos son aspectos a mejorar que se pueden conseguir más adelante si el proyecto continuase.

Por otra parte, queríamos desarrollar nuestra web de una forma que consideramos más idónea que emplea Fotoorla actualmente. En lugar de usar HTML + React, nos decantamos por String y Angular. Creemos que estos lenguajes nos permiten desarrollar una aplicación más robusta en términos de seguridad y funcionalidad, y permiten la escalabilidad según las necesidades de la empresa que la usa.

5 Análisis de objetivos y metodología.

5.1 Objetivos

El equipo se planteó los siguientes objetivos:

- Creación de una base de datos que se ajuste a nuestras necesidades.
- Implementación de un registro y login de usuarios, así como seguridad de la página.
- Escalabilidad del proyecto a futuro, tanto backend como frontend.
- Creación de un formulario que permita al usuario subir a la web las personas que van a aparecer en la orla.
- Desarrollar la lógica de la creación de la orla y su transformación en archivo en formato PDF.
- Implementar un estilo visual inspirado en la web fotoorla.es y que se adapte a distintos tamaños de pantalla (diseño web adaptable)
- Accesibilidad de la web
- Despliegue de la aplicación.

5.2 Metodología

Para la realización del proyecto se ha optado por la siguiente metodología:

Organización - Hemos empleado Discord como medio principal para compartir información técnica entre los miembros del equipo: guías, tutoriales, fragmentos de código, etc. También ha sido nuestro lugar de reunión, empleando para ello los canales de voz.

Comunicación - Whatsapp ha sido nuestra principal herramienta de comunicación diaria. Lo hemos empleado para organizar reuniones, resolver dudas, avisar de avances y bloqueos durante la realización del trabajo.

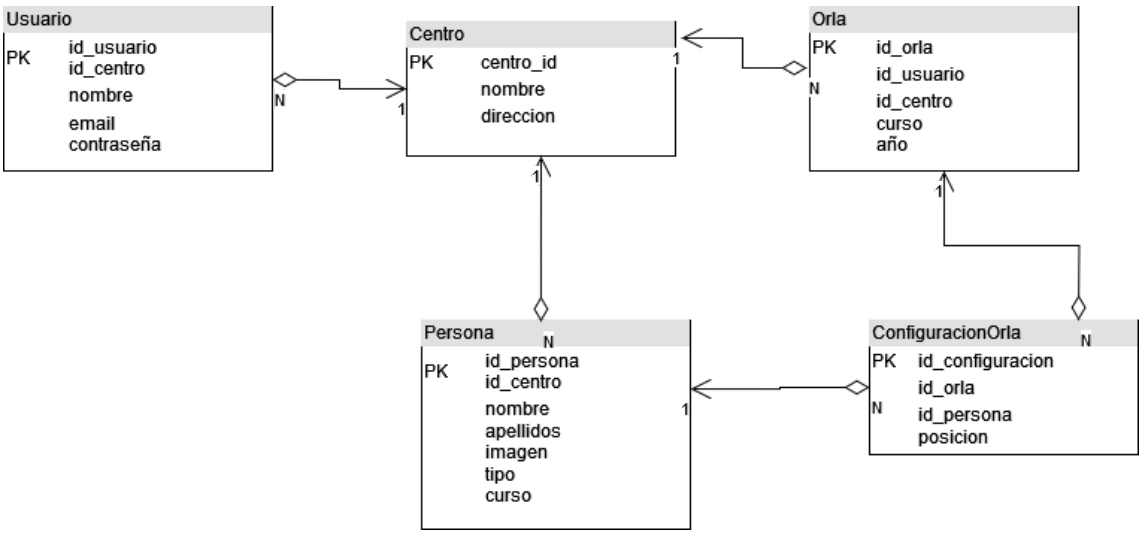
División de tareas - Hemos dividido las tareas del proyecto intentando adaptarnos a las habilidades y preferencias de cada miembro del equipo. Uno de los miembros del grupo se ha encargado del backend, mientras que los otros dos integrantes se han dividido la creación del frontend.

Control de versiones - Hemos empleado GitHub para guardar el código del proyecto y mantenerlo accesible para todos los integrantes del equipo. Esto ha sido especialmente importante en el frontend, donde era vital coordinar a varias personas trabajando sobre el mismo código. Afortunadamente, la comunicación ha sido excelente y no hemos tenido ningún problema desarrollando cada uno su parte del código.

6 Diseño y resolución del trabajo realizado

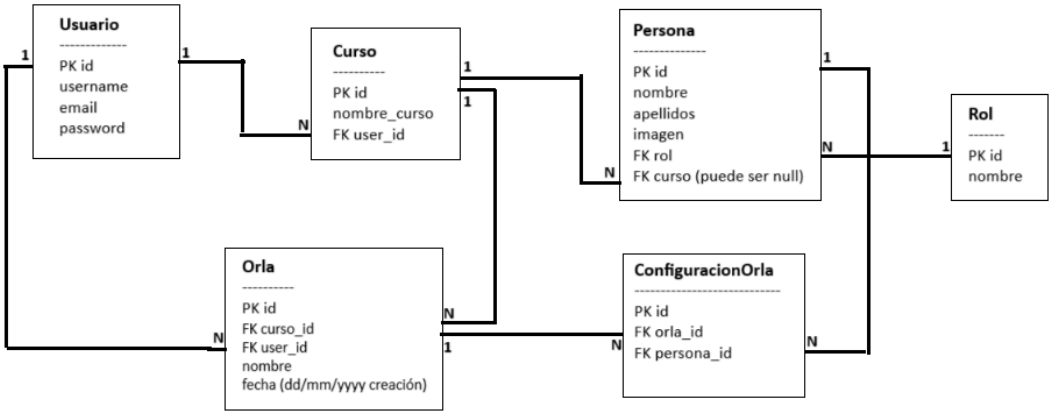
6.1 Base de datos

Originalmente ideamos un esquema para la base de datos basándonos en lo que pensábamos que sería la funcionalidad de nuestra web: El elemento principal sería el centro educativo. El usuario al registrarse crearía su centro educativo o se uniría a uno ya existente. El centro tendría asignadas personas (alumnos y profesores) y orlas. Se estableció una tabla intermedia entre orla y persona para saber qué personas aparecían en una orla y guardar su posición en la misma, de cara a poder editar la orla posteriormente.



Esquema original de la base de datos

Finalmente se decidió que era mejor que no hubiera más de un usuario por centro. Por tanto, no vimos necesaria la tabla centro y la eliminamos. En su lugar le hemos dado más importancia al curso y ahora tiene su propia tabla.



Versión final de la base de datos

Si bien creemos que el esquema se puede mejorar más para ajustarse a las necesidades de la web, este ha sido el modelo que hemos empleado finalmente durante el desarrollo del proyecto

6.2 Frontend en angular

Decidimos desarrollar el frontend de nuestra aplicación en Angular ya que es un framework que hemos aprendido durante el curso y nos han gustado las posibilidades que ofrece.

6.3 Diseño de la web

Nuestro proyecto consiste en el desarrollo de una versión “actualizada” de una web ya existente, fotoorla.es.

fotoorla.es

Debido a esta característica particular del trabajo, no hemos tenido que idear desde cero el diseño y la estética de la web, sino que están directamente inspirados por la web de Fotoorla. Hemos importado de Fotoorla aspectos como la paleta de colores y la distribución de los elementos de la página, siempre adaptándolo al enfoque más reducido de nuestra web.

Nuestra interfaz

Por otra parte, hemos tenido que diseñar la distribución interna de la página para que se adaptase a nuestras necesidades. Aun así, hemos intentado basarnos en la estética de fotoorla para desarrollar todos los componentes e interfaz de usuario, de forma que el resultado global fuese congruente.

6.4 Firebase Storage

Una de las cuestiones que nos planteamos al iniciar este proyecto fue dónde guardar las imágenes que los usuarios suban a la aplicación para añadirlas a sus orlas.

Una de las soluciones que encontramos fue Firebase, una plataforma de Google para el desarrollo de aplicaciones web y móviles. Firebase ofrece múltiples servicios, incluso la posibilidad de desplegar desde la propia herramienta, pero la función que más nos interesó fue Firebase Storage. Storage nos permitía guardar las imágenes en la nube, y su uso es gratuito hasta los 5GB de capacidad, por lo cual era idóneo para nuestro proyecto.

A pesar de esto, finalmente descartamos usar Firebase como base de datos, ya que nos convencía, lo que sí hemos implementado de Firebase en nuestro proyecto ha sido la subida de imágenes en la nube que nos ofrecía, para así solventar el problema de crear alumnos o profesores con una imagen ya definida en el momento de su creación.

