## PROCEDIMIENTO.

### Procedimiento del backend.

#### Creación del esqueleto del sitio web

Vamos a crear un proyecto de "esqueleto" de sitio web como base, que puedes continuar luego llenado de configuraciones específicas del sitio, urls, modelos, vistas y plantillas.

El proceso es sencillo:

1. Usar la herramienta django-admin para crear la carpeta del proyecto, los ficheros de plantillas básicos y el script de gestión del proyecto (manage.py).
2. Usar manage.py para crear una o más aplicaciones
3. Registrar las nuevas aplicaciones para incluirlas en el proyecto.
4. Conectar el mapeador url de cada aplicación.

**Instalación de django y creación del proyecto**

Para instalar Django se puede usar el administrador de paquetes pip, que es una herramienta para instalar y administrar paquetes de software escritos en Python. Para instalar Django, abra una ventana de comandos y ejecute el siguiente comando:

1. Vamos a crear y activar una unidad virtual en nuestro cmd con las siguientes líneas de comandos

* py -m venv revista
* revista\Scripts\activate

1. Vamos a instalar Django en nuestra unidad virtual y verificamos que se haya instalado la versión que queremos.

* pip install Django==4.1.7
* pip freeze

1. Una vez que se complete la instalación, puede crear un proyecto Django ejecutando el siguiente comando:

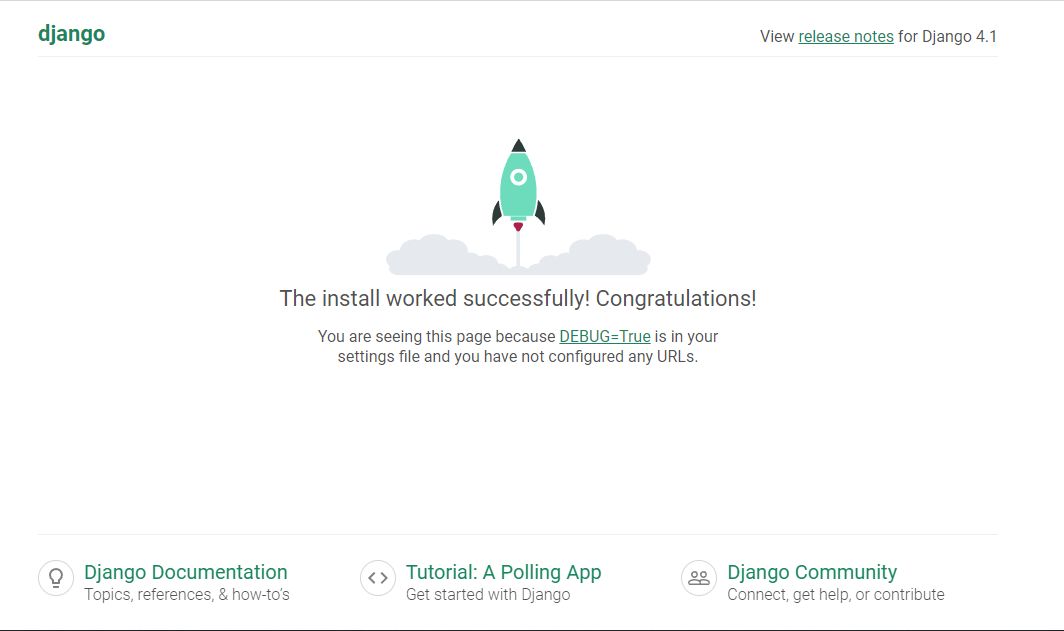
* django-admin startproject config
* cd config

Esto creará un directorio con el nombre del proyecto que contiene todos los archivos necesarios para ejecutar el proyecto. Para iniciar el servidor de desarrollo, vaya al directorio del proyecto y ejecute el siguiente comando:

* python manage.py runserver

Esto iniciará el servidor de desarrollo en el puerto 8000. Ahora puede abrir su navegador y navegar a http://127.0.0.1:8000/ para ver la página de inicio de Django.

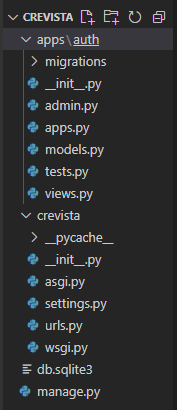
##### **Inicio basico Django**



Después creamos una carpeta llamada apps que es donde vamos a crear las aplicaciones que vamos a utilizar en nuestro proyecto, nos ubicamos en la carpeta apps y con el comando:

* django-admin startapp authenticacion
* django-admin startapp revista

Tendremos nuestro proyecto de la siguiente manera.



**Registro de la aplicación auth**

Ahora que se ha creado la aplicación tenemos que registrarla en el proyecto de manera que sea incluida cuando cualquiera de las herramientas se ejecute (por ejemplo, para añadir modelos a la base de datos). Las aplicaciones se registran añadiendoles a la lista de INSTALLED\_APPS en los ajustes del proyecto.

Abre el fichero de ajustes del proyecto crevista/crevista/settings.py y encuentra la definición de la lista INSTALLED\_APPS. Añade a continuación una nueva línea al final de la lista, como se muestra en negrilla abajo.

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'apps.auth.apps.AuthConfig',

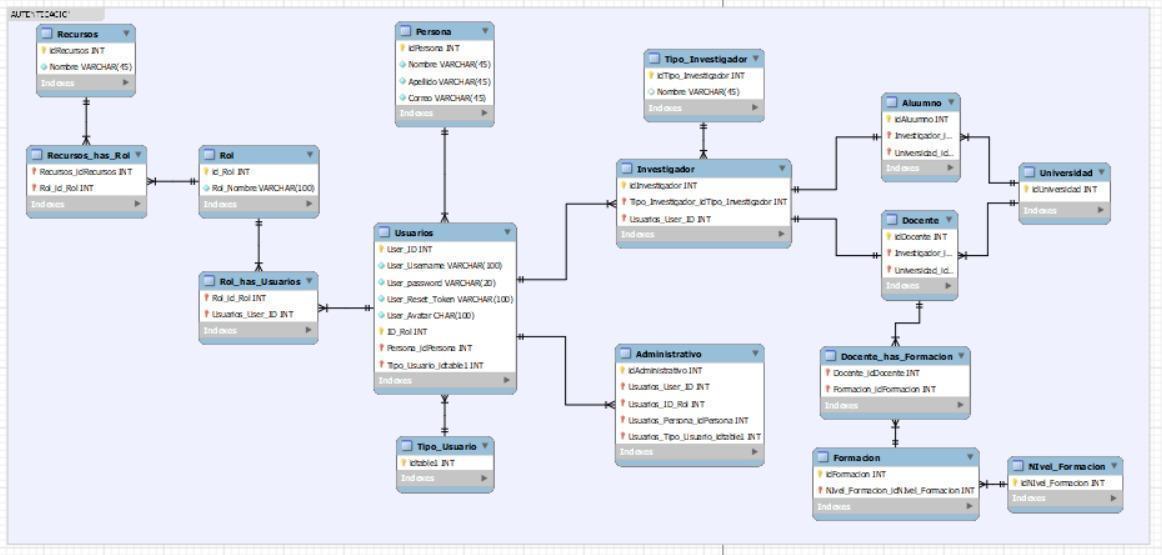
'apps.autenticacion.apps.AutenticacionConfig',

'apps.revista.apps.RevistaConfig',

****Uso de modelos.****

Para la creación de los modelos debemos tener claro que tablas vamos a utilizar y tenerlas en diagrama donde se te facilite llevar a cabo la realización de los mismos

##### **Diagrama de modelos**



**Código de los Modelos**

**Aplicación Auth**

Los códigos de los modelos se realizan dentro de cada aplicación, que para este caso en particular se crearían en apps/auth/models.py de la siguiente manera:

En este caso tomaremos la tabla de clientes dentro de la aplicación correspondiente

### Modelo de Auth

from django.db import models

from django.contrib.auth.models import AbstractUser

from django.utils.translation import gettext\_lazy as \_

from django.contrib.auth.models import AbstractUser

class BaseModel(models.Model):

createdAt = models.DateField(auto\_now\_add=True)

updateAt = models.DateField(auto\_now=True, blank=True, null=True)

class Meta:

abstract = True

class Personas(BaseModel):

nombre = models.CharField(max\_length=150)

apellido = models.CharField(max\_length=150)

correo = models.CharField(max\_length=150, blank=True, null=True)

def \_\_str\_\_(self) -> str:

return self.nombre

class Meta:

unique\_together = (('nombre', 'correo'))

verbose\_name = 'Persons'

verbose\_name\_plural = 'Persons'

class TipoUsuarios(BaseModel):

nombre = models.CharField(max\_length=100)

class Meta:

verbose\_name = 'Tipo\_users'

verbose\_name\_plural = 'Tipo\_users'

def \_\_str\_\_(self) -> str:

return self.nombre

class Usuarios(AbstractUser, BaseModel):

username = models.CharField(max\_length=100)

password = models.CharField(max\_length=100)

resetToken = models.CharField(max\_length=256, blank=True, null=True)

avatar = models.CharField(max\_length=256, blank=True, null=True)

roles = models.ManyToManyField(

'Roles', through='User\_roles', related\_name='user\_roles')

tipousuario = models.ForeignKey(TipoUsuarios, null=True, black=True, on\_delete=models.CASCADE,

verbose\_name='TipoUsuario')

class Meta:

verbose\_name = 'Users'

verbose\_name\_plural = 'Users'

def \_\_str\_\_(self) -> str:

return self.username

class Roles(BaseModel):

nombre = models.CharField(max\_length=200, unique=True)

estado = models.IntegerField()

users = models.ManyToManyField(

Usuarios, through='User\_roles', related\_name='roles\_user')

resources = models.ManyToManyField(

'Resources', through='Resources\_roles', related\_name='roles\_resources')

def \_\_str\_\_(self) -> str:

return self.nombre

class Meta:

verbose\_name = 'Roles'

verbose\_name\_plural = 'Roles'

class User\_roles(BaseModel):

estado = models.IntegerField()

userId = models.ForeignKey(

Usuarios, on\_delete=models.CASCADE, related\_name='users')

rolesId = models.ForeignKey(

Roles, on\_delete=models.CASCADE, related\_name='roles')

def \_\_str\_\_(self) -> str:

return self.userId.username + '-' + self.rolesId.nombre

class Meta:

unique\_together = (('userId', 'rolesId'))

verbose\_name = 'User\_roles'

verbose\_name\_plural = 'user\_roles'

class Resources(BaseModel):

path = models.CharField(max\_length=256)

id\_padre = models.IntegerField()

estado = models.IntegerField()

method = models.CharField(max\_length=256)

icono = models.CharField(max\_length=256)

link = models.CharField(max\_length=256)

titulo = models.CharField(max\_length=100)

roles = models.ManyToManyField(

Roles, through='Resources\_roles', related\_name='resources\_roles')

class Meta:

verbose\_name = 'Resources'

verbose\_name\_plural = 'Resources'

class Resources\_roles(BaseModel):

resourcesId = models.ForeignKey(

Resources, on\_delete=models.CASCADE, related\_name='resources')

rolesId = models.ForeignKey(

Roles, on\_delete=models.CASCADE, related\_name='resouces\_roles')

estado = models.IntegerField()

def \_\_str\_\_(self) -> str:

return self.resourcesId.path + '' + self.rolesId.name

class Meta:

verbose\_name = 'Resources\_roles'

verbose\_name\_plural = 'Resources\_roles'

Cada modelo obedece a una clase que hereda o extiende de la clase models.Model, la cual debemos importar.

Tipo de los Campos

El tipo del campo puede ser (Numérico, Fecha, Boolean, entre otros). Para saber todos los tipos de los campos: [TIPOS DE CAMPOS](https://docs.djangoproject.com/en/1.10/ref/models/fields/#field-options)

Los siguientes tipos de campos comunes:

|  |  |
| --- | --- |
| **NOMBRE DEL CAMPO** | **DESCRIPCIÓN** |
| [AutoField](https://es.acervolima.com/autofield-modelos-de-django/) | Es un IntegerField que se incrementa automáticamente. |
| [BigAutoField](https://es.acervolima.com/bigautofield-modelos-django/) | Es un entero de 64 bits, muy parecido a un AutoField, excepto que se garantiza que se ajusta a los números del 1 al 9223372036854775807. |
| [BigIntegerField](https://es.acervolima.com/bigintegerfield-modelos-de-django/) | Es un número entero de 64 bits, muy parecido a un IntegerField, excepto que se garantiza que se ajusta a números de -9223372036854775808 a 9223372036854775807. |
| [BinaryField](https://es.acervolima.com/binaryfield-modelos-django/) | Un campo para almacenar datos binarios sin procesar. |
| [BooleanField](https://es.acervolima.com/booleanfield-modelos-de-django/) | Un campo verdadero / falso. El widget de formulario predeterminado para este campo es CheckboxInput. |
| [CharField](https://es.acervolima.com/charfield-modelos-django/) | Es una fecha, representada en Python por una instancia de datetime.date. |
| [DateField](https://es.acervolima.com/datefield-modelos-de-django/) | Una fecha, representada en Python por una instancia de datetime.date |
|  | Se utiliza para la fecha y la hora, representada en Python por una instancia de datetime.datetime. |
| [DecimalField](https://es.acervolima.com/decimalfield-modelos-django/) | Es un número decimal de precisión fija, representado en Python por una instancia Decimal. |
| [DurationField](https://es.acervolima.com/durationfield-modelos-django/) | Un campo para almacenar períodos de tiempo. |
| [EmailField](https://es.acervolima.com/emailfield-modelos-de-django/) | Es un CharField que verifica que el valor sea una dirección de correo electrónico válida. |
| [FileField](https://es.acervolima.com/filefield-modelos-django/) | Es un campo de carga de archivos. |
| [FloatField](https://es.acervolima.com/floatfield-modelos-django/) | Es un número de punto flotante representado en Python por una instancia flotante. |
| [ImageField](https://es.acervolima.com/imagefield-modelos-django/) | Hereda todos los atributos y métodos de FileField, pero también valida que el objeto cargado es una imagen válida. |
| [IntegerField](https://es.acervolima.com/integerfield-modelos-django/) | Es un campo entero. Los valores de -2147483648 a 2147483647 están seguros en todas las bases de datos compatibles con Django. |
| [GenericIPAddressField](https://es.acervolima.com/genericipaddressfield-modelos-de-django/) | Una dirección IPv4 o IPv6, en formato de cadena(por ejemplo, 192.0.2.30 o 2a02: 42fe :: 4). |
| [NullBooleanField](https://es.acervolima.com/nullbooleanfield-formularios-de-django/) | Como BooleanField, pero permite NULL como una de las opciones. |
| [PositiveIntegerField](https://es.acervolima.com/positiveintegerfield-modelos-de-django/) | Como un IntegerField, pero debe ser positivo o cero(0). |
| [PositivoSmallIntegerField](https://es.acervolima.com/positivesmallintegerfield-modelos-de-django/) | Como PositiveIntegerField, pero solo permite valores bajo un cierto punto(dependiente de la base de datos). |
| [SlugField](https://es.acervolima.com/slugfield-modelos-django/) | Slug es un término periodístico. Una slug es una etiqueta corta para algo, que contiene solo letras, números, guiones bajos o guiones. Generalmente se utilizan en URL. |
| [SmallIntegerField](https://es.acervolima.com/smallintegerfield-modelos-django/) | Es como un IntegerField, pero solo permite valores bajo un cierto punto(dependiente de la base de datos). |
| [TextField](https://es.acervolima.com/textfield-modelos-de-django/) | Un campo de texto grande. El widget de formulario predeterminado para este campo es un área de texto. |
| [TimeField](https://es.acervolima.com/timefield-modelos-django/) | Una hora, representada en Python por una instancia de datetime.time. |
| [URLField](https://es.acervolima.com/urlfield-modelos-django/) | Un CharField para una URL, validado por URLValidator. |
| [UUIDField](https://es.acervolima.com/uuidfield-modelos-django/) | Un campo para almacenar identificadores únicos universales. Utiliza la clase UUID de Python. Cuando se usa en PostgreSQL, esto se almacena en un tipo de datos uuid, de lo contrario en un char(32). |

### Argumento de los Campos

Los siguientes argumentos son comunes a la mayoría de los tipos de campo y pueden usarse al declararlos:

* max\_length=20 --- Establece que la longitud máxima del valor de este campo es 20 caracteres.
* help\_text="Enter field documentation" --- proporciona una etiqueta de texto para mostrar que ayuda a los usuarios a saber qué valor proporcionar cuando un usuario ha de introducirlo vía un formulario HTML.
* [help\_text](https://docs.djangoproject.com/en/1.10/ref/models/fields/#help-text): Proporciona una etiqueta de texto para formularios HTML (ej. en el sitio de Administración), tal como se describe arriba.
* [verbose\_name](https://docs.djangoproject.com/en/1.10/ref/models/fields/#verbose-name): Nombre de fácil lectura que se usa en etiquetas para el campo. Si no se especifica, Django inferir el valor por defecto del verbose name a partir del nombre del campo.
* [default](https://docs.djangoproject.com/en/1.10/ref/models/fields/#default): Valor por defecto para el campo. Puede ser un valor o un callable object (objeto que puede ser llamado como una función), en cuyo caso el objeto será llamado cada vez que se cree un nuevo registro.
* [null](https://docs.djangoproject.com/en/1.10/ref/models/fields/#null): Si es True, Django guardará valores en blanco o vacíos como NULL en la base de datos para campos donde sea apropiado (un CharField guardará una cadena vacía en su lugar). Por defecto es False.
* [blank](https://docs.djangoproject.com/en/1.10/ref/models/fields/#blank): Si es True, se permite que el campo quede en blanco en tus formularios. El valor por defecto es False, lo que significa que la validación de formularios de Django te forzará a introducir un valor. Con frecuencia se usa con null=True, porque si vas a permitir valores en blanco, también querrás que la base de datos sea capaz de representarlos de forma apropiada.
* [choices](https://docs.djangoproject.com/en/1.10/ref/models/fields/#choices): Un grupo de valores de selección para este campo. Si se proporciona, el widget correspondiente por defecto del formulario será una caja de selección con estos valores de selección en vez del campo de texto estándar.
* [primary\_key](https://docs.djangoproject.com/en/1.10/ref/models/fields/#primary-key): Si es True, establece el campo actual como clave primaria para el modelo (Una clave primaria es una columna especial de la base de datos, diseñada para identificar de forma única todos los diferentes registros de una tabla). Si no se especifica ningún campo como clave primaria, Django añadirá automáticamente un campo para este propósito.

**Métodos**

Un modelo puede tener también métodos Mínimamente, en cada modelo deberías definir el método estándar de las clases de Python \_\_str\_\_() para devolver una cadena de texto legible por humanos para cada objeto. Esta cadena se usa para representar registros individuales en el sitio de administración (y en cualquier otro lugar donde necesites referirse a una instancia del modelo). Con frecuencia éste devolverá un título o nombre de campo del modelo.

def \_\_str\_\_(self):

return self.nombre

Recuerde que self.nombre ==> self, indica que estamos en una función dentro de una clase, es decir un método y nombre hace referencia a la propiedad o variable de la clase.

### Metadatos

Puedes declarar metadatos a nivel de modelo para tu Modelo declarando class Meta, tal como se muestra.

class Meta:

verbose\_name = "cliente"

verbose\_name\_plural = "clientes"

La forma en que aparecen los modelos en el admin que genera Django.

### Especificación de la base de datos

Primero se deben instalar las extensiones en Django que permiten comunicarse con el motor de la base de datos.

Para este caso utilizaremos MySql, por lo cual instalamos:

mysqlclient

python3 -m pip install mysqlclient

En caso que se genere algún error probar con la instalación de la siguiente manera:

sudo apt-get install libmysqlclient-dev

Y luego intentar nuevamente

python3 -m pip install mysqlclient

Éste es también el punto donde normalmente especificamos la base de datos a utilizar en el proyecto --- tiene sentido usar la misma base datos para desarrollo y producción donde sea posible, para evitar diferencias menores en el comportamiento

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'db\_sistemas',

'USER': 'root',

'PASSWORD': '',

'HOST': 'localhost',

'PORT': '3306',

}

}

Esto nos indica que procedemos a varios procesos.

* Instalar el motor de la base de datos que para este caso es MySql.  
  <https://docs.microsoft.com/en-us/windows/wsl/tutorials/wsl-database>
* Una vez instalado el motor debemos iniciar dicho motor.  
  Abrimos la terminal de Ubuntu WSL y escribimos.

sudo /etc/init.d/mysql start Si aparece este mensaje, quiere decir que ya está inicializado.

\* Starting MySQL database server mysqld

* Crear la base de datos indicada: 'NAME': 'almacen2022\_mysql', para ello, en la terminal de Ubuntu WSL ingresar sudo mysql -u root -p Y luego la contraseña.

Si aparece en la terminal mysql> Indica que ya entró al motor de MySQL.  
Luego realice: show databases; Para mostrar las bases de datos que tiene creadas create database almacen2022\_mysql; Debe aparecer en pantalla, para indicar que ha sido creada. Query OK, 1 row affected (0.01 sec)

Ya podemos probar realizando las migraciones, sin aun haber realizado los modelos. Esto es debido a que Django ya trae unos modelos preestablecidos.

Para ello, Escribimos lo siguiente:

python3 manage.py makemigrations => Crea las migraciones

python3 manage.py migrate ==> Crea la base de datos físicamente, de acuerdo a las migraciones actuales.

1. Vamos a admin.py donde vamos a Importar los módulos necesarios de Django:

from django.contrib import admin

from django.contrib.auth import get\_user\_model

from django.contrib.auth.admin import UserAdmin

from .models import Persons, Genders, Document\_types, Resources, Resources\_roles, Roles, User\_roles

from django.contrib.auth.models import Group

1. Registrar los modelos para la administración usando admin.site.register():

admin.site.register(Persons)

admin.site.register(Genders)

admin.site.register(Document\_types)

admin.site.register(Resources)

admin.site.register(Resources\_roles)

admin.site.register(Roles)

#admin.site.register(User);

admin.site.register(User\_roles)

1. Personalizar el modelo User usando @admin.register(get\_user\_model()) y la clase CustomUserAdmin:

@admin.register(get\_user\_model())

class CustomUserAdmin(UserAdmin):

pass

1. Anular el registro del modelo Group en la interfaz de administración usando admin.site.unregister(Group):

admin.site.unregister(Group)

Creamos un archivo serializer.py donde vamos a Importar los módulos necesarios de Django y Django REST Framework:

from django.contrib.auth import get\_user\_model

from django.contrib.auth.hashers import make\_password

from rest\_framework import serializers

from rest\_framework.validators import UniqueValidator

from rest\_framework.serializers import CharField, ModelSerializer, SlugField

from .models import CustomUser

from .api.serializer.serializers import RolesSerializers

from .api.serializer.customValidators import UserValidatorBefore

User = get\_user\_model()

Definir un modelo de serializador UserChangePassword para permitir que un usuario cambie su contraseña. Este serializador es un ModelSerializer de DRF que tiene dos campos, password e id, que son campos del modelo CustomUser y deben validarse antes de permitir que se cambie la contraseña.

class UserChangePassword(ModelSerializer):

password = CharField()

class Meta:

model = CustomUser

fields = ('password', 'id')

validators = [UserValidatorBefore()]

Definir un modelo de serializador CreateUserSerializers para crear un nuevo usuario. Este serializador es un ModelSerializer de DRF que tiene tres campos, username, password y email, que son campos del modelo CustomUser y deben validarse antes de permitir que se cree un nuevo usuario. Además, el campo username tiene un validador UniqueValidator que verifica que el valor ingresado no esté en uso por otro usuario existente.

class CreateUserSerializers(ModelSerializer):

username = SlugField(

max\_length=100,

validators=[UniqueValidator(queryset=User.objects.all())]

)

class Meta:

model = CustomUser

fields = ('username', 'password', 'email')

validators = [UserValidatorBefore()]

Definir un modelo de serializador UserSerializersSimpleRegister para crear un nuevo usuario con campos adicionales como first\_name y last\_name. Este serializador es similar al CreateUserSerializers, pero tiene campos adicionales.

**class UserSerializersSimpleRegister(ModelSerializer):**

**username = SlugField(**

**max\_length=100,**

**validators=[UniqueValidator(queryset=User.objects.all())]**

**)**

**class Meta:**

**model = CustomUser**

**fields = ('username', 'password', 'email', 'first\_name', 'last\_name')**

**validators = [UserValidatorBefore()]**

Definir un modelo de serializador UserSerializer para mostrar la información del usuario. Este serializador es un ModelSerializer de DRF que tiene campos personalizados para mostrar la información del usuario, como roles, email, username, password y avatar. Además, tiene un método to\_representation() para convertir la representación del objeto en el formato deseado y validar los roles del usuario.

**alizers.ModelSerializer para serializar los objetos User:**

class UserSerializer(serializers.ModelSerializer):

roles = RolesSerializers(many=True, read\_only=True)

email = serializers.EmailField(

required=True)

username = serializers.CharField(

required=True)

password = serializers.CharField(

min\_length=8)

avatar = serializers.ImageField(

required=False, allow\_null=True)

def to\_representation(self, instance):

representation = super().to\_representation(instance)

try:

if (len(representation['roles'])):

representation['roles'][0] = representation['roles'][0]['id']

return representation

except Exception as e:

return representation

class Meta:

model = get\_user\_model()

fields = ('email', 'username', 'roles', 'password', 'avatar')

def validate\_password(self, value):

return make\_password(value)

def validate\_username(self, value):

value = value.replace(" ", "") # Ya que estamos borramos los espacios

try:

user = get\_user\_model().objects.get(username=value)

# Si es el mismo usuario mandando su mismo username le dejamos

if user == self.instance:

return value

except get\_user\_model().DoesNotExist:

return value

raise serializers.ValidationError("Nombre de usuario en uso")

def validate\_email(self, value):

A la vista vamos a crear un archivo views.py donde este es un código en Python, utilizando el framework Django, que contiene varios endpoints para realizar acciones relacionadas con los usuarios del sistema. A continuación, se describe el código por cada una de las clases y métodos:

from django.dispatch import receiver

from django.contrib.auth import authenticate, login, logout

from django.contrib.auth.hashers import make\_password

from django\_rest\_passwordreset.signals import reset\_password\_token\_created

from rest\_framework import generics, status

from rest\_framework.views import APIView

from rest\_framework.response import Response

from django.http import HttpResponse

from rest\_framework.generics import CreateAPIView, UpdateAPIView, RetrieveAPIView

from .serializers import UserSerializer, CreateUserSerializers, UserChangePassword

#from ...serializers.user.users\_serializers import UserSerializers, CreateUserSerializers, UserChangePassword

from .models import CustomUser

from .mudules import create\_response

from apps.authenticacion.api.serializer.auth\_serializer import LoginSerializers, RegisterSerializers

from apps.authenticacion.api.serializer.serializers import ResourcesSerializers, ResourcesRolesSerializers

from rest\_framework\_simplejwt.tokens import RefreshToken, TokenError

from helps.flatList import flatList

from django.http import JsonResponse

import bcrypt, logging

En esta sección se importan los paquetes necesarios para el correcto funcionamiento del código. Se importan las funciones de Django y de Django Rest Framework necesarias para crear endpoints de API, modelos y serializadores.

Además, se importan algunos módulos personalizados (create\_response y flatList) y los serializadores (LoginSerializers, RegisterSerializers, ResourcesSerializers, ResourcesRolesSerializers, UserSerializer, CreateUserSerializers y UserChangePassword) y el modelo (CustomUser) necesarios para el funcionamiento de los endpoints.

class UsersViewPublic(RetrieveAPIView):

queryset = CustomUser.objects.all()

serializer\_class = UserSerializer

def get(self, request, \*args, \*\*kwargs):

users = self.get\_queryset()

serializers = UserSerializer(users, many=True)

response, code = create\_response(

status.HTTP\_200\_OK, 'User Public', serializers.data)

return Response(response, status=code)

Este es un endpoint que permite obtener una lista de usuarios en el sistema, a través del método HTTP GET. Se define una clase UsersViewPublic que hereda de RetrieveAPIView, la cual es una clase que ya tiene implementado el método HTTP GET para obtener un objeto específico del modelo.

En la clase se define el atributo queryset con el valor de todos los objetos del modelo CustomUser. Luego se define el atributo serializer\_class con el valor del serializador UserSerializer.

Finalmente, se sobreescribe el método get() para obtener la lista de usuarios, serializarlos con UserSerializer y crear una respuesta a través de la función create\_response.

class UserCreateView(CreateAPIView):

queryset = CustomUser.objects.all()

serializer\_class = CreateUserSerializers

def perform\_create(self, serializer):

password = make\_password(self.request.data['password'])

serializer.save(password=password)

def post(self, request, \*args, \*\*kwargs):

userSerializers = self.get\_serializer(data=request.data)

if userSerializers.is\_valid():

self.perform\_create(userSerializers)

response, code = create\_response(

status.HTTP\_200\_OK, 'User Create', userSerializers.data)

return Response(userSerializers.data, status=code)

response, code = create\_response(

status.HTTP\_200\_OK, 'Error', userSerializers.data)

return Response(response, status=code)

### UserUpdateView

La vista UserUpdateView es una vista de actualización que se utiliza para actualizar los datos de un usuario existente en la base de datos. Se requiere autenticación para acceder a esta vista. Devuelve una respuesta JSON que contiene los datos actualizados del usuario.

class UserUpdateView(UpdateAPIView):

queryset = CustomUser.objects.all()

serializer\_class = UserSerializer

def get\_object(self):

try:

request\_user = self.kwargs['pk']

user = CustomUser.objects.get(pk=request\_user)

return user

except CustomUser.DoesNotExist:

return None

except Exception as e:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', e)

return Response(response, status=code)

def perform\_update(self, serializer):

serializer.save()

def put(self, request, \*args, \*\*kwargs):

partial = kwargs.pop('partial', False)

user = self.get\_object()

if user is None:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Password Error', 'User Not found')

return Response(response, status=code)

try:

userSerializers = UserSerializer(

user, data=request.data, partial=partial)

if userSerializers.is\_valid():

self.perform\_update(userSerializers)

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Password Error', 'User Not found')

return Response(response, status=code)

return Response(userSerializers.errors, 'Error', status=status.HTTP\_400\_BAD\_REQUEST)

except (AttributeError, Exception) as e:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Not Found', e.args)

return Response(response, status=code)

### UserChangePasswordView

La vista UserChangePasswordView es una vista de actualización que se utiliza para cambiar la contraseña de un usuario existente en la base de datos. Se requiere autenticación para acceder a esta vista. Devuelve una respuesta JSON que indica si la contraseña se ha actualizado correctamente o no.

