



pyGAMEcol

Documentación del proyecto final del
Bootcamp de Backend con Python de



códigofacilito

Sergio San Quirico Ortega

Índice

- Presentación
- Tecnologías
- Recursos
- Requerimientos
- Desarrollo
- Uso
- Dificultades

Presentación

pyGAMEcol es un sitio web desarrollado como proyecto final del Bootcamp de Backend para Python de Código Facilito.

Tal y como indico en el README.md del repositorio del proyecto en GitHub, el proyecto ha sido desarrollado a pocos días de la fecha de entrega, debido a que, en su momento, por motivos personales entre los que se encontraban la desmotivación y frustración de un neófito enfrentándose a su primer proyecto con tecnologías que desconocía, decidí abandonar el proyecto.

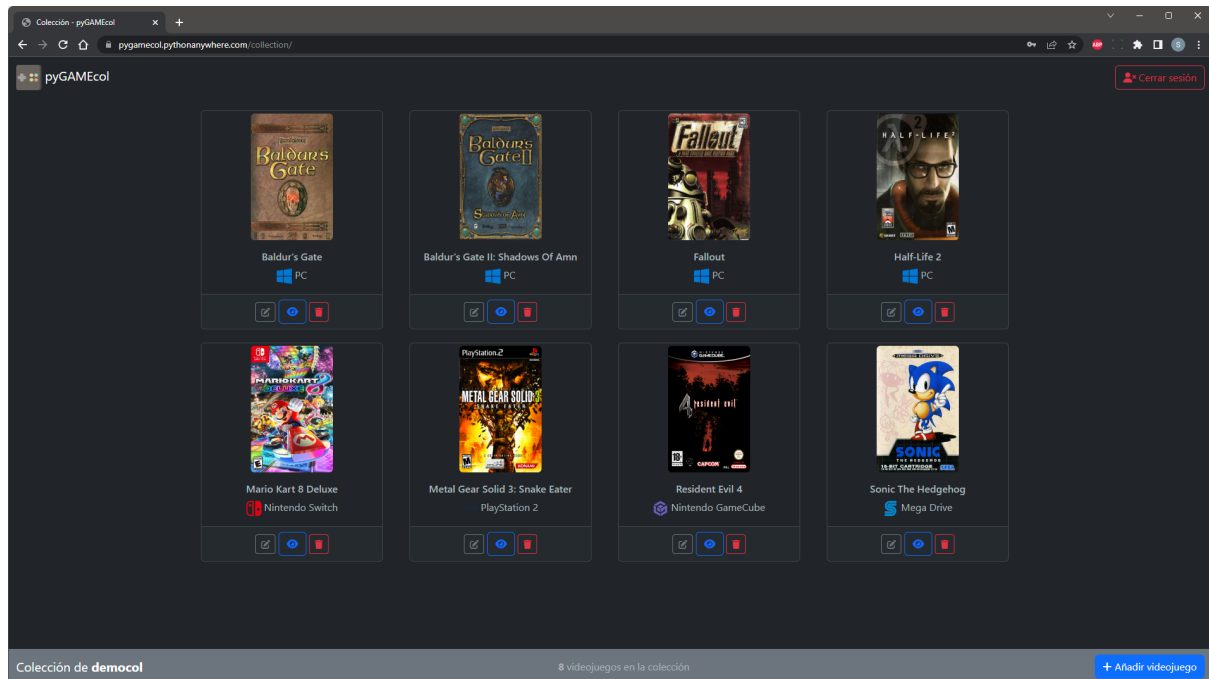
Apenas disponía de tiempo, entre el trabajo y un bebé en casa, las noches eran el único momento en el que podía ponerme a estudiar, y era realmente duro perder horas de sueño, ir cansado al trabajo y ver cómo el proyecto no avanzaba... así que el abandono era la opción más fácil.

