Coder Dojo y la IEEE CS Unsa

```
Integrantes:
- Chambilla Perca Ricardo  Mauricio
- Jhonatan Arias Quispe Jhonatan
- Carbajal Gonzales Diego Alejandro

Aplicacion para el evento de coder dojo que esta hosteando la ieee
Computational Society rama Peru por parte de la UNSA.
```

Las Especificaciones que recibimos de los coordinadores fueron:

- 2 tipos de usuario: Profesor y Estudiante, y un admin
- Salones virtuales, los profesores pueden crear su salon y asignarle tareas a los alumnos de ese salon
- Los Salones contienen a los estudiantes y a un profesor, ademas, posteos de parte de los profesores que los alumnos pueden ver para obtener material de estudio
- El usuario objetivo son alumnos de secundaria, por lo que debe tener un cirto nivel de gamificacion que es un framework reciente para aumentar la atencion y enganche surgido por Octolasys group

Los Modelos

```
  Una vez entendido las relaciones que deben tener los modelos se puede
pasar a programarlos y a migrarlos
```

Modelo para los Usuarios:

```
class AppUserManager(BaseUserManager):
  def create_user(self, email, password=None,):
    if not email:
        raise ValueError('An email is required.')
    if not password:
        raise ValueError('A password is required.')
    email = self.normalize_email(email)
    user = self.model(email=email,)
    user.set_password(password)
    user.save()
    return user
  def create_superuser(self, email, password=None, **extra_fields):
    if not email:
        raise ValueError('An email is required.')
    if not password:
        raise ValueError('A password is required.')
    user = self.create_user(email, password, extra_fields)
    user.is_superuser = True
    user.is_staff = True
```

```
    user.save()
    return user


class AppUser(AbstractBaseUser, PermissionsMixin):
    user_id = models.AutoField(primary_key=True)
    email = models.EmailField(max_length=50, unique=True)
    username = models.CharField(max_length=50)
    is_staff = models.BooleanField(default=False)
    is_student = models.BooleanField(default=False)
    is_teacher = models.BooleanField(default=False)


  USERNAME_FIELD = 'email'
  REQUIRED_FIELDS = ['username']
  objects = AppUserManager()
  def __str__(self):
    return self.username
```

Decidimos usar el usuario base de django para obtener todas sus caracteristicas de la autenticacion por default de django. Las cuales son comprobaciones en la contraseña, sesion de usuario y autenticacion de usuario usando la encriptacion de los credenciales para cada usuario

Modelos para los salones

```
  from django.db import models
  from django.contrib.auth.models import AbstractUser

  class User(AbstractUser):
      is_estudiante = models.BooleanField(default=False)
      is_docente = models.BooleanField(default=False)

  class Estudiante(models.Model):
      user = models.OneToOneField(User, on_delete=models.CASCADE)
      name = models.CharField(max_length=100)

  class Docente(models.Model):
      user = models.OneToOneField(User, on_delete=models.CASCADE)
      name = models.CharField(max_length=100)

  class Clase(models.Model):
      name = models.CharField(max_length=100)
      estudiantes = models.ManyToManyField(Estudiante,
related_name='clases')
      docente = models.ForeignKey(Docente, on_delete=models.CASCADE,
related_name='clases')

  class Publicacion(models.Model):
      content = models.TextField()
      clase = models.ForeignKey(Clase, on_delete=models.CASCADE,
related_name='publicaciones')
```

```
  class Asignacion(models.Model):
      title = models.CharField(max_length=100)
      description = models.TextField()
      due_date = models.DateTimeField()
      clase = models.ForeignKey(Clase, on_delete=models.CASCADE,
related_name='asignaciones')

  class Entrega(models.Model):
      asignacion = models.ForeignKey(Asignacion, on_delete=models.CASCADE,
related_name='entregas')
      estudiante = models.ForeignKey(Estudiante, on_delete=models.CASCADE,
related_name='entregas')
      file = models.FileField(upload_to='entregas/')
      submitted_at = models.DateTimeField(auto_now_add=True)
```