Cada vista contiene un queryset, que define el conjunto de objetos de modelo que se utilizarán para esta vista. También contiene un serializador, que se utiliza para serializar los datos de entrada y salida en un formato JSON. El método get\_object se utiliza para obtener un objeto de modelo específico de la base de datos. El método perform\_create se utiliza para crear un nuevo objeto de modelo en la base de datos. El método perform\_update se utiliza para actualizar un objeto de modelo existente en la base de datos.

Además, el archivo contiene importaciones de varias bibliotecas y módulos necesarios para el funcionamiento del código, como rest\_framework, django\_rest\_passwordreset, bcrypt, etc.

En resumen, este archivo de vista contiene varias vistas basadas en clases que se utilizan para crear, actualizar, leer y eliminar objetos CustomUser de la base de datos. Cada vista está diseñada para manejar una tarea específica y proporciona una respuesta JSON adecuada para cada solicitud.

class UserChangePasswordView(UpdateAPIView):

queryset = CustomUser.objects.all()

serializer\_class = UserSerializer

def get\_object(self):

try:

request\_user = self.kwargs['pk']

user = CustomUser.objects.get(pk=request\_user)

return user

except (CustomUser.DoesNotExist, TypeError):

return None

except (BaseException, TypeError) as e:

return None

def perform\_update(self, serializer):

if 'original-password' in self.request.data:

password = self.request.data['password'].encode('utf-8')

hashed\_password = bcrypt.hashpw(password, bcrypt.gensalt())

serializer.save(password=hashed\_password.decode('utf-8'))

else:

serializer.save()

def patch(self, request, \*args, \*\*kwargs):

partial = kwargs.pop('partial', False)

user = self.get\_object()

if user is None:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Not Found', e.args)

return Response(response, status=code)

if 'original-password' not in self.request.data:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Password Error', 'Password not found')

return Response(response, status=code)

if not user.check\_password(request.data['original-password']):

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Password Error', 'Password is not correct.')

return Response(response, status=code)

userSerializers = UserChangePassword(

user, data=request.data, partial=partial, context={'context': request})

try:

if userSerializers.is\_valid():

self.perform\_update(userSerializers)

response, code = create\_response(

status.HTTP\_200\_OK, 'Password', 'Password Change')

return Response(response, status=code)

return Response(userSerializers.errors, status=status.HTTP\_400\_BAD\_REQUEST)

except (AttributeError, Exception) as e:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Not Found', e.args)

return Response(response, status=code)

AuthLogin que hereda de la clase APIView. Tiene dos métodos: get\_tokens\_for\_user y post. El método get\_tokens\_for\_user toma un usuario como argumento y devuelve un diccionario con dos claves: "refresh" y "access". Estas claves contienen tokens de actualización y de acceso respectivamente.

El método post toma una solicitud request y algunos argumentos adicionales. Comprueba si la solicitud contiene una clave "email" en sus datos. Si es así, establece el nombre de usuario en la clave "email" y la contraseña en la clave "password". Si no, utiliza los datos tal como están. Luego, se crea un objeto LoginSerializers con los datos de la solicitud y un contexto que contiene la solicitud. Si el objeto serializers no es válido, se devuelve una respuesta HTTP 400 con un mensaje de error. Si es válido, se llama a la función login para autenticar al usuario y se obtiene un token de acceso utilizando el método get\_tokens\_for\_user.

A continuación, se construye un menú de recursos utilizando un objeto ResourcesSerializers. Se establece un token de actualización en la sesión de la solicitud y se devuelve una respuesta HTTP 200 con un mensaje de éxito que contiene el token de acceso, información del usuario (nombre e id) y el menú de recursos.

class AuthLogin(APIView):

def get\_tokens\_for\_user(self, user):

refresh = RefreshToken.for\_user(user)

return {

'refresh': str(refresh),

'access': str(refresh.access\_token),

}

def post(self, request, \*args, \*\*kwargs):

data = {}

if 'email' in request.data:

data['username'] = request.data['email']

data['password'] = request.data['password']

else:

data = request.data

serializers = LoginSerializers(

data=data, context={'request': self.request})

if not serializers.is\_valid():

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', serializers.errors)

return Response(response, status=code)

login(request, serializers.validated\_data)

token = self.get\_tokens\_for\_user(serializers.validated\_data)

resources = flatList([e.resources.prefetch\_related(

'resources') for e in serializers.validated\_data.roles.all()])

menu = ResourcesSerializers(set(resources), many=True)

request.session['refresh-token'] = token['refresh']

response, code = create\_response(

status.HTTP\_200\_OK, 'Login Success', {'token': token, 'user': {'name': serializers.validated\_data.username,

'id': serializers.validated\_data.id},

'menu': menu.data})

return Response(response, status=code)

Este código es para cerrar sesión de un usuario autenticado. Comienza obteniendo el token de actualización de sesión, luego lo invalida y elimina todas las cookies de la respuesta. También limpia la sesión y finalmente responde con un mensaje de éxito.

class LogoutView(APIView):

def get(self, request, \*args, \*\*kwargs):

try:

jwt\_token = request.session.get('refresh-token', None)

resp = HttpResponse('content')

resp.cookies.clear()

resp.flush()

token = RefreshToken(jwt\_token)

token.blacklist()

logout(request)

request.session.clear()

resp.flush()

request.session.flush()

response, code = create\_response(

status.HTTP\_200\_OK, 'Logout Success', 'Ok')

return Response(response, code)

except TokenError as TkError:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', f'{TkError}')

return Response(response, code)

except Exception as e:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', e)

return Response(e.args, code)

Este código es para registrar a un usuario. Verifica si los datos proporcionados son válidos y, si es así, crea una contraseña cifrada y guarda al usuario. Responde con un mensaje de éxito o un mensaje de error en caso contrario.

class RegistroView(APIView):

def post(self, request):

serializer = RegistroSerializzer(data=request.data)

if serializer.is\_valid():

user = serializer.save()

return Response({'message': 'Registro exitoso'}, status=201)

return Response(serializer.errors, status=400)

Este código es para ver y actualizar el perfil de un usuario autenticado. Se asegura de que el usuario esté autenticado y devuelve los datos del usuario si es así.

class ProfileView(generics.RetrieveUpdateAPIView):

serializer\_class = UserSerializer

http\_method\_names = ['get', 'patch']

def get\_object(self):

if self.request.user.is\_authenticated:

return self.request.user

Este código es un decorador que se activa cuando se crea un token de restablecimiento de contraseña. Imprime un mensaje en la consola que indica que se puede usar el token para restablecer la contraseña del usuario.

@receiver(reset\_password\_token\_created)

def password\_reset\_token\_created(sender, instance, reset\_password\_token, \*args, \*\*kwargs):

print(f"\nRecupera la contraseña del correo '{reset\_password\_token.user.email}' usando el token '{reset\_password\_token.key}' desde la API http://localhost:8000/api/auth/reset/confirm/.")

Este archivo se llama urls.py y es utilizado para definir las URLs de la aplicación web. Primero se importan las funciones y clases necesarias, como AuthLogin, LogoutView, AuthRegister, ProfileView, UsersViewPublic.

UserCreateView, UserUpdateView y UserChangePasswordView. Luego, se definen las rutas de la aplicación web utilizando la función path.

* 'auth/login/' - Esta ruta llama a la función AuthLogin.as\_view() y se utiliza para iniciar sesión en la aplicación.
* 'auth/logout/' - Esta ruta llama a la función LogoutView.as\_view() y se utiliza para cerrar sesión en la aplicación.
* 'auth/register/' - Esta ruta llama a la función AuthRegister.as\_view() y se utiliza para registrarse en la aplicación.
* 'auth/reset/' - Esta ruta incluye las URLs del paquete django\_rest\_passwordreset, que se utiliza para permitir a los usuarios restablecer sus contraseñas.
* 'user/profile/' - Esta ruta llama a la función ProfileView.as\_view() y se utiliza para ver y actualizar el perfil del usuario autenticado.
* 'user/viewpublic/' - Esta ruta llama a la función UsersViewPublic.as\_view() y se utiliza para ver la lista de usuarios públicos en la aplicación.
* 'user/createview/' - Esta ruta llama a la función UserCreateView.as\_view() y se utiliza para crear nuevos usuarios en la aplicación.
* 'user/update/int:pk/' - Esta ruta llama a la función UserUpdateView.as\_view() y se utiliza para actualizar los detalles de un usuario existente en la aplicación.
* 'int:pk/change/password/' - Esta ruta llama a la función UserChangePasswordView.as\_view() y se utiliza para permitir a un usuario cambiar su contraseña.

from django.urls import path, include

from .views import (AuthLogin, LogoutView, AuthRegister, ProfileView, RegistroView,

UsersViewPublic, UserCreateView, UserUpdateView, UserDetailView,

CustomUserListAPIView, UserChangePasswordView)

urlpatterns = [

# Auth views

path('registro/', RegistroView.as\_view(), name='registro'),

path('auth/login/', AuthLogin.as\_view(), name='auth\_login'),

path('auth/logout/', LogoutView.as\_view(), name='auth\_logout'),

path('auth/reset/', include('django\_rest\_passwordreset.urls',

namespace='password\_reset')),

# User

path('user/', CustomUserListAPIView.as\_view(), name='customuser-list'),

path('user/<int:pk>/', UserDetailView.as\_view(), name='user-detail'),

path('user/createview/', UserCreateView.as\_view(), name='user\_createview'),

path('user/update/<int:pk>/', UserUpdateView.as\_view(), name='user\_createview'),

path('user/profile/', ProfileView.as\_view(), name='user\_profile'),

path('user/viewpublic/', UsersViewPublic.as\_view(), name='user\_viewpublic'),

path('<int:pk>/change/password/', UserChangePasswordView.as\_view(), name='user\_changepassword'),

]

**Creamos una carpeta api**

Aquí se presentarán las serializers, vistas y permisos necesarios para el sistema de autenticación, login y otros modelos en esta aplicación.

Serializers: Se deben definir serializers para cada modelo que se desee exponer a través de la API REST. Las serializers son responsables de convertir los objetos de los modelos en formato JSON o viceversa.

Aquí se presentarán las serializers, vistas y permisos necesarios para el sistema de autenticación, login y otros modelos en esta aplicación.

**Vistas:** Se deben definir vistas para cada modelo que se desee exponer a través de la API **REST.** Las vistas se encargan de manejar las solicitudes HTTP y proporcionar una respuesta adecuada

**Permisos:** Se deben definir permisos para cada vista que se desee proteger. Los permisos controlan quién tiene acceso a cada vista y qué operaciones están permitidas.

En la carpeta de permisos creamos un archivo isadmin.py donde el código define un permiso personalizado llamado IsAdminRole, que hereda de la clase BasePermission proporcionada por Django REST Framework. Este permiso permite el acceso solo a usuarios que tengan un rol de "Administrador" en la aplicación.

La función has\_permission es el método principal del permiso que se utiliza para determinar si un usuario tiene permiso para acceder a una vista en particular. Recibe dos argumentos: request y view. El argumento request es el objeto de solicitud HTTP enviado por el cliente y view es la vista de Django que está siendo solicitada.

La lógica del permiso se implementa en la función has\_permission. En este caso, el permiso se otorga a los usuarios que cumplan cualquiera de las dos condiciones siguientes:

* El usuario es un miembro del personal de la aplicación (request.user.is\_staff es True).
* El usuario tiene al menos un rol que se llama "Admin" (request.user.roles.all().filter(name='Admin') devuelve un queryset que no está vacío).

Si el usuario no cumple ninguna de estas condiciones, el permiso se niega (False se devuelve). Esto significa que el usuario no tendrá acceso a la vista protegida por este permiso.

from rest\_framework.permissions import BasePermission

class IsAdminRole(BasePermission):

def has\_permission(self, request, view):

if request.user and request.user.is\_staff:

return True

if request.user.roles.all().filter(name='Admin'):

return True

return False

En la carpeta de serializer vamos agregar tres archivos nuevos uno para el register y login, otro para las validaciones, y el último para las demás serializers.

Este código es un archivo de serializers de Django REST Framework que contiene dos clases: RegisterSerializers y LoginSerializers.

La clase RegisterSerializers es un serializer que se utiliza para crear un nuevo usuario en la aplicación. Tiene campos para username, email y password. El campo username está configurado para usar un SlugField y se valida con un UniqueValidator para garantizar que el valor no se repita en la base de datos. La clase CustomUser se especifica como modelo y se utilizan todos los campos (fields = '\_\_all\_\_'). Además, se ha definido un validador personalizado en la variable validators para asegurarse de que los datos del usuario sean válidos antes de crear el usuario.

La clase LoginSerializers es un serializer que se utiliza para autenticar a un usuario en la aplicación. Tiene campos para username y password. Además, se ha definido un campo adicional “roles” que utiliza otro serializer llamado RolesSimpleSerializers y que es opcional (required=False). La función validate se utiliza para validar las credenciales del usuario ingresadas en los campos username y password. Si son válidas, se devuelve el objeto User autenticado. Si no son válidas, se lanza una excepción de ValidationError con un mensaje de error personalizado.

from rest\_framework import serializers

from django.contrib.auth import get\_user\_model

from django.contrib.auth import authenticate

from rest\_framework.validators import UniqueValidator

from .customValidators import UserValidatorBefore

from apps.authenticacion.models import CustomUser

from .serializers import RolesSimpleSerializers, PersonsSimpleSerializers

User = get\_user\_model()

class RegisterSerializers(serializers.ModelSerializer):

username = serializers.SlugField(

max\_length=100,

validators=[UniqueValidator(queryset=User.objects.all())]

)

email = serializers.EmailField()

password = serializers.CharField()

class Meta:

model = CustomUser

fields = '\_\_all\_\_'

validators = [UserValidatorBefore()]

person = PersonsSimpleSerializers(read\_only=True)

def create(self, validated\_data):

#person = validated\_data.pop('person')

user = CustomUser.objects.create(\*\*validated\_data)

#Persons.objects.create(\*\*person, user=user)

return user

class LoginSerializers(serializers.ModelSerializer):

username = serializers.CharField(label='Email/username')

password = serializers.CharField(

min\_length=8, error\_messages={

'min\_length': 'La contraseña debe tener al menos 8 caracteres.'})

roles = RolesSimpleSerializers(many=True, required=False)

class Meta:

model = CustomUser

fields = ('id', 'password', 'username', 'roles')

def validate(self, attrs):

user = authenticate(\*\*attrs)

if user and user.is\_active:

return user

raise serializers.ValidationError(

{'detail': 'Las credenciales ingresadas son incorrectas.'})

Este código es un archivo de validadores personalizados para los serializers de Django REST Framework.

La clase UserValidatorBefore es un validador personalizado que se utiliza para verificar que los datos del usuario ingresados sean válidos antes de crear un nuevo usuario. En su inicialización, se definen los nombres de los campos username, email y password. La función \_\_call\_\_ se utiliza para validar los datos del usuario ingresados en el serializer. Primero, verifica que la contraseña tenga al menos 8 caracteres y sea alfanumérica. Si no, se agrega un mensaje de error a message. Luego, verifica si el correo electrónico ya existe en la base de datos. Si es así, se agrega otro mensaje de error a message. Si hay mensajes de error en message, se lanza una excepción de ValidationError. Si no, se devuelve attrs como están.

La clase ChangeValidator es un validador personalizado que se utiliza para verificar que la nueva contraseña del usuario tenga al menos 8 caracteres y sea alfanumérica. La función \_\_call\_\_ se utiliza para validar la contraseña ingresada en el serializer. Si la contraseña no cumple con los requisitos mencionados anteriormente, se agrega un mensaje de error a message. Si hay mensajes de error en message, se lanza una excepción de ValidationError. Si no, se devuelve attrs como están.

from django.contrib.auth import get\_user\_model

from rest\_framework import serializers

from rest\_framework.utils.representation import smart\_repr

from ...mudules import create\_response

User = get\_user\_model()

class UserValidatorBefore:

def \_\_init\_\_(self, username='username', email='email', password='password'):

self.username = username

self.email = email

self.password = password

def \_\_call\_\_(self, attrs):

message = {}

if 'password' in attrs and len(attrs[self.password]) < 8 and attrs[self.password].isalnum():

message[self.password] = 'The password must be alphanumeric and more than 8 digits'

if 'email' in attrs and User.objects.filter(email=attrs[self.email]).exists():

message[self.email] = 'Email already exist'

if message:

raise serializers.ValidationError(message, code='before')

return attrs

def \_\_repr\_\_(self):

return '<%s(email=%s, password=%s, username=%s)>' % (

self.\_\_class\_\_.\_\_name\_\_,

smart\_repr(self.email),

smart\_repr(self.password),

smart\_repr(self.username)

)

class ChangeValidator:

def \_\_init\_\_(self, password='password'):

self.password = password

def \_\_call\_\_(self, attrs):

message = {}

if len(attrs[self.password]) < 8 and attrs[self.password].isalnum():

message[self.password] = 'The password must be alphanumeric and more than 8 digits'

if message:

raise serializers.ValidationError(message, code='before')

return attrs

def \_\_repr\_\_(self):

return '<%s(password=%s)>' % (

self.\_\_class\_\_.\_\_name\_\_,

smart\_repr(self.password),

)

El archivo comienza con importaciones de varios modelos de Django, y algunos serializadores simples como UserSerializersSimple, que convierte objetos CustomUser en JSON que contiene solo los campos username y email. También hay algunos serializadores más complejos para modelos como Persons, Roles y Resources.

En particular, ResourcesRolesSerializers es un serializador que se utiliza para crear recursos y asignarlos a roles en la base de datos. Este serializador toma una lista de recursos como entrada y los crea en la base de datos. Luego crea una instancia de Resources\_roles para cada recurso y rol, y las guarda en la base de datos.

RolesUserSerializers se utiliza para asignar roles a un usuario en la base de datos. Toma un objeto de usuario y una lista de roles como entrada, y crea una instancia de User\_roles para cada combinación de usuario y rol. Si se intenta asignar un rol duplicado a un usuario, se lanzará una excepción ValidationError.

En general, este código parece ser una parte importante de una API REST construida con Django y Django REST Framework, que permite a los usuarios autenticados acceder a recursos y roles específicos.

from apps.authenticacion.models import (Document\_types, Genders, Persons, Resources, Roles, CustomUser, \_roles, Resources\_roles, Resources\_roles, Roles)

from rest\_framework.serializers import ModelSerializer, CharField, ValidationError, Serializer, IntegerField

from ...mudules import create\_response, menuResources

class UserSerializersSimple(ModelSerializer):

class Meta:

model = CustomUser

fields = ('username', 'email')

#GENDER

class GenderSerializers(ModelSerializer):

name = CharField()

class Meta:

model = Genders

exclude = ('createdAt', 'updateAt')

#DOCUMENT

class DocumentSerializers(ModelSerializer):

class Meta:

model = Document\_types

fields = '\_\_all\_\_'

#PERSON

class PersonsSerializers(ModelSerializer):

document\_type = DocumentSerializers(read\_only=True)

gender\_type = GenderSerializers(read\_only=True)

user = UserSerializersSimple(read\_only=True)

class Meta:

model = Persons

fields = '\_\_all\_\_'

class PersonsSimpleSerializers(ModelSerializer):

document\_type = DocumentSerializers(read\_only=True)

class Meta:

model = Persons

fields = ('name', 'surname', 'document\_type',

'phone', 'status', 'date\_of\_birth')

#RESOURCES

class ResourcesSerializers(ModelSerializer):

class Meta:

model = Resources

exclude = ('roles',)

class ResourcesRolesSerializers(Serializer):

rolesId = IntegerField()

resources = ResourcesSerializers(many=True)

def create(self, validated\_data):

try:

resources = []

list\_resources\_roles = []

id\_last\_resources = 0

last = Resources.objects.last()

if last:

id\_last\_resources = last.id + 1

menuResources(validated\_data['resources'],

resources, Resources, id\_last\_resources)

resources = Resources.objects.bulk\_create(resources)

roles = Roles.objects.get(pk=validated\_data['rolesId'])

list\_resources\_roles = [Resources\_roles(

rolesId=roles, resourcesId=r) for r in resources]

Resources\_roles.objects.bulk\_create(list\_resources\_roles)

return None

except Exception as e:

raise e

#ROLES

class RolesSerializers(ModelSerializer):

class Meta:

model = Roles

fields = ('id', 'name')

class RolesSimpleSerializers(ModelSerializer):

resources = ResourcesSerializers(many=True)

class Meta:

model = User\_roles

fields = '\_\_all\_\_'

class RolesUserSerializers(ModelSerializer):

class Meta:

model = User\_roles

exclude = ('rolesId',)

def create(self, validated\_data):

user = validated\_data['userId']

rolesForUser = [User\_roles(

userId=user, rolesId=x) for x in validated\_data['roles']]

try:

response = User\_roles.objects.bulk\_create(rolesForUser)

return response[0]

except Exception as e:

response, code = create\_response(

404, '', 'Duplicate Key User - Rol')

raise ValidationError(response, code=code)

A continuación, las vistas de las demás tablas del modelo de autenticación

Este código se aplica a las vistas basadas en clases para un API relacionado con la manipulación de tipos de documentos en el contexto de una revista de ciencia e ingeniería. A continuación, describiré brevemente lo que hace cada parte del código y cómo podrías documentarlo adecuadamente.

Estas son las importaciones necesarias para las clases y módulos que se utilizan en las vistas.Estas importaciones proporcionan las clases base necesarias para crear vistas basadas en API que manejan los tipos de documentos.

Esta clase define una vista que lista los tipos de documentos disponibles. Esta vista permite obtener una lista de todos los tipos de documentos disponibles y devuelve una respuesta con el código de estado y los datos de los tipos de documentos en formato JSON.

Esta clase define una vista para crear nuevos tipos de documentos. Esta vista permite crear un nuevo tipo de documento utilizando los datos proporcionados en la solicitud POST y devuelve una respuesta con el código de estado y los datos del nuevo tipo de documento en formato JSON. Si la validación falla, se devolverán los errores de validación.

Esta clase define una vista para actualizar tipos de documentos existentes. Esta vista permite actualizar un tipo de documento existente utilizando los datos proporcionados en la solicitud PUT y devuelve una respuesta con el código de estado y los datos actualizados del tipo de documento en formato JSON. Si la actualización falla, se devolverán los errores correspondientes.

**from .....mudules import (ListAPIView, CreateAPIView, Response, status, UpdateAPIView,**

**create\_response, IsAdminRole, DestroyAPIView)**

**from apps.authenticacion.models import Document\_types**

**from ....serializer.serializers import DocumentSerializers**

**class DocumentListView(ListAPIView):**

**queryset = Document\_types.objects.all()**

**serializer\_class = DocumentSerializers**

**def get(self, request, \*args, \*\*kwargs):**

**data = self.get\_queryset()**

**serializers = DocumentSerializers(data, many=True)**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Document', serializers.data)**

**return Response(response, status=code)**

**class DocumentCreateView(CreateAPIView):**

**queryset = Document\_types.objects.all()**

**serializer\_class = DocumentSerializers**

**def post(self, request, \*args, \*\*kwargs):**

**documentSerializers = DocumentSerializers(data=request.data)**

**if documentSerializers.is\_valid():**

**documentSerializers.save()**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Document', documentSerializers.data)**

**return Response(response, status=code)**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Error', documentSerializers.errors)**

**return Response(response, status=code)**

**class DocumentUpdateView(UpdateAPIView):**

**queryset = Document\_types.objects.all()**

**serializer\_class = DocumentSerializers**

**def get\_object(self):**

**try:**

**pk = self.kwargs.get('pk')**

**return Document\_types.objects.get(pk=pk)**

**except Document\_types.DoesNotExist:**

**return None**

**def put(self, request, \*args, \*\*kwargs):**

**document = self.get\_object()**

**if document is None:**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Error', documentSerializers.errors)**

**return Response(response, status=code)**

**try:**

**documentSerializers = DocumentSerializers(**

**document, data=request.data)**

**if documentSerializers.is\_valid():**

**documentSerializers.save()**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Document Update', documentSerializers.data)**

**return Response(response, status=code)**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Error', documentSerializers.errors)**

**return Response(response, status=code)**

**except (AttributeError, Exception) as e:**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Not Found', e.args)**

**return Response(response, status=code)**

**class DocumentDestroyView(DestroyAPIView):**

**queryset = Document\_types.objects.all()**

**serializer\_class = DocumentSerializers**

**#permission\_classes = [IsAdminRole]**

**def get\_object(self):**

**try:**

**pk = self.kwargs.get('pk')**

**return Document\_types.objects.get(id=pk)**

**except Document\_types.DoesNotExist:**

**return None**

**def delete(self, request, \*args, \*\*kwargs):**

**document = self.get\_object()**

**if document is None:**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Error', 'Type document Not Exist')**

**return Response(response, status=code)**

**document.delete()**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Error', 'Ok')**

**return Response(response, status=code)**

### Vista de la tabla gender

**GenderListView**

Vista que lista los géneros disponibles. Esta vista permite obtener una lista de todos los géneros disponibles en la base de datos. Retorna una respuesta con el código de estado y los datos de los géneros en formato JSON.

**GenderCreateView**

Vista para crear nuevos géneros. Esta vista permite crear un nuevo género utilizando los datos proporcionados en la solicitud POST. Retorna una respuesta con el código de estado y los datos del nuevo género en formato JSON. Si la validación falla, se devolverán los errores de validación.

**GenderUpdateView**Vista para actualizar géneros existentes. Esta vista permite actualizar un género existente utilizando los datos proporcionados en la solicitud PUT. Retorna una respuesta con el código de estado y los datos actualizados del género en formato JSON. Si la actualización falla, se devolverán los errores correspondientes.

**GendersDestroyView**Vista para eliminar géneros existentes. Esta vista permite eliminar un género existente. Retorna una respuesta con el código de estado indicando si la eliminación fue exitosa.

from .....mudules import (CreateAPIView, ListAPIView, UpdateAPIView, status,

Response, create\_response,IsAdminRole,DestroyAPIView)

from apps.authenticacion.models import Genders

from ....serializer.serializers import GenderSerializers

class GenderListView(ListAPIView):

queryset = Genders.objects.all()

serializer\_class = GenderSerializers

def get(self, request, \*args, \*\*kwargs):

data = self.get\_queryset()

serializers = GenderSerializers(data, many=True)

response, code = create\_response(

status.HTTP\_200\_OK, 'Genders', serializers.data)

return Response(response, status=code)

class GenderCreateView(CreateAPIView):

queryset = Genders.objects.all()

serializer\_class = GenderSerializers

def post(self, request, \*args, \*\*kwargs):

genderSerializers = GenderSerializers(data=request.data)

if genderSerializers.is\_valid():

genderSerializers.save()

response, code = create\_response(

status.HTTP\_200\_OK, 'Genders', genderSerializers.data)

return Response(response, status=code)

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', genderSerializers.errors)

return Response(response, status=code)

class GenderUpdateView(UpdateAPIView):

queryset = Genders.objects.all()

serializer\_class = GenderSerializers

def get\_object(self):

try:

pk = self.kwargs.get('pk')

return Genders.objects.get(pk=pk)

except Genders.DoesNotExist:

return None

def put(self, request, \*args, \*\*kwargs):

gender = self.get\_object()

if gender is None:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Not Found', 'Gender Not Found')

return Response(response, status=code)

try:

genderSerializers = GenderSerializers(gender, data=request.data)

if genderSerializers.is\_valid():

genderSerializers.update(

gender, genderSerializers.validated\_data)

response, code = create\_response(

status.HTTP\_200\_OK, 'Gender Update', genderSerializers.data)

return Response(response, status=code)

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', genderSerializers.errors)

return Response(response, status=code)

except (AttributeError, Exception) as e:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Not Found', e.args)

return Response(response, status=code)

class GendersDestroyView(DestroyAPIView):

queryset = Genders.objects.all()

serializer\_class = GenderSerializers

#permission\_classes = [IsAdminRole]

def get\_object(self):

try:

pk = self.kwargs.get('pk')

return Genders.objects.get(id=pk)

except Genders.DoesNotExist:

return None

def delete(self, request, \*args, \*\*kwargs):

gender = self.get\_object()

if gender is None:

response, code = create\_response(

status.HTTP\_200\_OK, 'Error', 'Gender Not Exist')

return Response(response, status=code)

gender.delete()

response, code = create\_response(

status.HTTP\_200\_OK, 'Error', 'Ok')

return Response(response, status=code)

### Vista de la tabla persona

**PersonView**

Vista que lista las personas registradas. Esta vista permite obtener una lista de todas las personas registradas en la base de datos. Retorna una respuesta con el código de estado y los datos de las personas en formato JSON.

**PersonCreateView**

Vista para crear nuevas personas. Esta vista permite crear una nueva persona utilizando los datos proporcionados en la solicitud POST. Retorna una respuesta con el código de estado y los datos de la nueva persona en formato JSON. Si la validación falla, se devolverán los errores de validación.

PersonUpdateView

Vista para actualizar los datos de la persona registrada. Esta vista permite actualizar los datos de la persona registrada utilizando los datos proporcionados en la solicitud PUT. Retorna una respuesta con el código de estado y los datos actualizados de la persona en formato JSON. Si la actualización falla, se devolverán los errores correspondientes.

**PersonDeleteView**

Vista para ocultar una persona registrada. Esta vista permite ocultar una persona registrada. Si la persona ya está oculta, se devolverá un error. Retorna una respuesta con el código de estado indicando el resultado de la operación.

from ....serializer.serializers import PersonsSerializers

from apps.authenticacion.models import Persons

from rest\_framework import status

from rest\_framework.views import APIView

from .....mudules import ListAPIView, CreateAPIView, UpdateAPIView, Response, create\_response

class PersonView(ListAPIView):

queryset = Persons.objects.all()

serializer\_class = PersonsSerializers

def get(self, request, \*args, \*\*kwargs):

data = self.get\_queryset()

serializers = PersonsSerializers(data, many=True)

response, code = create\_response(

status.HTTP\_200\_OK, 'Person', serializers.data)

return Response(response, status=code)

class PersonCreateView(CreateAPIView):

queryset = Persons.objects.all()

serializer\_class = PersonsSerializers

def post(self, request, \*args, \*\*kwargs):

personSerializers = PersonsSerializers(data=request.data)

if personSerializers.is\_valid():

personSerializers.save()

response, code = create\_response(

status.HTTP\_200\_OK, 'Person', personSerializers.data)

return Response(response, status=code)

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', personSerializers.errors)

return Response(response, status=code)

class PersonUpdateView(UpdateAPIView):

queryset = Persons.objects.all()

serializer\_class = PersonsSerializers

def get\_object(self):

try:

pk = self.request.user.id

return Persons.objects.filter(user\_\_id=pk).first()

except Persons.DoesNotExist:

return None

def put(self, request, \*args, \*\*kwargs):

person = self.get\_object()

if person is None:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', None) # Update this line

return Response(response, status=code)

personSerializers = PersonsSerializers(person, data=request.data) # Move this line up

try:

if personSerializers.is\_valid():

personSerializers.save() # Use save() instead of update()

response, code = create\_response(

status.HTTP\_200\_OK, 'Person Update', personSerializers.data)

return Response(response, status=code)

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Error', personSerializers.errors)

return Response(response, status=code)

except (AttributeError, Exception) as e:

response, code = create\_response(

status.HTTP\_400\_BAD\_REQUEST, 'Not Found', str(e))

return Response(response, status=code)

class PersonDeleteView(APIView):

def delete(self, request, pk, \*args, \*\*kwargs):

try:

person = Persons.objects.get(pk=pk)

except Persons.DoesNotExist:

return Response({"detail": "Person not found"}, status=status.HTTP\_404\_NOT\_FOUND)

# Check if status is True before deleting

if person.status:

person.status = False

person.save()

return Response({"detail": "Person hidden successfully"}, status=status.HTTP\_204\_NO\_CONTENT)

else:

return Response({"detail": "Person is already hidden"}, status=status.HTTP\_400\_BAD\_REQUEST)

### Vista de la tabla recursos y recursos roles

**ResourcesListCreateView**

Vista para listar y crear recursos. Esta vista permite obtener una lista de todos los recursos disponibles y crear nuevos recursos. Retorna una respuesta con el código de estado y los datos de los recursos en formato JSON.