Pero una de las cosas que hacen diferente a este tipo de formación es su comunidad. Cuando apenas quedaba poco más de una semana para que finalizara la fecha de entrega del proyecto, algunos compañeros y profesores me animaron a intentarlo, a hacer y entregar lo que pudiera, así que me armé de valor para enfrentarme a varias noches durmiendo poco, a muchas vueltas en la cabeza pensando en cómo resolver los problemas del proyecto durante los descansos del trabajo, conduciendo, haciendo la compra...

Al final, como es obvio, el poco tiempo no me ha permitido realizar un proyecto complejo, pyGAMEcol es un proyecto muy simple pero que considero estable, escalable y que me ha permitido aprender las bases para que mis siguientes proyectos tengan cierta solidez.

Por favor, cuando revises el proyecto, no seas demasiado exigente. He ido aprendiendo algunas cosas en tiempo real mientras desarrollaba (no sabía nada de HTML ni Bootstrap), y, como he comentado al principio, este es mi

primer proyecto de desarrollo de cualquier tipo, anteriormente, lo más complicado que había realizado era mostrar en “Hola Mundo!” por una consola de texto.



Tecnologías

Cuando te enfrentas por primera vez a un proyecto sin haber desarrollado nada anteriormente, la primera duda que te asalta es qué tecnologías son las indicadas para llevar a buen puerto tu idea.

De entre las opciones disponibles para desarrollar el proyecto, o bien una API o bien un sitio web, la única válida que cumplía los requisitos que necesitaba mi proyecto era la del **sitio web**. Además no quería dejar pasar la oportunidad de enfrentarme a uno de mis temores más primigenios, el lenguaje de marcado HTML, estilos CSS y diseño de la interfaz, algo que también era totalmente nuevo para mi.

El lenguaje de programación estaba claro, al fin y al cabo, forma parte del nombre del bootcamp: **Python**. Si bien es cierto que mi conocimiento del lenguaje era muy limitado antes de empezar el bootcamp, durante este tiempo he consolidado conocimientos como la programación orientada a objetos, decoradores, trabajar con módulos y paquetes, incluso los test unitarios.

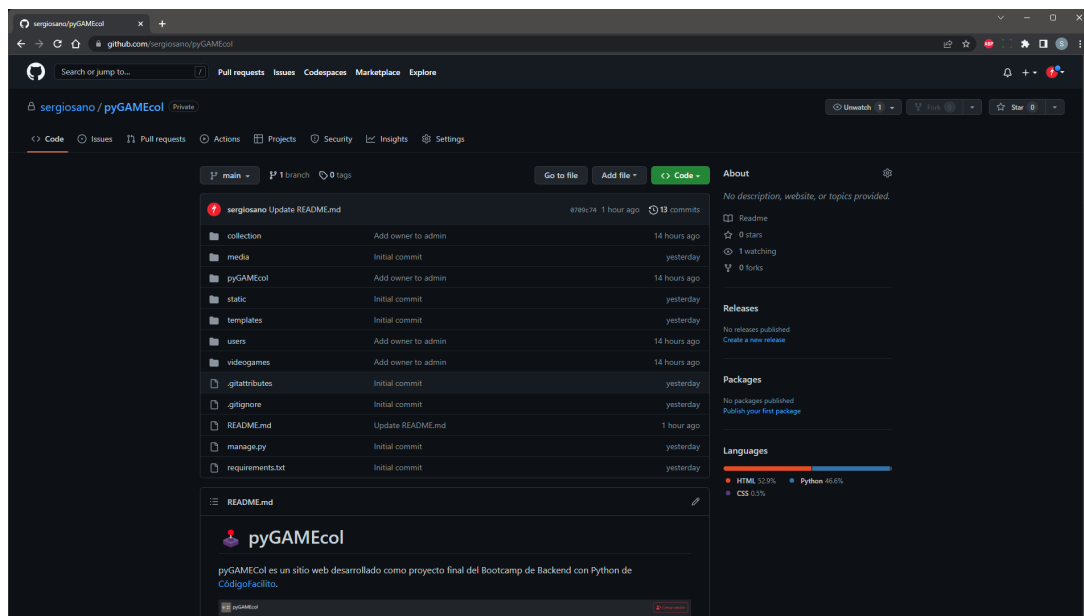
En cuanto al framework de desarrollo web para el backend, dudé poco. En el bootcamp nos mostraron dos de los más utilizados, Flask y Django, y aunque me atraía la idea de trabajar con Flask y desarrollar por mi mismo algunas de las funcionalidades que Django integraba por defecto, el tiempo, documentación disponible y elección del restos de compañeros me hizo decantarme por **Django**.