Resumen del codigo para el backend:

```
from rest_framework import generics, permissions
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from django.shortcuts import get_object_or_404
from .models import Clase, Publicacion, Asignacion, Entrega, Estudiante,
Docente
from .serializers import ClaseSerializer, PublicacionSerializer,
AsignacionSerializer

class LoginView(APIView):
    permission_classes = (permissions.AllowAny,)

    def post(self, request):
        username = request.data.get('username')
        password = request.data.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return HttpResponseRedirect('/clases')
        return Response({'error': 'Invalid Credentials'},
status=status.HTTP_401_UNAUTHORIZED)

class LogoutView(APIView):
    def post(self, request):
        logout(request)
        return HttpResponseRedirect('/')

class ClaseListView(generics.ListCreateAPIView):
    serializer_class = ClaseSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        user = self.request.user
```

```
        if user.is_estudiante:
            estudiante = get_object_or_404(Estudiante, user=user)
            return estudiante.clases.all()
        elif user.is_docente:
            docente = get_object_or_404(Docente, user=user)
            return docente.clases.all()
        return Clase.objects.none()

    def perform_create(self, serializer):
        user = self.request.user
        if user.is_docente:
            docente = get_object_or_404(Docente, user=user)
            serializer.save(docente=docente)

class PublicacionListView(generics.ListCreateAPIView):
    serializer_class = PublicacionSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        user = self.request.user
        if user.is_estudiante:
            estudiante = get_object_or_404(Estudiante, user=user)
            return
Publicacion.objects.filter(clase__in=estudiante.clases.all())
        elif user.is_docente:
            docente = get_object_or_404(Docente, user=user)
            return Publicacion.objects.filter(clase__docente=docente)
        return Publicacion.objects.none()

class AsignacionListView(generics.ListCreateAPIView):
    serializer_class = AsignacionSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        user = self.request.user
        if user.is_estudiante:
            estudiante = get_object_or_404(Estudiante, user=user)
            return
Asignacion.objects.filter(clase__in=estudiante.clases.all())
        elif user.is_docente:
            docente = get_object_or_404(Docente, user=user)
            return Asignacion.objects.filter(clase__docente=docente)
        return Asignacion.objects.none()

class ClaseDetailView(generics.RetrieveUpdateDestroyAPIView):
    queryset = Clase.objects.all()
    serializer_class = ClaseSerializer
```

No nos olvidemos que los unicos que pueden crear una nueva clase son los profesores, por lo que necesitamos verificar permisos, nos podemos ayudar de una clase de permisos:

```
  from rest_framework import permissions

  class IsDocente(permissions.BasePermission):
    def has_permission(self, request, view):
        return request.user and request.user.is_authenticated and
request.user.is_docente
```

Tambien debemos serializar la creacion de las clases para que los alumnos y el profesor puedan guardar esta clase que recien se crea:

```
  class PublicacionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Publicacion
        fields = '__all__'

  class AsignacionSerializer(serializers.ModelSerializer):
      class Meta:
          model = Asignacion
          fields = '__all__'

  class ClaseSerializer(serializers.ModelSerializer):
      publicaciones = PublicacionSerializer(many=True, required=False)
      asignaciones = AsignacionSerializer(many=True, required=False)
      estudiantes = serializers.ListField(child=serializers.IntegerField(),
required=False)

  class Meta:
      model = Clase
      fields = ['id', 'nombre', 'docente', 'publicaciones', 'asignaciones',
'estudiantes' ]

  def create(self, validated_data):
      publicaciones_data = validated_data.pop('publicaciones', [])
      asignaciones_data = validated_data.pop('asignaciones', [])
      estudiantes_data = validated_data.pop('estudiantes', [])

      clase = Clase.objects.create(**validated_data)

      for publicacion_data in publicaciones_data:
          Publicacion.objects.create(clase=clase, **publicacion_data)

      for asignacion_data in asignaciones_data:
          Asignacion.objects.create(clase=clase, **asignacion_data)

      for estudiante_id in estudiantes_data:
          estudiante = Estudiante.objects.get(id=estudiante_id)
          estudiante.clases.add(clase)

      return clase
```