**ResourcesUpdateDeleteView**

Vista para actualizar y eliminar recursos. Esta vista permite actualizar y eliminar recursos existentes utilizando los datos proporcionados en las solicitudes PUT y DELETE. Retorna una respuesta con el código de estado indicando el resultado de la operación.

**ResourcesRolesListView**

Vista para listar y crear asignaciones de recursos a roles. Esta vista permite obtener una lista de todas las asignaciones de recursos a roles disponibles y crear nuevas asignaciones. Retorna una respuesta con el código de estado y los datos de las asignaciones en formato JSON.

**ResourcesRolesDetailView**

Vista para actualizar y eliminar asignaciones de recursos a roles. Esta vista permite actualizar y eliminar asignaciones de recursos a roles existentes utilizando los datos proporcionados en las solicitudes PUT y DELETE. Retorna una respuesta con el código de estado indicando el resultado de la operación.

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework import status

from rest\_framework.permissions import IsAuthenticated

from django.shortcuts import get\_object\_or\_404

from apps.authenticacion.models import Resources, Roles, Resources\_roles

from ....serializer.serializers import ResourcesSerializers, ResourcesRolesSerializers

class ResourcesListCreateView(APIView):

#permission\_classes = [IsAuthenticated]

def get(self, request):

queryset = Resources.objects.all()

serializer = ResourcesSerializers(queryset, many=True)

return Response(serializer.data)

def post(self, request):

serializer = ResourcesSerializers(data=request.data)

if serializer.is\_valid():

serializer.save()

response = {'response': 'Resource created', 'data': serializer.data}

return Response(response, status=status.HTTP\_201\_CREATED)

response = {'response': 'Error', 'errors': serializer.errors}

return Response(response, status=status.HTTP\_400\_BAD\_REQUEST)

class ResourcesUpdateDeleteView(APIView):

permission\_classes = [IsAuthenticated]

def get\_object(self):

try:

pk = self.kwargs.get('pk')

return Resources.objects.get(id=pk)

except Resources.DoesNotExist:

return None

def put(self, request, \*args, \*\*kwargs):

resources = self.get\_object()

if resources is None:

response = {'response': 'Resource Not Found'}

return Response(response, status=status.HTTP\_404\_NOT\_FOUND)

serializer = ResourcesSerializers(resources, data=request.data)

if serializer.is\_valid():

serializer.save()

response = {'response': 'Resource updated', 'data': serializer.data}

return Response(response, status=status.HTTP\_200\_OK)

response = {'response': 'Error', 'errors': serializer.errors}

return Response(response, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, \*args, \*\*kwargs):

resources = self.get\_object()

if resources is None:

response = {'response': 'Resource Not Found'}

return Response(response, status=status.HTTP\_404\_NOT\_FOUND)

resources.delete()

response = {'response': 'Resource deleted'}

return Response(response, status=status.HTTP\_204\_NO\_CONTENT)

########### RECURSOS ROLES ############

class ResourcesRolesListView(APIView):

def get(self, request):

resources\_roles = Resources\_roles.objects.all()

serializer = ResourcesRolesSerializers(resources\_roles, many=True)

return Response(serializer.data)

def post(self, request):

serializer = ResourcesRolesSerializers(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class ResourcesRolesDetailView(APIView):

def get\_object(self, pk):

try:

return Resources\_roles.objects.get(pk=pk)

except Resources\_roles.DoesNotExist:

return None

def put(self, request, pk):

resources\_roles = self.get\_object(pk)

if resources\_roles is None:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

serializer = ResourcesRolesSerializers(resources\_roles, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, pk):

resources\_roles = self.get\_object(pk)

if resources\_roles is None:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

resources\_roles.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

### Vista de la tabla roles y user roles

RolesListView

Vista para listar roles. Esta vista permite obtener una lista de todos los roles disponibles en la base de datos. Retorna una respuesta con el código de estado y los datos de los roles en formato JSON.

**RolescreateView**

Vista para crear nuevos roles. Esta vista permite crear un nuevo rol utilizando los datos proporcionados en la solicitud POST. Retorna una respuesta con el código de estado y los datos del nuevo rol en formato JSON. Si la validación falla, se devolverán los errores de validación.

**RoleUpdateView**

Vista para actualizar roles. Esta vista permite actualizar un rol existente utilizando los datos proporcionados en la solicitud PUT. Retorna una respuesta con el código de estado y los datos actualizados del rol en formato JSON. Si la actualización falla, se devolverán los errores correspondientes.

RolesDestroyView

Vista para eliminar roles. Esta vista permite eliminar un rol existente. Retorna una respuesta con el código de estado indicando el resultado de la operación.

**User\_rolesView**

Vista para obtener roles de usuario. Esta vista permite obtener una lista de todas las asignaciones de roles a usuarios disponibles en la base de datos. Retorna una respuesta con el código de estado y los datos de las asignaciones en formato JSON.

**UserRolesCreateAPIView**

Vista para asignar roles a usuarios. Esta vista permite asignar roles a los usuarios utilizando los datos proporcionados en la solicitud POST. Retorna una respuesta con el código de estado y los datos de la asignación en formato JSON. Si la validación falla, se devolverán los errores de validación.

**User\_rolesDetailView**

Vista para obtener, actualizar y eliminar asignaciones de roles a usuarios. Esta vista permite obtener, actualizar y eliminar asignaciones de roles a usuarios existentes utilizando los datos proporcionados en las solicitudes GET, PUT y DELETE. Retorna una respuesta con el código de estado y los datos de la asignación en formato JSON (en caso de GET y PUT). Si la actualización falla, se devolverán los errores correspondientes.

**from apps.authenticacion.models import Roles, User\_roles**

**from rest\_framework.views import APIView**

**from rest\_framework.generics import CreateAPIView**

**from ....serializer.serializers import RolesSerializers, UserRolesSerializer, RolesUserSerializers**

**from .....mudules import (CreateAPIView, ListAPIView, Response, UpdateAPIView,**

**create\_response, status, DestroyAPIView, IsAdminRole)**

**class RolesListView(ListAPIView):**

**queryset = Roles.objects.all()**

**serializer\_class = RolesSerializers**

**def get(self, request, \*args, \*\*kwargs):**

**data = self.get\_queryset()**

**serializers = RolesSerializers(data, many=True)**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Roles', serializers.data)**

**return Response(response, status=code)**

**class RolescreateView(CreateAPIView):**

**queryset = Roles.objects.all()**

**serializer\_class = RolesSerializers**

**def post(self, request, \*args, \*\*kwargs):**

**roleSerializers = RolesSerializers(data=request.data)**

**if roleSerializers.is\_valid():**

**roleSerializers.save()**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Role', roleSerializers.data)**

**return Response(response, status=code)**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Error', roleSerializers.errors)**

**return Response(response, status=code)**

**class RoleUpdateView(UpdateAPIView):**

**queryset = Roles.objects.all()**

**serializer\_class = RolesSerializers**

**def get\_object(self):**

**try:**

**pk = self.kwargs.get('pk')**

**return Roles.objects.get(id=pk)**

**except Roles.DoesNotExist:**

**return None**

**def put(self, request, \*args, \*\*kwargs):**

**role = self.get\_object()**

**if role is None:**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Error', 'Role Not Exist')**

**return Response(response, status=code)**

**try:**

**roleSerializers = RolesSerializers(role, data=request.data)**

**if roleSerializers.is\_valid():**

**roleSerializers.save()**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Role', roleSerializers.data)**

**return Response(response, status=code)**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Error', roleSerializers.errors)**

**return Response(response, status=code)**

**except (AttributeError, Exception) as e:**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Not Found', e.args)**

**return Response(response, status=code)**

**class RolesDestroyView(DestroyAPIView):**

**queryset = Roles.objects.all()**

**serializer\_class = RolesSerializers**

**#permission\_classes = [IsAdminRole]**

**def get\_object(self):**

**try:**

**pk = self.kwargs.get('pk')**

**return Roles.objects.get(id=pk)**

**except Roles.DoesNotExist:**

**return None**

**def delete(self, request, \*args, \*\*kwargs):**

**role = self.get\_object()**

**if role is None:**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Error', 'Role Not Exist')**

**return Response(response, status=code)**

**if role.name.lower() == 'admin' or role.name.lower() == 'egresado':**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Error', 'No se puede borrar este rol')**

**return Response(response, status=code)**

**role.delete()**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Error', 'Ok')**

**return Response(response, status=code)**

**class User\_rolesView(APIView):**

**def get(self, request):**

**user\_roles = User\_roles.objects.all()**

**serializer = RolesUserSerializers(user\_roles, many=True)**

**return Response(serializer.data)**

**class UserRolesCreateAPIView(CreateAPIView):**

**queryset = User\_roles.objects.all()**

**serializer\_class = UserRolesSerializer**

**def post(self, request, \*args, \*\*kwargs):**

**userroles = UserRolesSerializer(data=request.data)**

**if userroles.is\_valid():**

**userroles.save()**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Role', userroles.data)**

**return Response(response, status=code)**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Error', userroles.errors)**

**return Response(response, status=code)**

**class User\_rolesDetailView(APIView):**

**def get\_object(self, pk):**

**try:**

**return User\_roles.objects.get(pk=pk)**

**except User\_roles.DoesNotExist:**

**raise status.HTTP\_404\_NOT\_FOUND**

**def get(self, request, pk):**

**user\_roles = self.get\_object(pk)**

**serializer = RolesUserSerializers(user\_roles)**

**return Response(serializer.data)**

**def put(self, request, pk):**

**user\_roles = self.get\_object(pk)**

**serializer = RolesUserSerializers(user\_roles, data=request.data)**

**if serializer.is\_valid():**

**serializer.save()**

**return Response(serializer.data)**

**return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)**

**def delete(self, request, pk):**

**user\_roles = self.get\_object(pk)**

**user\_roles.delete()**

**return Response(status=status.HTTP\_204\_NO\_CONTENT)**

### Vista de la tabla Security

SecurityResourcesCreate

Vista para crear recursos de seguridad. Esta vista permite crear nuevos recursos de seguridad utilizando los datos proporcionados en la solicitud POST. Retorna una respuesta con el código de estado indicando el resultado de la operación.

**SecurityRolesUser**

Vista para asignar roles a un usuario. Esta vista permite asignar roles a un usuario utilizando los datos proporcionados en la solicitud POST. Retorna una respuesta con el código de estado indicando el resultado de la operación.

**from rest\_framework.generics import CreateAPIView**

**from ....serializer.serializers import ResourcesRolesSerializers, RolesUserSerializers, RolesSerializers**

**from rest\_framework import status**

**from apps.authenticacion.models import Resources, User\_roles, Roles**

**from rest\_framework.response import Response**

**from .....mudules import create\_response**

**class SecurityResourcesCreate(CreateAPIView):**

**queryset = Resources.objects.all()**

**serializer\_class = ResourcesRolesSerializers**

**def post(self, request, \*args, \*\*kwargs):**

**try:**

**resources = ResourcesRolesSerializers(data=request.data)**

**resources.is\_valid(raise\_exception=True)**

**resources.create(request.data)**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'Resources', 'Resources Create')**

**return Response(response, code)**

**except BaseException as e:**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Error', e.args[0])**

**return Response(response, status=code)**

**class SecurityRolesUser(CreateAPIView):**

**queryset = User\_roles.objects.all()**

**serializer\_class = RolesUserSerializers**

**def post(self, request, \*args, \*\*kwargs):**

**user = request.data['user']**

**rolesId = request.data['roles']**

**roles = Roles.objects.filter(id\_\_in=rolesId)**

**rolesUser = RolesUserSerializers(**

**data={'userId': user})**

**if rolesUser.is\_valid():**

**rolesUser.save(roles=roles)**

**response, code = create\_response(**

**status.HTTP\_200\_OK, 'User-Rol', 'successfully assigned roles')**

**return Response(response, status=code)**

**response, code = create\_response(**

**status.HTTP\_400\_BAD\_REQUEST, 'Error', rolesUser.errors)**

**return Response(response, status=code)**

### Urls de las vistas de los modelos

**from django.urls import path, include**

**urlpatterns = [**

**path('roles/', include('apps.authenticacion.api.view.models\_view.roles.urls')),**

**path('resources/', include('apps.authenticacion.api.view.models\_view.resources.urls')),**

**path('resourcesr/', include('apps.authenticacion.api.view.models\_view.resources.urlss')),**

**path('persons/', include('apps.authenticacion.api.view.models\_view.persons.urls')),**

**path('genders/', include('apps.authenticacion.api.view.models\_view.gender.urls')),**

**path('documents/', include('apps.authenticacion.api.view.models\_view.documents.urls')),**

**path('security/',include('apps.authenticacion.api.view.models\_view.security.urls'))**

**]**

### Middlewares

El middleware que se define es un mecanismo de autorización que verifica si un usuario tiene permiso para acceder a ciertas rutas en función de su rol y los recursos asociados a ese rol. Aquí hay un resumen de cómo funciona tu middleware:

* Cuando una solicitud HTTP llega a tu aplicación, el middleware ServiceMiddleware se ejecuta antes de llegar a la vista correspondiente.
* El middleware verifica si el usuario está autenticado y no es anónimo (request.user contiene la instancia del usuario logueado). Si no está autenticado o es un usuario anónimo, la solicitud continúa normalmente sin ninguna restricción.
* Si el usuario está autenticado, el middleware obtiene el ID del usuario logueado (idUsuario) a partir del modelo CustomUser.
* Luego, se obtienen los roles del usuario (rolUsuario) utilizando el modelo User\_roles y filtrando por el ID del usuario.
* Se realiza una consulta para obtener los recursos a los que el usuario tiene acceso. Se busca en el modelo Resources aquellos recursos cuyos roles coincidan con los roles del usuario (rolUsuario).
* Luego, el middleware itera a través de los recursos obtenidos y verifica si la URL actual de la solicitud (url) coincide con alguno de los recursos obtenidos (path en el recurso).
* Si la URL coincide con alguno de los recursos permitidos, el interruptor sw se establece en 1, lo que indica que el usuario tiene acceso a la ruta.
* Si el interruptor sw sigue siendo 0 después de recorrer todos los recursos permitidos, se devuelve una respuesta HTTP 403 Forbidden, indicando que el usuario no tiene permiso para acceder a esa ruta.
* Si el interruptor sw es 1 o si el usuario no está autenticado, la solicitud continúa normalmente.
* Finalmente, después de que se haya realizado toda la verificación y procesamiento, la solicitud pasa a la vista correspondiente.

Este middleware agrega una capa de seguridad a tus vistas al restringir el acceso a ciertas rutas según los roles y recursos asociados. Esto ayuda a garantizar que los usuarios solo tengan acceso a las partes de la aplicación para las que tienen permiso.

**from django.http import HttpResponseForbidden**

**from apps.authenticacion.models import CustomUser, Roles, Resources, User\_roles, Resources\_roles**

**class ServiceMiddleware:**

**def \_\_init\_\_(self, get\_response):**

**self.get\_response = get\_response**

**def \_\_call\_\_(self, request):**

**url = request.path\_info**

**if request.user and not request.user.is\_anonymous:**

**idUsuario = CustomUser.objects.get(username = request.user)**

**print(idUsuario.id)**

**rolUsuario = User\_roles.objects.filter(userId = idUsuario.id).values('rolesId')**

**print(rolUsuario)**

**role\_recursos = Resources.objects.all().prefetch\_related('roles').values('path','titulo','roles').filter(roles\_\_in= (rolUsuario))**

**sw = 0**

**for rolerec in role\_recursos:**

**if url in rolerec['path']:**

**sw = 1 # Cambiar el interruptor a 1 si se encuentra el título en la URL**

**break**

**if sw == 0:**

**return HttpResponseForbidden("Acceso denegado. El usuario no tiene acceso a esta ruta.")**

**print(url)**

**response = self.get\_response(request)**

**return response**

**def get\_user\_resources(self, user\_roles):**

**user\_resources = []**

**for role in user\_roles:**

**resources = Resources.objects.filter(roles\_\_pk=role).values\_list('path', flat=True)**

**user\_resources.extend(resources)**

**return user\_resources**

### App solicitudes

En nuestra carpeta llamada apps que es donde vamos a crear las aplicaciones que vamos a utilizar en nuestro proyecto, nos ubicamos en la carpeta apps y con el comando:

* django-admin startapp solicitudes

En la cual vamos a llenar primero los modelos sobre los cuales vamos a necesitar.

**ContenidoSolicitud:** Esta clase representa el contenido asociado a una solicitud de artículo. Contiene detalles como resumen, palabras clave, introducción, métodos, resultados, agradecimientos, literatura citada y archivos adjuntos. Se utiliza para almacenar la información esencial relacionada con la solicitud.

**Solicitud**: Esta clase modela una solicitud de artículo. Contiene información sobre el título del artículo, autores, contenido de la solicitud, fechas relevantes, URL, ORCID, afiliación y estado. Cada solicitud se asocia con un único contenido y puede seguir diferentes estados de seguimiento.

**PasosSeguimiento**: Define los diferentes pasos de seguimiento en el proceso de revisión de una solicitud. Cada paso tiene un nombre, nivel y número de días de programación. Esta clase ayuda a organizar y gestionar el flujo de trabajo durante el seguimiento de una solicitud.

**EstadoSeguimiento**: Representa los posibles estados en los que puede encontrarse una solicitud en el proceso de seguimiento. Cada estado tiene un nombre y una descripción asociada. Esta clase ayuda a identificar el estado actual de una solicitud en su ciclo de vida.

**Seguimiento**: Esta clase captura el seguimiento de una solicitud específica a lo largo de su ciclo de vida. Almacena información sobre la solicitud, fechas de asignación y evaluación, pasos de seguimiento, estado actual, responsables y correcciones realizadas. Ayuda a rastrear y gestionar la evolución de una solicitud.

**Anexos**: Modelo simple que representa los anexos asociados a una solicitud. Puede contener detalles sobre los archivos adjuntos relacionados con la solicitud. Ayuda a mantener una relación entre las solicitudes y los archivos adjuntos.

**RolAutor**: Define los diferentes roles de autor que pueden tener los usuarios en relación con un artículo. Cada rol tiene un nombre y un estado para determinar su activación o desactivación.

**AutorXSolicitud**: Esta clase asocia autores individuales con solicitudes específicas. Almacena información sobre el usuario, el rol de autor y la solicitud a la que pertenece. Ayuda a identificar la contribución de cada autor a una solicitud.

**NivelFormacion**: Define los diferentes niveles de formación que pueden tener los usuarios. Cada nivel tiene un nombre y un estado para determinar su activación o desactivación. Puede utilizarse para indicar la educación formal de un autor.

**UsuarioXFormacion**: Asocia usuarios con sus niveles de formación y detalles relacionados. Almacena información sobre la formación académica de un usuario, incluyendo nombre de institución, fecha de grado, resolución de convalidación, certificado de grado y más.

from django.db import models

from django.contrib.auth import get\_user\_model

from apps.authenticacion.models import CustomUser

CustomUser = get\_user\_model()

class ContenidoSolicitud(models.Model):

resumen = models.CharField(max\_length=1500)

palabras\_claves = models.CharField(max\_length=256)

abstract = models.CharField(max\_length=256)

Keywords = models.CharField(max\_length=256)

introduccion = models.CharField(max\_length=2000)

materi\_metodos = models.CharField(max\_length=2000)

result\_discu = models.CharField(max\_length=2000)

agradecimientos = models.CharField(max\_length=1500)

literact\_citada = models.CharField(max\_length=2000)

archivo\_adjunto = models.FileField(upload\_to='archivos\_adjuntos/')

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return self.resumen

class Solicitud(models.Model):

titulo\_articulo = models.CharField(max\_length=200)

autores = models.CharField(max\_length=500)

contenidoSolicitud = models.OneToOneField(ContenidoSolicitud, on\_delete=models.CASCADE, null=True)

fecha = models.DateField()

urls = models.CharField(max\_length=256)

orcid = models.CharField(max\_length=256)

afiliacion = models.CharField(max\_length=256)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return self.urls

class Meta:

verbose\_name = 'Solicitud'

class PasosSeguimiento(models.Model):

nivel = models.CharField(max\_length=256)

nombre = models.CharField(max\_length=256)

dias\_programacion = models.CharField(max\_length=256)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return self.nombre

class Meta:

verbose\_name = 'PasosSeguimiento'

class EstadoSeguimiento(models.Model):

nombre = models.CharField(max\_length=256)

descripcion = models.CharField(max\_length=2000)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return str(self.nombre)

class Seguimiento(models.Model):

solicitudId = models.ForeignKey(Solicitud,on\_delete=models.CASCADE)

fecha\_asignacion = models.DateField()

fecha\_programacion = models.DateField()

fecha\_evaluacion = models.DateField()

pasos\_seguimiento = models.ForeignKey(PasosSeguimiento,on\_delete=models.CASCADE, related\_name='pasos\_seguimiento')

estado\_seguimiento = models.ForeignKey(EstadoSeguimiento,on\_delete=models.CASCADE, related\_name='estado\_seguimiento')

responsableId = models.ForeignKey(CustomUser,on\_delete=models.CASCADE, related\_name='responsable')

correciones = models.FileField(upload\_to='archivos\_adjuntos/')

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return str(self.solicitudId)

class Anexos(models.Model):

solicitudId = models.ForeignKey(Solicitud,on\_delete=models.CASCADE)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return str(self.solicitudId)

class RolAutor(models.Model):

nombre = models.CharField(max\_length=200)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return str(self.nombre)

class AutorXSolicitud(models.Model):

usuario = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)

rolautor = models.ForeignKey(RolAutor, on\_delete=models.CASCADE)

solicitud = models.ForeignKey(Solicitud, on\_delete=models.CASCADE)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return f"{self.usuario.username} - {self.rolautor.nombre} - {self.solicitud.urls}"

class NivelFormacion(models.Model):

nombre = models.CharField(max\_length=256)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return self.nombre

class UsuarioXFormacion(models.Model):

nombre = models.CharField(max\_length=256)

fecha\_grado = models.CharField(max\_length=256)

resol\_conv = models.CharField(max\_length=256)

cert\_grado = models.CharField(max\_length=256)

nombre\_institucion = models.CharField(max\_length=256)

cert\_resol = models.CharField(max\_length=256)

nivel\_formacion = models.ForeignKey(NivelFormacion,on\_delete=models.CASCADE)

autor = models.ForeignKey(CustomUser,on\_delete=models.CASCADE)

status = models.BooleanField(default=True)

def \_\_str\_\_(self):

return self.nombre

### Registro de los modelos en el panel de administración de django

**from django.contrib import admin**

**from .models import \***

**admin.site.register(UsuarioXFormacion)**

**admin.site.register(RolAutor)**

**admin.site.register(AutorXSolicitud)**

**admin.site.register(Solicitud)**

**admin.site.register(Seguimiento)**

**admin.site.register(Anexos)**

**admin.site.register(PasosSeguimiento)**

**admin.site.register(NivelFormacion)**

**admin.site.register(ContenidoSolicitud)**

**admin.site.register(EstadoSeguimiento)**

### Serializer de los modelos de solicitudes

Esta clase proporciona campos comunes para los serializadores en tu proyecto. Los campos incluyen información sobre el ID del objeto, las fechas de creación y actualización, así como el usuario que creó y actualizó el objeto.

Campos:

* id: Un campo de clave principal de solo lectura. Representa el ID del objeto serializado.
* createdAt: Un campo de fecha de solo lectura. Representa la fecha de creación del objeto.
* updateAt: Un campo de fecha de solo lectura. Representa la fecha de última actualización del objeto.
* userCreate: Un campo de solo lectura que muestra el nombre de usuario del usuario que creó el objeto.
* userUpdate: Un campo de sólo lectura que muestra el nombre de usuario del usuario que realizó la última actualización en el objeto.

Método \_\_init\_\_: El método \_\_init\_\_ se ha personalizado para manejar un argumento adicional llamado meta. Si meta se establece en True, los campos de fechas y usuarios se mantendrán en el serializador. De lo contrario, estos campos se eliminarán del serializador.

Uso: Puedes heredar de esta clase en tus serializadores específicos y personalizar aún más su comportamiento si es necesario. Algunos serializadores pueden requerir la información de fechas y usuarios, mientras que otros pueden necesitar una representación más compacta sin estos detalles.

class BaseSerializers(serializers.Serializer):

id = serializers.PrimaryKeyRelatedField(read\_only=True)

createdAt = serializers.DateField(read\_only=True)

updateAt = serializers.DateField(read\_only=True)

userCreate = serializers.SlugRelatedField("username",read\_only=True)

userUpdate = serializers.SlugRelatedField("username",read\_only=True)

def \_\_init\_\_(self, instance=None, data=..., \*\*kwargs):

meta = bool(kwargs.pop('meta', None))

super().\_\_init\_\_(instance, data, \*\*kwargs)

if meta != True or meta is None:

self.fields.pop("createdAt")

self.fields.pop("updateAt")

self.fields.pop("userCreate")

self.fields.pop("userUpdate")

Estos serializadores a continuación están diseñados para convertir los objetos de Django en representaciones de Python que se pueden convertir fácilmente en JSON o XML, y viceversa.

**NivelFormacionSerializer:**

* Serializador para el modelo NivelFormacion.
* Todos los campos del modelo se incluyen en la representación.

**UsuarioXFormacionSerializer:**

* Serializador para el modelo UsuarioXFormacion.
* Incluye detalles del nivel de formación y el autor asociado.
* Se utiliza el serializador NivelFormacionSerializer para el campo nivel\_formacion.
* Se utiliza el serializador CustomUserSerializer para el campo autor.

**AutorSerial:**

* Serializador para el modelo CustomUser, que es el modelo de usuario personalizado.
* Incluye la relación inversa usuarioxformacion a través del serializador UsuarioXFormacionSerializer.
* El campo usuario se representa como una clave principal relacionada y se establece automáticamente en el usuario actual a través de serializers.CurrentUserDefault().

**RolAutorSerializer:**

* Serializador para el modelo RolAutor.
* Incluye todos los campos del modelo.

**AutorXSolicitudSerializer:**

* Serializador para el modelo AutorXSolicitud.
* Incluye solo los campos id, usuario, rolautor y solicitud.

from rest\_framework import serializers

from ....models import \*

from .....authenticacion.models import CustomUser

from ..solicitud.solicitud\_serializers import \*

from .....authenticacion.serializers import CustomUserSerializer

class NivelFormacionSerializer(serializers.ModelSerializer):

class Meta:

model = NivelFormacion

fields = '\_\_all\_\_'

class UsuarioXFormacionSerializer(serializers.ModelSerializer):

nivel\_formacion = NivelFormacionSerializer()

autor = CustomUserSerializer()

class Meta:

model = UsuarioXFormacion

fields = '\_\_all\_\_'

class AutorSerial(serializers.ModelSerializer):

usuarioxformacion = UsuarioXFormacionSerializer(many=True)

usuario = serializers.PrimaryKeyRelatedField(

queryset=get\_user\_model().objects.all(),

default=serializers.CurrentUserDefault()

)

class Meta:

model = CustomUser

fields = '\_\_all\_\_'

class RolAutorSerializer(serializers.ModelSerializer):

class Meta:

model = RolAutor

fields = '\_\_all\_\_'

class AutorXSolicitudSerializer(serializers.ModelSerializer):

class Meta:

model = AutorXSolicitud

fields = ['id', 'usuario', 'rolautor', 'solicitud']

**ContenidoSolicitudSerializer para Archivos Adjuntos**

Este serializador está diseñado para manejar la serialización y deserialización de objetos de tipo ContenidoSolicitud, que incluyen campos relacionados con archivos adjuntos. La adición de la línea archivo\_adjunto = serializers.FileField() permite gestionar y representar archivos adjuntos en la API.

Este serializador se utiliza para convertir los objetos de tipo ContenidoSolicitud en representaciones JSON (u otros formatos) y viceversa, lo que facilita la comunicación entre la base de datos y la API en una aplicación Django con el uso de Django REST framework.

El serializador se configura para incluir todos los campos del modelo ContenidoSolicitud, y se agrega el campo archivo\_adjunto para manejar archivos adjuntos. Esto permite que el serializador se encargue de validar y deserializar los datos entrantes relacionados con archivos adjuntos y representarlos adecuadamente en la respuesta de la API.

Este serializador se puede utilizar junto con vistas basadas en clases de Django REST framework para construir endpoints de API que manejen la creación, actualización y recuperación de objetos ContenidoSolicitud con archivos adjuntos.

from rest\_framework import serializers

from ....models import \*

class ContenidoSolicitudSerializer(serializers.ModelSerializer):

archivo\_adjunto = serializers.FileField() # Agregar esta línea para manejar archivos adjuntos

class Meta:

model = ContenidoSolicitud

fields = '\_\_all\_\_'

**Serializadores para Modelos Relacionados con Seguimiento**

Estos serializadores están diseñados para manejar la serialización y deserialización de objetos relacionados con el modelo Seguimiento. Cada serializador representa un modelo relacionado y es útil para convertir objetos de Django en representaciones JSON (u otros formatos) y viceversa, lo que facilita la comunicación entre la base de datos y la API en una aplicación Django que utiliza Django REST framework.

**SeguimientoSerializer**:

* Serializador para el modelo Seguimiento.
* Incluye todos los campos del modelo Seguimiento en la representación.
* Los campos relacionados como solicitud y pasos\_seguimiento se pueden incluir descomentando las líneas correspondientes, utilizando PrimaryKeyRelatedField para representar claves principales relacionadas.

**AnexosSerializer:**

* Serializador para el modelo Anexos.
* Incluye todos los campos del modelo Anexos en la representación.

**EstadoSeguimientoSerializer:**

* Serializador para el modelo EstadoSeguimiento.
* Incluye todos los campos del modelo EstadoSeguimiento en la representación.