En cuanto a la persistencia de datos, elegir un sistema gestor de bases de datos fue sencillo. A este nivel inicial, **MySQL** era la opción más sencilla, además de que era el SGBD en el cual nos formaron en el bootcamp. Durante parte del desarrollo utilicé SQLite, y el hecho de poder migrar de un SGBD a otro de forma tan sencilla y transparente, me ha abierto los ojos a la utilización del ORM, concepto que no entendía al principio pudiendo usar SQL.

Recursos

Los recursos del proyecto, incluida esta documentación, las instrucciones de puesta en marcha, el proyecto desplegado, etc. se encuentran en:

- Repositorio GitHub: <https://github.com/sergiosano/pyGAMEcol>



- Sitio web desplegado: <https://pygamecol.pythonanywhere.com/>



Requerimientos

Estos son los requisitos indicados para un proyecto basado en sitio web y cómo los he cumplido:

Requerimientos generales

- El proyecto debe conectarse a una base de datos (MySQL o PostgreSQL):

Durante parte del desarrollo utilicé SQLite, pero en un momento determinado migré a MySQL, instalando una VM dedicada para ello. Obviamente en producción, el despliegue se ha realizado con MySQL como SGBD, pero en el repositorio de GitHub incluyo las instrucciones para trabajar con cualquier SGBD soportado por Django. La configuración del SGBD se realiza con un archivo `.env` utilizando el paquete **python-decouple**.

- **El proyecto debe apoyarse de un ORM para realizar las consultas:** Al utilizar Django como framework web, utilizo el ORM de Django para realizar todas las consultas necesarias en mi proyecto.