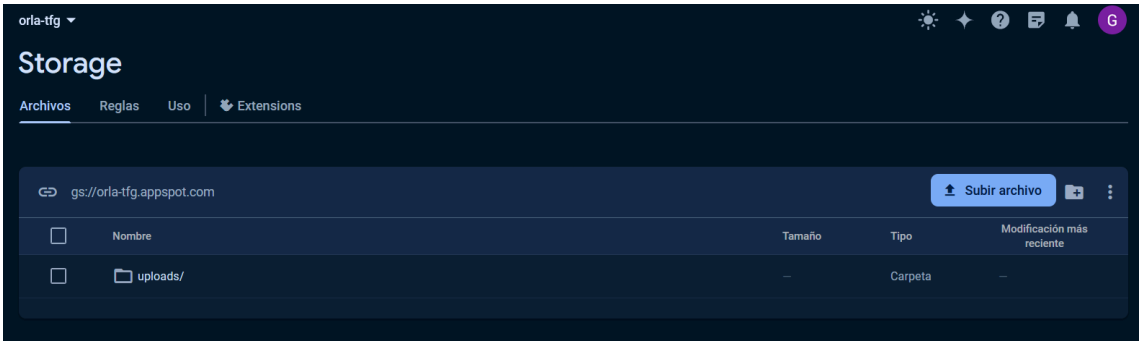
src > app > TS app.config.ts > ...

```
1  import { ApplicationConfig } from '@angular/core';
2  import { provideRouter } from '@angular/router';
3
4  import { routes } from './app.routes';
5  import { initializeApp, provideFirebaseApp } from '@angular/fire/app';
6  import { getStorage, provideStorage } from '@angular/fire/storage';
7
8  export const appConfig: ApplicationConfig = {
9    providers: [
10     provideRouter(routes),
11     provideFirebaseApp(() => initializeApp({
12       "projectId":"orla-tfg",
13       "appId":"[redacted]",
14       "storageBucket":"orla-tfg.appspot.com",
15       /* "locationId":"europe-west", */
16       "apiKey":"[redacted]",
17       "authDomain":"orla-tfg.firebaseapp.com",
18       "messagingSenderId":"[redacted]"})),
19     provideStorage(() => getStorage())]
20 };
21
```

Configuración de Firebase en el proyecto

```
/* Metodo para subir la imagen a Firebase */
uploadFile():void {
  const storageRef = ref(this._storage, `uploads/${this.file.name}`)
  uploadBytesResumable(storageRef, this.file)
}
```

La imagen subida al formulario se captura en la variable “file” y el método **uploadFile()** se ejecuta con el submit del formulario

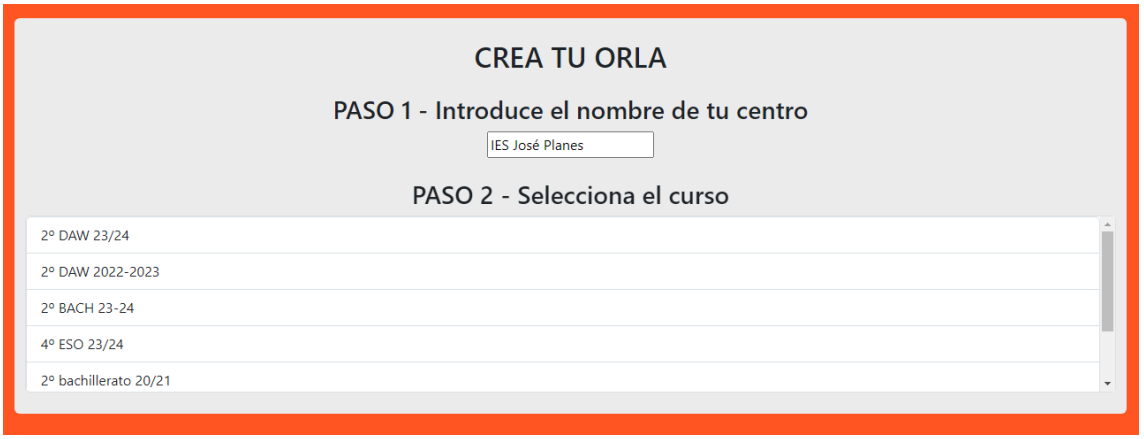


Panel de Firebase Storage. Creamos la carpeta uploads, dentro de la cual se almacenaban las imágenes

6.5 Creación de orla y conversión a PDF

Para la creación de la orla y su posterior descarga barajamos varias posibilidades, teníamos la posibilidad de gestionarlo tanto en el back como en el front, en nuestro caso al final decidimos gestionarlo completamente desde el front en Angular, todo esto se gestiona de la siguiente manera:

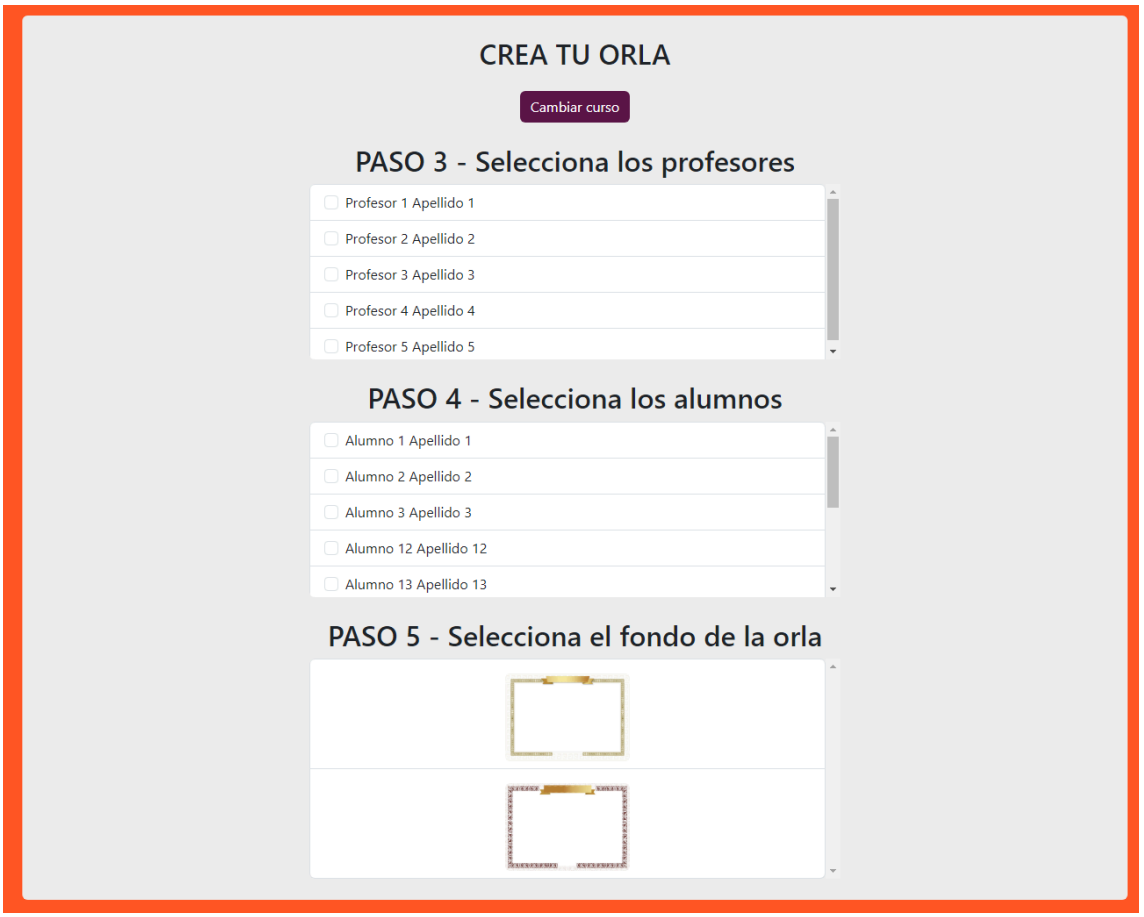
Al darle a la sección de crear orla, se nos pedirá que pongamos un nombre del centro y que seleccionemos el curso de unos de los cursos ya creados, el centro por defecto será el IES José Planes:



El código detrás de esto es el siguiente:

```
<div class="row justify-content-center py-3 bg-naranja">
  <div class="col-11 col-md-6 rounded py-4 bg-offWhite">
    <h2 class="text-center mb-4">CREA TU ORLA</h2>

    <!-- Nombre del centro escolar -->
    @if (!cursoSeleccionado) {
      <h3 class="text-center">PASO 1 - Introduce el nombre de tu centro</h3>
      <div class="row justify-content-center mb-4">
        <div class="col text-center">
          <input  type="text" [(ngModel)]="nombreInstituto" (ngModelChange)="setNombreInstituto($event)">
        </div>
      </div>

      <!-- Selección de curso -->
      <h3 class="text-center">PASO 2 - Selecciona el curso</h3>
      <div id="cursos-list" class="list-group border rounded bg-white">
        @for (c of getCursos(); track c.id){
          <button (click)="selectCurso(c.id)" type="button"
          class="list-group-item list-group-item-action align-middle d-flex justify-content-between align-items-center" >
            {{c.name}}
        </button>
        }
      </div>
```

Funciona de tal manera → Si no hay ningún cursos seleccionado, muestra una caja de texto para escribir el nombre del centro, y luego un listado de todos los cursos disponibles ya creados por el usuario, al seleccionar el curso, automáticamente avanza al siguiente paso para crear la orla, la selección de profesores, alumnos y fondo para la orla.



Nos aparecen todos los profesores que tenemos dados de alta, puesto que un profesor puede impartir varias asignaturas en varios cursos diferentes, tienen que estar todos, y posteriormente nos aparecen los alumnos del curso que hemos seleccionado con anterioridad, pudiendo seleccionar aquellos que quieran aparecer en la orla, y por último, el fondo que queremos utilizar para nuestra orla, de momento hemos añadido dos fondos.

También tenemos un botón en la parte superior con el que podemos cambiar de curso en caso de haber escogido uno erróneo para no tener que volver hacia atrás con la barra de navegación.