**from rest\_framework import serializers**

**from ....models import \***

**from .....authenticacion.models import CustomUser**

**from ..solicitud.solicitud\_serializers import \***

**class SeguimientoSerializer(serializers.ModelSerializer):**

**class Meta:**

**model = Seguimiento**

**fields = '\_\_all\_\_'**

**class AnexosSerializer(serializers.ModelSerializer):**

**class Meta:**

**model = Anexos**

**fields = '\_\_all\_\_'**

**class EstadoSeguimientoSerializer(serializers.ModelSerializer):**

**class Meta:**

**model = EstadoSeguimiento**

**fields = '\_\_all\_\_'**

**Serializadores para Solicitud y Pasos de Seguimiento.**

Estos serializadores están diseñados para manejar la serialización y deserialización de objetos relacionados con los modelos Solicitud y PasosSeguimiento. Cada serializador representa un modelo relacionado y es útil para convertir objetos de Django en representaciones JSON (u otros formatos) y viceversa, lo que facilita la comunicación entre la base de datos y la API en una aplicación Django que utiliza Django REST framework.

**SolicitudSerializer:**

* Serializador para el modelo Solicitud.
* Incluye todos los campos del modelo Solicitud en la representación.
* Agrega campos fecha\_asignacion, fecha\_programacion y fecha\_evaluacion como campos de formulario para permitir el ingreso de fechas en formato "YYYY-MM-DD" en la API.

**PasosSeguimientoSerializer:**

* Serializador para el modelo PasosSeguimiento.
* Incluye todos los campos del modelo PasosSeguimiento en la representación.

### Vista de los modelos de app solicitudes

**Vistas basadas en clases para el modelo ContenidoSolicitud.**

Estas vistas basadas en clases están diseñadas para interactuar con el modelo ContenidoSolicitud en una aplicación Django utilizando Django REST framework. Proporcionan endpoints de API que permiten la creación, recuperación, actualización y eliminación de objetos de tipo ContenidoSolicitud, así como la descarga de archivos adjuntos.

**ContenidoSolicitudList:**

* Vista basada en clases que proporciona un endpoint para listar y crear objetos ContenidoSolicitud.
* Utiliza el serializador ContenidoSolicitudSerializer para manejar la serialización y deserialización.
* La lista de objetos ContenidoSolicitud se filtra para mostrar solo aquellos con el atributo status en True.
* Permite la creación de nuevos objetos ContenidoSolicitud.

**ContenidoSolicitudDetail:**

* Vista basada en clases que proporciona un endpoint para recuperar, actualizar y eliminar objetos ContenidoSolicitud.
* Utiliza el serializador ContenidoSolicitudSerializer para manejar la serialización y deserialización.
* La lista de objetos ContenidoSolicitud se filtra para mostrar solo aquellos con el atributo status en True.
* La función perform\_destroy cambia el atributo status del objeto a False en lugar de eliminarlo físicamente.

**descargar\_archivo:**

* Función de vista que permite descargar el archivo adjunto de un objeto ContenidoSolicitud específico.
* Utiliza el ID de un objeto ContenidoSolicitud para recuperar el archivo adjunto y enviarlo como una respuesta de archivo (FileResponse) al cliente.

from django.http import Http404

from rest\_framework import status

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework import generics

from django.shortcuts import get\_object\_or\_404

from django.http import FileResponse

from ....models import

from ...serializers.literatura.literatura\_serilizers import ContenidoSolicitudSerializer

class ContenidoSolicitudList(generics.ListCreateAPIView):

queryset = ContenidoSolicitud.objects.filter(status=True)

serializer\_class = ContenidoSolicitudSerializer

def create(self, request, \*args, \*\*kwargs):

serializer = self.get\_serializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class ContenidoSolicitudDetail(generics.RetrieveUpdateDestroyAPIView):

queryset = ContenidoSolicitud.objects.filter(status=True)

serializer\_class = ContenidoSolicitudSerializer

def perform\_destroy(self, instance):

instance.status = False

instance.save()

def delete(self, request, \*args, \*\*kwargs):

instance = self.get\_object()

self.perform\_destroy(instance)

return Response(status=status.HTTP\_204\_NO\_CONTENT)

def descargar\_archivo(request, pk):

contenido = get\_object\_or\_404(ContenidoSolicitud, pk=pk, status=True)

archivo = contenido.archivo\_adjunto

response = FileResponse(archivo)

return response

Vistas basadas en clases para el modelo Solicitud

Estas vistas basadas en clases están diseñadas para interactuar con el modelo Solicitud en una aplicación Django utilizando Django REST framework. Proporcionan endpoints de API que permiten la creación, recuperación, actualización y eliminación de objetos de tipo Solicitud, y se aseguran de que solo se manejen objetos con el atributo status en True.

**SolicitudList:**

* Vista basada en clases que proporciona un endpoint para listar y crear objetos Solicitud.
* Utiliza el serializador SolicitudSerializer para manejar la serialización y deserialización.
* La lista de objetos Solicitud se filtra para mostrar solo aquellos con el atributo status en True.
* Permite la creación de nuevos objetos Solicitud.

**SolicitudDetail:**

* Vista basada en clases que proporciona un endpoint para recuperar, actualizar y eliminar objetos Solicitud.
* Utiliza el serializador SolicitudSerializer para manejar la serialización y deserialización.
* La función get\_object recupera el objeto Solicitud con el ID proporcionado.
* La recuperación de un objeto Solicitud se verifica primero si su atributo status es True.
* La actualización y eliminación de un objeto Solicitud cambia su atributo status a False.

from django.http import Http404

from rest\_framework import status

from rest\_framework.views import APIView

from rest\_framework.response import Response

from ....models import Solicitud

from ...serializers.solicitud.solicitud\_serializers import SolicitudSerializer

from rest\_framework.response import Response

from rest\_framework.permissions import IsAuthenticated

class SolicitudList(APIView):

def get(self, request, format=None):

solicitudes = Solicitud.objects.filter(status=True) # Filtrar por status=True

serializer = SolicitudSerializer(solicitudes, many=True)

data = {'solicitudes': serializer.data}

return Response(data)

def post(self, request, format=None):

serializer = SolicitudSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class SolicitudDetail(APIView):

def get\_object(self, pk):

try:

return Solicitud.objects.get(pk=pk)

except Solicitud.DoesNotExist:

raise Http404

def get(self, request, pk, format=None):

solicitud = self.get\_object(pk)

# Check if status is True before serializing and returning the data

if solicitud.status:

serializer = SolicitudSerializer(solicitud)

data = {'solicitud': serializer.data}

return Response(data)

else:

# Return a 404 response when status is False

print("No encontrado")

return Response('No encontrado',status=status.HTTP\_404\_NOT\_FOUND)

def put(self, request, pk, format=None):

solicitud = self.get\_object(pk)

serializer = SolicitudSerializer(solicitud, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, pk):

solicitud = self.get\_object(pk)

if solicitud is None:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

solicitud.status = False

solicitud.save()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

### Vistas basadas en clases para el modelo PasosSeguimiento

Estas vistas basadas en clases son parte de la API diseñada para interactuar con el modelo PasosSeguimiento en una aplicación Django utilizando Django REST framework. Proporcionan endpoints de API para listar, crear, recuperar, actualizar y eliminar objetos de tipo PasosSeguimiento. Estas vistas también aseguran que solo se manejen objetos con el atributo status en True.

**PasosSeguimientoList:**

* Vista basada en clases que proporciona un endpoint para listar y crear objetos PasosSeguimiento.
* Utiliza el serializador PasosSeguimientoSerializer para manejar la serialización y deserialización.
* La lista de objetos PasosSeguimiento se filtra para mostrar solo aquellos con el atributo status en True.
* Permite la creación de nuevos objetos PasosSeguimiento.

**PasosSeguimientoDetail:**

* Vista basada en clases que proporciona un endpoint para recuperar, actualizar y eliminar objetos PasosSeguimiento.
* Utiliza el serializador PasosSeguimientoSerializer para manejar la serialización y deserialización.
* La función get\_object recupera el objeto PasosSeguimiento con el ID proporcionado.
* La recuperación de un objeto PasosSeguimiento se verifica primero si su atributo status es True.
* La actualización de un objeto PasosSeguimiento cambia sus datos según la solicitud del cliente.
* La eliminación de un objeto PasosSeguimiento cambia su atributo status a False.

from django.http import Http404

from rest\_framework import status

from rest\_framework import generics

from rest\_framework.views import APIView

from rest\_framework.response import Response

from ....models import PasosSeguimiento

from ...serializers.solicitud.solicitud\_serializers import PasosSeguimientoSerializer

class PasosSeguimientoList(generics.ListCreateAPIView):

queryset = PasosSeguimiento.objects.all()

serializer\_class = PasosSeguimientoSerializer

def list(self, request, \*args, \*\*kwargs):

queryset = PasosSeguimiento.objects.filter(status=True) # Filtrar por status=True

serializer = self.get\_serializer(queryset, many=True)

data = {'pasos\_seguimiento': serializer.data}

return Response(data)

def create(self, request, \*args, \*\*kwargs):

serializer = self.get\_serializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class PasosSeguimientoDetail(generics.RetrieveUpdateDestroyAPIView):

queryset = PasosSeguimiento.objects.all()

serializer\_class = PasosSeguimientoSerializer

def retrieve(self, request, \*args, \*\*kwargs):

instance = self.get\_object()

serializer = self.get\_serializer(instance)

data = {'paso\_seguimiento': serializer.data}

return Response(data)

def update(self, request, \*args, \*\*kwargs):

instance = self.get\_object()

serializer = self.get\_serializer(instance, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, \*args, \*\*kwargs):

instance = self.get\_object()

instance.status = False # Establecer el estado en "oculto"

instance.save()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

### Vistas basadas en clases para el modelo Seguimiento

Estas vistas basadas en clases forman parte de la API creada para interactuar con el modelo Seguimiento en una aplicación Django mediante Django REST framework. Proporcionan endpoints de API para listar, crear, recuperar, actualizar y eliminar objetos de tipo Seguimiento. Además, se incluye una función para descargar archivos adjuntos asociados a un seguimiento.

**SeguimientoList:**

* Esta vista proporciona un endpoint para listar y crear objetos Seguimiento.
* Se filtran los seguimientos con el atributo status en True.
* Los seguimientos se utilizan utilizando SeguimientoSerializer.
* Permite la creación de nuevos seguimientos.

**SeguimientoDetail:**

* Esta vista proporciona un endpoint para recuperar, actualizar y eliminar objetos Seguimiento.
* La función get\_object recupera un objeto Seguimiento por su ID.
* La recuperación de un objeto Seguimiento verifica si su atributo status es True.
* Se utilizan los métodos HTTP GET, PUT y DELETE para realizar las acciones correspondientes.
* Cuando se elimina un seguimiento, su atributo status se establece en False.

**descargar\_archivo:**

* Esta función descarga un archivo adjunto asociado a un seguimiento.
* Utiliza la función get\_object\_or\_404 para recuperar el seguimiento con el ID proporcionado y el atributo status en True.
* Si el archivo existe en el seguimiento, se devuelve como una respuesta de archivo (FileResponse).
* Si no se encuentra un archivo adjunto, se lanza una excepción Http404.

from django.http import Http404

from rest\_framework import status

from rest\_framework.views import APIView

from rest\_framework.viewsets import ModelViewSet

from rest\_framework.response import Response

from django.shortcuts import get\_object\_or\_404

from django.http import FileResponse

from ....models import Seguimiento

from ...serializers.seguimiento.seguimiento\_serializers import SeguimientoSerializer

class SeguimientoList(APIView):

def get(self, request):

seguimientos = Seguimiento.objects.filter(status=True) # Filtrar por status=True

serializer = SeguimientoSerializer(seguimientos, many=True)

data = {'seguimientos': serializer.data}

return Response(data)

def post(self, request):

serializer = SeguimientoSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class SeguimientoDetail(APIView):

def get\_object(self, pk):

try:

return Seguimiento.objects.get(pk=pk)

except Seguimiento.DoesNotExist:

raise Http404

def get(self, request, pk):

seguimiento = self.get\_object(pk)

if seguimiento.status:

serializer = SeguimientoSerializer(seguimiento)

data = {'seguimiento': serializer.data}

return Response(data)

else:

return Response('No encontrado... Realice otra busquedad.',status=status.HTTP\_404\_NOT\_FOUND)

def put(self, request, pk):

seguimiento = self.get\_object(pk)

serializer = SeguimientoSerializer(seguimiento, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, pk):

seguimiento = self.get\_object(pk)

if seguimiento is None:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

seguimiento.status = False # Establecer el estado en "oculto"

seguimiento.save()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

def descargar\_archivo(request, pk):

seguimiento = get\_object\_or\_404(Seguimiento, pk=pk, status=True)

archivo = seguimiento.correciones

if archivo:

response = FileResponse(archivo)

return response

else:

raise Http404("Archivo no encontrado")

### Vistas basadas en clases para el modelo EstadoSeguimiento

Estas vistas basadas en clases forman parte de la API diseñada para interactuar con el modelo EstadoSeguimiento en una aplicación Django mediante Django REST framework. Proporcionan endpoints de API para listar, crear, recuperar, actualizar y eliminar objetos de tipo EstadoSeguimiento.

**EstadoSeguimientoList:**

* Esta vista proporciona un endpoint para listar y crear objetos EstadoSeguimiento.
* Se filtran los estados de seguimiento con el atributo status en True.
* Los estados de seguimiento se serializan utilizando EstadoSeguimientoSerializer.
* Permite la creación de nuevos estados de seguimiento.

**EstadoSeguimientoDetail:**

* Esta vista proporciona un endpoint para recuperar, actualizar y eliminar objetos EstadoSeguimiento.
* La función get\_object recupera un objeto EstadoSeguimiento por su ID.
* La recuperación de un objeto EstadoSeguimiento verifica si su atributo status es True.
* Se utilizan los métodos HTTP GET, PUT y DELETE para realizar las acciones correspondientes.
* Cuando se elimina un estado de seguimiento, su atributo status se establece en False.

from django.http import Http404

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework import status

from ....models import EstadoSeguimiento

from ...serializers.seguimiento.seguimiento\_serializers import EstadoSeguimientoSerializer

class EstadoSeguimientoList(APIView):

def get(self, request):

estados\_seguimiento = EstadoSeguimiento.objects.filter(status=True) # Filtrar por status=True

serializer = EstadoSeguimientoSerializer(estados\_seguimiento, many=True)

return Response(serializer.data)

def post(self, request):

serializer = EstadoSeguimientoSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class EstadoSeguimientoDetail(APIView):

def get\_object(self, pk):

try:

return EstadoSeguimiento.objects.get(pk=pk)

except EstadoSeguimiento.DoesNotExist:

raise Http404

def get(self, request, pk):

estado\_seguimiento = self.get\_object(pk)

if estado\_seguimiento.status:

serializer = EstadoSeguimientoSerializer(estado\_seguimiento)

return Response(serializer.data)

else:

return Response('No encontrado... Realice otra busquedad.',status=status.HTTP\_404\_NOT\_FOUND)

def put(self, request, pk):

estado\_seguimiento = self.get\_object(pk)

serializer = EstadoSeguimientoSerializer(estado\_seguimiento, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, pk):

estado\_seguimiento = self.get\_object(pk)

estado\_seguimiento.status = False

estado\_seguimiento.save()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

### urls de de las vistas de apps solicitudes

**from django.urls import path, include**

**urlpatterns = [**

**path('usuarioxf/', include('apps.solicitudes.api.views.autor.urls\_autorXF')),**

**path('nivel/', include('apps.solicitudes.api.views.autor.urls\_nivelF')),**

**path('rol/', include('apps.solicitudes.api.views.autor.urls\_rolautor')),**

**path('autsoli/', include('apps.solicitudes.api.views.autor.urls\_autorsolicitud')),**

**path('contenido/', include('apps.solicitudes.api.views.literatura.urls\_contenido')),**

**path('seguimiento/', include('apps.solicitudes.api.views.seguimiento.urls\_seguimiento')),**

**path('estado/', include('apps.solicitudes.api.views.seguimiento.urls\_estado')),**

**path('anexo/', include('apps.solicitudes.api.views.seguimiento.urls\_anexo')),**

**path('solicitud/', include('apps.solicitudes.api.views.solicitud.urls\_solicitud')),**

**path('pasossolicitud/', include('apps.solicitudes.api.views.solicitud.urls\_pasosS')),**

**]**

## PROCEDIMIENTO DEL FRONTEND.

Stoney (2020), asegura que: “La parte frontal de una aplicación o sitio web es la que permite a los usuarios interactuar con ella, también llamada el lado del usuario. En esencia, abarca todo lo que aparece en la pantalla para presentar la información de una manera más agradable para quien la usa.”

Pérez, Mullicundo, Lamas, (2021), nos dice que: “El desarrollo del frontend implica comprender técnicas y estrategias de diseño que faciliten la interacción del usuario. Esto implica ubicar los elementos de manera eficiente para que los usuarios puedan navegar con facilidad y rapidez. El objetivo es lograr una experiencia ideal entre el usuario y la aplicación o sitio web.”

EPS Alicante (2021), define lo siguiente: El frontend se compone de diversos elementos, como tipos de letra variados, una paleta de colores adecuada y un diseño que se adapte a diferentes tamaños de pantalla. También incluye efectos que responden a la interacción con el teclado y el ratón, elementos visuales atractivos, transiciones suaves y otras características que hacen posible la navegación en la aplicación o sitio web. Todo esto se traduce en una experiencia satisfactoria para el usuario.

En base a estos autores podemos

En este procedimiento, explicaremos los pasos fundamentales para iniciar y configurar nuestro frontend, estableciendo la conexión con el API proporcionado por el backend. Para lograr esto, aprovecharemos los archivos de configuración de entorno que se encuentran en la carpeta "environments" en el directorio "src" de nuestro proyecto. Estos archivos, llamados "environment.ts" y "environment.prod.ts", nos permiten ajustar las variables y configuraciones específicas según el entorno en el que se encuentre nuestra aplicación.

### Configuración del Entorno

Dentro del archivo "environment.ts", encontramos una constante crucial denominada "environment". Esta constante juega un papel esencial al consolidar y mantener la URL principal (API\_URI) de nuestro backend. Esta URL actúa como el punto de entrada para nuestras solicitudes de API y comunicación con el servidor backend. La estructura de esta constante es la siguiente:

export const environment = {

  production: false,

  API\_URI : 'http://localhost:8000'

};

Para el archivo "environment.prod.ts", repetimos el proceso, pero esta vez ajustamos el campo "production" a "true". Esto asegura que, en el entorno de producción, nuestra aplicación se conecte al backend de manera adecuada. La estructura sería:

export const environment = {

  production: true,

  API\_URI : 'http://localhost:8000',

};

**Preparación del HTML Principal**

Antes de sumergirnos en nuestro proyecto, debemos preparar el archivo HTML principal llamado "index.html". En este archivo, estableceremos el entorno para que nuestra aplicación pueda ejecutarse de manera efectiva. A continuación, se muestra el código esencial que debes incluir en este archivo:

<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>Ciencia e Ingenieria</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500&display=swap" rel="stylesheet">

  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

  <link rel="icon" type="image/x-icon" href="assets/logo\_r.jpg">

</head>

<body>

  <app-root></app-root>

</body>

</html>

### Configuración de la Estructura de Carpetas y Modelos

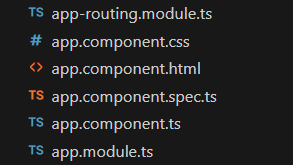
**Organización de Carpetas**

Una vez establecidas las constantes de entorno y el archivo principal "index.html", procedemos a explorar la estructura de carpetas en el directorio "src" y "app" de nuestro proyecto. Esta organización nos ayudará a mantener una estructura clara y modular para nuestro código.

Dentro de la carpeta "src", encontramos el componente básico "app.component" junto con sus archivos asociados (css, html, spec.ts y ts). Además, para crear un sistema de enrutamiento coherente, requerimos dos archivos adicionales: "routing.module.ts" y "module.ts". Estos archivos se encargarán de establecer las rutas y las importaciones necesarias en nuestra aplicación. Para generar el archivo "app.module.ts" en la carpeta "app", ejecutamos el siguiente comando en la terminal: **ng generate module app.** Posteriormente, ingresamos a la carpeta "app" utilizando el comando "cd app". Aquí, creamos un archivo de rutas utilizando el comando: **ng generate module estados-routing --flat --module=app,** Esto dará como resultado el archivo "app-routing.module.ts".

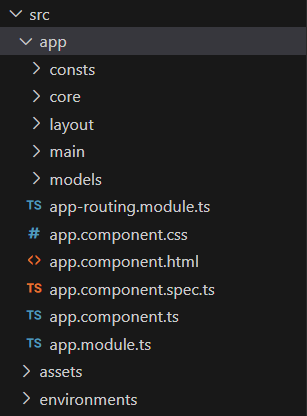
****Estructura de Componentes y Modelos****

La estructura básica de nuestros componentes principales incluye los archivos "routing.module.ts", css, html, spec.ts y module.ts. Esta configuración será similar para cada componente que creemos.

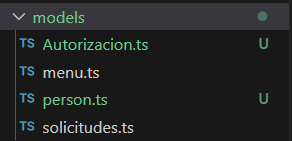


Dentro de la carpeta "app", creamos las siguientes subcarpetas para una organización más eficiente: "consts", "core", "layout", "main" y "models". Esta disposición nos permitirá mantener un orden lógico en nuestro proyecto.

* **"consts":** Aquí guardaremos constantes y configuraciones globales de la aplicación.
* **"core":** Esta carpeta contendrá servicios y lógica central de la aplicación.
* **"layout":** Almacena componentes relacionados con la estructura de la interfaz de usuario, como encabezados y pies de página.
* **"main":** En esta carpeta, ubicamos componentes principales de nuestras páginas.
* **"models":** Aquí definiremos los modelos necesarios para estructurar la información recibida del backend.



Los modelos se crean para mantener una estructura coherente en los datos que interactuamos con el backend. Siguiendo esta lógica, creamos una separación adicional en la carpeta "models":



Dentro de esta carpeta conseguimos los siguientes archivos:

**autorizacion.ts:** En este archivo, definimos una serie de modelos esenciales para el acceso y registro de usuarios. Importamos el modelo "Person" del archivo "person.ts" para su uso. Los modelos aquí incluidos son: Roles, Recursos, Mantenimiento, asignación de recursos por rol, cambio de contraseña, Usuario, roles de usuario, usuario login, respuesta de usuario login, menú y nombre de usuario.

import { Person } from "./person";

Dichos modelos son representados a continuación:

export interface RoleI {

    id: number;

    name: string;

    Users?: UserI[];

    rolesUsers?: any[]

}

export interface ResourceI {

    id?: number;

    path: string;

    method: string;

    id\_padre: string | number;

    icono: string;

    link: string;

    titulo: string;

    Roles?: RoleI[]

}

export interface MaintenanceI {

    id?: number

    name: string

    url: string

    date: string

    status\_maintenance: string

}

export interface assinRoleResourceI

{

    id?: number;

    RecursosRoles: [

        {

            ResourceId: string;

            RoleId: string;

        }

    ]

}

export interface CambiarPasswordI {

    id?: number;

    oldPassword: string;

    newPassword: string;

}

export interface UserI {

    id?: number;

    username: string;

  email: string;

    fullName: string;

    password?: string;

    Roles?: [

        {

            name: string

        }

    ]

    Person?: Person

    avatar?: string

    People?: Person[]

    UserRoles?: UserRoleI[]

    todo?: string

}

export interface UserRoleI {

    UserId: number,

    RoleId: number,

    Role?: RoleI

}

export interface UserLoginI {

    username: string;

    email: string;

    password: string;

}

export interface UserLoginResponseI

{

    ok?: boolean;

    data?: any;

    request\_id?: any;

    user?: UsernameI;

    token: any;

    menu?: any

}

export interface MenuResponseI {

    id: number;

    id\_padre: number;

    icono: string;

    link: string;

    titulo: string;

}

export interface UsernameI {

    username: string;

    fullName?: string;

    id: number;

}

**menu.ts:** Este archivo contiene los modelos para la recolección y ordenamiento de datos relacionados con el menú de la aplicación. Los modelos incluidos son: menú, lista de menú, hijos, submenú y opciones de menú.

Dichos modelos son representados a continuación:

export interface menuGen {

    mainSesion: listaMenuI[]

}

export interface listaMenuI {

    id: number;

    id\_padre: number;

    icono: string;

    link: string;

    titulo: string;

}

export interface hijosI {

id: number;

id\_padre: number;

icono: string;

link: string;

    titulo: string;

    menu?: listaMenuI[]

}

export interface menuI {

    menu: opcMenu[];

}

export interface opcMenu {

    id: number;

    id\_padre: number;

    icono: string;

    link: string;

    titulo: string;

    menu?: [

        {

            id: number;

            id\_padre: number;

            icono: string;

            link: string;

            titulo: string;

            menu?: [

                {

                    id: number;

                    id\_padre: number;

                    icono: string;

                    link: string;

                    titulo: string;

                }

            ]

        }

    ]

}

**person.ts:** hemos establecido una serie de modelos esenciales para recopilar y estructurar los datos relacionados con los usuarios. Cada modelo desempeña un papel crucial en la organización y presentación de la información. A continuación, presentamos los modelos que hemos definido:

* **Tipo de Documento:** Este modelo representa los diferentes tipos de documentos de identificación, como cédulas, pasaportes, etc.
* **Género:** Aquí definimos los géneros disponibles para clasificar a los usuarios.
* **Usuario:** Este modelo describe los atributos principales de un usuario, como su nombre, correo electrónico y contraseña.
* **Nivel de Formación:** Definimos los niveles educativos que un usuario puede tener, como bachillerato, pregrado, posgrado, etc.
* **Roles:** Este modelo representa los roles posibles dentro de la aplicación.
* **Roles de Usuario:** En este modelo, establecemos la relación entre un usuario y los roles que puede tener.

export interface DocumentType {

id: number;

createdAt: string;

updateAt: string;

name: string;

status: boolean;

}

export interface GenderType {

id: number;

name: string;

status: boolean;

}

export interface User {

id: number;

username: string;

email: string;

avatar: string | null;

}

export interface NivelFormacion {

id: number;

nombre: string;

}

export interface Usuario {

id: number;

email: string;

username: string;

password: string;

avatar: string | null;

roles: number[];

}

export interface Rol {

id: number;

createdAt: string;

updateAt: string;

name: string;

status: boolean;

users: number[];

resources: number[];

}

export interface UserRole {

id: number;

status: boolean;

userId: {

id: number;

last\_login: string | null;

is\_superuser: boolean;

username: string;

first\_name: string;

last\_name: string;

is\_staff: boolean;

is\_active: boolean;

date\_joined: string;

email: string;

password: string;

resetToken: string | null;

avatar: string | null;

groups: number[];

user\_permissions: number[];

roles: number[];

};

rolesId: {

id: number;

createdAt: string;

updateAt: string;

name: string;

status: boolean;

users: number[];

resources: number[];];}

**solicitudes.ts:** mantenemos la misma práctica de definir modelos por separado, lo cual contribuye a la claridad y organización del código. En este contexto, hemos definido una serie de modelos necesarios para manejar los datos relacionados con las solicitudes. Los modelos que encontraremos aquí son:

* **"Estado":** Representa los posibles estados de una solicitud.
* **"Pasos":** Define los pasos que implica una solicitud.
* **"Seguimiento":** Establece el seguimiento de una solicitud a lo largo de su proceso.
* **"Solicitud":** Describe una solicitud específica, incluidos detalles como su título y contenido.
* **"Contenido":** Representa el contenido de una solicitud.
* **"AutorSolicitud":** Define el autor de una solicitud.
* **"RolAutores":** Establece los roles de los autores de una solicitud.
* **"Anexos":** Representa los anexos asociados a una solicitud.

Estos modelos son representados de la siguiente manera:

export interface Estado {

    id: number;

    nombre: string;

    descripcion: string;

    status: boolean;

}

export interface Pasos {

    id: number;

    nivel: string;

    nombre: string;

    dias\_programacion: string;

    status: boolean;

}

export interface Seguimiento {

    id: number;

    solicitudId: number;

    pasos\_seguimiento: number;

    fecha\_asignacion: string;

    fecha\_programacion: string;

    fecha\_evaluacion: string;

    estado\_seguimiento: number;

    responsableId: number;

    correciones: string;

    status: boolean;

}

export interface Solicitud {

    id: number;

    titulo\_articulo: string;

    autores: string;

    fecha: string;

    urls: string;

    orcid: string;

    afiliacion: string;

    contenidoSolicitud: number;

    status: boolean;

}

export interface Contenido {

    id: number;

    resumen: string;

    palabras\_claves: string;

    abstract: string;

    Keywords: string;

    introduccion: string;

    materi\_metodos: string;

    result\_discu: string;

    agradecimientos: string;

    literact\_citada: string;

    archivo\_adjunto: string;

    status: boolean;

}

export interface Anexos {

    id: number;

    solicitud: number;

    status: boolean;

}

export interface Autor\_Solicitud {

    id: number;

    RolAutor: number;

    solicitud: number;

    status: boolean;

}

export interface Rol\_Autor {

    id: number;

    nombre: string;

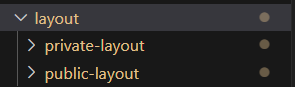
    status: boolean;

}

### Estructura y Rutas de Vistas en la Carpeta "layout"

En la fase siguiente de desarrollo, una vez que hayamos definido nuestros modelos de datos, nos adentraremos en la creación de la estructura de vistas y la configuración de rutas en nuestro proyecto. Esta etapa es fundamental para establecer las bases de la navegación y presentación de nuestro software. Para lograr esto, nos enfocaremos en la carpeta "src > app > layout", donde desplegamos una organización eficiente mediante subcarpetas "private" y "public".

Dentro de la carpeta "layout", creamos dos subcarpetas clave: "private" y "public". Estas subcarpetas tienen la función de albergar las vistas respectivas del software, dividiéndolas en públicas y privadas. La estructura se visualiza de esta manera:



La subcarpeta "private" contendrá las vistas privadas, accesibles sólo para usuarios autorizados. Por otro lado, la subcarpeta "public" albergará las vistas públicas, disponibles para todos los usuarios.

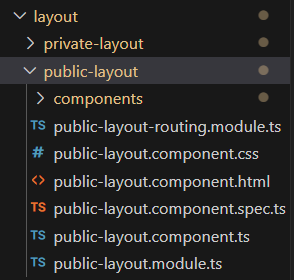
**Generación de Módulos y Rutas**

Para establecer una estructura de rutas eficiente, utilizamos la terminal. Dentro de la carpeta "layout", ejecutamos el siguiente comando: **ng generate module public-layout**

Este comando crea el archivo "public-layout.module.ts", que será fundamental para organizar las rutas de las vistas públicas. A continuación, ingresamos a la subcarpeta "public-layout" con el comando "cd public-layout" y generar el archivo "public-layout-routing.module.ts" utilizando el comando: **ng generate module estados-routing --flat --module=app**

Este archivo, "public-layout-routing.module.ts", será vital para definir y gestionar las rutas en el diseño de vistas públicas. Estos módulos y rutas aseguran una estructura coherente y manejable en el proyecto.