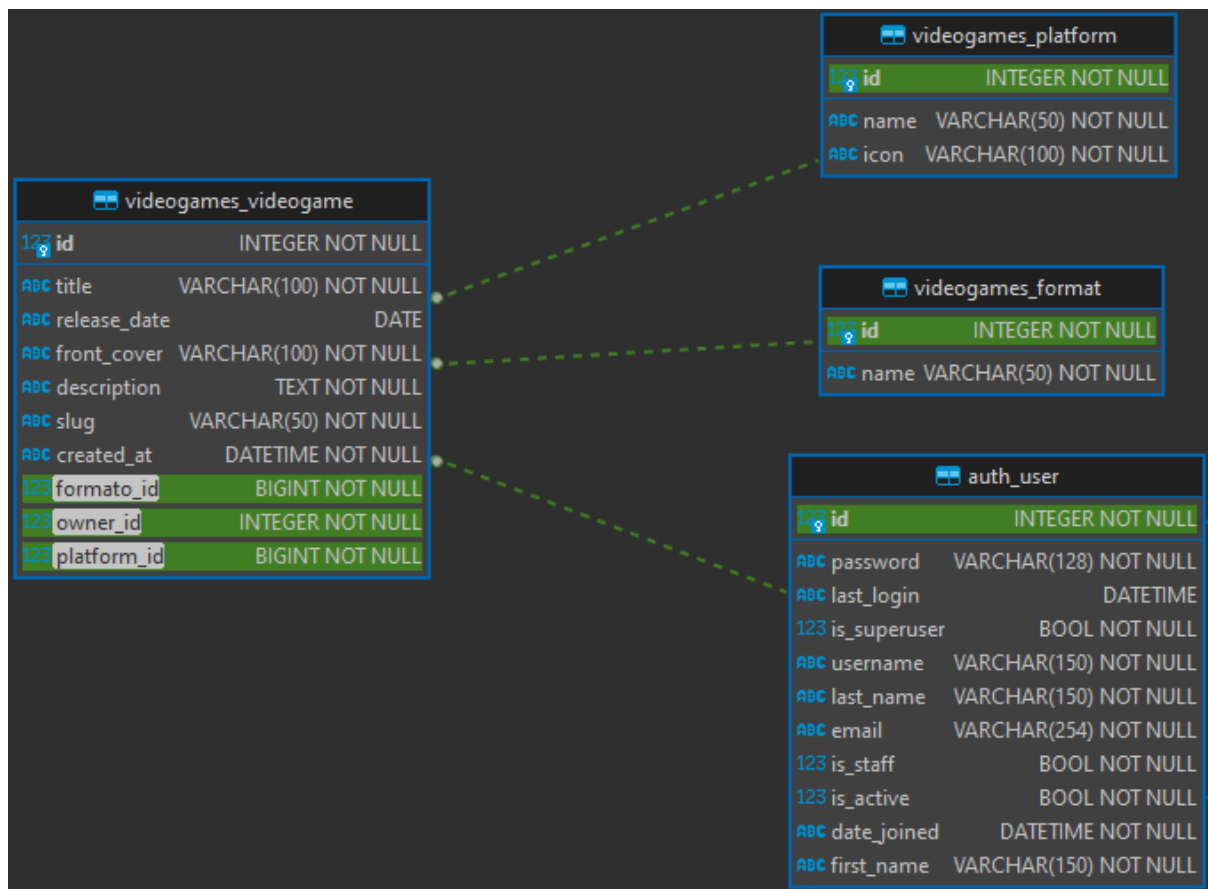
Algunos ejemplos:

```
def get_queryset(self):
    return Videogame.objects.filter(owner_id=self.request.user.id).order_by('title')
```

```
def dispatch(self, request, *args, **kwargs):
    if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=kwargs['slug']).owner:
        return redirect('index')

    return super().dispatch(request, *args, **kwargs)
```

- **Deben existir, por lo menos, 3 modelos relacionados entre sí:** Cumplido por la mínima:



- **Se debe implementar por lo menos un catálogo CRUD:** El proyecto permite realizar las 4 operaciones básicas de leer, crear, actualizar y eliminar con una combinación de vistas basadas en funciones y vistas basadas en clases, lo que me permitió ver las características y limitaciones de cada sistema.

Personalmente, al ser novato, me gustan más las vistas basadas en funciones, dado que toda la funcionalidad la desarrolla el programador, mientras que con las vistas basadas en clases, tengo la sensación de utilizar cajas negras, teniendo que sumergirte en la documentación oficial para averiguar cómo sobrescribir ciertos métodos para llegar a obtener la funcionalidad deseada.

Leer: Vista basada en clase

```
class VideogameDetailView(DetailView):
    model = Videogame
    template_name = 'videogames/videogame_view.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        return context

    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=kwargs['slug']).owner:
            return redirect('index')

        return super().dispatch(request, *args, **kwargs)
```

Crear: Vista basada en función

```
def videogame_create(request):
    if not request.user.is_authenticated:
        return redirect('index')

    form = VideogameCreateForm(request.POST or None, request.FILES or None)

    if request.method == 'POST' and form.is_valid():
        title = form.cleaned_data.get('title')
        platform = form.cleaned_data.get('platform')
        formato = form.cleaned_data.get('formato')
        release_date = form.cleaned_data.get('release_date')
        front_cover = form.cleaned_data.get('front_cover')
        if not front_cover:
            front_cover = 'videogames/front_covers/default_front_cover.png'
        description = form.cleaned_data.get('description')
        videogame = Videogame.objects.create(owner=request.user, title=title, platform=platform, formato=formato, release_date=release_date, front_cover=front_cover, description=description)

        if videogame:
            return redirect('collection')

    return render(request, 'videogames/videogame_create.html', {'form': form})
```