El código detrás de esta funcionalidad es el siguiente:

```
    <!-- Botón cambiar curso -->
    <div class="row justify-content-center">
      <div class="col text-center">
        <button class="btn btn-granate" (click)="unselectCurso()">Cambiar curso</button>
      </div>
    </div>

    <!-- Selección de profesores -->
    <h2 class="mt-4 text-center">PASO 3 - Selecciona los profesores</h2>
    <div class="row justify-content-center">
      <div class="col-md-6">
        <div class="list-group">
          @for (profesor of getProfesores(); track profesor.id) {
            <div class="list-group-item">
              <input type="checkbox" class="form-check-input me-2" (change)="seleccionarPersona(profesor)" />
              <label class="form-check-label">{{ profesor.nombre }} {{ profesor.apellidos }}</label>
            </div>
          }
        </div>
      </div>
    </div>


    <!-- Selección de alumnos -->
    <h2 class="mt-4 text-center">PASO 4 - Selecciona los alumnos</h2>
    <div class="row justify-content-center">
      <div class="col-md-6">
        <div id="lista-alumnos" class="list-group">
          @for (alumno of getAlumnosByCurso(selectedCursoId); track alumno.id) {
            <div class="list-group-item">
              <input type="checkbox" class="form-check-input me-2" (change)="seleccionarPersona(alumno)" />
              <label class="form-check-label">{{ alumno.nombre }} {{ alumno.apellidos }}</label>
            </div>
          }
        </div>
      </div>
    </div>

    <!-- Selección de fondo -->
    <h2 class="mt-4 text-center">PASO 5 - Selecciona el fondo de la orla</h2>
    <div class="row justify-content-center">
      <div class="col-md-6 text-center">
        <div class="list-group" id="fondosOrla">
          @for (f of fondos; track $index) {
            <div class="list-group-item fondo-item"
            (click)="seleccionarFondo(f)"
              [class.selected]="fondoSeleccionado === f"
              [style.cursor]="'pointer'">
              <input type="radio" class="form-check-input d-none" name="fondo" value="{{f.id}}" />
              <img src="{{f.url}}" class="img-fluid rounded mt-2" style="max-height: 100px;" alt="Fondo de la Orla" />
            </div>
          }
        </div>
      </div>
    </div>
  }
  </div><!-- col offWhite end-->
</div><!-- row naranja end-->
```

Todas las secciones están divididas dentro del código para que se interprete de forma sencilla dónde se encuentran.

Después, al seleccionar todos los profesores, alumnos, y el fondo que queremos, automáticamente se desplegará una vista previa de la orla con todos los ajustes que hemos seleccionado, con la opción de editar la asignatura impartida por cada profesor para que así se especifique dentro de la orla, y al final, un botón para descargar esa misma orla en formato PDF.

El código que hace que todo esto funcione es el siguiente:

```
<!-- Mostrar Orla -->
<div class="row justify-content-center bg-naranja">
  <div class="col-md-10 mb-3 rounded text-center bg-offWhite">
    @if (seleccionados.length > 0 && fondoSeleccionado) {
      <h2 class="mt-3 mb-3">Vista previa de la orla</h2>
      <div class="orla-wrapper">
      <div id="orla" [ngStyle]="{'background-image': fondoSeleccionado ? 'url(' + fondoSeleccionado.url + ')' : 'none'}" class="orla mx-auto">
        <h3 class="nombre-instituto">{{ nombreInstituto }}</h3>
        <!-- Profesores -->
        <div class="orla-row profesores">
          @for (p of seleccionados; track $index) {
            @if (!p.curso) {
              <div class="orla-item profesor text-center">
                <img src="{{p.imagen}}" alt="" class="img-fluid rounded-circle mb-2">
                <p class="mb-0">{{p.nombre}}<br>{{p.apellidos}}</p>
                <input class="inputAsignatura text-center" type="text" placeholder="Editar asignatura">
              </div>
            }
          }
        </div>
        <!-- Alumnos -->
        <div class="orla-row alumnos">
          @for (p of seleccionados; track $index) {
            @if (p.curso) {
              <div class="orla-item">
                <img src="{{p.imagen}}" alt="" class="img-fluid rounded-circle mb-2">
                <p>{{p.nombre}}<br>{{p.apellidos}}</p>
              </div>
            }
          }
        </div>
      </div>
      </div>
      <!-- Botón crear orla -->
      <h2 class="mt-4 text-center">PASO 6 - Descarga tu orla</h2>
      <button class="btn btn-granate mb-4" (click)="generarPDF()">Descargar orla</button>
    }
  </div><!-- col offWhite end-->
</div><!-- row naranja end-->
</div>
```

Bien, para empezar, la previsualización de la orla se despliega si tenemos profesores, alumnos y un fondo seleccionado.

La orla está contenida dentro de un div (contenedor) para que, en caso de tener una pantalla con una baja resolución, podamos ver la orla a tamaño real y

sea posible el desplazamiento lateral de la misma dentro de la página para verla completamente. Esto debe ser así debido al funcionamiento del método para descargar la orla en pdf, el cual explicaremos posteriormente.

Dentro del contenedor de la orla, tendremos el propio contenedor de la orla (el div que tiene el id orla), mediante el atributo "**ngStyle**", agregamos el fondo seleccionado para que sea visible en tiempo real para el usuario, después listamos tanto los profesores como los alumnos, primero colocamos a los profesores, para distinguirlos de los alumnos es sencillo, puesto que los alumnos tienen curso y los profesores no, y luego colocamos a los alumnos.

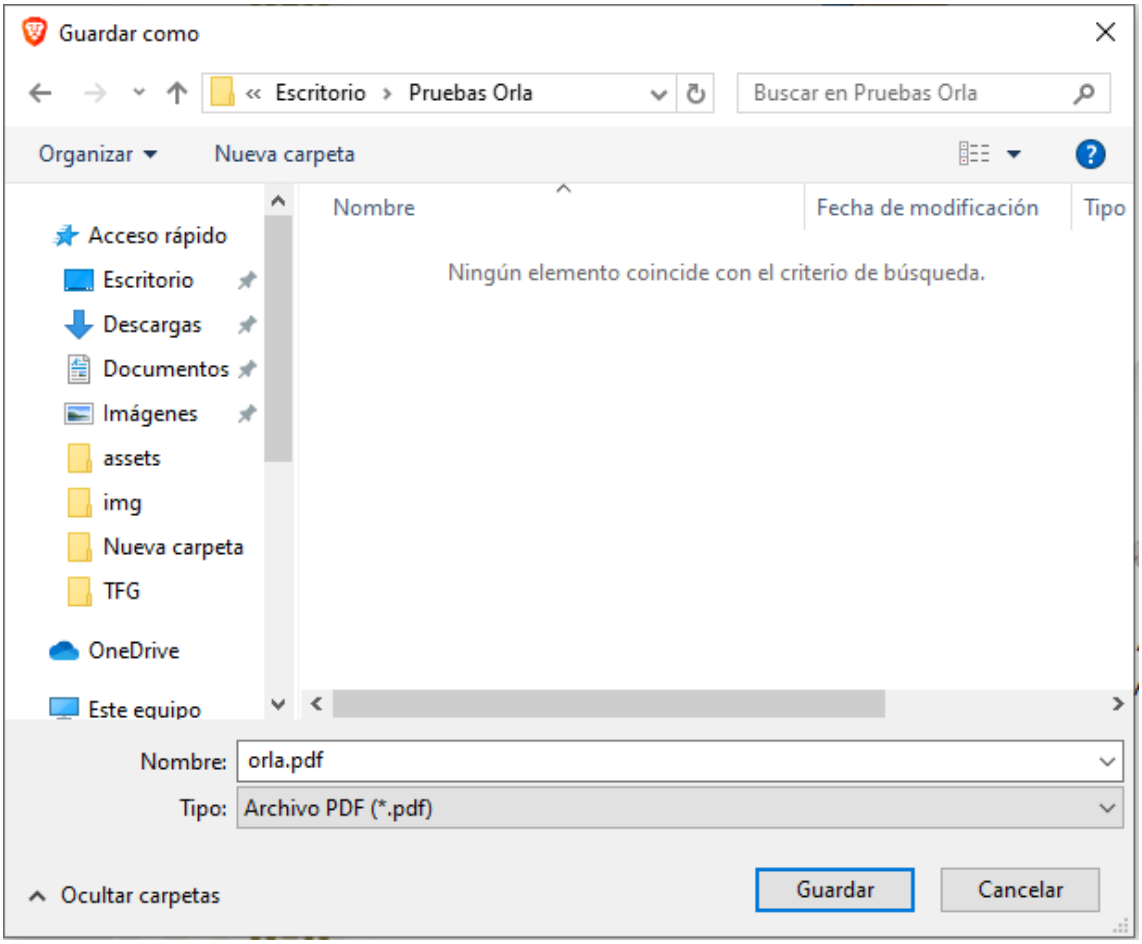
Con todo esto y unos ajustes en css para que esté todo ordenado y bien centrado en nuestra orla, procederíamos a descargar la orla en PDF, esto lo haremos mediante la librería **jsPDF,** y el método para descargar la orla en pdf sería el siguiente:

```
generarPDF(): void {
  const orlaElement = document.getElementById('orla')!;
  html2canvas(orlaElement).then(canvas => {
    const imgData = canvas.toDataURL('image/png');
    const imgWidth = 297; // Anchura de A4 en mm (landscape)
    const pageHeight = 210; // Altura de A4 en mm (landscape)
    const imgHeight = canvas.height * imgWidth / canvas.width;
    let heightLeft = imgHeight;

    const pdf = new jsPDF('l', 'mm', 'a4');
    let position = 0;

    pdf.addImage(imgData, 'PNG', 0, position, imgWidth, imgHeight);
    heightLeft -= pageHeight;

    pdf.save('orla.pdf');
  });
}
```