Dentro de "public-layout", creamos la subcarpeta "components", que albergará los componentes necesarios para construir las vistas públicas del sistema. Los componentes desempeñan un papel fundamental en la construcción modular de la interfaz de usuario y facilitan la reutilización de elementos en diferentes vistas.



### Generación de Componentes en la Carpeta "public-layout > components"

Ahora nos adentraremos en la creación de componentes en la subcarpeta "components" de la carpeta "public-layout". Esta fase nos permitirá construir los bloques esenciales de nuestras vistas públicas. Dos componentes fundamentales que generamos son "form-login" y "form-register". Estos componentes se encargan de las funcionalidades de inicio de sesión y registro en nuestro sistema.

### Generación de Componente "form-login"

Dentro de la carpeta "public-layout > components", utilizamos el siguiente comando en la terminal: **ng generate component form-login**

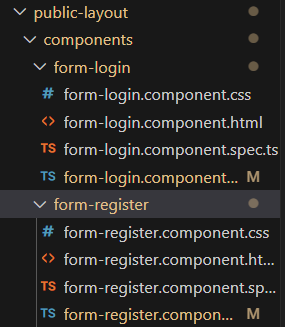
Este comando generará una nueva carpeta llamada "form-login" junto con los archivos correspondientes, que son vitales para la construcción del componente. Los archivos generados incluirán código TypeScript (.ts), HTML (.html), hojas de estilo CSS (.css) y un archivo de prueba (.spec.ts). Esta estructura modular nos permite desarrollar la lógica y la apariencia del formulario de inicio de sesión de manera separada.

### Generación de Componente "form-register"

De manera similar, para el componente "form-register", ejecutamos el siguiente comando: **ng generate component form-register**

Este comando generará una nueva carpeta llamada "form-register" junto con los archivos esenciales. Al igual que con el componente anterior, los archivos .ts, .html, .css y .spec.ts nos brindarán el lienzo para crear la funcionalidad y la presentación del formulario de registro.

Una vez que hayamos ejecutado estos comandos, habremos creado los componentes "form-login" y "form-register" en la subcarpeta "components" de "public-layout". Estos componentes actuarán como elementos modulares clave en la construcción de nuestras vistas públicas. La organización modular y la generación de componentes nos permiten desarrollar funcionalidades específicas de manera eficiente y reutilizable, allanando el camino para la creación de una experiencia de usuario sólida y coherente.



### Configuración de Rutas en "public-layout-routing.module.ts"

Siguiendo con la construcción de nuestro sistema, dirigimos nuestra atención al archivo "public-layout-routing.module.ts" ubicado en la carpeta "public-layout". Aquí realizamos importaciones esenciales para los componentes que hemos creado, específicamente "FormLoginComponent" y "FormRegisterComponent". También importamos las funcionalidades de enrutamiento proporcionadas por Angular a través de "RouterModule" y "Routes". Estas herramientas nos permiten definir cómo se mapean las URL a los componentes en nuestro sistema.

### Importación de Componentes y Módulos de Enrutamiento

Dentro del archivo "public-layout-routing.module.ts", realizamos las siguientes importaciones:

import { NgModule } from '@angular/core';

import { FormLoginComponent } from './components/form-login/form-login.component';

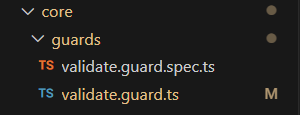
import { FormRegisterComponent } from './components/form-register/form-register.component';

import { RouterModule, Routes } from '@angular/router';

Antes de sumergirnos en la lógica interna de nuestros componentes, es fundamental establecer mecanismos de seguridad que regulen el acceso a ciertas partes del sistema. Para lograr esto, crearemos "guards" que actúen como guardianes de seguridad para nuestras rutas. Esto se logra mediante la implementación de componentes específicos que aplican reglas de autorización antes de permitir que un usuario acceda a una ruta determinada.

### Creación de Validaciones "guards"

Dirigimos nuestros pasos hacia la carpeta "core" ubicada en "src>app". Aquí generamos un nuevo guard llamado "guards". Ejecutamos el siguiente comando para crearlo: ng generate guard validate



Este comando generará los archivos básicos necesarios para el componente "guards", incluidos los archivos "guard.spec.ts" y "guard.ts".

### Roles de los Archivos Generados

El componente "guards" contendrá los archivos "guard.spec.ts" y "guard.ts". Estos archivos desempeñarán roles específicos:

* **guard.ts:** Aquí implementamos la lógica de nuestro guardia de seguridad. Definimos reglas que determinan si un usuario tiene el permiso necesario para acceder a una ruta específica. Si no se cumplen estas reglas, el guardia bloqueará el acceso y redirigirá al usuario a una página o acción alternativa.
* **guard.spec.ts:** Este archivo contiene pruebas y casos de prueba para asegurarse de que nuestro guardia de seguridad funcione correctamente y cumpla con sus funciones de autorización.

### Guardianes de Seguridad y Protección de Rutas

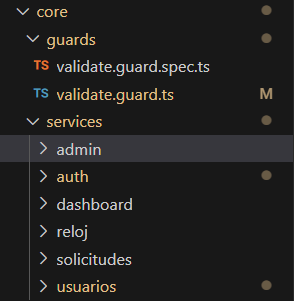
Los "guards" son componentes esenciales para proteger las rutas en nuestra aplicación. Al implementarlos, garantizamos que solo los usuarios autorizados puedan acceder a ciertas partes del sistema. Los "guards" actúan como una capa adicional de seguridad y ayudan a prevenir el acceso no autorizado. A medida que continuamos construyendo nuestro sistema, los "guards" serán una parte clave de la estrategia de seguridad global, lo que asegurará que los datos y la funcionalidad del sistema estén protegidos contra acceso no autorizado.

Creamos una nueva carpeta llamada "services" dentro de la carpeta "core". En esta carpeta, alojaremos todos los servicios necesarios para interactuar con el backend y realizar operaciones específicas.

La creación de la carpeta "services" proporciona una organización ordenada y coherente para los servicios del proyecto. Al separar estas funcionalidades en servicios individuales, mejoramos la modularidad y la mantenibilidad de nuestro código. Cada servicio se centra en una tarea específica, lo que facilita su reutilización en diferentes partes de la aplicación.

### Lógica de los Servicios

Dentro de cada servicio, implementamos la lógica correspondiente. Esto puede incluir la comunicación con el backend a través de peticiones HTTP, la manipulación y procesamiento de datos, la autenticación y autorización, entre otras tareas clave para el funcionamiento de la aplicación.



Dentro de la carpeta services>auth creamos un servicio llamado “auth” con el comando **ng generate service auth**, lo cual crea los siguientes archivos:



Dentro del archivo auth.service.ts, es importante realizar las importaciones necesarias para utilizar los componentes y elementos requeridos en la implementación del servicio de autenticación. A continuación, se muestra cómo se realizan estas importaciones:

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { listaMenuI } from 'src/app/models/menu';

import { catchError, retry, tap } from 'rxjs/operators';

import { BehaviorSubject } from 'rxjs';

import { UserI, UserLoginI, UserLoginResponseI } from 'src/app/models/Autorizacion';

import { environment } from 'src/environments/environment';

import \*as moment from 'moment';

import { Router } from '@angular/router';

A su vez, dentro de este archivo establecemos constantes y variables que desempeñarán un papel clave en el servicio de autenticación. Estas constantes y variables ayudan a administrar datos importantes y a mantener la lógica coherente. A continuación, detallamos cómo se realizan estas definiciones:

const KEY\_TOKEN = 'token';

const KEY\_USER = 'user';

const KEY\_MENU = 'menu';

const KEY\_FECHA = 'fecha';

Dentro de la clase AuthService, creamos variables esenciales para gestionar la funcionalidad del servicio de autenticación. Al declarar y definir estas variables, estamos estableciendo el marco necesario para manejar la autenticación del usuario. A continuación, describimos cómo se lleva a cabo esta definición:

@Injectable({

  providedIn: 'root'

})

export class AuthService {

  API\_URI = environment.API\_URI;

  public \_KEY\_CODE\_TOKEN = '';

  public \_KEY\_MENU = '';

  private \_isLoggedIn = false;

  public isLoggedIn$ = new BehaviorSubject(this.\_isLoggedIn);

  public userLoginResponse: UserLoginResponseI = {

    user: { username: '', id: 0 },

    token: '',

    menu: undefined

  }

public userLoginResponse$ = new BehaviorSubject(this.userLoginResponse);

Dado que hemos establecido la URL base de la API en la variable API\_URI, construimos las bases de las URL necesarias para interactuar con el backend:

**base\_path\_get:** Una URL base para acceder a ciertas rutas de la API relacionadas con la sesión administrativa principal. Esta URL se compone concatenando la API\_URI con la ruta específica.

**base\_path\_get2:** Similar a base\_path\_get, pero esta URL se usa para acceder a rutas de la API relacionadas con la sesión principal.

**base\_path\_get\_login:** Una URL base para realizar solicitudes de inicio de sesión. Se combina con la API\_URI y la ruta específica para el inicio de sesión.

base\_path\_get = `${this.API\_URI}/api/mainSesionAdmin`;

base\_path\_get2 = `${this.API\_URI}/api/mainSesion`;

base\_path\_get\_login = `${this.API\_URI}/api/auth/login/`;

El constructor del servicio se define a continuación, en el cual inyectamos las dependencias necesarias:

constructor(

    private http: HttpClient,

    private router: Router,

    ) {}

Luego, creamos una nueva función llamada getToken, la cual se encarga de obtener el token de autenticación. En esta función, utilizamos el valor almacenado en localStorage para calcular la diferencia en días entre la fecha almacenada y la fecha actual utilizando la librería Moment.js. Si la diferencia es igual a cero, sabemos que estamos obteniendo el token en ese mismo día y podemos proceder con el proceso.

 public getToken(): Observable<boolean> | boolean {

    let fechanueva: string | undefined = localStorage.getItem('fecha')?.replace(/["]/g, '');

    let a = moment(moment(fechanueva).format("YYYY-MM-DD"));

    let b = moment(moment().format("YYYY-MM-DD"));

    let algo: number = b.diff(a, 'days');

    if (algo == 0) {

### Lógica de Obtención de Token y Validación en auth.service.ts

En esta sección, creamos la función getToken dentro del servicio AuthService para gestionar la obtención del token de autenticación y realizar las validaciones correspondientes. El objetivo es verificar si la acción se está realizando en el mismo día o si ha pasado más de un día desde la última obtención del token.

La función getToken comienza imprimiendo un mensaje de depuración en la consola para rastrear el proceso de verificación del token y la fecha. Luego, obtenemos la fecha almacenada en el navegador a través de localStorage y la fecha actual utilizando la biblioteca Moment.js. Calculamos la diferencia en días entre estas dos fechas para determinar si la acción se realizó en el mismo día.

Si la diferencia de días es cero, procedemos a realizar la verificación y validación:

* Obtenemos el token y el menú almacenados en el navegador.
* Si el usuario ya está autenticado (valor isLoggedIn$ es verdadero), se permite el acceso.
* Si hay un token y un menú almacenados, asumimos que el usuario ha estado autenticado. Creamos un objeto con los datos del usuario, lo marcamos como autenticado y actualizamos los datos del usuario en el servicio.
* Si no se cumplen las condiciones anteriores, redirigimos al usuario a la página de aterrizaje.

Si la diferencia de días no es cero, procedemos a invalidar el token y redirigir al usuario a la página de inicio de sesión:

* Eliminamos los datos almacenados relacionados con el token, el usuario y la fecha.
* Marcamos al usuario como no autenticado.
* Redirigimos al usuario a la página de inicio de sesión.

En resumen, esta parte del código contiene la lógica para verificar la validez del token de autenticación y gestionar las acciones correspondientes según el resultado de la verificación. Se encarga de mantener la autenticación del usuario y redirigirlo adecuadamente en función de la fecha de obtención del token.

public getToken(): Observable<boolean> | boolean {

    let fechanueva: string | undefined = localStorage.getItem('fecha')?.replace(/["]/g, '');

    let a = moment(moment(fechanueva).format("YYYY-MM-DD"));

    let b = moment(moment().format("YYYY-MM-DD"));

    let algo: number = b.diff(a, 'days');

    if (algo == 0) {

      var token: string | null = JSON.stringify(localStorage.getItem('token'));

      var menu: string | null = localStorage.getItem('menu');

      if (this.isLoggedIn$.value == true) {

        return true;

      } else {

        if (token != null && menu !== undefined) {

          let userLoginResponse = {

            token: token,

            menu: undefined

          }

          this.setLogin(true)

          this.userLoginResponse = userLoginResponse

          this.userLoginResponse$.next(userLoginResponse)

          return true;

        }

      }

      this.router.navigateByUrl('/landing');

      return false

    } else {

      localStorage.removeItem(KEY\_TOKEN);

      this.\_KEY\_CODE\_TOKEN = '';

      localStorage.removeItem(KEY\_USER);

      localStorage.removeItem(KEY\_FECHA);

      localStorage.removeItem(KEY\_MENU);

      this.setLogin(false);

      this.router.navigateByUrl('/login');

      return false

    }

  }

Una vez tenemos nuestro servicio getToken creado volvemos a guards>validate.guards.ts y lo importamos.

import { AuthService } from '../services/auth/auth.service';

Creamos la clase y la implementamos cuando esté activa y cuando cargue también en nuestro constructor declaramos las rutas y el servicio necesario de esta manera:

@Injectable({

  providedIn: 'root'

})

export class ValidateGuard implements CanActivate, CanLoad {

  constructor(

    private router: Router,

    private authService: AuthService,

  ) { }

Una vez tenemos nuestros elementos importados y nuestro constructor creado lo siguiente es crear las funciones canActive y canLoad lo cual hacemos y explicamos a continuación:

canActivate(

  route: ActivatedRouteSnapshot,

  state: RouterStateSnapshot

): Observable<boolean> | boolean {

  if (this.authService.getToken() == of(false)) {

    this.router.navigateByUrl('/login');

    return of(false);

  }

  return of(true);

}

canLoad(

  route: Route,

  segments: UrlSegment[]

): Observable<boolean> | boolean {

  if (this.authService.getToken() == of(false)) {

    this.router.navigateByUrl('/login');

    return of(false);

  }

  return of(true);

}

### Creación de Rutas Protegidas en public-layout-routing.module.ts

En esta sección, vamos a definir las rutas protegidas para el módulo public-layout, que contiene las vistas públicas como el inicio de sesión y el registro. Utilizamos la validación del guardia ValidateGuard para asegurarnos de que solo los usuarios no autenticados puedan acceder a estas rutas.

Primero, importamos ValidateGuard y los componentes necesarios:

import { NgModule } from '@angular/core';

import { ValidateGuard } from 'src/app/core/guards/validate.guard';

import { FormLoginComponent } from './components/form-login/form-login.component';

import { FormRegisterComponent } from './components/form-register/form-register.component';

import { RouterModule, Routes } from '@angular/router';

Luego, definimos las rutas utilizando los componentes de inicio de sesión y registro, y asociamos el guardia !ValidateGuard para asegurarnos de que los usuarios no autenticados puedan acceder a estas rutas:

const routes: Routes = [

  {

    path: 'login',

    component:FormLoginComponent,

    canLoad:[!ValidateGuard]

  },

  {

    path: 'register',

    component: FormRegisterComponent,

    canLoad:[!ValidateGuard]

  },

];

En el módulo PublicLayoutModule, importamos todos los elementos y módulos que se utilizarán en los componentes del módulo public-layout. Esto incluye importar componentes de PrimeNG, módulos relacionados con la interfaz de usuario y servicios:

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { PublicLayoutRoutingModule } from './public-layout-routing.module';

import { PublicLayoutComponent } from './public-layout.component';

import { LandingComponent } from '../private-layout/landing/landing.component';

import {MultiSelectModule} from 'primeng/multiselect';

import { FormRegisterComponent } from './components/form-register/form-register.component';

import { FormLoginComponent } from './components/form-login/form-login.component';

import { ButtonModule } from 'primeng/button';

import { DialogModule } from 'primeng/dialog';

import { DividerModule } from 'primeng/divider';

import { InputTextModule } from 'primeng/inputtext';

import { AvatarGroupModule } from 'primeng/avatargroup';

import { AvatarModule } from 'primeng/avatar';

import { ToastModule } from 'primeng/toast';

import { ConfirmPopupModule } from 'primeng/confirmpopup';

import { ConfirmationService, MessageService, SharedModule } from 'primeng/api';

import { CarouselModule } from 'primeng/carousel';

import { GalleriaModule } from 'primeng/galleria';

import { HttpClientModule } from '@angular/common/http';

import { CheckboxModule } from 'primeng/checkbox';

import { RadioButtonModule } from 'primeng/radiobutton';

import { RippleModule } from 'primeng/ripple';

import { MenuModule } from 'primeng/menu';

import { MessagesModule } from 'primeng/messages';

import { CardModule } from 'primeng/card';

import { MessageModule } from 'primeng/message';

import { SplitterModule } from 'primeng/splitter';

import { MenubarModule } from 'primeng/menubar';

import { SidebarModule } from 'primeng/sidebar';

import { PanelMenuModule } from 'primeng/panelmenu';

import { ChartModule } from 'primeng/chart';

import { ToolbarModule } from 'primeng/toolbar';

import { SplitButtonModule } from 'primeng/splitbutton';

import { TreeModule } from 'primeng/tree';

import { TableModule } from 'primeng/table';

import { KeyFilterModule } from 'primeng/keyfilter';

import { DropdownModule } from 'primeng/dropdown';

import { CalendarModule } from 'primeng/calendar';

import { FieldsetModule } from 'primeng/fieldset';

import { FileUploadModule } from 'primeng/fileupload';

import { InputTextareaModule } from 'primeng/inputtextarea';

import {PasswordModule} from 'primeng/password';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import {AccordionModule} from 'primeng/accordion';

@NgModule({

  declarations: [

    PublicLayoutComponent,

    FormLoginComponent,

    FormRegisterComponent,

  ],

  imports: [

    CommonModule,

    ReactiveFormsModule,

    ButtonModule,

    DialogModule,

    DividerModule,

    InputTextModule,

    FormsModule,

    ReactiveFormsModule,

    CarouselModule,

    GalleriaModule,

    HttpClientModule,

    MultiSelectModule,

    ConfirmPopupModule,

    ToastModule,

    AvatarGroupModule,

    AvatarModule,

    PublicLayoutRoutingModule,

    FormsModule,

    SharedModule,

    HttpClientModule,

    FormsModule,

    ReactiveFormsModule,

    AccordionModule,

    DividerModule,

    ButtonModule,

    CheckboxModule,

    RadioButtonModule,

    RippleModule,

    CardModule,

    MenuModule,

    MessagesModule,

    MessageModule,

    SplitterModule,

    MenubarModule,

    SidebarModule,

    PanelMenuModule,

    ChartModule,

    ToolbarModule,

    SplitButtonModule,

    TreeModule,

    TableModule,

    KeyFilterModule,

    DropdownModule,

    CalendarModule,

    FieldsetModule,

    FileUploadModule,

    InputTextareaModule,

  PasswordModule

  ],providers: [ConfirmationService,MessageService],

})

export class PublicLayoutModule { }

Dentro del archivo HTML de nuestro public-layout, utilizamos la directiva <router-outlet></router-outlet> para insertar el contenido de la vista de la ruta activa en ese lugar. Esto permitirá que los componentes de inicio de sesión y registro se carguen en este componente según las rutas definidas.

<router-outlet></router-outlet>

**Definición de la Vista del Formulario de Inicio de Sesión**

En esta sección, se describe cómo se estructura y presenta el formulario de inicio de sesión dentro del componente FormLoginComponent. El formulario utiliza elementos y estilos de PrimeNG para lograr una apariencia moderna y atractiva. Además, se muestra cómo se maneja la lógica de autenticación en el archivo .ts del componente.

El archivo form-login.component.html contiene la estructura del formulario de inicio de sesión:

<form (ngSubmit)="onSubmit()" [formGroup]="form" novalidate>

  <p-dialog [(visible)]="displayMaximizable" [modal]="true"

    [style]="{'width': '77%','max-height':'80%','text-align': 'center','background':'#333333'}"

    contentStyleClass="overflow-y-hidden" [closable]="false" [closeOnEscape]="false" [resizable]="false" zindex="10000"

    [showHeader]="false">

    <div class="grid grid-nogutter surface-section text-800">

      <div

        class="col-12 sm:col-6 md:col-6 lg:col-4 xl:col-4 pl-3 pr-3 pb-6 text-center md:text-center align-items-center ">

        <section class="mr-1 md:mt-4 md:mb-3">

          <span class="block text-6xl font-bold mb-3" style="color:black">Revista

            <strong class="block" style="color:gray;font-size: 0.8rem;">

              Ciencia e ingenieria - Uniguajira

            </strong>

          </span>

          <p-avatar styleClass="mr-2 " shape="circle" size="xlarge"><img src="assets/logo\_r.jpg"></p-avatar>

          <div [style]="{'margin-top':'-1px'}">

            <div fxLayout="row" class="titleCard">

              <i class="pi pi-shield" style="font-size: 2rem;margin-right: 10px;"></i>

              <h2 class="fondebabutton" style="margin-top: -1px;">Inicio de sesión</h2>

            </div>

            <div class="p-grid p-fluid">

              <div class="p-col-12 p-md-6">

                <div class="form-field">

                  <span class="pi pi-user"></span>

                  <input type="text" pInputText formControlName="username" placeholder="Nombre de usuario">

                </div>

              </div>

              <div class="p-col-12 p-md-6">

                <div class="form-field">

                  <span class="pi pi-key"></span>

                  <input type="password" pInputText formControlName="password" placeholder="Contraseña">

                </div>

              </div>

            </div>

          </div>

          <ng-template [ngIf]="motrar == false">

            <div fxLayout="row" class="titleCard ">

              <button pButton pRipple type="submit" icon="pi pi-check" label="Entrar" [disabled]="!form.valid"

                style="width:auto" styleClass="mr-2" class="mr-3 p-button-raised mt-3"></button>

              <button pButton pRipple type="button" routerLink="/register" icon="pi pi-users" styleClass="p-button-text"

                style="width:auto" label="Registrarse" class="p-button-outlined mt-3"></button>

            </div>

          </ng-template>

          <ng-template [ngIf]="motrar == true">

            <br>

            <br>

            <p-progressBar mode="indeterminate" [style]="{'height': '6px'}"></p-progressBar>

          </ng-template>

        </section>

      </div>

      <div class="col-12 sm:col-6 md:col-6 lg:col-8 xl:col-8 overflow-hidden " style="border-radius: 20px;">

        <img src="assets/revista\_principal.jpg" alt="Image" class="md:ml-auto block "

          style="clip-path: polygon(8% 0, 100% 0%, 100% 100%, 0 100%);height:100%">

      </div>

    </div>

  </p-dialog>

</form>

<img width="100%" height="45%" src="assets/Revista\_c.jpg">

<p-toast></p-toast>

<p-confirmPopup></p-confirmPopup>

El presente formulario utiliza la directiva ngSubmit para llamar al método onSubmit() cuando se envía el formulario. Además, se utiliza la directiva formGroup para vincular el formulario de Angular al formulario HTML.

Dentro del componente FormLoginComponent “form-login.component.ts”, se maneja la lógica del formulario de inicio de sesión:

import { Component, Inject, OnInit } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { Router } from '@angular/router';

import { MessageService } from 'primeng/api';

import { AuthService } from 'src/app/core/services/auth/auth.service';

import { UserLoginI } from 'src/app/models/Autorizacion';

import { HttpClient } from '@angular/common/http';

@Component({

  selector: 'app-form-login',

  templateUrl: './form-login.component.html',

  styleUrls: ['./form-login.component.css']

})

export class FormLoginComponent implements OnInit {

  displayMaximizable: boolean = true

  public form: FormGroup = this.formBuilder.group({

    username: ['', [Validators.required]],

    password: ['', [Validators.required]],

  });

  public motrar: boolean = false

  public image3: string = 'assets/demo.png'

  constructor(

    private formBuilder: FormBuilder,

    private authService: AuthService,

    private router: Router,

    private messageService: MessageService,

    private http: HttpClient

  ) { }

  ngOnInit(): void {

    const token: string | null = localStorage.getItem('token');

    const user: string | null = localStorage.getItem('user');

    console.log('token:', token);

    console.log('user:', user);

    if (token !== null && user !== null) {

      console.log('landing');

      this.router.navigateByUrl('/landing');

    } else {

      console.log('login');

      this.router.navigateByUrl('/login');

    }

  }

  onSubmit() {

    const form: UserLoginI = this.form.value;

    const requestBody = JSON.stringify(form);

    this.authService.login(form).subscribe((result) => {

      if (result)

        this.motrar = true;

      if (result && result.data && result.data.user && result.data.user.name) {

        this.messageService.add({ severity: 'success', summary: `Bienvenido ${result.data.user.name}` });

      }

      var date = new Date('2020-01-01 00:00:04');

      function padLeft(n: any) {

        return n = "00".substring(0, "00".length - n.length) + n;

      }

      var interval = setInterval(() => {

        var minutes = padLeft(date.getMinutes() + "");

        var seconds = padLeft(date.getSeconds() + "");

        date = new Date(date.getTime() - 1000);

        if (minutes === '00' && seconds === '01') {

          this.router.navigateByUrl('/landing');

          clearInterval(interval);

        }

      }, 1000);

    }, async error => {

      this.motrar = false;

      if (error != undefined) {

        if (error.error) {

          this.messageService.add({ severity: 'error', summary: 'Error', detail: `Credenciales incorrectas` });

        }

        console.log(error);

      }

    })

    console.log(requestBody);

  }

}

Dentro de onSubmit(), se obtienen los valores del formulario, se realiza una llamada al servicio AuthService para iniciar sesión y se maneja la respuesta del servidor. Si el inicio de sesión es exitoso, se muestra un mensaje de bienvenida y se configura una cuenta regresiva para redirigir a la página de aterrizaje. En caso de error, se muestra un mensaje de error y se oculta la barra de progreso todo este proceso se lleva a cabo con el servicio presentado a continuación:

login(dataLogin: UserLoginI): Observable<UserLoginResponseI> {

    return this.http.post<UserLoginResponseI>(this.base\_path\_get\_login, dataLogin).pipe(tap(

      (res: UserLoginResponseI) => {

        if (res) {

          console.log(res)

          this.setLoginData(res.data);

          this.userLoginResponse$.next(res)

          this.setLogin(true);

        }

      })

    );

  }

Finalmente, el archivo de estilos form-login.component.css se utiliza para aplicar estilos específicos al formulario de inicio de sesión, los campos de entrada y otros elementos de la página.

/\* Estilos generales \*/

.back-to-top {

  position: relative;

  visibility: hidden;

  opacity: 0;

  right: 15px;

  bottom: 75px;

  z-index: 996;

  background: #0026ff;

  height: 40px;

  border-radius: 4px;

  transition: all 0.4s;

}

.back-to-top i {

  font-size: 28px;

  color: #fff;

  line-height: 0;

}

.back-to-top:hover {

  background: #0026ff;

  color: #fff;

}

.back-to-top.active {

  visibility: visible;

  opacity: 1;

}

/\* Estilos de formularios y campos de entrada \*/

.form-field {

  position: relative;

  margin-bottom: 20px;

  width: 100%;

}

.form-field input {

  font-size: 16px;

  border: none;

  background-color: #f2f2f2;

  border-radius: 5px;

  padding: 10px 20px 10px 40px;

  width: 100%;

  box-shadow: none;

  transition: box-shadow 0.3s ease-in-out;

}

.form-field input:focus {

  outline: none;

  box-shadow: 0 0 5px rgba(50, 50, 93, 0.15), 0 2px 2px rgba(0, 0, 0, 0.1);

}

.form-field .pi {

  position: absolute;

  left: 10px;

  top: 50%;

  transform: translateY(-50%);

  color: #5f5f5f;

  font-size: 20px;

}

/\* Estilos de títulos y secciones \*/

.titleCard {

  margin: 0;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

.fondebabutton {

  color: #707070;

}

/\* Estilos de botones \*/

.buttons-container {

  color: #707070;

  display: flex;

  align-items: center;

  justify-content: center;

  height: 75px;

  padding: 0 5px;

  text-decoration: none;

  margin-top: -15px;

  margin-right: 10px;

}

.buttons-container:hover {

  background-color: #c9c9c9f3;

  color: black;

}

.buttons-container .mat-icon {

  padding-top: 10px;

}

**Registro de Usuarios**

El componente de registro de usuario es una parte fundamental de esta aplicación Angular que permite a los usuarios crear nuevas cuentas. A continuación, se detallan los pasos necesarios para implementar y entender este componente.

**Estructura de Archivos**

A continuación, se describen los archivos y carpetas clave relacionados con el registro de usuario:

* **user.service.ts:** Este archivo define el servicio UserService, que se encarga de manejar las solicitudes HTTP relacionadas con la creación de usuarios.
* **register.component.ts:** En este archivo se encuentra el componente FormRegisterComponent, que implementa la lógica del formulario de registro y maneja la interacción con el servicio.
* **form-register.component.html:** El archivo HTML contiene la estructura y los elementos visuales del formulario de registro, que se presenta en un diálogo emergente.
* **form-register.component.css:** Este archivo CSS define los estilos para el componente de registro, incluyendo la apariencia de los campos de formulario, íconos, títulos y más.

**UserService - Servicio de Registro**

El servicio UserService es responsable de gestionar las solicitudes HTTP para la creación de nuevos usuarios. Las URLs necesarias para las solicitudes se definen en el servicio y se importan desde el archivo environment.ts. El método createUser envía una solicitud POST con los datos del nuevo usuario y utiliza los operadores tap y catchError para gestionar las respuestas y los errores respectivamente.



Accedemos al archivo user.service.ts y en su interior llevamos a cabo todas las importaciones que emplearemos más adelante:

import { HttpClient, HttpHeaders } from '@angular/common/http';

import { Injectable } from '@angular/core';

import { Observable, throwError } from 'rxjs';

import { catchError, retry, tap } from 'rxjs/operators';

import { CambiarPasswordI, UserI } from 'src/app/models/Autorizacion';

import { environment } from 'src/environments/environment';

import { Person, Person\_request, NivelFormacion} from 'src/app/models/person';

import { map } from 'rxjs/operators';

Posteriormente, recurrimos al entorno (environment) para forjar las URL que se requiere en este servicio.