Actualizar: Vista basada en función

```
def videogame_update(request, slug):
    if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=slug).owner:
        return redirect('index')

    videogame = Videogame.objects.get(slug=slug)

    if request.method == 'POST':
        form = VideogameUpdateForm(request.POST or None, request.FILES or None, instance=videogame)
        if form.is_valid():
            form.save()
            return redirect('videogame_view', videogame.slug)
    else:
        form = VideogameUpdateForm(instance=videogame)

    return render(request, 'videogames/videogame_update.html', {'form': form, 'videogame': videogame})
```

Eliminar: Vista basada en clase

```
class VideogameDeleteView(DeleteView):
    model = Videogame
    success_url = reverse_lazy("collection")
    template_name = "videogames/videogame_delete.html"

    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=kwargs['slug']).owner:
            return redirect('index')

        return super().dispatch(request, *args, **kwargs)
```

- **Validar modelo de negocio mediante pruebas unitarias:** Tengo que reconocerlo, en este caso me falta experiencia para saber cómo aplicar los test unitarios de forma coherente. Entiendo que es una herramienta imprescindible cuando el proyecto tiene cierta entidad, agilizando todas las pruebas y comprobando que la funcionalidad sigue estable después de las modificaciones.

En mi caso he implementado 2 test unitarios:

Comprobación de la creación de un usuario correctamente:

```
from django.test import TestCase
from django.contrib.auth import get_user_model

class CustomUserTests(TestCase):
    def test_create_user(self):
        User = get_user_model()
        user = User.objects.create_user(username='testusertest', email='testusertest@domain.com', password='testpass123')
        self.assertEqual(user.username, "testusertest")
        self.assertEqual(user.email, "testusertest@domain.com")
        self.assertTrue(user.is_active)
        self.assertFalse(user.is_staff)
        self.assertFalse(user.is_superuser)
```

Comprobación de la creación de un videojuego correctamente:

```
from django.test import TestCase
from django.contrib.auth import get_user_model
from models import Videogame, Platform, Format

class VideogameTests(TestCase):
    @classmethod
    def setUpTestData(cls):
        User = get_user_model()
        cls.user = User.objects.create_user(username='testusertest', email='testusertest@domain.com', password='testpass123')
        cls.platform = Platform.objects.create(name='PC', icon='platforms/icons/PC.png')
        cls.format = Format.objects.create(name='CD-ROM')

    def test_create_videogame(self):
        videogame = Videogame.objects.create(owner=self.user, title='Baldur's Gate', platform=self.platform, format=self.format, release_date='1998-12-21', front_cover='videogames/front_covers/BG.png',
                                             description='Baldur's Gate es un videojuego de rol de fantasía desarrollado por BioWare y publicado en 1998 por Interplay Entertainment.')
        self.assertEqual(videogame.owner, self.user)
        self.assertEqual(videogame.title, "Baldur's Gate")
        self.assertEqual(videogame.platform, self.platform)
        self.assertEqual(videogame.format, self.format)
        self.assertEqual(videogame.release_date, '1998-12-21')
        self.assertEqual(videogame.front_cover, 'videogames/front_covers/BG.png')
        self.assertEqual(videogame.description, "Baldur's Gate es un videojuego de rol de fantasía desarrollado por BioWare y publicado en 1998 por Interplay Entertainment.")
```

- **El proyecto debe encontrarse desplegado en producción:** Si bien al principio pensé en utilizar Digital Ocean como recomendaban en la clase destinada a despliegue, debido a problemas con el método de pago y a que mi situación económica no me permitía un desembolso mayor, al final he utilizado la opción gratuita de <https://www.pythonanywhere.com/> para desplegar en producción el proyecto, el cual se encuentra en: <https://pygamecol.pythonanywhere.com/>

Al no contar con experiencia previa utilizando esta plataforma, los conocimientos adquiridos durante el bootcamp han sido claves para poder desplegar este proyecto con éxito.