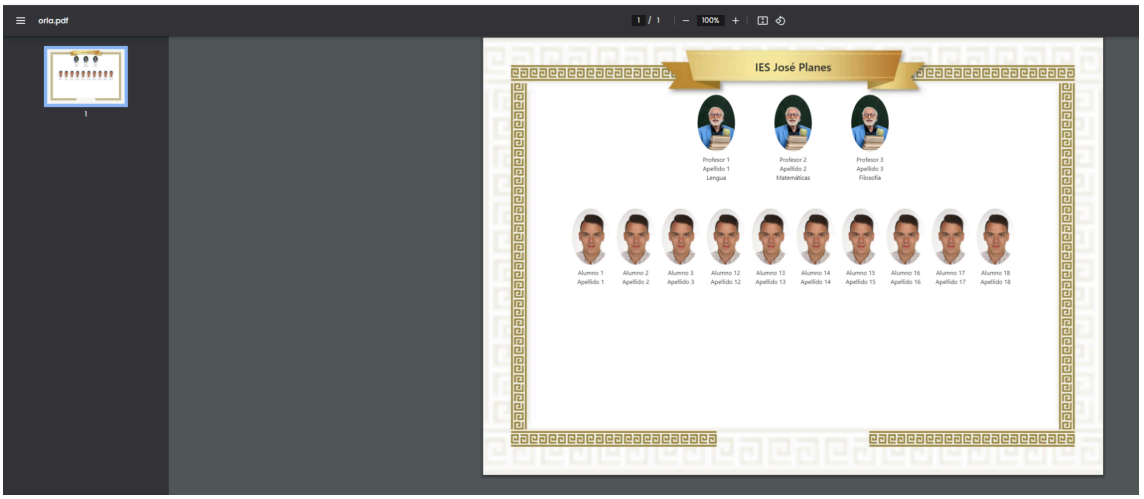
1. Obtiene una referencia al elemento HTML con el id "orla" utilizando **document.getElementById('orla')**. El símbolo de exclamación (**!**) al final del elemento **document.getElementById('orla')** indica que estamos seguros de que el elemento existe y no es nulo.
2. Utiliza la función **html2canvas** para convertir el elemento HTML en un lienzo de lienzo (canvas).

3. Una vez que se ha generado el lienzo, se extrae la imagen en formato PNG utilizando el método **toDataURL('image/png')**. Esto devuelve una cadena de texto que representa la imagen codificada en base64.
4. Se definen algunas variables para el tamaño del PDF. **imgWidth** representa el ancho del PDF en milímetros (en este caso, se utiliza el ancho de un A4 en modo paisaje). **pageHeight** representa la altura de una página en el PDF (también se utiliza el valor correspondiente a un A4 en modo paisaje).
5. Se calcula la altura de la imagen en el PDF utilizando la relación de aspecto del lienzo. Esto se hace multiplicando la altura del lienzo por el ancho del PDF y dividiendo el resultado por el ancho del lienzo.
6. Se crea una instancia de **jsPDF** con los parámetros 'l', 'mm', 'a4'. Esto crea un nuevo objeto PDF con orientación horizontal, unidades en milímetros y tamaño de página A4 y se establece la posición inicial en la parte superior de la página utilizando **position = 0.**
7. Se agrega la imagen al PDF utilizando el método **addImage** de **jsPDF**. Los argumentos son la imagen codificada en base64, el formato de imagen (en este caso, **'PNG'**), la posición x e y (0, **position**), el ancho y alto de la imagen (**imgWidth**, **imgHeight**) y se resta la altura de una página al valor de **heightLeft** para indicar que se ha utilizado una página.
8. Finalmente, se guarda el PDF con el nombre "orla.pdf" utilizando el método **save** de **jsPDF**.

Al pulsar el botón descargar orla, aparecerá el explorador de carpetas de windows para indicar dónde queremos guardar la orla descargada:



Y cuando la guardemos ya podremos disponer de la orla que acabamos de generar en formato PDF y lista para imprimir:



Como hemos explicado anteriormente, era imprescindible que el contenedor inicial de la orla estuviera encapsulado en otro contenedor, puesto que el método **generarPDF** funciona de tal manera que captura el contenedor con el

id “orla” y hace una captura de lo que se ve en el contenedor, si el tamaño del contenedor varía, todo se descuadrar y no se descargaría correctamente la orla, este también fue uno de los problemas que tuvimos que solucionar en un principio al decidir generar el método **generarPDF** directamente en Angular.

6.6 Bootstrap y Sass

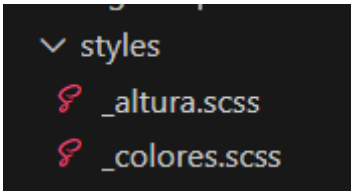
En nuestro proyecto usamos Bootstrap para dar estilos al código HTML. Este framework de CSS nos ha permitido obtener resultados vistosos y efectivos con poco trabajo. Gracias al uso de la grilla de Bootstrap hemos conseguido un HTML que es fácil de entender, organizado por filas y columnas, y que es adaptable a todos los tamaños de pantalla. Por otra parte, con las clases de bootstrap hemos podido establecer los márgenes y paddings de nuestros componentes sin tener que recurrir a extensos archivos CSS.

Sin embargo, bootstrap también nos presenta la desventaja de que su estilo es muy reconocible. Afortunadamente gracias a Sass podemos personalizar los estilos de bootstrap.

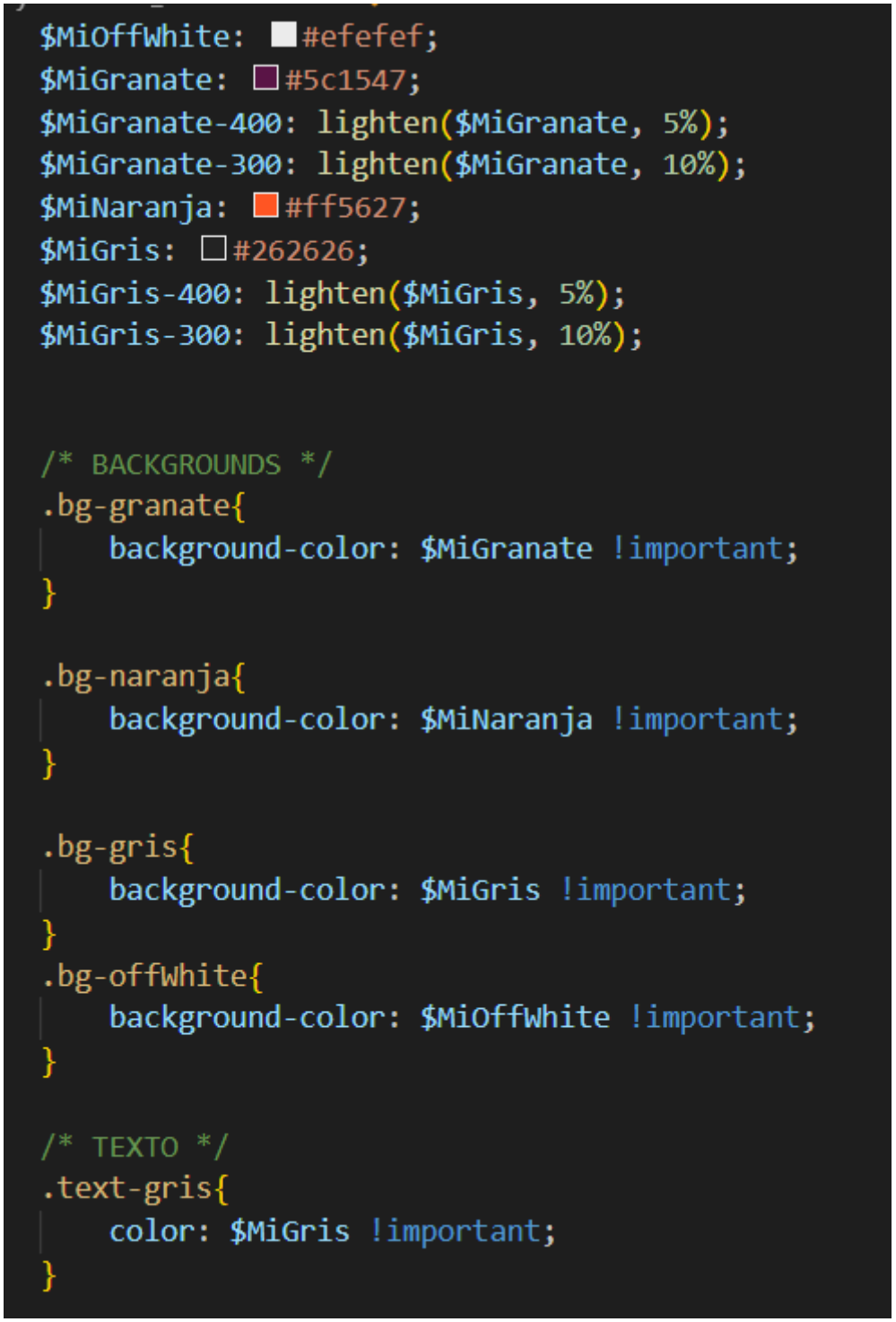
Puesto que tomamos la decisión de emplear Sass con el proyecto ya empezado, fue necesario realizar algunos cambios en la configuración, indicando dentro del archivo **angular.json** cuál es la ruta del documento que queremos usar para los estilos.

```
{} angular.json > {} projects > {} orla-tfg > {} architect > {} build > {} options > [ ] assets
  5      "projects": {
  6        "orla-tfg": {
 12          "architect": {
 74            "test": {
 76              "options": {
 85                ],
 86                "styles": [
 87                  "src/styles.css"
 88                ],
 89                "scripts": []
 90              }
 91            }
 92          }
 93        }
 94      }
 95    }
 96
```

Dividimos el código Sass en diferentes archivos dependiendo de su funcionalidad y lo organizamos dentro de una carpeta para los estilos.