@Injectable({

  providedIn: 'root'

})

export class UserService {

  API\_URI = environment.API\_URI;

  // API path

  base\_path = `${this.API\_URI}/api/registro/`;

  base\_path\_post = `${this.API\_URI}/api/registro/`;

  base = `${this.API\_URI}/api/auth/reset/`;

  base\_nivel\_formacion = `${this.API\_URI}/solicitud/nivel/niveles\_formacion/`;

  base\_personas = `${this.API\_URI}/persons/`;

  base\_personas\_crear = `${this.API\_URI}/persons/create/`;

  base\_personas\_editar = `${this.API\_URI}/persons/update/`;

  base\_documentos = `${this.API\_URI}/documents/`;

  base\_usuarios = `${this.API\_URI}/api/users/`;

  base\_gender = `${this.API\_URI}/genders/`;

  base\_usuario\_rol = `${this.API\_URI}/roles/user\_roles/create/`;

  base\_editar\_user = `${this.API\_URI}/api/user/update/`;

  constructor(private http: HttpClient,) { }

A continuación, instauramos el método "**createUser**", el cual ejecuta una petición POST con el propósito de instaurar un recién creado usuario. Se apoya en el operador "**tap**" para llevar a cabo acciones no obligatorias tras recibir la respuesta, incluyendo lógica vinculada al éxito del resultado. Asimismo, hace uso del operador "**catchError**" para gestionar posibles errores que puedan acontecer durante la solicitud.

createUser(person: any): Observable<any> {

    return this.http.post<any>(this.base\_path\_post, person).pipe(

      tap((res: any) => {

        if (res) {

        }

      }),

      catchError(this.handleError))

  }

**FormRegisterComponent - Lógica del Componente**

El componente FormRegisterComponent se encarga de gestionar la lógica del formulario de registro. Contiene la lógica para verificar el dominio del correo electrónico, construir el formulario, enviarlo y realizar validaciones. El método verificar valida el dominio del correo electrónico en función de un conjunto de dominios permitidos.

import { Component, Inject, OnInit } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { Router } from '@angular/router';

import { MessageService } from 'primeng/api';

import { UserService } from 'src/app/core/services/usuarios/user.service';

import { RoleI } from 'src/app/models/Autorizacion';

@Component({

  selector: 'app-form-register',

  templateUrl: './form-register.component.html',

  styleUrls: ['./form-register.component.css']

})

export class FormRegisterComponent implements OnInit {

  public form: FormGroup = this.formBuilder.group({});

  public mostrar: boolean = false;

  displayMaximizable: boolean = true

  selectedCities3: any[] = [];

  cities: RoleI[] = [];

  public image: string = 'assets/demo.png'

  blockSpecial: RegExp = /^[^<>\*!@.,]+$/

  public Roles1: any[] = []

  public bandera: boolean = false

  constructor(

    private formBuilder: FormBuilder,

    private router: Router,

    private messageService: MessageService,

    private userService: UserService,

  ) {

  }

  ngOnInit(): void {

    var token: string | null = localStorage.getItem('token');

    var user: string | null = localStorage.getItem('user');

    if (token != null && user != null) {

      let userObjeto: any = JSON.parse(user);

      let userLoginResponse = {

        user: userObjeto,

        token: token,

      }

      this.router.navigateByUrl('/welcome');

    } else {

    }

    this.buildForm();

  }

  public verificar() {

    let email = this.form;

    let dominios = new Array('uniguajira.edu.co');

    if (email.value.email1 == '' || email.value.email1 == 'undefined') {

      console.log('El campo es obligatorio');

      return false;

    } else {

      let value = email.value.email1.split('@');

      if (value[1] == undefined) {

        return false;

      } else {

        console.log(value[1]);

        if (dominios.indexOf(value[1]) == -1) {

          console.log('dominio no encontrado');

          dominios.forEach(function (dominio) {

            console.log(`Los dominios aceptados son: ${dominio}`);

          })

          this.messageService.add({ severity: 'warn', summary: 'Warn', detail: `El dominio aceptado es: ${dominios[0]}`, life: 1000 });

          return false;

        } else {

          console.log('dominio aceptado');

          return true;

        }

      }

    }

  }

  private buildForm() {

    this.form = this.formBuilder.group({

      username: ['', [Validators.required]],

      first\_name: ['', [Validators.required]],

      last\_name: ['', [Validators.required]],

      email1: ['', [Validators.required]],

      password: ['', [Validators.required]],

      password2: ['', [Validators.required]],

    });

  }

  onSubmit() {

    let value = this.form.value.email1.split('@');

    let formValue = {

      username: this.form.value.username,

      first\_name: this.form.value.first\_name,

      last\_name: this.form.value.last\_name,

      email: `${value[0]}@uniguajira.edu.co`,

      password: this.form.value.password,

    };

    if (this.form.value.password !== this.form.value.password2) {

      this.messageService.add({ severity: 'warn', summary: 'Alerta', detail: 'Las contraseñas no coinciden' });

    } else {

      if (formValue.username != "" &&

        formValue.email != "" &&

        formValue.password != ""

      ) {

        this.bandera = true

        console.log(formValue)

        this.userService.createUser(formValue).subscribe(

          (algo) => {

            var date = new Date('2020-01-01 00:00:03');

            function padLeft(n: any) {

              return n = "00".substring(0, "00".length - n.length) + n;

            }

            var interval = setInterval(() => {

              var minutes = padLeft(date.getMinutes() + "");

              var seconds = padLeft(date.getSeconds() + "");

              if (seconds == '03') {

                this.messageService.add({

                  severity: 'success', summary: 'Success',

                  detail: 'Registro exitoso.'

                });

              }

              if (seconds == '01') {

                this.bandera = false

              }

              date = new Date(date.getTime() - 1000);

              if (algo.ok && minutes == '00' && seconds == '01') {

                this.bandera = false

                this.router.navigateByUrl('/login');

                clearInterval(interval);

              }

            }, 1000);

          }, async (responseError) => {

            if (responseError.error && responseError.error.errors && responseError.error.errors.error) {

              this.messageService.add({ severity: 'error', summary: 'Error', detail: `Error. ${responseError.error.errors.error.person}` });

            }

            console.log(responseError);

            this.bandera = false;

          });

      } else {

        this.messageService.add({ severity: 'warn', summary: 'Warn', detail: 'Faltan datos' });

        this.bandera = false

      }

    }

  }

}

**Formulario de Registro - Interfaz de Usuario**

El archivo HTML form-register.component.html define la estructura visual del formulario de registro. Utiliza elementos de PrimeNG para crear un diálogo emergente con campos de formulario organizados en columnas y secciones. También muestra una imagen de fondo y elementos visuales para mostrar progreso.

<form (ngSubmit)="onSubmit()" [formGroup]="form" novalidate>

  <p-dialog [(visible)]="displayMaximizable" [modal]="true"

    [style]="{'width': '77%','max-height':'80%','text-align': 'center','background':'#333333'}" [closable]="false"

    [closeOnEscape]="false" [resizable]="false" zindex="10000" [showHeader]="false">

    <div class="grid grid-nogutter surface-section text-800 ">

      <div class="col-12 sm:col-7 md:col-7 lg:col-7 xl:col-7 overflow-hidden">

        <img src="assets/revista\_principal.jpg" alt="Image" class="sm:ml-auto block sm:h-full sm:w-full "

          style="clip-path: polygon(0% 0%, 95% 0%, 85% 100%, 0% 100%) ">

      </div>

      <div class="col-12 sm:col-5 md:col-5 lg:col-5 xl:col-5 pl-3 pr-3  text-center

      md:text-center  align-items-center ">

        <section class=" text-center ">

          <div fxLayout="row" class="titleCard" style="margin-left: -25px;">

            <p-avatar styleClass="" shape="circle" size="xlarge"> <img src="assets/logo\_r.jpg"></p-avatar>

            <span class="block text-6xl font-bold mb-2" style="color:black">

              REVISTA

              <strong class="block" style="color:gray;font-size: 0.8rem;">

                Ciencia e ingenieria - Uniguajira

              </strong>

            </span>

          </div>

          <div [style]="{'margin-top':'-6px'}">

            <br>

            <div fxLayout="row" class="titleCard">

              <i class="pi pi-shield" style="font-size: 2rem;margin-right: 10px;margin-top: -10px;"></i>

              <h2 class="fondebabutton" style="margin-top: -12px;">Registro de Usuario</h2>

            </div>

            <br>

            <div class="grid">

              <div class="col-12 sm:col-6" style="width: 100%;">

                <div class="p-inputgroup mb-2">

                  <span class="p-inputgroup-addon"><i class="pi pi-user"></i></span>

                  <input type="text" pInputText formControlName="username" placeholder="username" title="Username">

                </div>

                <div class="p-inputgroup  mb-2 ">

                  <span class="p-inputgroup-addon"><i class="pi pi-id-card"></i></span>

                  <input type="text" pInputText formControlName="first\_name" placeholder="Nombres" required

                    title="Nombres">

                </div>

                <div class="p-inputgroup  mb-2 ">

                  <span class="p-inputgroup-addon"><i class="pi pi-id-card"></i></span>

                  <input type="text" pInputText formControlName="last\_name" placeholder="Apellidos" required

                    title="Apellidos">

                </div>

                <div class="p-inputgroup  mb-2 ">

                  <input type="text" pInputText [pKeyFilter]="blockSpecial" class="p-inputtext" name="email1"

                    formControlName="email1" placeholder="Correo" required title="Correo Electronico">

                  <span class="p-inputgroup-addon">@uniguajira.edu.co</span>

                </div>

                <div class="grid">

                  <div class="col-12 sm:col-6">

                    <div class="p-inputgroup mb-1 ">

                      <span class="p-inputgroup-addon"><i class="pi pi-key"></i></span>

                      <input type="password" pInputText formControlName="password" placeholder="Contraseña"

                        title="Contraseña" minlength="8" maxlength="50">

                    </div>

                  </div>

                  <div class="col-12 sm:col-6">

                    <div class="p-inputgroup mb-1 ">

                      <span class="p-inputgroup-addon"><i class="pi pi-key"></i></span>

                      <input type="password" pInputText formControlName="password2"

                        placeholder="Confirmación de Contraseña" title="Confirmación Contraseña" minlength="8"

                        maxlength="50">

                    </div>

                  </div>

                </div>

              </div>

            </div>

          </div>

          <ng-template [ngIf]="bandera == false">

            <br>

            <div fxLayout="row" class="titleCard ">

              <button pButton pRipple type="submit" icon="pi pi-check" label="Enviar" [disabled]="!form.valid"

                class=" mr-2 p-button-success "></button>

              <button pButton pRipple type="button" routerLink="/login" icon="pi pi-sync" styleClass="p-button-text"

                label="Login" class="p-button-outlined "></button>

            </div>

          </ng-template>

          <ng-template [ngIf]="bandera == true">

            <br>

            <br>

            <p-progressBar mode="indeterminate" [style]="{'height': '6px'}"></p-progressBar>

          </ng-template>

        </section>

      </div>

    </div>

  </p-dialog>

</form>

<img width="100%" height="45%" src="assets/Revista\_c.jpg">

<p-toast></p-toast>

<p-confirmPopup></p-confirmPopup>

**Estilos CSS - Diseño y Apariencia**

El archivo CSS form-register.component.css define los estilos visuales y el diseño general del componente de registro. Controla la apariencia de los campos de formulario, tarjetas, íconos, títulos, áreas de la cuadrícula, botones y enlaces.

/\* Estilos para el formulario \*/

mat-form-field {

  width: 100%;

  min-width: 385px;

}

.mat-input-element {

  caret-color: #F16257;

}

/\* Estilos para las tarjetas \*/

mat-card {

  width: 40%;

  overflow-x: auto;

  overflow-y: hidden;

}

/\* Estilos para íconos \*/

.mat-icon {

  font-size: 30px;

  background-color: #707070;

}

/\* Estilos para títulos de tarjetas \*/

.titleCard {

  margin: 0 0 10px 0;

  display: flex;

  justify-content: center;

  align-items: center;

  text-align: center;

}

.fondebabutton {

  color: #707070;

}

/\* Estilos para el contenido y estructura de la página \*/

.grid-container {

  display: grid;

  grid-template-columns: 250px 3fr;

  grid-template-rows: 40px 3fr 80px;

  grid-template-areas:

    "sidenav subtitle"

    "sidenav main"

    "sidenav footer";

  height: 100%;

  width: 100vmax;

}

.sidenav {

  grid-area: sidenav;

  overflow: auto;

  scrollbar-width: none;

  box-shadow: 3px 0 5px rgba(57, 63, 72, 0.3);

}

.sidenav::-webkit-scrollbar {

  display: none;

}

.subtitle {

  grid-area: subtitle;

  margin-top: 20px;

}

.main {

  grid-area: main;

  margin: 5px;

}

.footer {

  grid-area: footer;

  display: flex;

  align-content: center;

  justify-content: space-around;

  margin: 3px 0 0 2px;

}

/\* Estilos para los botones \*/

.clickable {

  cursor: pointer;

}

.buttons-container {

  color: #707070;

  display: flex;

  align-items: center;

  justify-content: center;

  height: 75px;

  padding: 0 5px;

  text-decoration: none;

  margin-top: -15px;

  margin-right: 10px;

}

.buttons-container:hover {

  background-color: #c9c9c9f3;

  color: black;

}

.buttons-container .mat-icon {

  padding-top: 10px;

}

a {

  color: #707070;

  text-decoration: none;

}

a .text {

  font-weight: 500;

  font-size: 16px;

  margin-left: 5px;

  padding-top: 7px;

}

.padre-container {

  color: #707070;

  display: flex;

  align-items: center;

  justify-content: center;

  height: inherit;

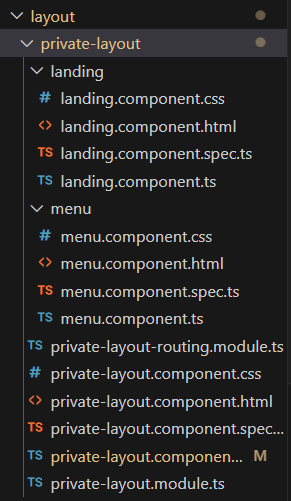
  padding: 0 10px;

  text-decoration: none;

}

### Desarrollo de las Páginas Solo para Usuarios: Incorporación Estratégica

En esta etapa del proceso, nos sumergimos en la tarea de construir las páginas que solo estarán disponibles para los usuarios registrados. Para lograr esto, vamos a crear archivos similares a los que creamos anteriormente para la sección pública. Sin embargo, esta vez los vamos a integrar en el entorno privado que creamos previamente, agregando los componentes esenciales como "landing" y "menú". Una vez que hayamos completado la creación de los archivos y componentes necesarios, obtendremos una estructura similar a la que se muestra a continuación:



**Importando Módulos y Componentes**

Una vez establecida esta estructura, el siguiente paso es llevar todos los módulos y componentes que necesitamos al archivo "private-layout.module.ts". Esto implica agruparlos de manera organizada en las secciones correspondientes: "imports", "declarations" y "providers". Este enfoque organizado es fundamental para establecer una base sólida en la que construir nuestras funcionalidades y asegurarnos de que todo funcione de manera eficiente.

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { PrivateLayoutRoutingModule } from './private-layout-routing.module';

import { PrivateLayoutComponent } from './private-layout.component';

import { HttpClientModule } from '@angular/common/http';

import { DashboardRoutingModule } from 'src/app/main/dashboard/dashboard-routing.module';

import { BrowserModule } from '@angular/platform-browser';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { DividerModule } from 'primeng/divider';

import { ButtonModule } from 'primeng/button';

import { CheckboxModule } from 'primeng/checkbox';

import { RadioButtonModule } from 'primeng/radiobutton';

import { RippleModule } from 'primeng/ripple';

import { InputTextModule } from 'primeng/inputtext';

import { CardModule } from 'primeng/card';

import { MenuModule } from 'primeng/menu';

import { MessagesModule } from 'primeng/messages';

import { MessageModule } from 'primeng/message';

import { SplitterModule } from 'primeng/splitter';

import { MenubarModule } from 'primeng/menubar';

import { AvatarGroupModule } from 'primeng/avatargroup';

import { AvatarModule } from 'primeng/avatar';

import { SidebarModule } from 'primeng/sidebar';

import { PanelMenuModule } from 'primeng/panelmenu';

import { ChartModule } from 'primeng/chart';

import { ToolbarModule } from 'primeng/toolbar';

import { SplitButtonModule } from 'primeng/splitbutton';

import { DialogModule } from 'primeng/dialog';

import { ConfirmPopupModule } from 'primeng/confirmpopup';

import { ToastModule } from 'primeng/toast';

import { TreeModule } from 'primeng/tree';

import { SharedModule } from 'primeng/api';

import { TableModule } from 'primeng/table';

import { KeyFilterModule } from 'primeng/keyfilter';

import { DropdownModule } from 'primeng/dropdown';

import { CalendarModule } from 'primeng/calendar';

import { PanelModule } from 'primeng/panel';

import { InputNumberModule } from 'primeng/inputnumber';

import { CarouselModule } from 'primeng/carousel';

import { FieldsetModule } from 'primeng/fieldset';

import { FileUploadModule } from 'primeng/fileupload';

import { InputTextareaModule } from 'primeng/inputtextarea';

import { AutoCompleteModule } from 'primeng/autocomplete';

import { SlideMenuModule } from 'primeng/slidemenu';

import { ConfirmDialogModule } from 'primeng/confirmdialog';

import { LandingComponent } from './landing/landing.component';

import { DynamicDialogConfig, DynamicDialogRef } from 'primeng/dynamicdialog';

import { MenuComponent } from './menu/menu.component';

import { BadgeModule } from 'primeng/badge';

import { DataViewModule } from 'primeng/dataview';

import { AccordionModule } from 'primeng/accordion';

import { MultiSelectModule } from 'primeng/multiselect';

import { StyleClassModule } from 'primeng/styleclass';

import { OverlayPanelModule } from 'primeng/overlaypanel';

import { TabMenuModule } from 'primeng/tabmenu';

import { TabViewModule } from 'primeng/tabview';

import { ScrollTopModule } from 'primeng/scrolltop';

import { StepsModule } from 'primeng/steps';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { AdminRoutingModule } from 'src/app/main/admin/admin-routing.module';

@NgModule({

  declarations: [

    PrivateLayoutComponent,

    LandingComponent,

    MenuComponent,

  ],

  imports: [

    CommonModule,

    ReactiveFormsModule,

    DataViewModule,

    DashboardRoutingModule,

    PrivateLayoutRoutingModule,

    AdminRoutingModule,

    AccordionModule,

    HttpClientModule,

    FormsModule,

    StepsModule,

    TabMenuModule,

    TabViewModule,

    ScrollTopModule,

    BrowserModule,

    BrowserAnimationsModule,

    MultiSelectModule,

    StyleClassModule,

    DividerModule,

    ButtonModule,

    CheckboxModule,

    RadioButtonModule,

    RippleModule,

    InputTextModule,

    OverlayPanelModule,

    CardModule,

    MenuModule,

    MessagesModule,

    MessageModule,

    SplitterModule,

    MenubarModule,

    AvatarGroupModule,

    AvatarModule,

    SidebarModule,

    PanelMenuModule,

    ChartModule,

    ToolbarModule,

    SplitButtonModule,

    DialogModule,

    ConfirmPopupModule,

    ToastModule,

    TreeModule,

    TableModule,

    KeyFilterModule,

    DropdownModule,

    CalendarModule,

    PanelModule,

    AutoCompleteModule,

    InputNumberModule,

    CarouselModule,

    SlideMenuModule,

    FieldsetModule,

    FileUploadModule,

    InputTextareaModule,

    ConfirmDialogModule,

    BadgeModule

  ],

  providers: [DynamicDialogRef, DynamicDialogConfig]

})

export class PrivateLayoutModule { }

**Rutas en "private-layout-routing.module.ts"**

Una vez que hemos incorporado estos componentes y módulos en nuestro módulo, el siguiente paso es crear las rutas correspondientes en el archivo "private-layout-routing.module.ts". Esta tarea se aborda de manera similar a la anterior, y la estructura se muestra a continuación:

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { ValidateGuard } from 'src/app/core/guards/validate.guard';

import { LandingComponent } from './landing/landing.component';

import { PrivateLayoutComponent } from './private-layout.component';

const routes: Routes = [

  {

    path: '',

    component: PrivateLayoutComponent,

    canActivate: [ValidateGuard],

    canLoad:[ValidateGuard],

    children: [

      {

        path: 'landing',

        component: LandingComponent

      },

      {

        path: 'dashboard',

        loadChildren: () => import('../../main/dashboard/dashboard.module').then(m => m.DashboardModule),

      },

      {

        path: 'administrador',

        loadChildren: () => import('../../main/admin/admin.module').then(m => m.AdminModule),

      },

      {

        path: 'solicitudes',

        loadChildren: () => import('../../main/solicitudes/solicitudes.module').then(m => m.SolicitudesModule),

      },

      {

        path: 'usuarios',

        loadChildren: () => import('../../main/usuarios/usuarios.module').then(m => m.UsuariosModule),

      },

    ]

  }

];

@NgModule({

  imports: [RouterModule.forChild(routes)],

  exports: [RouterModule]

})

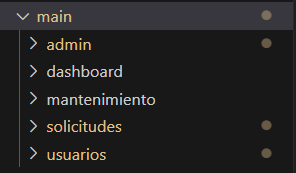
export class PrivateLayoutRoutingModule { }

Este archivo define las rutas que estarán disponibles para los usuarios con acceso privado. Cada ruta está asociada a un componente específico y, en algunos casos, a módulos adicionales que se cargarán según la necesidad. A través de este enfoque, creamos una ruta clara y definida para la navegación dentro de nuestra área privada.

### Generando Componentes en "main"

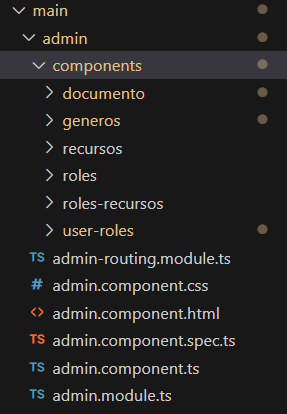
Como es evidente, se han añadido componentes que no están ubicados en el espacio del "private-layout". Para corregir esta discrepancia y aportar un método organizado a nuestra estructura, es necesario generar estos componentes en la ruta "src > app > main". Esta acción genera un método de organización más coherente en nuestro proyecto.

En la carpeta "main" que reside dentro de la carpeta "app", se procede a crear los componentes necesarios. Estos componentes clave, que incluyen "admin", "dashboard", "mantenimiento", "solicitudes" y "usuarios", se crean de la siguiente manera:



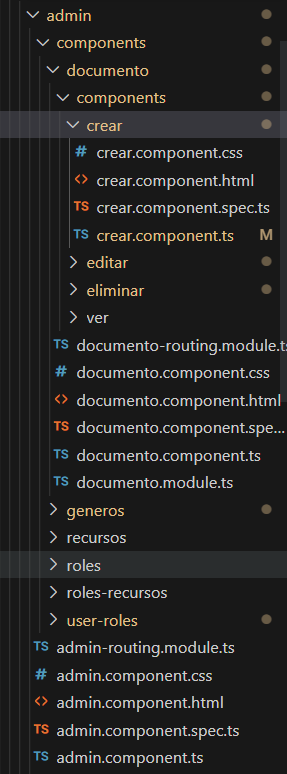
### Construyendo la Estructura Interna de los Componentes

Dentro de cada uno de los componentes que hemos creado, tenemos que generar algunos archivos esenciales para su funcionamiento. Estos archivos son el "module.ts", "routing.module.ts" y una carpeta llamada "components". Dentro de esta última carpeta, alojaremos todos los subcomponentes que serán necesarios para respaldar cada uno de nuestros componentes principales. El proceso se lleva a cabo de la siguiente manera:



### Estableciendo la Base para los Subcomponentes.

En el interior de cada uno de los subcomponentes que hemos creado, debemos seguir una estructura similar. También debemos generar los archivos "module.ts" y "routing.module.ts". Además, creamos una carpeta llamada "components" donde organizaremos los componentes necesarios para llevar a cabo las operaciones CRUD (CREATE, READ, UPDATE y DELETE) en relación con nuestros subcomponentes. Aquí hay una vista clara de la estructura del mismo:



Es importante notar que hemos incluido archivos de enrutamiento dentro de los subcomponentes. Esta práctica se lleva a cabo con el propósito de dirigir de manera efectiva hacia las operaciones CRUD de cada subcomponente. Al adoptar esta estrategia, estamos dotando a nuestro proyecto de una organización coherente y facilitando la navegación entre las diferentes partes de la aplicación.

### Integrando Lógica en el Diseño Privado

Ahora, al adentrarnos en el archivo "private-layout.component.ts", encontramos la esencia de la lógica necesaria para nuestro diseño privado. En este escenario, estamos interconectando servicios y recursos que aún no están presentes, lo que nos lleva a la importancia y creación estos servicios en nuestra carpeta "services".

import { MenuItem, MessageService, PrimeNGConfig } from 'primeng/api';

import { Component, OnInit } from '@angular/core';

import { forkJoin, Observable } from 'rxjs';

import { valorReloj, XsegundoService } from 'src/app/core/services/reloj/Xsegundo.service';

import { listaMenuI } from 'src/app/models/menu';

import { AuthService } from 'src/app/core/services/auth/auth.service';

import { UserService } from 'src/app/core/services/usuarios/user.service';

import { createMenu, listMenu } from 'src/app/consts/menu';

import { DialogService } from 'primeng/dynamicdialog';

import { Router } from '@angular/router';

import { Person, GenderType, DocumentType, User } from 'src/app/models/person';

interface menu {

  label: string,

  data?: string

  expandedIcon: string,

  collapsedIcon: string,

  children?: any[]

}

@Component({

  selector: 'app-private-layout',

  templateUrl: './private-layout.component.html',

  styleUrls: ['./private-layout.component.css'],

  providers: [DialogService]

})

export class PrivateLayoutComponent implements OnInit {

  display = false;

  items: MenuItem[] = [];

  items2: MenuItem[] = [];

  public genderTypes: GenderType[] = [];

  public documentTypes: DocumentType[] = [];

  userDataLoaded: boolean = false;

  public isLoggedIn = false;

  public menu1: listaMenuI[] = [];

  public algo: listaMenuI[] = [];

  public publicMenu: listaMenuI[] = [];

  public privateMenu: listaMenuI[] = [];

  public profileImageUrl = 'assets/demo1.png';

  public ImageUrl = 'assets/demo1.png';

  public nombre: string = 'Demo';

  public subcribe: any;

  public token: string | null = null;

  public displayMaximizable: boolean = true

  public mostrar: boolean = false;

  public username: string | undefined = undefined

  public password: string | undefined = undefined

  public files1: menu[] = []

  datos$: Observable<valorReloj> = this.segundo.getInfoReloj();

  hora?: number = 0;

  minutos?: string = '';

  dia?: string = '';

  fecha?: string = '';

  ampm?: string = '';

  segundos?: string = '';

  public ref1: any;

  public image3: string = 'assets/user.png'

  public image2: string = 'assets/demo.png'

  public Dialog: boolean = false

  public Dialog2: boolean = false

  private UserId: number = 0

  public mensaje: boolean = false

  enFormacion: boolean = false;

  public users: User[] = [];

  public usuarioCopy: Person | null = null;

  usuarioId: number | undefined;

  public usuario: Person = {

    id: 0,

    document\_type: 0,

    gender\_type: 0,

    user: 0,

    createdAt: '',

    updateAt: '',

    name: '',

    surname: '',

    identification: '',

    address: '',

    nationality: '',

    date\_of\_birth: '',

    phone: '',

    status: true,

  };

  constructor(

    private messageService: MessageService,

    private primengConfig: PrimeNGConfig,

    private authService: AuthService,

    private userService: UserService,

    private router: Router,

    private segundo: XsegundoService,

    public dialogService: DialogService,

  ) { }

  displayDialog: boolean = false;

  imageSource: string = 'assets/PQR.png';

  displayDialog2: boolean = false;

  avataresPredefinidos = [

    'assets/avatars/avataaars-1.png',

    'assets/avatars/avataaars-2.png',

    'assets/avatars/avataaars-3.png',

    'assets/avatars/avataaars-4.png',

    'assets/avatars/avataaars-5.png',

    'assets/avatars/avataaars-6.png',

    'assets/avatars/avataaars-7.png',

    'assets/avatars/avataaars-8.png',

    'assets/avatars/avataaars-9.png',

    'assets/avatars/avataaars-10.png',

  ];

  selectedAvatar: string = '';

  ngOnInit() {

    this.usuarioId = this.authService.getUserId();

    if (this.usuarioId !== undefined) {

      console.log("Esta es la ID del usuario Loggeado: ", this.usuarioId)

    } else {

      console.error('El usuario no tiene un ID válido.');

    }

    this.datos$ = this.segundo.getInfoReloj();

    this.datos$.subscribe(x => {

      this.hora = x.hora;

      this.minutos = x.minutos;

      this.dia = x.diadesemana;

      this.fecha = x.diaymes;

      this.ampm = x.ampm;

      this.segundos = x.segundo;

    });

    this.files1 =

      [

        {

          "label": "Pictures",

          "data": "Pictures Folder",

          "expandedIcon": "pi pi-folder-open",

          "collapsedIcon": "pi pi-folder",

          "children": [

            { "label": "barcelona.jpg", "icon": "pi pi-image", "data": "Barcelona Photo" },

            { "label": "logo.jpg", "icon": "pi pi-image", "data": "PrimeFaces Logo" },

            { "label": "primeui.png", "icon": "pi pi-image", "data": "PrimeUI Logo" }]

        },

      ]

    console.log('valores')

    this.verificar()

    this.primengConfig.ripple = true;

    this.items2 = [

      {

        label: 'File',

        items: [{

          label: 'New',

          icon: 'pi pi-fw pi-plus',

          items: [

            { label: 'Project' },

            { label: 'Other' },

          ]

        },

        { label: 'Open' },

        { label: 'Quit' }

        ]

      },

      {

        label: 'Edit',

        icon: 'pi pi-fw pi-pencil',

        items: [

          { label: 'Delete', icon: 'pi pi-fw pi-trash' },

          { label: 'Refresh', icon: 'pi pi-fw pi-refresh' }

        ]

      }

    ];

    this.items = [

      {

        label: 'Avatar', icon: 'pi pi-user', command: () => {

          this.abrirEditarImagen();

        }

      },

      { separator: true },

      {

        label: 'Editar perfil', icon: 'pi pi-cog', command: () => {

          this.abrirEditarPerfil();

        }

      },

      { separator: true },

      {

        label: 'Cerrar Sesion', icon: 'pi pi-power-off', command: () => {

          this.showConfirm();

        }

      },

      { separator: true },

    ];

  }

  openDialog() {

    this.displayDialog = true;