Cosas como crear el entorno virtual, instalar las dependencias, clonar el repositorio, etc son tareas a las que me he podido enfrentar trasladando los conocimientos adquiridos a otro sistema que, aunque con ciertas similitudes, no es como manejar tu propio VPS:



Bash console 27602606

```
12:08 ~/pyGAMEcol (main)$ mkvirtualenv myvirtualenv --python=/usr/bin/python3.10
```



Bash console 27602606

```
(myvirtualenv) 12:05 ~/pyGAMEcol (main)$ git clone https://github.com/sergiosano/pyGAMEcol.git
```



Bash console 27602606

```
12:08 ~/pyGAMEcol (main)$ python manage.py check --deploy
```



/var/www/pygamecol-pythonanywhere_com_wsgi.py

```
1 import os
2 import sys
3
4 path = os.path.expanduser('~/.pyGAMEcol')
5 if path not in sys.path:
6     sys.path.insert(0, path)
7 os.environ['DJANGO_SETTINGS_MODULE'] = 'pyGAMEcol.settings'
8 from django.core.wsgi import get_wsgi_application
9 from django.contrib.staticfiles.handlers import StaticFilesHandler
10 application = StaticFilesHandler(get_wsgi_application())
```

MySQL settings

Connecting:

Use these settings in your web applications.

Database host address:	pygamecol.mysql.pythonanywhere-services.com
Username:	pygamecol

- Debe existir un script para poder automatizar el deploy para cada nuevo release: Lamentablemente, Python Anywhere, si bien dispone de consolas bash, estas están basadas en servicios web, no es un VPS real, por lo que la parte de automatización del despliegue remoto no es posible. He intentado hacer el script con *fabric* para que funcionara aunque fuera ejecutándose directamente desde el servidor remoto, pero no hay acceso a los servicios por lo que no se pueden reiniciar ni interactuar con ellos. Con fines formativos, he realizado la misma prueba en un servidor basado en Linux implementado en una VM y todo ha funcionado sin problema.

Requerimientos para un sitio web

- **Implementar un sistema de autenticación:** Mi proyecto se apoya en los paquetes y módulos de Django para realizar el sistema de registro de usuarios y autenticación, existiendo una aplicación específica para ello llamada *users*.

Importación de los módulos necesarios

```
from django.shortcuts import render, redirect
from django.contrib import messages
from django.contrib.auth import login, logout, authenticate
from . import forms
```

Inicio de sesión: Vista basada en función

```
def user_login(request):
    if request.user.is_authenticated:
        return redirect('collection')

    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')

        user = authenticate(username=username, password=password)

        if user:
            login(request, user)
            return redirect('collection')
        else:
            messages.error(request, "Usuario y/o contraseña incorrectos.")

    return render(request, 'users/login.html')
```

Cierre de sesión: Vista basada en función

```
def user_logout(request):
    logout(request)
    return redirect('index')
```

Registro de usuario: Vista basada en función

```
def register(request):
    if request.user.is_authenticated:
        return redirect('collection')

    form = forms.RegisterForm(request.POST or None)

    if request.method == 'POST' and form.is_valid():
        user = form.save()

        if user:
            login(request, user)
            return redirect('collection')

    return render(request, 'users/register.html', {'form': form})
```

- **Implementar patrón MVC o MTV:** Utilizando Django, no sabría como no implementar su patrón Model-Template-View, creo que al estar basado en este modelo es obligatorio su uso.