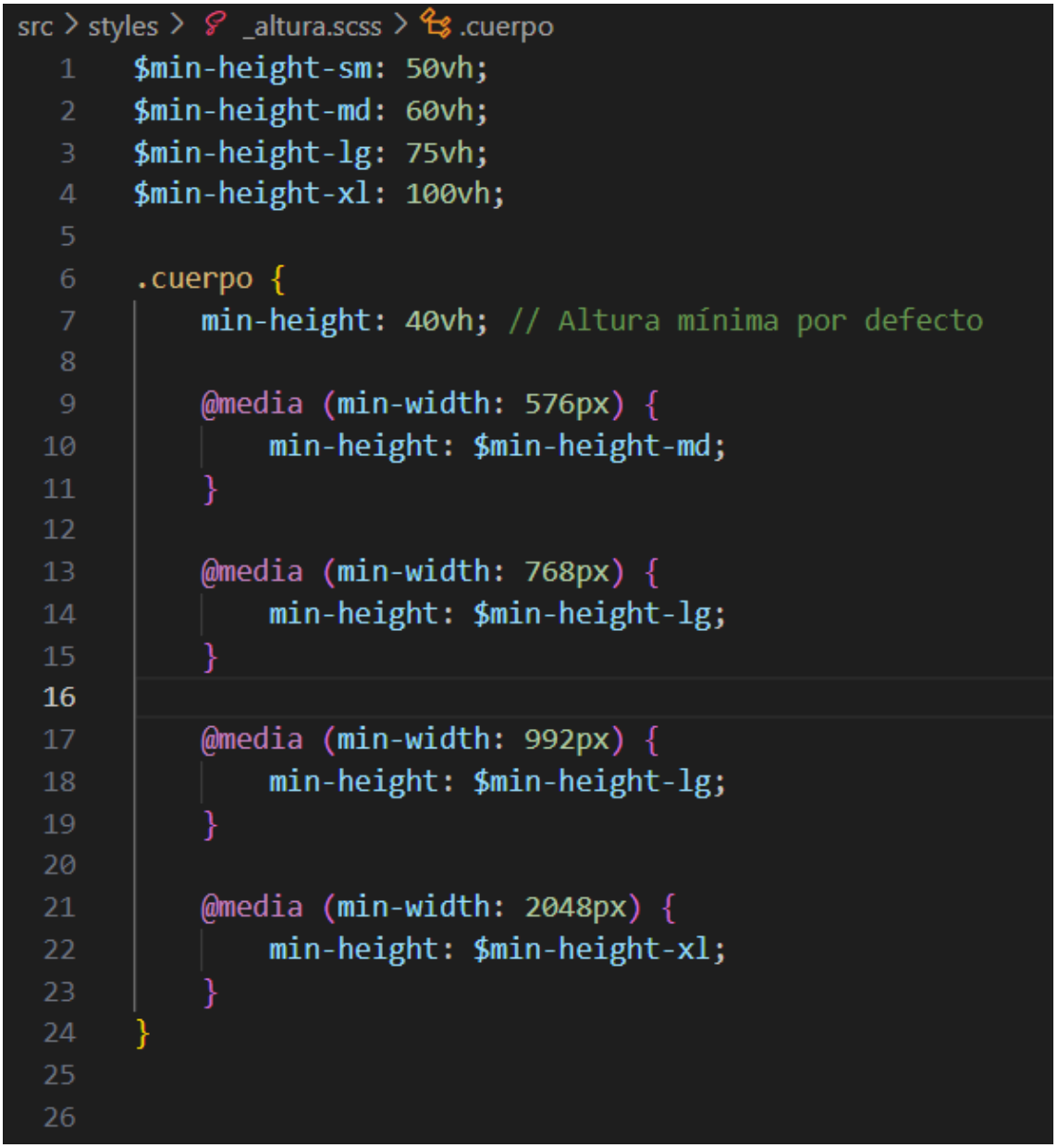
De esta forma hemos podido crear nuestras propias variables de color y emplearlos en colores de fondo, en botones, etc



```scss
$MiOffWhite: #efefef;
$MiGranate: #5c1547;
$MiGranate-400: lighten($MiGranate, 5%);
$MiGranate-300: lighten($MiGranate, 10%);
$MiNaranja: #ff5627;
$MiGris: #262626;
$MiGris-400: lighten($MiGris, 5%);
$MiGris-300: lighten($MiGris, 10%);


/* BACKGROUNDS */
.bg-granate{
    background-color: $MiGranate !important;
}

.bg-naranja{
    background-color: $MiNaranja !important;
}

.bg-gris{
    background-color: $MiGris !important;
}
.bg-offWhite{
    background-color: $MiOffWhite !important;
}

/* TEXTO */
.text-gris{
    color: $MiGris !important;
}
```

También gracias a Sass hemos podido resolver uno de los mayores problemas que nos hemos encontrado durante el desarrollo del front. Tuvimos bastantes dificultades para conseguir que la página fuese responsive verticalmente. Es decir, cuando había poco contenido en la página se veía una franja blanca vacía en la parte inferior de la pantalla. Esto era especialmente evidente en las resoluciones más grandes, de más de 1080 píxeles de anchura. Si bien no es

la solución más idónea, optamos por crear media queries en Sass que cambiasen el virtual height (vh) mínimo que ocupa el cuerpo de la página dependiendo de la resolución.



Estos archivos Sass son finalmente importados en el archivo **styles.scss**, que es el archivo principal de estilos de nuestro proyecto.

```
src > styles.scss
1  @import './styles/colores';
2  @import './styles/altura';
3  @import 'bootstrap/scss/bootstrap';
```

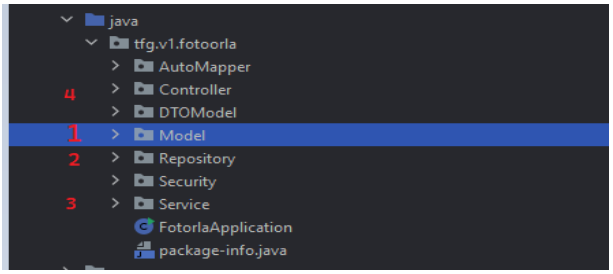
Sin embargo, a consecuencia de haber implementado Sass hacia el final del desarrollo del proyecto, existen todavía archivos .css en el código que por falta de tiempo material no han sido pasados a .scss. Es uno de los aspectos a mejorar del trabajo.

6.7 Backend bajo el framework SPRING

Para enfocar el backend hemos elegido, partiendo desde los cimientos, la elección de una buena arquitectura para su futura escalabilidad, para ello trás haber investigado elegimos la estructura clean multicapa,

La arquitectura Clean Multicapa organiza el código en diferentes capas, cada una con una responsabilidad específica. Esto facilita el mantenimiento, la escalabilidad y las pruebas del software. Al separar la lógica de negocio de la implementación, se obtiene un código más limpio y manejable.

Hemos seguido la estrategia de que el **modelo** se comunica con el **repositorio,** este se utiliza exclusivamente en el **servicio,** que este en particular se encarga de transformar la entidad con la que se trabaja previamente y enviar un **DTO** (Data Transfer Object) a la capa de el **controlador,** que es el que se encarga de ofrecer las solicitudes HTTP. Recibe los datos de los clientes, llama a los servicios apropiados para procesar esos datos y devuelve las respuestas.



Para ello hemos empezado desde la capa de entidades siguiendo una implementación del diagrama UML en base a FIRST-CODE, que se basa en crear nuestra capa de modelos, y añadir las entidades previamente diseñadas en base de datos. Estas entidades las íbamos a relacionar mediante anotaciones JPA que es una forma de mapear clases Java a tablas de base de datos relacional. En Spring Boot, JPA se integra a través de Hibernate, que es el ORM (Object Relational Mapping) o Mapeo Objeto-Relacional que consiste en la simplificación del desarrollo de aplicaciones al permitir que el desarrollador trabaje con objetos en lugar de consultas SQL para interactuar con la base de datos en nuestro caso, POSTGRES SQL, levantada en un contenedor DOCKER en local por falta de tiempo de desarrollo.

Gracias a hibernate se puede generar automáticamente el esquema de la base de datos basado en las clases de entidad y sus mapeos, eliminando la necesidad de crear manualmente las tablas de la base de datos, y así hemos hecho con nuestras entidades, os dejamos un ejemplo de código:

```java
@Entity
@Table(name = "Usuario")
@Data
@NoArgsConstructor
public class Usuario {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id_usuario;
    no usages
    private String username;
    no usages
    private String password;

    no usages
    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(
            name = "usuario_roles",
            joinColumns = @JoinColumn(name = "usuario_id", referencedColumnName = "id_usuario"),
            inverseJoinColumns = @JoinColumn(name = "role_id", referencedColumnName = "id_role")
    )
    private List<Roles> roles;

    no usages
    @OneToMany(mappedBy = "usuario")
    @JsonIgnore
    private List<Curso> cursos;
}
```

Como podéis ver anotaciones como **@Entity** indican que es una entidad y **@Table,** el nombre que le vamos a dar a la tabla en bases de datos, **@ManyToMany** indica que es una relación de muchos a muchos y **@JoinTable** nos indica que tablas queremos referenciar.