  }

  closeDialog() {

    this.displayDialog = false;

  }

  ocultarMenu(boolean: boolean) {

    // this.display=boolean

  }

  setLogin(value: boolean): void {

    this.authService.setLogin(value);

  }

  openDialogResgister(event: Event) {

    event.preventDefault();

  }

  public confirm() {

    this.showSuccess()

    this.displayMaximizable = false

  }

  showSuccess() {

    this.messageService.add({ severity: 'success', summary: 'Success', detail: 'Ingreso exitoso' });

  }

  save(id: string) { }

  openDialogLogin(event: Event) {

    event.preventDefault();

    this.displayMaximizable = true

  }

  cerrarSesion() {

    this.setLogin(false)

    this.authService.logout()

    this.ngOnInit()

    this.router.navigateByUrl('/login')

  }

  showConfirm() {

    this.Dialog = true;

  }

  hideDialog() {

    this.Dialog = false;

  }

  abrirEditarImagen() {

    console.log('Abriendo diálogo de edición de imagen');

    this.displayDialog2 = true;

  }

  cerrarEditarImagen() {

    this.displayDialog2 = false;

  }

  abrirEditarPerfil() {

    console.log('Abriendo diálogo de edición de perfil');

    this.loadDropdownData();

    this.loadUserData();

    this.Dialog2 = true;

    this.usuarioCopy = { ...this.usuario };

  }

  cerrarEditarPerfil() {

    this.Dialog2 = false;

  }

  public verificar() {

    var token: string | null = localStorage.getItem('token');

    var user: string | null = localStorage.getItem('user');

    var menu: string | null = localStorage.getItem('menu');

    console.log(token, menu, 'valores')

    if (token != null && menu != null && user != null) {

      this.showSuccess()

      let userObjeto: any = JSON.parse(user);

      let menuObjeto: any = JSON.parse(menu);

      console.log(menuObjeto)

      console.log("SI MENU")

      this.privateMenu = createMenu(menuObjeto) as any;

      this.menu1 = this.privateMenu;

      this.nombre = userObjeto.name;

      this.isLoggedIn = true

      this.setLogin(true)

    } else {

      console.log("NO MENU")

      this.isLoggedIn = false

      this.setLogin(false)

      this.menu1 = [];

      this.router.navigateByUrl('/login');

    }

  }

  loadUserData(): void {

    if (!this.userDataLoaded && this.usuarioId !== undefined) {

      this.userService.getPersonsByUserId(this.usuarioId).subscribe(

        persons => {

          if (persons.length > 0) {

            this.usuario = persons[0];

          }

          this.userDataLoaded = true;

          this.loadDropdownData();

        },

        error => {

          console.error('Error loading user data:', error);

        }

      );

    }

  }

  loadDropdownData() {

    this.userService.getDocumentTypes().subscribe(

      response => {

        if (response) {

          this.documentTypes = response;

          console.log("documentos", this.documentTypes);

        }

      },

      error => {

        console.error('error al cargar los documentos:', error);

      }

    );

    this.userService.getGenderTypes().subscribe(

      response => {

        if (response) {

          this.genderTypes = response;

          console.log("generos", this.genderTypes);

        }

      },

      error => {

        console.error('error al cargar los generos:', error);

      }

    );

  }

  guardarDatos() {

    console.log('Guardando datos del usuario');

    const usuarioActualizado: Person = {

      id: this.usuario.id,

      document\_type: this.usuario.document\_type,

      gender\_type: this.usuario.gender\_type,

      user: this.usuario.user,

      createdAt: this.usuario.createdAt,

      updateAt: this.usuario.updateAt,

      name: this.usuario.name,

      surname: this.usuario.surname,

      identification: this.usuario.identification,

      address: this.usuario.address,

      nationality: this.usuario.nationality,

      date\_of\_birth: this.usuario.date\_of\_birth,

      phone: this.usuario.phone,

      status: this.usuario.status !== undefined ? this.usuario.status : false

    };

    console.log('Datos a guardar:', usuarioActualizado);

    this.userService.editarUsuario(usuarioActualizado).subscribe(

      (response) => {

        console.log('Datos del usuario guardados exitosamente', response);

        this.Dialog2 = false;

      },

      (error) => {

        console.error('Error al guardar los datos del usuario', error);

      }

    );

  }

  seleccionarAvatar(avatar: string) {

    this.selectedAvatar = avatar;

  }

  guardarCambios() {

    if (this.selectedAvatar) {

      const userId = 1;

      this.userService.actualizarImangeUsuario(userId, this.selectedAvatar)

        .subscribe(

          (response) => {

            console.log('Imagen de perfil actualizada con éxito', response);

            this.cerrarEditarImagen();

          },

          (error) => {

            console.error('Error al actualizar la imagen de perfil', error);

          }

        );

    }

  }

}

Observamos que en este código se utilizan servicios que aún no están disponibles. Para solucionar esto, es esencial crear los servicios correspondientes dentro de la carpeta "services".

### Generando servicios para el componente privado

Nos dirigimos a nuestra carpeta services y dentro de la carpeta “reloj” creamos el servicio “Xsegndo” con el comanod **ng generate service Xsegundo.**

Dentro de este tenemos el siguiente código:

import { Injectable } from '@angular/core';

import { map, Observable, shareReplay, Subject, timer } from 'rxjs';

const KEY\_ANUNCIOS = 'anuncios';

export interface valorReloj {

  hora?: number;

  minutos?: string;

  ampm?: string;

  diadesemana?: string;

  diaymes?: string;

  segundo?: string;

}

@Injectable({

  providedIn: 'root'

})

export class XsegundoService {

  clock: Observable <Date>;

  infofecha$ = new Subject<valorReloj>();

  vr: valorReloj={};

  ampm: string='';

  hours: number=0;

  minute: string='';

  weekday: string='';

  months: string='';

  verificar:any

  constructor(

  ) {

    this.clock = timer(0,1000).pipe(map(t => new Date()),shareReplay(1));

   }

   getInfoReloj(): Observable<valorReloj>{

     this.clock.subscribe(t => {

      this.hours = t.getHours() % 12;

      this.hours = this.hours ? this.hours : 12;

       this.vr = {

         hora: this.hours,

         minutos: (t.getMinutes() < 10) ? '0' + t.getMinutes() : t.getMinutes().toString(),

         ampm: t.getHours() > 11 ? 'PM' : 'AM',

         diaymes: t.toLocaleString('es-MX', { day: '2-digit', month: 'long' }).replace('.', '').replace('-', ' '),

         diadesemana: t.toLocaleString('es-MX', { weekday: 'long' }).replace('.', ''),

         segundo: t.getSeconds() < 10 ? '0' + t.getSeconds() : t.getSeconds().toString()

       }

       this.infofecha$.next(this.vr);

     });

     return this.infofecha$.asObservable();

   }

}

En este servicio, se realizan las siguientes acciones:

* Se importan las clases y operadores necesarios de **RxJS** y se define la interfaz valorReloj que define la estructura de la información de tiempo y fecha.
* Se marca el servicio como "injectable" y se proporciona a nivel de raíz (root) para que esté disponible en toda la aplicación.
* Se declara el observable clock, que utiliza el operador timer para emitir un valor (una instancia de Date) cada segundo. El operador map transforma este valor en la hora actual.
* El uso de **shareReplay** permite que los suscriptores futuros reciban el último valor emitido, evitando la ejecución redundante de la lógica de obtención de tiempo y fecha.
* El método **getInfoReloj** suscribe la función al observable clock y, en cada emisión de tiempo, se realiza la lógica para obtener la hora, los minutos, el período AM/PM, el día del mes, el nombre del día de la semana y los segundos.
* La información obtenida se almacena en el objeto vr (valorReloj) y se notifica a través del sujeto **infofecha$,** que es un observable especial que puede ser observado por otros componentes.
* El método **getInfoReloj** devuelve el observable **infofecha$** para que los componentes puedan suscribirse y recibir actualizaciones de tiempo y fecha.

Este servicio "Xsegundo" proporciona un flujo constante de información de tiempo y fecha a través de un observable, lo que permite a los componentes de la aplicación mantenerse actualizados con la información de tiempo y reaccionar en consecuencia.

**Método getUserId(): number | undefined**

Este método recupera el ID del usuario activo a partir de la información almacenada en el almacenamiento local (localStorage) de la aplicación. Utiliza el ID almacenado en el objeto UserI para proporcionar una identificación única del usuario.

getUserId(): number | undefined {

    const userJson = localStorage.getItem(KEY\_USER);

    if (userJson) {

      const user: UserI = JSON.parse(userJson);

      return user.id;

    }

    return undefined;

  }

Con este código se hace lo siguiente:

* Dentro del servicio AuthService, se define el método getUserId(), que se utiliza para obtener el ID del usuario actualmente autenticado.
* Se obtiene la representación JSON del usuario desde el almacenamiento local utilizando la clave KEY\_USER.
* Si se encuentra una representación JSON válida del usuario en el almacenamiento local (userJson), se analiza (parse) el JSON para convertirlo en un objeto UserI, asumiendo que UserI es la interfaz que define la estructura del objeto de usuario.
* Se devuelve el ID del usuario (user.id) si se encontró información de usuario en el almacenamiento local.
* Si no se encuentra información de usuario, el método devuelve undefined para indicar que no se pudo obtener el ID del usuario.

Este método es útil para recuperar el ID del usuario actualmente autenticado y se puede utilizar en diferentes partes de la aplicación donde se necesite esta información, como en el componente **PrivateLayoutComponent.**

**Método setLogin(value: boolean): void**

Este método permite cambiar el estado de inicio de sesión de un usuario. Al proporcionar un valor booleano value, el método actualiza internamente el estado de autenticación. Esto es fundamental para controlar las áreas y funciones de la aplicación a las que se puede acceder según el estado de inicio de sesión del usuario.

setLogin(value: boolean): void {

    this.\_isLoggedIn = value;

    this.isLoggedIn$.next(value);

  }

Dentro del servicio AuthService, se define el método setLogin(value: boolean), que se utiliza para establecer el estado de inicio de sesión, lo cual hace lo siguiente:

* La propiedad privada \_isLoggedIn se actualiza con el valor booleano proporcionado (value), lo que refleja si el usuario está o no autenticado.
* Se crea un sujeto observable isLoggedIn$ que se utilizará para notificar a los observadores (como componentes) sobre los cambios en el estado de inicio de sesión.
* En el método setLogin(), después de actualizar \_isLoggedIn, se utiliza el método next() en isLoggedIn$ para emitir el nuevo valor a todos los observadores registrados.

Este método es útil para controlar y notificar cambios en el estado de inicio de sesión en toda la aplicación. Cuando se llama a **setLogin(),** los componentes que estén suscritos a **isLoggedIn$** recibirán la notificación del cambio en el estado de inicio de sesión y podrán reaccionar en consecuencia, actualizando la interfaz de usuario y tomando otras acciones relevantes, permitiendo de esta forma que el usuario este activo o inactivo dentro del sistema y detecte cuando se realice el cierre de sección de un usuario.

**Recuperación de Datos de Usuario por ID de Usuario**

Este método habilita la recuperación de datos de usuario basados en su ID. Cuando se proporciona un valor userId, el método realiza una solicitud HTTP para obtener la información relacionada con el usuario asociado. Los datos obtenidos se filtran y se devuelven como un array de objetos Person. El método emite un observable que proporciona acceso a los datos de usuario recuperados.

 getPersonsByUserId(userId: number): Observable<Person[]> {

    return this.http.get<Person\_request>(`${this.base\_personas}`).pipe(

      map(response => {

        if (response && response.ok) {

          const filteredPersons = response.data.filter(person => person.user && person.user === userId);

          return filteredPersons;

        } else {

          throw new Error(response && response.message ? response.message : "Unexpected response");

        }

      }),

      catchError(error => {

        console.error("Error in getPersonsByUserId:", error);

        throw error;

      })

    );

  }

Se define el método getPersonsByUserId(userId: number), que recupera un array de objetos Person asociados a un ID de usuario específico.

* El método realiza una solicitud GET a la API utilizando HttpClient. La URL de la API se compone utilizando la URL base.
* La respuesta de la API se procesa usando el operador map. Si la respuesta es exitosa (ok es verdadero), se filtran las personas cuyo campo user coincide con el ID del usuario proporcionado.
* Si la respuesta no es exitosa o no es como se esperaba, se lanza un error con un mensaje descriptivo. Esto es útil para manejar situaciones inesperadas y proporcionar información clara sobre el error.
* Se utiliza el operador catchError para manejar errores y registrarlos en la consola. También se utiliza throwError para propagar el error para que los componentes que llaman a este método puedan manejarlo adecuadamente.
* La estructura del código es clara y bien organizada, lo que facilita la comprensión y el mantenimiento.

Este método es esencial para obtener información de personas asociadas a un usuario específico y demuestra cómo se manejan las respuestas de la API y los errores en un servicio Angular.

Después de establecer los fundamentos mencionados anteriormente, el proceso continúa mediante la creación de dos servicios esenciales: getDocumentTypes y getGenderTypes. Estos servicios desempeñan un papel importante al permitir la recuperación de información clave relacionada con los tipos de documento y géneros de los usuarios dentro del contexto de la aplicación.

getDocumentTypes(): Observable<any[]> {

    return this.http.get<any>(`${this.base\_documentos}`).pipe(

      map(response => response.data)

    );

  }

 getGenderTypes(): Observable<any[]> {

    return this.http.get<any>(`${this.base\_gender}`).pipe(

      map(response => response.data)

    );

Dentro de estos métodos tenemos las siguientes caracteristicas:

* Cada método utiliza una estructura similar y coherente. Realizan solicitudes HTTP GET a las URLs correspondientes y utilizan el operador map para extraer y devolver la propiedad data de la respuesta.
* La URL base para las API de tipos de documento y géneros se define claramente en las propiedades privadas base\_documentos y base\_gender. Esto centraliza la información y facilita los cambios futuros en las URLs.
* Los métodos devuelven observables que emiten arrays de objetos representando los tipos de documento y géneros disponibles.
* La estructura y el formato del código son consistentes con las mejores prácticas de Angular y facilitan la legibilidad y el mantenimiento.

Estos métodos son esenciales para obtener la información de tipos de documento y géneros disponibles, lo que es útil para crear interfaces de usuario más ricas y personalizadas.

**Método editarUsuario(usuario: Person): Observable<any>**

Este método acepta un objeto usuario que encapsula los datos del perfil a editar o crear. Dependiendo del valor de la propiedad id del objeto usuario, el método decide si debe realizar una solicitud HTTP PUT para editar un perfil existente o una solicitud HTTP POST para crear un nuevo perfil. La función devuelve un observable que emite la respuesta de la solicitud HTTP, lo que permite controlar y reaccionar según el resultado de la operación de edición o creación.

 editarUsuario(usuario: Person): Observable<any> {

    if (usuario.id !== 0) {

      return this.http.put(`${this.base\_personas\_editar}${usuario.id}/`, usuario);

    } else {

      return this.http.post(this.base\_personas\_crear, usuario);

    }

  }

**Actualización de la Imagen de Usuario.**

Dentro del servicio userservice, se desarrolla una función esencial denominada actualizarImagenUsuario. Este servicio se encarga de actualizar la imagen de perfil de un usuario específico. La actualización de la imagen es crucial para proporcionar una representación visual personalizada y distintiva de cada usuario en la aplicación.

**Método actualizarImagenUsuario.**

Este método permite actualizar la imagen de perfil de un usuario específico. Toma como entrada el ID del usuario cuya imagen se actualizará y la ruta de acceso a la nueva imagen de perfil (avatar). Utilizando estos datos, el servicio construye la URL de la API correspondiente para llevar a cabo la actualización.

actualizarImangeUsuario(userId: number, avatar: string): Observable<any> {

    const apiUrl = `${this.base\_editar\_user}${userId}/`;

    const userData = {

      id: userId,

      avatar: avatar

    };

    return this.http.put(apiUrl, userData);

  }

Gracias a la implementación de estos servicios, el componente "private-layout.component.ts" será capaz de operar de manera óptima y eficaz, desempeñando una serie de funciones esenciales, tales como:

* **Obtención de Usuario Logueado:** El método de autenticación ("getUserId") permite identificar al usuario que ha iniciado sesión, proporcionando información crítica sobre su estado de autenticación. Esto garantiza que el componente pueda adaptarse a las necesidades y permisos específicos de cada usuario.
* **Obtención de Datos de Usuario:** Mediante el servicio "getPersonsByUserId" del "user.service", el componente puede acceder a los detalles completos de un usuario en particular. Esto es fundamental para personalizar la experiencia del usuario y presentar información relevante en su perfil.
* **Cierre de Sesión:** Gracias al método "logout" implementado en "private-layout.component.ts", los usuarios tienen la capacidad de cerrar sesión de manera controlada. Esto asegura la seguridad de la aplicación y permite que los usuarios finalicen su interacción de forma adecuada.
* **Actualización de Datos de Usuario:** Los métodos "getDocumentTypes", “getGenderTypes” y “editarUsuario” proporciona la funcionalidad para actualizar y mantener la información de perfil de un usuario. Esto asegura que los datos de usuario, como nombre, dirección y detalles de contacto, se mantengan precisos y actualizados.
* **Actualización de Imagen de Usuario:** Utilizando el método "actualizarImagenUsuario", el componente puede permitir que los usuarios cambien su imagen de perfil. Esta característica ofrece una manera visualmente atractiva de personalizar la representación del usuario en la aplicación.

Estas funciones en conjunto enriquecen la experiencia del usuario y aseguran que el componente "private-layout.component.ts" opere de manera fluida y coherente, proporcionando las herramientas necesarias para gestionar y actualizar la información de los usuarios de manera eficiente.

### Interfaz private-layout.component.html

El código HTML proporciona la interfaz visual para el componente "private-layout.component.ts". Esta interfaz está diseñada para brindar una experiencia de usuario efectiva y agradable. Esta interfaz tiene las siguientes características.

**Encabezado de la Aplicación:**

* Contiene el logo de la aplicación y un botón que muestra un menú lateral al hacer clic.
* Incluye un reloj que muestra la fecha y hora actual junto con un icono de notificaciones.
* Presenta un botón para mostrar un menú desplegable con opciones de perfil y avatar.

**Menú Lateral:**

* Se despliega al hacer clic en el botón de menú en el encabezado.
* Contiene el logo y título de la aplicación, así como un botón para cerrar el menú.
* Muestra el nombre de usuario en la parte superior y ofrece opciones de navegación en forma de menú desplegable.

**Contenido Principal:**

* Contiene el enrutador de la aplicación para cargar diferentes componentes según la navegación del usuario.
* Se encuentra bajo el menú lateral y ocupa la mayor parte de la pantalla.

**Pie de Página:**

* Incluye información de derechos de autor y reconocimiento.
* Se encuentra al final del contenido principal y es una parte estática de la página.

**Diálogos de Confirmación:**

* Se utilizan para confirmar acciones importantes, como cerrar sesión.
* Presenta un ícono de advertencia y un mensaje de confirmación.
* Ofrece botones para confirmar o cancelar la acción.

**Estilos y Clases:**

* Se aplican estilos a los elementos para mejorar la apariencia y la experiencia de usuario.
* Se utilizan clases como p-button, p-ripple, y otros estilos específicos de PrimeNG para mejorar la interfaz.

**Interactividad:**

* Se utilizan eventos como click para activar acciones, como mostrar o cerrar menús y diálogos.
* Se utiliza la interpolación “{{ }}” para mostrar datos dinámicos, como el nombre de usuario y la hora actual.

**Elementos PrimeNG:**

* Se utilizan componentes de PrimeNG, como p-button, p-sidebar, p-menu, p-avatar, p-dialog, y otros, para agregar funcionalidad y estilo a la interfaz.

En conjunto, esta interfaz está diseñada para proporcionar una experiencia de usuario atractiva y funcional al componente "private-layout.component.ts", con características como navegación intuitiva, visualización de la hora actual, menús desplegables y opciones de perfil personalizables, este código se representa a continuación:

<ng-template [ngIf]="isLoggedIn === true">

  <div class="grid main-header" [style]="{'background':' #fa6357',

  'color':'#fff','font-weight':'bold','margin':'0',

  'border':'1px solid gray','border-radius':'0px'}">

    <div class="col-12 sm:col-2 md:col-3 lg:col-3 xl:col-3">

      <p-button styleClass="p-button-danger p-button-outlined" (click)="display = true" class="mr-3">

        <i class="pi pi-align-justify" style="font-size: 1.5rem; margin-top: 5px;color:whitesmoke"></i>

        <span class="ml-2 font-bold" style="color:whitesmoke">GRECIUG</span>

      </p-button>

    </div>

    <div class="col-12 sm:col-2 md:col-3 lg:col-4 xl:col-5 "></div>

    <div class="col-12 sm:col-7 md:col-6 lg:col-5 xl:col-4">

      <div class="grid">

        <div class="col-12 sm:col-12 md:col-1 lg:col-1 xl:col-1 bell">

          <i class="pi pi-bell mt-1 mr-4" style="font-size:2rem"></i>

        </div>

        <div class="col-12 sm:col-12 md:col-9 lg:col-9 xl:col-9 clock">

          <button pButton pRipple type="button" [disabled]="false" class="p-button-danger p-button-outlined"

            style="padding: 7px;">

            <div class="clock" style="color:whitesmoke">

              {{dia | titlecase}} {{fecha | titlecase}} [<span class="mishoras">{{hora}}:{{minutos}}:<span

                  class="segundos">{{segundos}}</span></span><span class="ampm">{{ampm}}</span>]

            </div>

          </button>

        </div>

        <div class="col-12 sm:col-12 md:col-2 lg:col-2 xl:col-2 avatar">

          <button type="button" pButton pRipple class="p-button-rounded p-button-text boton-perfil" icon="pi pi-plus"

            (click)="menu.toggle($event)">

            <p-avatar image="{{image3}}" styleClass="ml-3" [ngStyle]="{'height': '99%'}" shape="circle"></p-avatar>

          </button>

        </div>

      </div>

      <p-menu #menu [popup]="true" [model]="items" [style]="{'width':'200px','margin-top':'5px'}"></p-menu>

      <div #menu2 class="box hidden" header="Anuncios" style="display: inline-block;margin:0px">

      </div>

    </div>

  </div>

  <div class="container margen">

    <p-sidebar [(visible)]="display" [style]="{'width':'345px','min-height': '88vh'}" [showCloseIcon]="false"

      [blockScroll]="false">

      <br>

      <br>

      <br>

      <div class="grid">

        <div class="col-11">

          <a routerLink="/landing" (click)="display=false">

            <img alt="logo" src="assets/Uniguajira\_logo.png" width="260px" height="92px"

              style="margin-left: 20px ;margin-top: -5px;" styleClass="p-button-outlined" />

          </a>

        </div>

        <div class="col-1">

          <a pripple="" (click)="display=false" class="p-ripple p-element

          cursor-pointer inline-flex align-items-center

            justify-content-center text-white border-2 surface-border

            hover:bg-gray-600 text-600 transition-colors transition-duration-150 border-circle" style="width: 40px; height: 40px;margin-left: -20px;margin-top:22px;

            background-color:  #fa6357;"><i class="pi pi-align-justify text-xl"></i>

            <span class="p-ink"></span>

          </a>

        </div>

      </div>

      <div>

        <hr>

        <h4 class="fondebabutton" style="text-align: center;margin-bottom:-5px">{{nombre}}</h4>

        <hr>

      </div>

      <app-menu [menu]="menu1" (funcion)="ocultarMenu($event)"></app-menu>

      <span style="margin:5px"></span>

    </p-sidebar>

    <div class="content">

      <router-outlet></router-outlet>

      <br>

      <br>

      <hr>

      <footer class="footer" style="padding-top: 20px;">

        <br>

        <br>

        <br>

        <div class="footer\_\_copyright">

          <p style="color:gray;font-size: 1.1rem">© 2023 Desarrollo <b>Universidad de la Guajira</b></p>

        </div>

        <div class="footer\_\_copyright">

          <p style="color:gray;font-size: 1rem">Todos los derechos reservados - <b>Semillero SETEC - RASTOR</b></p>

        </div>

      </footer>

    </div>

    <div>

      <img src="assets/Revista\_banner\_i.jpg" height="200px" width="100%">

    </div>

  </div>

</ng-template>

<div style="display:relative;">

  <p-menu #menu [popup]="true" [model]="items" [style]="{'width':'200px','margin-top':'5px'}"></p-menu>

</div>

<p-dialog [(visible)]="Dialog" [style]="{'width': '40vw'}" [resizable]="false" zindex="10000" [showHeader]="false"

  [closeOnEscape]="false" [closable]="false" [modal]="true">

  <div class="flex flex-column " style="flex: 1">

    <div class="text-center">

      <i class="pi pi-exclamation-triangle" style="font-size: 3rem"></i>

      <h4>Esta seguro de Cerrar la Sesion?</h4>

      <p>Si para continuar</p>

    </div>

  </div>

  <ng-template pTemplate="footer" class="titleCard">

    <div style="text-align: center;">

      <button pButton pRipple label="No" icon="pi pi-times" style="color: rgb(31, 45, 64);" class="p-button-text"

        (click)="hideDialog()"></button>

      <button pButton pRipple label="Si" style="background: #fa6357" class=" p-button-raised p-button-rounded"

        icon="pi pi-check" (click)="cerrarSesion()"></button>

    </div>

  </ng-template>

</p-dialog>

<p-dialog [(visible)]="Dialog2" [style]="{'width': '60vw', 'top': '40px'}" [resizable]="false" zindex="10000"

  [showHeader]="false" [closeOnEscape]="false" [closable]="false" [modal]="true">

  <div class="p-grid p-nogutter p-align-center"

    style="padding: 0px; text-align: center; border: 2px solid #F16257; border-radius: 10px;">

    <div class="p-col">

      <h3 class="p-text-2xl font-weight-bold"><i class="pi pi-pencil"></i>editar perfil</h3>

    </div>

  </div>

  <br>

  <br>

  <section>

    <p-accordion \*ngIf="usuario" [activeIndex]="[0, 1]" [multiple]="true" class="p-accordion-rounded">

      <p-accordionTab header="Datos básicos" [selected]="0">

        <div class="p-grid p-fluid">

          <div class="p-col-12 p-md-6">

            <div class="p-grid">

              <div class="p-col-12 ">

                <label for="nombres">Nombres:</label>

                <input type="text" id="nombres" pInputText class="p-inputtext" [(ngModel)]="usuario.name">

              </div>

              <div class="p-col-12">

                <label for="apellidos">Apellidos:</label>

                <input type="text" id="apellidos" pInputText class="p-inputtext" [(ngModel)]="usuario.surname">

              </div>

            </div>

          </div>

          <div class="p-col-12 p-md-6">

            <div class="p-grid">

              <div class="p-col-12 p-md-6">

                <label for="documentType">Tipo de documento:</label>

                <p-dropdown \*ngIf="documentTypes" id="documentType" [options]="documentTypes" optionLabel="name"

                  [(ngModel)]="usuario.document\_type"></p-dropdown>

              </div>

              <div class="p-col-12">

                <label for="identificacion">Identificación:</label>

                <input type="text" id="identificacion" pInputText class="p-inputtext"

                  [(ngModel)]="usuario.identification">

              </div>

            </div>

          </div>

          <div class="p-col-12">

            <label for="genderType">Género:</label>

            <p-dropdown id="genderType" [options]="genderTypes" optionLabel="name"

              [(ngModel)]="usuario.gender\_type"></p-dropdown>

          </div>

          <div class="p-col-12">

            <label for="direccion">Dirección:</label>

            <input type="text" id="direccion" pInputText class="p-inputtext" [(ngModel)]="usuario.address">

          </div>

          <div class="p-col-12 p-md-6">

            <label for="nacionalidad">Nacionalidad:</label>

            <input type="text" id="nacionalidad" pInputText class="p-inputtext" [(ngModel)]="usuario.nationality">

          </div>

          <div class="p-col-12 p-md-6">

            <label for="fechaNacimiento">Fecha de Nacimiento:</label>

            <input type="date" id="fechaNacimiento" pCalendar class="p-inputtext" [(ngModel)]="usuario.date\_of\_birth">

          </div>

          <div class="p-col-12 p-md-6">

            <label for="telefono">Número de Teléfono:</label>

            <input type="tel" id="telefono" pInputText class="p-inputtext" [(ngModel)]="usuario.phone">

          </div>

        </div>

      </p-accordionTab>

    </p-accordion>

  </section>

  <ng-template pTemplate="footer" class="titleCard">

    <div style="text-align: center;">

      <button pButton pRipple label="Cancelar" style="background: gray" class="p-button-raised p-button-rounded"

        icon="pi pi-times" (click)="cerrarEditarPerfil()"></button>

      <button pButton pRipple label="Guardar datos" style="background: #fa6357" class="p-button-raised p-button-rounded"

        icon="pi pi-check" (click)="guardarDatos()"></button>

    </div>

  </ng-template>

</p-dialog>

<a (click)="openDialog()">

  <img class="float-img" src="assets/PQR.png" width="6%" id="pqr-img">

</a>

<p-dialog [visible]="displayDialog" showEffect="fade" [modal]="true" [style]="{ width: '50%' }">

  <div class="ng-tns-c69-121 p-dialog-content">

    <form novalidate="" class="flex flex-column gap-3 mt-3 ng-untouched ng-pristine ng-valid ng-tns-c69-121">

      <div>

        <label for="cardholder" class="block mb-1 text-color text-base">Cardholder Name</label>

        <span class="p-input-icon-left w-full">

          <i class="pi pi-user"></i>

          <input name="cardholder" type="text" id="cardholder" pinputtext=""

            class="p-inputtext p-component p-element w-full">

        </span>

      </div>

      <div>

        <label for="credit-card" class="block mb-1 text-color text-base">Credit Card Number</label>

        <span class="p-input-icon-left w-full">

          <i class="pi pi-credit-card"></i>

          <input name="cc" type="text" id="credit-card" pinputtext="" placeholder="1234-1234-1234-1234"

            class="p-inputtext p-component p-element w-full">

        </span>

      </div>

      <div class="flex gap-3">

        <div class="w-full">

          <label for="cvc" class="block mb-1 text-color text-base">CVC</label>

          <span class="p-input-icon-left w-full">

            <i class="pi pi-lock"></i>

            <input name="cvc" type="text" id="cvc" pinputtext="" pkeyfilter="int" placeholder="123"

              class="p-inputtext p-component p-element w-full ng-untouched ng-pristine ng-valid">

          </span>

        </div>

        <div class="w-full">

          <label for="expiration" class="block mb-1 text-color text-base">Expiration</label>

          <span class="p-input-icon-left w-full">

            <i class="pi pi-clock"></i>

            <input name="exp" type="text" id="expiration" pinputtext="" placeholder="12/22"

              class="p-inputtext p-component p-element w-full ng-untouched ng-pristine ng-valid">

          </span>

        </div>

      </div>

    </form>

  </div>

  <div class="flex gap-3 justify-content-end border-top-1 surface-border pt-5 ng-star-inserted">

    <button pbutton="" pripple="" label="Cancel" class="p-element p-ripple p-button-text p-button p-component">

      <span class="p-button-label">Cancel</span>

      <span class="p-ink"></span>

    </button>

    <button pbutton="" pripple="" label="Update" class="p-element p-ripple p-button-rounded p-button p-component">

      <span class="p-button-label">Update</span>

      <span class="p-ink"></span>

    </button>

  </div>

</p-dialog>

<p-dialog [visible]="displayDialog2" [modal]="true" [style]="{ width: '50%' }" [showHeader]="false">

  <div style="text-align: center; padding: 20px;">

    <h2>Selecciona tu Avatar</h2>

    <div class="avatar-list">

      <div \*ngFor="let avatar of avataresPredefinidos"

           (click)="seleccionarAvatar(avatar)"

           [ngClass]="{'avatar-item': true, 'selected': selectedAvatar === avatar}">

        <img [src]="avatar" />

      </div>

    </div>

    <div class="button-group">

      <p-button type="button" class="p-button-outlined" (click)="guardarCambios()">Guardar</p-button>

      <p-button type="button" class="p-button-outlined" (click)="cerrarEditarImagen()">Cerrar</p-button>

    </div>

  </div>

</p-dialog>

**Estilos para private-layout.**

Estos estilos CSS se aplican a la interfaz de usuario definida en el código HTML que se ha creado. A continuación, se describen las características de los estilos:

**Estilos Generales:**

* Se define el estilo general del cuerpo de la página, incluyendo margen, relleno y la fuente utilizada.
* Estilos para el Encabezado Principal:
* Se establece la posición fija del encabezado en la parte superior de la página.
* Se define el ancho, la ubicación y el índice z para asegurar que el encabezado esté visible en todo momento.