Modelos

```
class Platform(models.Model):
    name = models.CharField(max_length=50, null=False, blank=False, verbose_name='Plataforma')
    icon = models.ImageField(upload_to='platforms/icons/', null=False, blank=False, default='platforms/icons/default_icon.png', verbose_name='Icono')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = 'Plataforma'
        verbose_name_plural = 'Plataformas'
        ordering = ['name']

class Format(models.Model):
    name = models.CharField(max_length=50, null=False, blank=False, verbose_name='Formato')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = 'Formato'
        verbose_name_plural = 'Formatos'
        ordering = ['name']

class Videogame(models.Model):
    owner = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='Propietario')
    title = models.CharField(max_length=100, null=False, blank=False, verbose_name='Título')
    platform = models.ForeignKey(Platform, on_delete=models.CASCADE, verbose_name='Plataforma')
    formato = models.ForeignKey(Format, on_delete=models.CASCADE, verbose_name='Formato') # format no disponible
    release_date = models.DateField(null=True, blank=True, verbose_name='Fecha de lanzamiento')
    front_cover = models.ImageField(upload_to='videogames/front_covers/', null=False, blank=False, default='videogames/front_covers/default_front_cover.png', verbose_name='Carátula')
    description = models.TextField(blank=True, default='', verbose_name='Descripción')
    slug = models.SlugField(null=False, blank=False, unique=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse_lazy('videogame_view', args=[str(self.slug)])

    class Meta:
        verbose_name = 'Videojuego'
        verbose_name_plural = 'Videojuegos'
        ordering = ['title']

def set_slug(sender, instance, *args, **kwargs):
    if instance.title and not instance.slug:
        slug = slugify(f'{str(uuid.uuid4())}')

        while Videogame.objects.filter(slug=slug).exists():
            slug = slugify(f'{str(uuid.uuid4())}')

        instance.slug = slug
```

Vistas

```
def videogame_create(request):
    if not request.user.is_authenticated:
        return redirect('index')

    form = VideogameCreateForm(request.POST or None, request.FILES or None)

    if request.method == 'POST' and form.is_valid():
        title = form.cleaned_data.get('title')
        platform = form.cleaned_data.get('platform')
        formato = form.cleaned_data.get('formato')
        release_date = form.cleaned_data.get('release_date')
        front_cover = form.cleaned_data.get('front_cover')
        if not front_cover:
            front_cover = 'videogames/front_covers/default_front_cover.png'
        description = form.cleaned_data.get('description')
        videogame = Videogame.objects.create(owner=request.user, title=title, platform=platform, formato=formato, release_date=release_date, front_cover=front_cover, description=description)

        if videogame:
            return redirect('collection')

    return render(request, 'videogames/videogame_create.html', {'form': form})

def videogame_update(request, slug):
    if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=slug).owner:
        return redirect('index')

    videogame = Videogame.objects.get(slug=slug)

    if request.method == 'POST':
        form = VideogameUpdateForm(request.POST or None, request.FILES or None, instance=videogame)
        if form.is_valid():
            form.save()
            return redirect('videogame_view', videogame.slug)
        else:
            form = VideogameUpdateForm(instance=videogame)

    return render(request, 'videogames/videogame_update.html', {'form': form, 'videogame': videogame})

class VideogameDeleteView(DeleteView):
    model = Videogame
    success_url = reverse_lazy("collection")
    template_name = "videogames/videogame_delete.html"

    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=kwargs['slug']).owner:
            return redirect('index')

        return super().dispatch(request, *args, **kwargs)
```