A partir de esta entidad creamos un repositorio

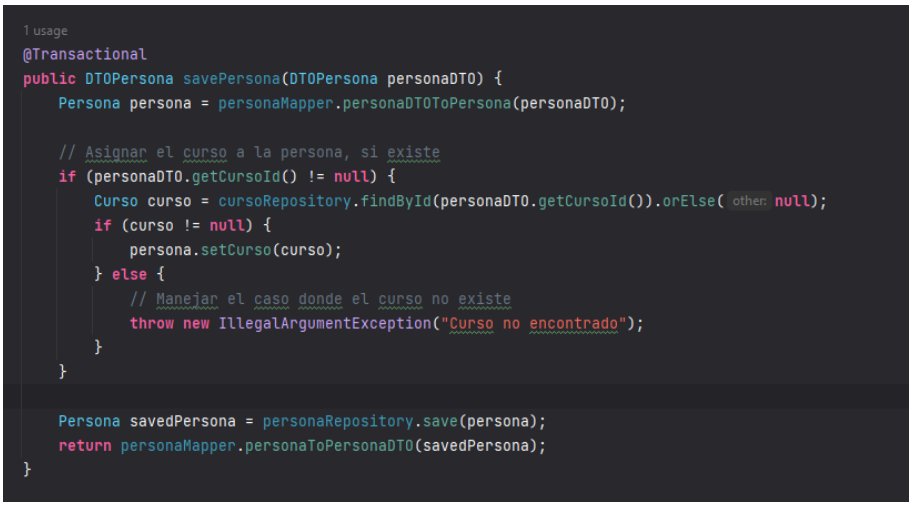
```
7 usages
@EnableJpaRepositories
@Repository
public interface IUsuarioRepository extends JpaRepository<Usuario, Long> {
    2 usages
    <Optional>Usuario findByUsername(String username);
    1 usage
    Boolean existsByUsername(String username);
}
```

El cual nos va a permitir tener todas las consultas básicas, seteos, eliminaciones a través de Jpa Repository gracias a Java para que el servicio se comunique y esté separado de la lógica de modelo y también de la lógica de negocio .

Luego, a través de un Mapper (es una clase o interfaz que se utiliza para convertir objetos de un tipo a otro y es especialmente útil para transformar datos entre distintas capas de la aplicación, como por ejemplo entre la capa de datos y la capa de presentación)

```
3 usages  1 implementation
@Mapper(componentModel = "spring")
public interface PersonaMapper {

    4 usages  1 implementation
    @Mapping(source = "curso.id", target = "cursoId")
    @Mapping(source = "configuracionesOrla", target = "orlasIds", qualifiedByName = "configuracionesOrlaToIds")
    DTOPersona personaToPersonaDTO(Persona persona);

    1 usage  1 implementation
    Persona personaDTOToPersona(DTOPersona personaDTO);

    1 usage
    @Named("configuracionesOrlaToIds")
    default List<Long> configuracionesOrlaToIds(List<ConfiguracionOrla> configuracionesOrla) {
        return configuracionesOrla.stream() Stream<ConfiguracionOrla>
                .map(ConfiguracionOrla::getId) Stream<Long>
                .collect(Collectors.toList());
    }
}
```

MapStruct es una biblioteca de Spring que simplifica este proceso de mapeo. Genera automáticamente el código de mapeo en tiempo de compilación, lo que lo hace eficiente y fácil de usar. No tienes que escribir manualmente todo el código necesario para convertir un objeto en otro luego en el siguiente paso de nuestra aplicación, el servicio. El cual este mapea la entidad a un DTO, que no



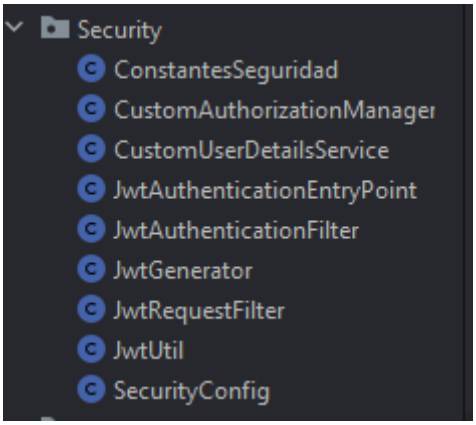
```
1 usage
@Transactional
public DTOPersona savePersona(DTOPersona personaDTO) {
    Persona persona = personaMapper.personaDTOToPersona(personaDTO);

    // Asignar el curso a la persona, si existe
    if (personaDTO.getCursoId() != null) {
        Curso curso = cursoRepository.findById(personaDTO.getCursoId()).orElse( other: null);
        if (curso != null) {
            persona.setCurso(curso);
        } else {
            // Manejar el caso donde el curso no existe
            throw new IllegalArgumentException("Curso no encontrado");
        }
    }

    Persona savedPersona = personaRepository.save(persona);
    return personaMapper.personaToPersonaDTO(savedPersona);
}
```

tiene que tener necesariamente los mismos datos, sino solamente los que necesitemos para enviar al controlador.

Por ultimo, este método es llamado por la capa de aplicación, que no incluye la lógica del negocio sino solamente la llamada al servicio y maneja las solicitudes HTTP

```
@RestController
@RequestMapping("/cursos")
@CrossOrigin(origins = "http://localhost:4200")
public class    CursoController {

    5 usages
    @Autowired
    private CursoServiceImpl cursoService;

    no usages
    @PostMapping("/getNombrePorId/{id}")
    public ResponseEntity<DTOResponse> getNombreCursoPorId(@PathVariable Long id) {
        DTOResponse nombreCurso = cursoService.getNombreCursoPorId(id);
        return ResponseEntity.ok(nombreCurso);
    }

    no usages
    @PostMapping("/create")
    public ResponseEntity<DTOCurso> createCurso(@RequestBody DTOCursoCreacion cursoDTO) {
        DTOCurso savedCursoDTO = cursoService.saveCurso(cursoDTO);
        return ResponseEntity.ok(savedCursoDTO);
    }
    no usages
    @GetMapping("/list")
    public ResponseEntity<List<DTOCurso>> getAllCursos() {
        List<DTOCurso> cursosDTO = cursoService.getAllCursos();
        return ResponseEntity.ok(cursosDTO);
    }
```

Todo ello está protegido por Spring Security, es un módulo del framework Spring que facilita la implementación de seguridad en aplicaciones Java. Proporciona mecanismos para la autenticación (verificar quién eres) y la autorización (verificar qué puedes hacer), protegiendo así tu aplicación contra accesos no autorizados. Es altamente configurable y se puede integrar con diversas fuentes de datos como bases de datos, LDAP o proveedores de identidad como Google. Con Spring Security, puedes definir qué partes de tu aplicación requieren inicio de sesión y controlar el acceso basado en roles, todo de manera sencilla y sin necesidad de desarrollar desde cero las funcionalidades de seguridad. Para ello hemos necesitado crear una serie de ficheros para poder implementarlo.



Lo primordial y en lo que nos vamos a centrar es en 3 ficheros, Security Config es el padre que maneja todo acerca la autorización y validación. .

```java
no usages
@Configuration
@EnableWebSecurity //habilita la seguridad web en la aplicacion de Spring Boot y proporciona la configuracion de seguridad por defecto
public class SecurityConfig {
    2 usages
    private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;


    no usages
    @Autowired
    public SecurityConfig(JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint){
        this.jwtAuthenticationEntryPoint = jwtAuthenticationEntryPoint;
    }
    no usages
    @Bean
    public JwtAuthenticationEntryPoint jwtAunthenticationEntryPoint() { return new JwtAuthenticationEntryPoint(); }

    //se encargara de verificar si el token es valido y si el usuario tiene los permisos necesarios para acceder a la solicitud
    no usages
    @Bean
    AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {
        return authenticationConfiguration.getAuthenticationManager();
    }
    //encriptamos todas las contraseña de los usuarios
    no usages
    @Bean
    PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

```java
//este bean incorporara el filtro de seguridad de json web token que creamos en la clase JwtAuthenticationFilter
1 usage
@Bean
JwtAuthenticationFilter jwtAuthenticationFilter() { return new JwtAuthenticationFilter(); }

//Establecerá una cadena de filtros de seguridad en nuestra aplicacion
//Es aqui donde determinaremos los permisos de acceso a las rutas de nuestra aplicacion segun el rol del usuario
no usages
@Bean
SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .exceptionHandling(handling -> handling.authenticationEntryPoint(jwtAuthenticationEntryPoint))
            .sessionManagement(sesionmng -> sesionmng.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeRequests(authorize -> authorize.
                    requestMatchers( ...patterns: "configuracionesOrla/**").authenticated().
                    requestMatchers( ...patterns: "cursos/**").authenticated().
                    requestMatchers( ...patterns: "orlas/**").authenticated().
                    requestMatchers( ...patterns: "personas/**").authenticated().
                    requestMatchers( ...patterns: "api/usuarios/**").permitAll().
                    requestMatchers( ...patterns: "api/**").authenticated()
                    .anyRequest().authenticated());

    http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Aqui vemos en la cadena authorize podemos especificar a través de todos los filtros previamente creados junto a la generación del JSON WEB TOKEN para validar los roles del usuario. Para ello utilizamos el JwtAuthenticationFilter