**Estilos para el Reloj:**

* Se configuran los estilos para mostrar el reloj en la barra de navegación.
* Se establece el color del texto y el tamaño de fuente para resaltar la hora actual.
* Se definen estilos específicos para los elementos de la hora y minutos.

**Estilos para la Barra de Navegación Lateral:**

* Se personaliza el estilo de la barra de desplazamiento en la barra de navegación lateral.

**Estilos para la Imagen Flotante:**

Se establece la posición y el comportamiento de la imagen flotante.

Al hacer hover sobre la imagen, se aplica una transformación para crear un efecto visual.

**Estilos para el Formulario PQR Flotante:**

* Se definen estilos para un formulario flotante en la esquina inferior derecha de la página.
* Se establecen propiedades como posición, tamaño, color de fondo y bordes redondeados.

**Estilos para Títulos Editables:**

* Se configura el estilo de títulos editables, como el título "Editar Perfil" en los diálogos.

**Estilos para Elementos en Pantallas Pequeñas:**

* Se definen estilos específicos para pantallas con un ancho máximo de 768px.
* Se ajusta la disposición y el tamaño de ciertos elementos para mejorar la experiencia en dispositivos móviles.

**Estilos CSS para la Selección de Avatares:**

* Se aplican estilos a la lista de avatares y a los elementos individuales.
* Al seleccionar un avatar, se resalta con un borde y se aplica un efecto de escala.

**Estilos para Grupos de Botones:**

* Se establecen estilos para agrupar botones, como los botones "Guardar" y "Cerrar" en los diálogos.

En conjunto, estos estilos CSS mejoran la apariencia y la interacción de la interfaz de usuario del componente "private-layout.component.ts", asegurando una experiencia de usuario coherente y agradable en diferentes tamaños de pantalla y dispositivos.

/\* Estilos generales \*/

body {

  margin: 0;

  padding: 0;

  font-family: Arial, sans-serif;

}

/\* ... Otros estilos generales ... \*/

/\* Estilos para el encabezado principal \*/

.main-header {

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  z-index: 99999;

}

/\* Estilos para el reloj \*/

.clock {

  display: flex;

  margin: 0;

  display: inline;

}

.clock p {

  color: #FBC02D;

  font-size: 1.5rem;

}

.hours {

  grid-column-start: 1;

  grid-column-end: 3;

  color: #FBC02D;

}

.horas {

  display: flex;

  margin: 0;

  font-size: 0.9rem;

  flex-wrap: nowrap;

  flex-direction: column-reverse;

  justify-content: center;

  align-items: baseline;

}

.mishoras {

  margin-left: 2px;

  font-weight: bold;

  font-size: 0.9rem;

}

.ampm {

  margin-left: 2px;

  margin-right: 2px;

  font-size: 0.9rem;

}

.segundos {

  width: 100%;

  text-align: right;

  font-size: 0.8rem;

  font-size: 100;

}

.app-sidebar-11::-webkit-scrollbar {

  width: 8px;

  height: 8px;

  background: #ccc;

}

.app-sidebar-11::-webkit-scrollbar-thumb {

  background: #F16257;

  border-radius: 4px;

}

.float-img {

  position: absolute;

  bottom: 20px;

  right: 20px;

  border-radius: 50%;

}

.float-img:hover {

  transform: translateY(-10px);

  transition: transform 0.2s ease-in-out;

}

/\* Estilos para el formulario PQR flotante \*/

.form-pqr {

  display: none;

  position: absolute;

  bottom: 50px;

  right: 50px;

  width: 400px;

  height: 300px;

  background-color: white;

  border-radius: 10px;

  padding: 20px;

}

.title edit {

  background-color: #f0f0f0;

  border: 1px solid #ccc;

  border-radius: 5px;

  padding: 10px;

  margin-bottom: 20px;

  font-size: 24px;

  font-weight: bold;

  color: #333;

  text-align: center;

  align-content: center;

}

.button-container {

  display: flex;

  gap: 10px;

  align-content: center;

}

.container-clock{

  display: flex;

}

footer {

  display: flex;

  padding: 20px;

}

.footer\_\_copyright{

  margin-right: 20%;

}

@media screen and (max-width: 768px) {

  /\* Estilos específicos para pantallas más pequeñas \*/

  .col-12 {

    display: flex;

    justify-content: space-between;

    align-items: center;

    padding: 0 10px;

  }

  .avatar {

    position: absolute;

    top: 10px;

    right: 10%;

  }

  .main-header{

    height: 60px;

    align-content: center;

  }

  .clock {

    display: none;

  }

  .bell {

    display: none;

  }

  .container-clock {

    display: none;

  }

  .boton-perfil {

    margin-right: 10px;

  }

  .float-img{

    width: 15%;

  }

}

.avatar-list {

  display: flex;

  flex-wrap: wrap;

  justify-content: center;

  gap: 20px;

  margin-top: 20px;

}

.avatar-item {

  cursor: pointer;

  border-radius: 10px;

  transition: transform 0.3s ease-in-out;

}

.avatar-item img {

  width: 100px;

  height: 100px;

  border-radius: 10px;

}

.avatar-item.selected {

  border: 2px solid #007bff;

  transform: scale(1.05);

}

.button-group {

  margin-top: 20px;

  display: flex;

  justify-content: center;

  gap: 10px;

}

Después de haber establecido el esqueleto básico de nuestro diseño principal "private-layout", el siguiente paso será construir dos elementos esenciales: la sección de bienvenida (landing) y el menú de navegación.

### Construcción del componente Landing

Ahora nos centraremos en el componente llamado "Landing". Este componente utiliza un archivo de tipo HTML para su diseño y apariencia.

Dentro de este archivo HTML, escribiremos el código que define cómo se verá y se estructurará la sección de bienvenida en nuestra página. En otras palabras, aquí es donde diseñamos la parte de la página que da la primera impresión a las personas que la visitan.

<br>

<br>

<br>

<header class="banner">

    <div>

        <img src="/assets/REVISTA-LOGO.png" width="60%" alt="revsita ciencia e ingenieria">

        <br>

        <br>

        <b><p>Explora, crea y gestiona contenido de calidad</p></b>

        <p>Revista de investigación interdisciplinar en biodiversidad y desarrollo sostenible,<br> ciencia, tecnología e

            innovación y procesos productivos industriales</p>

        <br>

    </div>

</header>

<div class="block-content content">

    <div class="ng-star-inserted">

        <div class="grid grid-nogutter surface-section text-800">

            <div class="col-12 md:col-6 p-6 text-center md:text-left flex align-items-center">

                <section><span class="block text-6xl font-bold mb-1">Crea el artículo</span>

                    <div class="text-6xl text-primary font-bold mb-3">Que tus lectores merecen</div>

                    <p class="mt-0 mb-4 text-700 line-height-3">Explorando los límites del conocimiento a través de la

                        ciencia y la ingeniería, nuestra revista te invita a un viaje de descubrimiento y asombro en

                        cada página.</p><button pbutton="" pripple="" label="Learn More" type="button"

                        class="p-element p-ripple mr-3 p-button-raised p-button p-component">

                        <a href="http://revistas.uniguajira.edu.co/rev/index.php/cei/article/view/e8091877/432"

                            target="\_blank" class="p-button-label"><span class="p-button-label"

                                style="color: aliceblue;">Ver mas</span></a><span class="p-ink"></span></button>

                </section>

            </div>

            <div class="col-12 md:col-6 overflow-hidden"><img src="assets/UNIGUAJIRA-5.jpg" width="100%" alt="Image"

                    class="md:ml-auto block md:h-full" style="clip-path: polygon(8% 0, 100% 0%, 100% 100%, 0 100%);">

            </div>

        </div>

    </div><!----><!---->

</div>

<div class="surface-section px-4 py-8 md:px-6 lg:px-8 revistas">

    <div>

        <div class="text-900 mb-3 font-bold text-4xl text-center">Puedes encontrarnos en</div>

        <div class="grid">

            <div class="col-12 md:col-6 lg:col-3">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/dialnet.png"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-3">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/worldcat.png"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-3">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/base.png"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-3">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/miar.png"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-2">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/latindex.jpg"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-2">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/aura.png"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-2">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/zenodo.png"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-2">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/erihplus.jpg"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-2">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/redib.jpg"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

            <div class="col-12 md:col-6 lg:col-2">

                <div class="flex align-items-center justify-content-center p-3"><img src="assets/logos/latinrev.jpg"

                        width="100px" alt="Image" class="hover-effect"></div>

            </div>

        </div>

    </div>

</div>

<section id="features" class="features-section">

    <div class="feature-item">

        <div class="feature">

            <h3>Gestión de Artículos</h3>

            <p>Optimiza tu proceso editorial con nuestra solución de gestión <br>

                de artículos. Administra tus solicitudes de manera eficiente.</p>

        </div>

        <div class="feature">

            <img src="/assets/logo\_r\_f.png" alt="Gestión de Suscripciones" width="150px">

        </div>

        <div class="feature">

            <h3>Gestión de Seguimientos</h3>

            <p>Sigue el proceso de tus artículos de manera eficiente,<br>

                conoce el resultado de tus evaluaciones de una manera sencilla.</p>

        </div>

    </div>

</section>

<section class="editions-section">

    <h2>Explora Nuestras Ediciones</h2>

    <div class="edition">

        <a href="http://revistas.uniguajira.edu.co/rev/index.php/cei/article/view/e8091877/432"><img

                src="/assets/portada\_edicion\_10.jpg" alt="Portada Edición 1"></a> <br>

        <a href="http://revistas.uniguajira.edu.co/rev/index.php/cei/article/view/e8091877/432" target="\_blank">

            Edición 1 | Volumen 10</a>

    </div>

</section>

<div class="surface-ground">

    <div class="grid">

        <div class="col-12 lg:col-4 p-3">

            <div class="shadow-2 border-round surface-card mb-3 h-full flex-column justify-content-between flex">

                <div class="p-4">

                    <div class="flex align-items-center">

                        <span

                            class="inline-flex border-circle align-items-center justify-content-center bg-green-100 mr-3"

                            style="width: 38px; height: 38px;">

                            <i class="pi pi-globe text-xl text-green-600"></i>

                        </span>

                        <span class="text-900 font-medium text-2xl">Plantilla para artículos</span>

                    </div>

                    <div class="text-900 my-3 text-xl font-medium">Una platnilla con todo el contenido y estructura que

                        debe tener su articulo.</div>

                </div>

                <div class="px-4 py-3 surface-100 text-right">

                    <button pbutton="" pripple="" icon="pi pi-arrow-right" iconpos="right" label="More"

                        class="p-element p-ripple p-button-rounded p-button-success p-button p-component">

                        <span class="p-button-icon p-button-icon-right pi pi-download" aria-hidden="true"></span>

                        <span class="p-button-label">Descargar</span>

                        <span class="p-ink"></span>

                    </button>

                </div>

            </div>

        </div>

        <div class="col-12 lg:col-4 p-3">

            <div class="shadow-2 border-round surface-card mb-3 h-full flex-column justify-content-between flex">

                <div class="p-4">

                    <div class="flex align-items-center">

                        <span

                            class="inline-flex border-circle align-items-center justify-content-center bg-yellow-100 mr-3"

                            style="width: 38px; height: 38px;">

                            <i class="pi pi-briefcase text-xl text-yellow-600"></i>

                        </span>

                        <span class="text-900 font-medium text-2xl">Declaración de originalidad</span>

                    </div>

                    <div class="text-900 my-3 text-xl font-medium">Formato donde se manifiesta la originalidad de su

                        articulo.</div>

                </div>

                <div class="px-4 py-3 surface-100 text-right">

                    <button pbutton="" pripple="" icon="pi pi-download" iconpos="right" label="Load"

                        class="p-element p-ripple p-button-rounded p-button-warning p-button p-component">

                        <span class="p-button-icon p-button-icon-right pi pi-download" aria-hidden="true"></span>

                        <span class="p-button-label">Descargar</span>

                        <span class="p-ink"></span>

                    </button>

                </div>

            </div>

        </div>

        <div class="col-12 lg:col-4 p-3">

            <div class="shadow-2 border-round surface-card mb-3 h-full flex-column justify-content-between flex">

                <div class="p-4">

                    <div class="flex align-items-center">

                        <span

                            class="inline-flex border-circle align-items-center justify-content-center bg-purple-100 mr-3"

                            style="width: 38px; height: 38px;">

                            <i class="pi pi-image text-xl text-purple-600"></i>

                        </span>

                        <span class="text-900 font-medium text-2xl">Normas de publicación</span>

                    </div>

                    <div class="text-900 my-3 text-xl font-medium">Normas de publicación Revista de Ciencias e

                        Ingeniería.</div>

                </div>

                <div class="px-4 py-3 surface-100 text-right">

                    <button pbutton="" pripple="" icon="pi pi-bookmark" iconpos="right" label="Save"

                        class="p-element p-ripple p-button-rounded p-button-help p-button p-component">

                        <span class="p-button-icon p-button-icon-right pi pi-arrow-right" aria-hidden="true"></span>

                        <span class="p-button-label">Ver</span>

                        <span class="p-ink"></span>

                    </button>

                </div>

            </div>

        </div>

    </div>

</div>

Dentro de este código HTML podemos encontrar las siguientes características:

**Encabezado de la Página:**

Esta sección contiene el logo de la revista y un mensaje de bienvenida en negrita. Además, se proporciona una descripción que informa sobre el enfoque temático de la revista, abarcando áreas como biodiversidad, desarrollo sostenible, ciencia, tecnología e innovación.

**Contenido Principal:**

Aquí se resalta la esencia de la revista, invitando a los usuarios a explorar y crear contenido de calidad. Se presenta un título llamativo y una descripción que enfatiza la experiencia de descubrimiento a través de la ciencia e ingeniería. Se incluye un botón para obtener más información.

**Logos de Plataformas:**

En esta sección se muestran logotipos de diferentes plataformas en línea donde la revista está disponible. Cada logotipo representa un sitio web o base de datos donde los usuarios pueden acceder a la revista.

**Características Especiales:**

Se presentan tres características destacadas de la revista: "Gestión de Artículos", "Gestión de Suscripciones" y "Gestión de Seguimientos". Cada característica incluye un título descriptivo y, en algunos casos, una imagen representativa.

**Ediciones Anteriores:**

En esta sección se invita a los usuarios a explorar ediciones anteriores de la revista. Se muestra la portada de una edición y se proporciona un enlace para acceder a más detalles sobre esa edición específica.

**Recursos Útiles:**

Aquí se presentan recursos descargables para los usuarios. Cada recurso está representado por una tarjeta que incluye un título y una breve descripción. Además, se ofrece un botón de descarga asociado a cada recurso.

Este código HTML crea una página principal que presenta la revista en línea de manera atractiva, resaltando su contenido, características y recursos disponibles para los usuarios. Cada sección tiene su propia función en la presentación y promoción de la revista.

### Landing.component.ts

Posteriormente debemos crear la lógica para esta vista html dentro de landing.component.ts para ello abrimos este archivo y hacemos lo siguiente:

import { Component, OnInit } from '@angular/core';

import { Router } from '@angular/router';

import { ConfirmationService, MessageService } from 'primeng/api';

import { UserService } from 'src/app/core/services/usuarios/user.service';

import { environment } from 'src/environments/environment';

@Component({

  selector: 'app-landing',

  templateUrl: './landing.component.html',

  styleUrls: ['./landing.component.css'],

  providers: [ConfirmationService,MessageService]

})

export class LandingComponent implements OnInit {

  public image3:string='assets/user.png'

  public bandera:boolean=false

  public nombre:string = '';

  private API\_URI:string=environment.API\_URI

  constructor(

    private userService:UserService,

    private router: Router,

    private messageService: MessageService

  ) {}

  ngOnInit(): void {

    var token :string | null= localStorage.getItem('token');

    if(token!=null){

        this.showSuccess()

        this.bandera=true

      }else{

        this.router.navigateByUrl('/login');

        this.bandera=false

      }

  }

 public showSuccess() {

    this.messageService.add({severity:'success', summary: 'Success', detail: 'Ingreso exitoso'});

}

}

Este archivo es esencial para dar vida y funcionalidad a la vista HTML previamente creada en "landing.component.html". A continuación, se ofrece una explicación detallada de las partes y acciones que contiene este archivo:

**Importación de Módulos y Servicios:**

En esta sección, se realizan importaciones necesarias para habilitar características y servicios. Estas importaciones son esenciales para el correcto funcionamiento de la lógica de la vista. Se importan módulos como Component y OnInit de Angular, así como varios servicios de manejo de mensajes y navegación.

**Definición del Componente:**

Se define el componente "LandingComponent" utilizando el decorador @Component. Se establecen las propiedades principales del componente, como el selector, la plantilla HTML y las hojas de estilo asociadas. Además, se proporciona una instancia de los servicios ConfirmationService y MessageService que estarán disponibles para este componente.

**Inicialización de Variables:**

Se declaran y definen varias variables que se utilizarán en el componente. Esto incluye la definición de image3, bandera, nombre y API\_URI. Estas variables son fundamentales para controlar y mostrar información en la vista HTML.

**Constructor y Dependencias:**

Se define el constructor del componente. Aquí se inyectan los servicios y recursos necesarios, como UserService, Router y MessageService. Estos recursos son utilizados para interactuar con la lógica de la aplicación y proporcionar funcionalidades de manejo de usuarios y navegación.

**Método ngOnInit():**

El método ngOnInit() se implementa como parte de la interfaz OnInit de Angular. En este método se realiza una serie de acciones al iniciar el componente. Primero, se verifica la existencia de un token en el almacenamiento local. Si se encuentra un token, se muestra un mensaje exitoso y se activa la bandera bandera. En caso contrario, se redirige al usuario a la página de inicio de sesión y se desactiva la bandera.

**Método showSuccess():**

El método showSuccess() se define para mostrar un mensaje de éxito utilizando el servicio MessageService. Este mensaje se visualiza mediante una notificación visual en la vista HTML. El mensaje muestra una breve confirmación de "Ingreso exitoso".

En resumen, el archivo "landing.component.ts" es crucial para gestionar la lógica detrás de la vista HTML de la página de aterrizaje. Controla la autenticación del usuario, maneja mensajes de éxito y redirecciones, y se comunica con los servicios de Angular para brindar una experiencia de usuario fluida y efectiva.

### Landing.component.css

Por último, le damos estilos a nuestro componente landing de la siguiente manera:

.margSup {

  display: flex;

  flex-direction: row;

  flex-wrap: nowrap;

  justify-content: flex-end;

  padding-right: 10%;

  padding-top: 10px;

}

.imgLogo {

  width: auto;

  border: 1px solid transparent;

  border-radius: 15px;

  display: flex;

  justify-content: center;

  align-content: center;

}

.titleCard {

  display: flex;

  justify-content: center;

  align-content: center;

  font-size: 45px;

  font-weight: bold;

  /\* color:rgba(51,102,0,1); \*/

  margin-top: 10px;

}

.imgLogo2 {

  width: 60%;

  /\* min-height: 250px; \*/

  /\* margin-left: 30px; \*/

  /\* max-width: 500px; \*/

  /\* max-height:700px; \*/

  border: 1px solid transparent;

  border-radius: 15px;

  display: flex;

  justify-content: center;

  align-content: center;

}

.titleCard2 {

  display: flex;

  justify-content: center;

  align-content: center;

  font-size: 45px;

  font-weight: bold;

  /\* color:rgba(51,102,0,1); \*/

  margin-top: 10px;

}

.margen {

  /\* padding: 10px; \*/

  margin-top: -5px;

  margin-bottom: -15px;

  background-color: #dcdcdc;

  min-height: auto;

}

.content {

  background: #f5f5f5;

  /\* border-radius: 13px; \*/

  padding: 30px;

  box-shadow: 7px 8px 19px 0px #00000047;

}

.content2 {

  /\* width: 100%; \*/

  height: 80%;

  margin-top: -5px;

  background: transparent;

  /\* border-radius: 13px; \*/

  padding: 20px;

  box-shadow: 7px 8px 19px 0px #00000047;

  /\* background-image: url('../assets/images/fondo2.jpg'); \*/

  /\* background: linear-gradient(0deg, rgba(51,102,0,1) 0%, rgba(34,193,195,0.5578606442577031) 100%); \*/

  background-size: cover;

}

.mat-toolbar-single-row {

  justify-content: space-between;

}

.menuimagen {

  margin-top: 10px;

  /\* display: flex; \*/

}

.mat-toolbar-single-row {

  display: flex;

  /\* padding-top:10px; \*/

  height: 70px;

  margin-bottom: 0

}

/\* .mat-drawer-content {

  position: relative;

  z-index: 1;

  display: block;

  height: 100%;

  overflow: auto;

} \*/

/\* Assign grid instructions to our parent grid container \*/

.grid-container {

  display: grid;

  grid-template-columns: 250px 3fr;

  grid-template-rows: 40px 3fr 80px;

  grid-template-areas:

    "sidenav subtitle"

    "sidenav main"

    "sidenav footer";

  height: 100%;

  width: 100vmax;

}

/\* Give every child element its grid name \*/

.sidenav {

  grid-area: sidenav;

  overflow: auto;

  -ms-overflow-style: none;

  /\* IE 11 \*/

  scrollbar-width: none;

  /\* Firefox 64 \*/

  box-shadow: 3px 0 5px rgba(57, 63, 72, 0.3);

  ;

}

.sidenav::-webkit-scrollbar {

  display: none;

}

.subtitle {

  grid-area: subtitle;

  margin-top: 20px;

}

.main {

  grid-area: main;

  margin: 5px;

  /\* margin-top: -px; \*/

}

.footer {

  grid-area: footer;

  display: flex;

  align-content: center;

  justify-content: space-around;

  margin: 3px 0 0 2px;

  /\* border: grayscale($color:  cadetblue) medium solid; \*/

  /\* background-color: rgb(179, 170, 170); \*/

  /\* background-color: rgb(243, 164, 85) \*/

  /\* background-color: #F1A157; \*/

}

.centered-content {

  width: 100%;

  max-width: 1000px;

  padding: 0 40px;

}

.hijo {

  display: flex;

  justify-content: flex-start;

  align-items: flex-start;

  /\*  height: 30px; \*/

  font-size: 14px;

  color: #6b778c;

}

.hijo1 {

  display: flex;

  justify-content: flex-end;

  align-items: flex-end;

  margin-top: -60px;

  height: 30px;

  font-size: 14px;

  color: #6b778c;

}

.footer\_\_copyright {

  line-height: 10px;

  margin: 5px;

}

.clickable {

  cursor: pointer;

}

.padre:hover {

  border-left: 5px solid #F16257;

  text-decoration-color: #F16257;

  color: #F16257

    /\* background-color: #F16257; \*/

}

.buttons-container {

  color: #707070;

  display: flex;

  align-items: center;

  justify-content: center;

  height: 75px;

  padding: 0 5px;

  text-decoration: none;

  margin-top: -15px;

  margin-right: 10px;

}

.buttons-container:hover {

  background-color: #c9c9c9f3;

  color: black

}

.buttons-container .mat-icon {

  padding-top: 10px;

}

a {

  color: #707070;

  text-decoration: none;

  /\* align-items: center; \*/

}

a .text {

  font-weight: 500;

  font-size: 16px;

  margin-left: 5px;

  padding-top: 7px;

}

.padre-container {

  color: #707070;

  /\* font-weight: 500; \*/

  display: flex;

  align-items: center;

  justify-content: center;

  height: inherit;

  padding: 0 10px;

  text-decoration: none;

}

li {

  line-height: 24px;

}

li {

  display: list-item;

  text-align: -webkit-match-parent;

}

.nav {

  margin-left: 0;

  margin-bottom: 24px;

  list-style: none;

}

#redes\_sociales li a {

  color: #fff;

  font-size: 0px;

  padding: 0;

  margin: 0;

}

#redes\_sociales li {

  margin: 8px 40px;

  padding: 0;

}

#redes\_sociales {

  width: 100%;

  margin: 0;

  margin-top: -5px;

  display: flex;

  justify-content: center;

  -webkit-justify-content: center;

  -ms-justify-content: center;

  -o-justify-content: center;

  -moz-justify-content: center;

}

.nav>li>a {

  display: block;

}

.fondebabutton {

  color: #707070;

}

.clock {

  display: flex;

}

.clock p {

  color: #FBC02D;

  font-size: 2rem;

}

.hours {

  grid-column-start: 1;

  grid-column-end: 3;

  color: #FBC02D;

}

.horas {

  display: flex;

  flex-wrap: nowrap;

  flex-direction: column-reverse;

  justify-content: center;

  align-items: baseline;

}

.mishoras {

  margin-left: 3px;

  font-weight: bold;

  font-size: 1rem;

}

.ampm {

  margin-left: 3px;

  margin-right: 3px;

  font-size: 1rem;

}

.segundos {

  width: 100%;

  text-align: right;

  font-size: 0.8rem;

  font-size: 100;

}

.titleCard1 {

  margin: 0 0 10px 0;

  display: flex;

  justify-content: center;

  align-content: center;

}

mat-card-title {

  padding-left: 10px;

  padding-top: 10px;

}

.categ-section {

  display: flex;

  align-content: center;

  align-items: center;

  justify-content: flex-end;

  height: 80px;

}

.categ-margin {

  margin: 0;

}

.opcStatus {

  margin: 0 40px;

}

.banner {

  display: flex;

  align-items: center;

  justify-content: center;

  padding: 50px 0;

  background-color: #f5f5f5;

}

.banner>div {

  text-align: center;

  padding: 0 20px;

}

.titulo {

  margin-top: 20px;

}

.titulo h1 {

  font-size: 36px;

  color: #333;

  font-weight: bold;

}

.banner p {

  font-size: 18px;

  margin: 20px 0;

}

.banner img {

  max-width: 100%;

  height: auto;

}

.cta-button {

  display: inline-block;

  padding: 10px 20px;

  background-color: #ffcc00;

  color: #333;

  font-weight: bold;

  text-decoration: none;

  border-radius: 5px;

  transition: background-color 0.3s;

}

.cta-button:hover {

  background-color: #ffdb4d;

}

.feature-item {

  display: flex;

  align-items: center;

  margin-left: 200px;

}

.features-section {

  padding: 50px 0;

  text-align: center;

  background-color: #f5f5f5f5;

  align-items: center;

  vertical-align: center;

}

.feature h3 {

  font-size: 24px;

  margin-top: 10px;

  margin-bottom: 10px;

}

.feature p {

  font-size: 16px;

}

.editions-section {

  padding: 50px 0;

  text-align: center;

}

.edition {

  display: inline-block;

  margin: 0 20px;

}

.edition img {

  width: 200px;

  height: 300px;

  border: 1px solid #ccc;

  border-radius: 5px;

}

.edition p {

  margin-top: 10px;

}

.cta-section {

  padding: 50px 0;

  text-align: center;

}

.cta-section h2 {

  margin-bottom: 20px;

}

.cta-button-big {

  display: inline-block;

  padding: 15px 30px;

  background-color: #ffcc00;

  color: #333;

  font-weight: bold;

  text-decoration: none;

  border-radius: 5px;

  transition: background-color 0.3s;

}

.cta-button-big:hover {

  background-color: #ffdb4d;

}

.footer {

  text-align: center;

  padding: 20px 0;

  background-color: #333;

  color: #fff;

}

.social-media a {

  margin: 0 10px;

}

.social-media img {

  width: 30px;

  height: 30px;

}

.revistas {

  align-items: center;

  vertical-align: center;

}

.hover-effect {

  transition: transform 0.3s ease-in-out;

}

.hover-effect:hover {

  transform: scale(1.5);

}

Este conjunto de pasos, que involucra la definición de rutas, la implementación de servicios, la creación de modelos de datos, la construcción de elementos visuales, la creación de componentes funcionales y la aplicación de estilos, se repite de manera consistente a lo largo de todo el proceso de desarrollo de software. Cada iteración de este proceso se adapta específicamente a las características particulares de cada componente.

Este flujo constante de desarrollo se lleva a cabo de manera continua para múltiples componentes en el sistema. A continuación, se presenta un resumen de cómo este proceso se aplica a diversos componentes:

**Solicitudes**

* Generar
  + Ver solicitud
  + Crear solicitud
  + Editar solicitud
  + Eliminar solicitud
  + Ver mis solicitudes
* **Pasos**
  + **Ver paso**
  + **Crear paso**
  + **Editar paso**
  + **Eliminar pasó**
* **Seguimiento**
  + **Ver seguimiento**
  + **Crear seguimiento**
  + **Editar seguimiento**
  + **Eliminar seguimiento**
  + **Ver mis seguimientos**
  + **Ver solicitudes asignadas**
* **Evaluación**

**Administrador**

* **Roles**
* **Recursos**
* **Asignar rol**
* **Asignar recursos**

**Usuarios**

* **Personas**

Cada uno de estos componentes sigue un proceso similar de desarrollo continuo, adaptándose y ajustándose según sus propias características y funcionalidades. Esta metodología garantiza que cada parte del sistema sea diseñada y construida con precisión y eficacia, cumpliendo con los requisitos y objetivos establecidos.