Por ultimo las urls en la app:

```
from django.urls import path
from .views import LoginView, LogoutView, ClaseListView, ClaseDetailView,
PublicacionListView, AsignacionListView

urlpatterns = [
    path('login/', LoginView.as_view(), name='login'),
    path('logout/', LogoutView.as_view(), name='logout'),
    path('clases/', ClaseListView.as_view(), name='clase-list'),
    path('clases/<int:pk>/', ClaseDetailView.as_view(), name='clase-
detail'),
    path('publicaciones/', PublicacionListView.as_view(),
name='publicacion-list'),
    path('asignaciones/', AsignacionListView.as_view(), name='asignacion-
list'),
]
```

Por el lado del front-end

Los frameworks que se utilizaron para el frontend son React y Tailwind css, la pagina principal del login, esta basado en componentes: los componentes que contiene la pagina principal son: el Login, los logos, el mensaje de bienvenida, el boton de cambio de tema y el widget que permite abrir el minigame (DojoType).

La pagina principal es el componente HomePage.jsx:

```
  function HomePage() {
  const [isDojoTypeOpen, setIsDojoTypeOpen] = useState(false);

  useEffect(() => {
    const handleKeyPress = (event) => {
      if (event.key === "k" || event.key === "K") {
        setIsDojoTypeOpen(true);
      }
    };

    window.addEventListener("keydown", handleKeyPress);
    return () => {
      window.removeEventListener("keydown", handleKeyPress);
    };
  }, []);

  const handleDojoTypeButtonClick = () => {
    setIsDojoTypeOpen(true);
  };

  const handleClosePopup = () => {
    setIsDojoTypeOpen(false);
  };
```

```
  return (
    <div className="bg-dojo-day dark:bg-dojo-night bg-cover bg-center h-
screen w-screen flex">
      <div className="relative flex-grow">

        <div className="relative flex flex-col items-center justify-center
min-h-screen ">
          <div className="absolute inset-0 flex items-center justify-
center">
            <div className="relative bg-white bg-opacity-70 backdrop-blur-
md p-8 rounded-lg shadow-lg text-center max-w-lg">
              <h1 className="text-4xl font-extrabold text-gray-900 mb-4">
                ¡Bienvenido a CoderDojo!
              </h1>
              <p className="text-lg text-gray-700">
                Estamos emocionados de tenerte con nosotros. Prepárate para
                aprender y crecer en el mundo de la programación. ¡Vamos a
hacer
                grandes cosas juntos!
              </p>
            </div>
          </div>
        <div className="absolute top-5 left-5 p-4">
          <Logo path="/src/assets/IEEE-CS-UNSA.png" />
        </div>
        <a href="https://coderdojo.com/en/" target="_blank"
className="inset-1">
          <div className="absolute top-5 right-5 p-4">
            <Logo path="/src/assets/CoderDojo.png" />
          </div>
        </a>
          <div className="absolute bottom-5 left-5 p-4">
            <DojoTypeButton onClick={handleDojoTypeButtonClick} />
          </div>
          <div className="absolute bottom-5 right-5 p-4">
            <ThemeSwitcher />
          </div>
        </div>
      </div>

      <LoginForm />

      {isDojoTypeOpen && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex items-
center justify-center z-50">
          <div className="bg-gray-900 bg-opacity-80 backdrop-blur-md p-4
rounded-3xl shadow-lg w-[90vw] h-5/6 relative">
            <DojoType />
            <button
              className="absolute top-1 right-1 text-4xl text-red-500 p-2
rounded"
              onClick={handleClosePopup}
            >
```

```
                <IoCloseCircle />
              </button>
            </div>
          </div>
        )}
      </div>
    );
  }
```

Mientras que la forma del classroom lo proporciona el componente Dashboard

```
  export default function Bashboard(){
  return (
    <div className="flex">
      <Sidebar />
      <div className="ml-64 w-full">
        <Navbar />
        <div className="pt-16">
          <StudentRoutes />
        </div>
        <CoursesList />
      </div>
    </div>
    )
```

}

URL pagina live: https://coder-dojo-front.vercel.app/