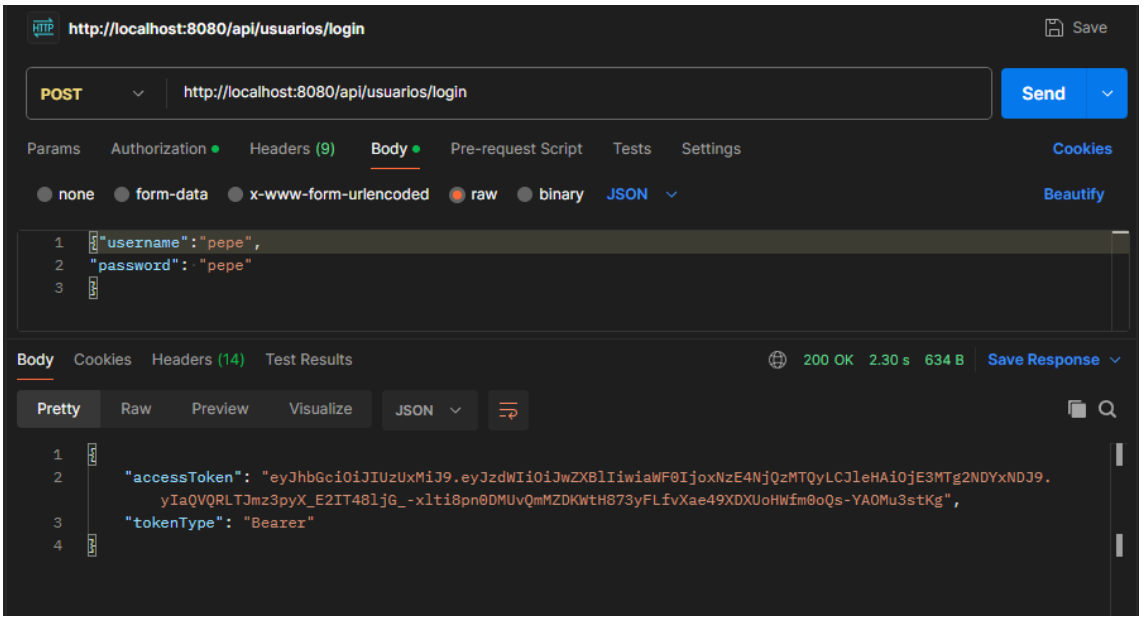
```java
//la funcion de este filtro es interceptar las peticiones y validar el token y si es valido se autentica al usuario en la solicitud
2 usages
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    1 usage
    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    2 usages
    @Autowired
    private JwtGenerator jwtGenerator;


    //Metodo para validar el token y autenticar al usuario
    1 usage
    private String getTokenRequest(HttpServletRequest request) {
        final String requestTokenHeader = request.getHeader( s: "Authorization");
        if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
            return requestTokenHeader.substring( beginIndex: 7);
        }
        return null;
    }
```

```java
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain) throws ServletException, IOException {
    String token = getTokenRequest(request);
    if (token != null && this.jwtGenerator.validarToken(token)){ //si el token es valido
        String username = this.jwtGenerator.obtenerUsernameDeJwt(token); //obtenemos el username del token
        UserDetails userDetails = this.customUserDetailsService.loadUserByUsername(username); //obtenemos los datos del usuario
        List<String> roles = userDetails.getAuthorities().stream().map(GrantedAuthority::getAuthority).toList(); //obtenemos los roles del usuario
        if(roles.contains("USER") || roles.contains("ADMIN")){//si el usuario tiene el rol de usuario o administrador
            UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(userDetails, //autenticamos al usuario
                    null, userDetails.getAuthorities()); //con sus roles
            authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request)); //retomamos los detalles de la solicitud
            SecurityContextHolder.getContext().setAuthentication(authenticationToken); //autenticamos al usuario
        }

    }
    filterChain.doFilter(request, response); //continuamos con la solicitud
}
```

Como vemos en los comentarios de la explicación del código con este filtro interceptamos el token, el username del token, datos del usuario, roles, comprobamos si es usuario o administrador ya que administrador tendrá acceso a todos los métodos y retornar los detalles.

```java
//Metodo para generar el token por medio de la autehnticacion
2 usages
public String generarToken(Authentication authentication) {
        String username = authentication.getName();
        Date tiempoActual = new Date();
        Date expiracionToken = new Date(tiempoActual.getTime() + ConstantesSeguridad.JWT_EXPIRATION_TOKEN);

    //generar token
    String token = Jwts.builder()
            .setSubject(username) //nombre de usuario
            .setIssuedAt(tiempoActual) //fecha de creacion del token
            .setExpiration(expiracionToken) //fecha de expiracion del token
            .signWith(SignatureAlgorithm.HS512, ConstantesSeguridad.JWT_SECRET)
            //firma el token con la clave secreta que solo el servidor conoce
            .compact(); //construye el token

    return token;
}
```

y como resultado por ejemplo, al loguearnos con un usuario y passworde existente en la aplicación, nos devuelve un TOKEN que es el que usaremos para acceder a los métodos.



y en caso de lo contrario nos devuelve un 401 UNAUTHORIZED

7 Presupuesto

Cada miembro del equipo ha dedicado aproximadamente unas 50 horas de trabajo, para un total de 150 horas.

Teniendo en cuenta que el sueldo promedio para un programador junior en España es de 10,77[1] euros por hora, obtenemos un coste total de 1615,5 euros.

8 Conclusiones y vías futuras

El objetivo principal del proyecto ha sido realizar una aplicación que fuese capaz de generar orlas de forma interactiva a partir de datos proporcionados por el usuario. En ese sentido, estamos satisfechos de haber logrado desarrollar esta funcionalidad.

Sin embargo, es cierto que nos ha faltado tiempo para poder implementar muchos elementos de la página, algunos esenciales. Desde el principio del proyecto sufrimos un bloqueo que nos hizo perder mucho tiempo efectivo de desarrollo. Creemos que esto se ha notado en los últimos momentos, cuando ese tiempo nos había venido muy bien para dar las últimas pinceladas al trabajo.

Por otra parte, la comunicación entre el desarrollo del frontend y del backend ha sido mejorable. No hemos estado suficientemente coordinados y eso nos ha traído problemas. Cuando unimos ambas partes, los datos que ofrecía el back no coincidían con los que pedía el front. Esto ha requerido de muchas horas de trabajo para ajustar ambas partes del proyecto y que encajaran entre sí.

Encontrarnos con este revés en los últimos días antes de entregar el proyecto también ha hecho que no tuviésemos tiempo material de desplegar la web. Por tanto, pese a que hemos desarrollado la funcionalidad principal que queríamos en nuestro proyecto, no hemos conseguido que la web pueda ser usada por nuestros hipotéticos clientes.

[1] Talent.com. (2024). *Salario programador junior en España*. Recuperado de https://es.talent.com/salary?job=programador+junior

Sin duda la unión del front y back y el despliegue son los aspectos más importantes a mejorar, pero también hay otros detalles menores que se han quedado pendientes en el tintero:

- Traspasar todo el código CSS a SCSS
- Comentar y documentar el código en profundidad
- Añadir más elementos a la orla, como el escudo del centro educativo
- Implementar drag & drop para añadir personas a la orla
- Poder editar orlas ya creadas (faltaría conectar con backend)
- Añadir más variedad de fondos a las orlas y diseñar fondos propios, evitando infringir el copyright
- Mejorar la accesibilidad
- Mejorar la experiencia del usuario (correos de confirmación, poder cambiar contraseñas, etc)
- Añadir información a la página de inicio sobre la empresa Fotoorla, servicios, precios, formulario de contacto, etc.
- Implementar las mejoras sugeridas inicialmente por Fotoorla.es (SEO, correos a los clientes, mejora de paneles internos, etc.)

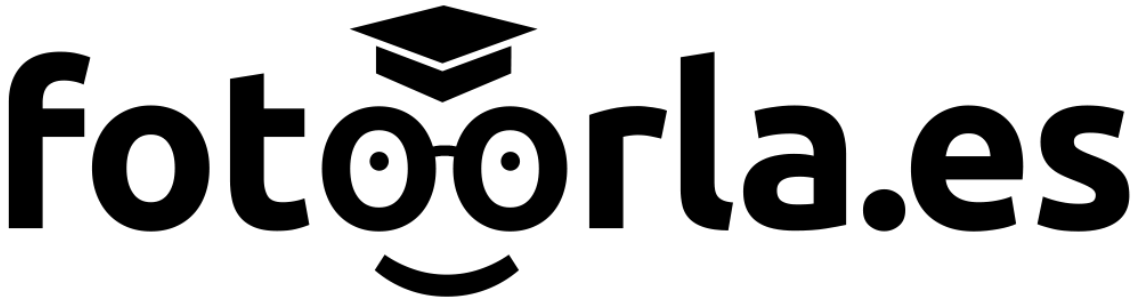
9 Bibliografía/Webgrafía.

9.1 Webgrafía

- Bootstrap Core Team. (n.d.). Get started with Bootstrap. Bootstrap. https://getbootstrap.com/docs/5.3/getting-started/introduction/
- Bootstrap Core Team. (n.d.). Bootstrap Icons. Bootstrap. https://icons.getbootstrap.com/
- Google Inc. (n.d.) Introduction to the Angular docs. Angular. https://v17.angular.io/docs/
- Talent.com. (2024). *Salario programador junior en España*. Recuperado de https://es.talent.com/salary?job=programador+junior
- Fotoorla.es
- https://stablediffusionweb.com/ IA Generated Images.

- Spring (Documentación) https://spring.io/projects/spring-restdocs

9.2 Imágenes

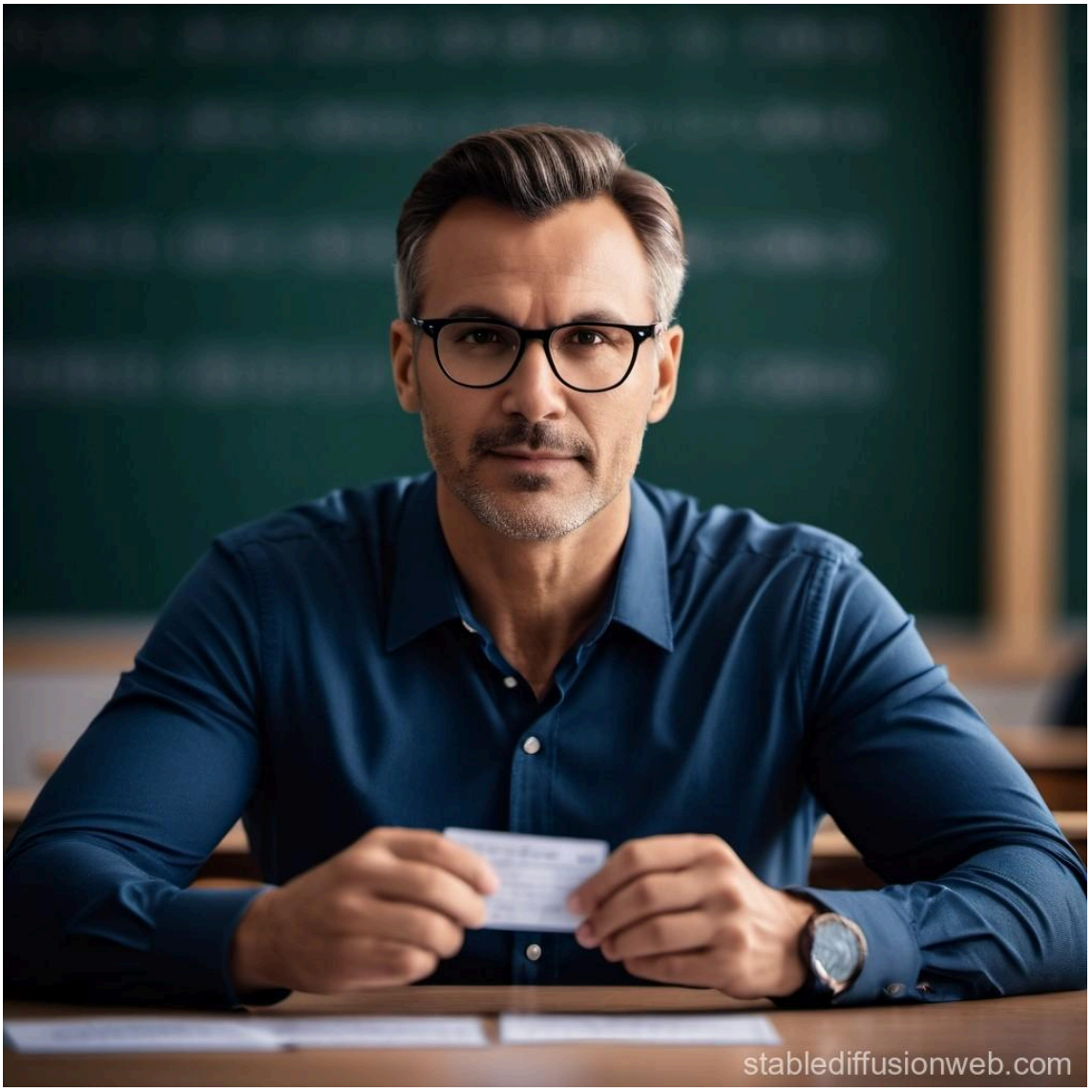


fotoorla.es. (n.d.). [Logo de la web Fotorla]. fotoorla.es. https://fotoorla.es/

fotoorla.es. (n.d.). [Favicon de la web Fotorla]. fotoorla.es. https://fotoorla.es/

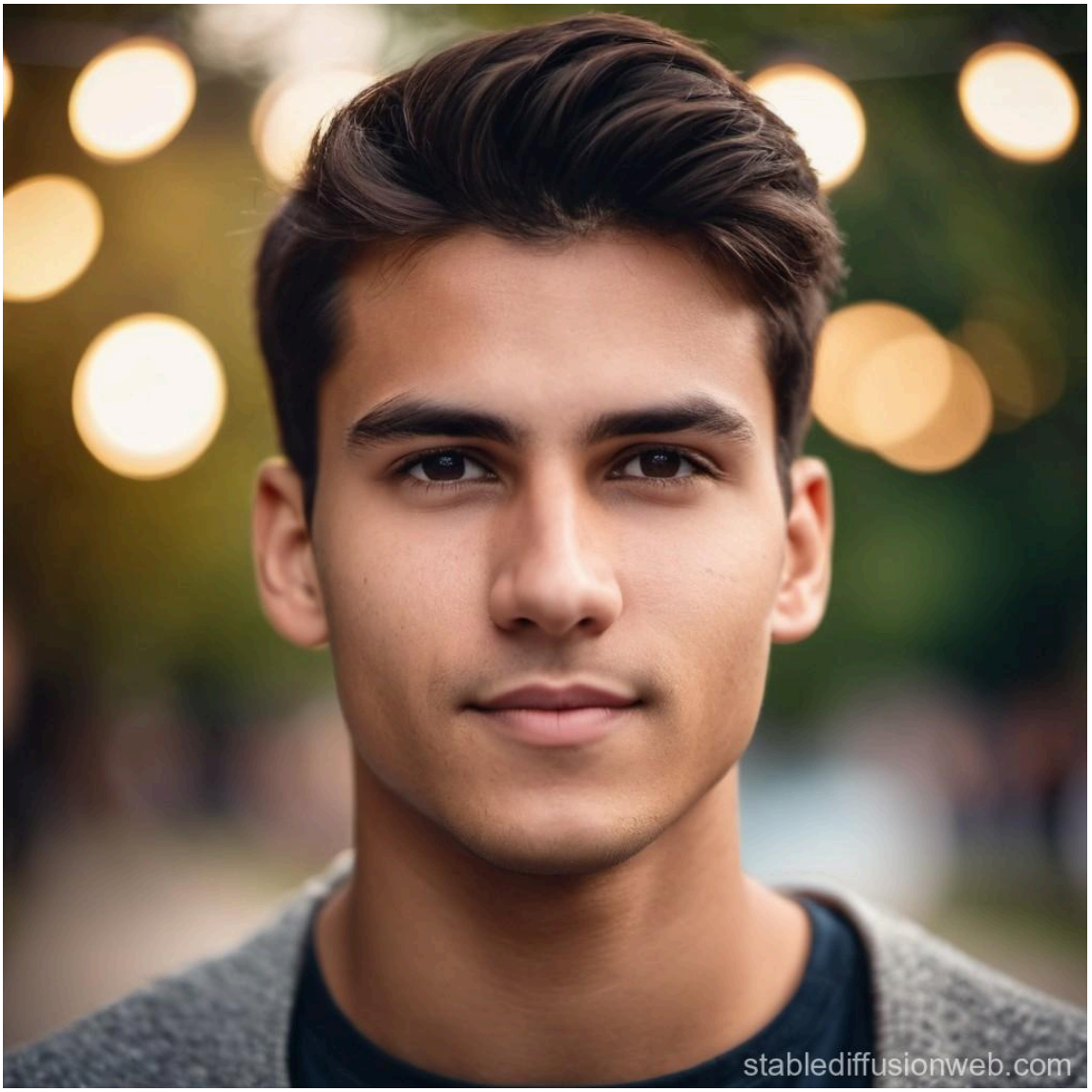
orlaonline.es. (n.d.). [Marco Flores I]. orlaonline.es. https://www.orlaonline.es/files/plantillas/web/ORLA-1_web_2.jpg?72

orlaonline.es. (n.d.). [Universitaria Moderna]. orlaonline.es. https://www.orlaonline.es/files/plantillas/web/ORLA_UNIVERSITAD-01_web.jpg?72



Sergio.Q. (2024). Imagen de un profesor [Imagen generada por IA]. Stable Diffusion Web. https://stablediffusionweb.com/

Sergio.Q. (2024). Imagen de una profesora [Imagen generada por IA]. Stable Diffusion Web. https://stablediffusionweb.com/



Sergio.Q. (2024). Imagen de un alumno [Imagen generada por IA]. Stable Diffusion Web. https://stablediffusionweb.com/

Sergio.Q. (2024). Imagen de una alumna[Imagen generada por IA]. Stable Diffusion Web. https://stablediffusionweb.com/

Sergio.Q. (2024). Imagen de un alumno [Imagen generada por IA]. Stable Diffusion Web. https://stablediffusionweb.com/