Templates

```
{% extends 'base/base.html' %}
{% load static %}
{% block title %}{{ videogame.title }} - pyGAMEcol{% endblock title %}
{% block navbar %}
<nav class="navbar-nav">
    <form class="form-inline" action="{% url 'logout' %}">
        <button class="btn btn-outline-danger" type="submit"><i class="fa-solid fa-user-xmark"></i> Cerrar sesión</button>
    </form>
</nav>
{% endblock navbar %}
{% block subnavbar %}
<nav class="navbar navbar-expand fixed-bottom navbar-dark bg-secondary">
    <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'collection' %}">
            Visualizar videojuego
        </a>
        <nav class="navbar-nav mx-auto">
            <strong>{{ videogame.title }}</strong> lleva en tu colección desde el {{ videogame.created_at }}
        </nav>
        <nav class="navbar-nav">
            <a href="{% url 'collection' %}" class="btn btn-success"><i class="fa-solid fa-house"></i> Colección</a>
        </nav>
    </div>
</nav>
{% endblock subnavbar %}
{% block body %}
<section class="vh-100">
    <div class="container py-5 h-100">
        <div class="row d-flex justify-content-center align-items-center h-100" style="margin-top: 25px;">
            <div class="col">
                <div class="card bg-dark text-white" style="border-radius: 1rem;">
                    <div class="card-body p-4 text-left">
                        <div class="mb-md-4 mt-md-1">
                            <h2 class="fw-bold mb-2 text-uppercase text-center">{{ videogame.title }}</h2>
                            <p class="text-white-50 mb-4 text-center fs-5"> {{ videogame.platform }}</p>
                            <div class="row d-flex justify-content-center align-items-center h-100">
                                <div class="card" style="width: 80rem;">
                                    
                                    <div class="card-body">
                                        <ul class="list-group list-group-flush">
                                            <li class="list-group-item"><strong>Título:</strong>{{ videogame.title }}</li>
                                            <li class="list-group-item"><strong>Plataforma:</strong>{{ videogame.platform }}</li>
                                            <li class="list-group-item"><strong>Formato:</strong>{{ videogame.formato }}</li>
                                            <li class="list-group-item"><strong>Fecha de lanzamiento:</strong>{{ videogame.release_date }}</li>
                                            <li class="list-group-item"><strong>Fecha de lanzamiento:</strong>{{ videogame.release_date }}</li>
                                            <li class="list-group-item"><strong>Descripción:</strong>{{ videogame.description }}</li>
                                        </ul>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

- **Implementar validaciones sobre los formularios:** El proyecto tiene varias validaciones para los campos de los formularios, desde comprobar si un usuario a registrar ya existe, si las contraseñas introducidas son iguales, si la longitud de los datos introducidos es correcta:

Comprobar si el usuario a registrar ya existe

```
def clean_username(self):
    username = self.cleaned_data.get('username')

    if User.objects.filter(username=username).exists():
        raise forms.ValidationError('Usuario previamente registrado.')

    return username
```

Comprobar si el correo electrónico fue previamente registrado

```
def clean_email(self):
    email = self.cleaned_data.get('email')

    if User.objects.filter(email=email).exists():
        raise forms.ValidationError('Correo electrónico previamente registrado.')

    return email
```

Comprobar si las contraseñas introducidas son iguales

```
def clean_passwordrpt(self):
    cleaned_data = super().clean()

    if cleaned_data.get('passwordrpt') != cleaned_data.get('password'):
        raise forms.ValidationError('Las contraseñas no coinciden.')
```

Validaciones implícitas de Django basadas en los tributos de las clases de los formularios (requerido, longitud, etc)

```
from django import forms
from .models import Videogame, Platform, Format

class VideogameCreateForm(forms.Form):
    title = forms.CharField(required=True, max_length=100, widget=forms.TextInput(attrs={'class': 'form-control form-control-lg', 'id': 'title'}))
    platform = forms.ModelChoiceField(queryset=Platform.objects.all().order_by('name'), widget=forms.Select(attrs={'class': 'form-control form-control-lg', 'id': 'platform'}))
    formato = forms.ModelChoiceField(queryset=Format.objects.all(), widget=forms.Select(attrs={'class': 'form-control form-control-lg', 'id': 'formato'}))
    release_date = forms.DateField(required=False, widget=forms.DateInput(attrs={'class': 'form-control form-control-lg text-center', 'id': 'release_date', 'type': 'date'}))
    front_cover = forms.ImageField(required=False, widget=forms.FileInput(attrs={'class': 'form-control form-control-lg', 'id': 'front_cover'}))
    description = forms.CharField(required=False, widget=forms.Textarea(attrs={'class': 'form-control form-control-lg', 'id': 'formato', 'rows': 4}))















class VideogameUpdateForm(forms.ModelForm):
    title = forms.CharField(required=True, max_length=100, widget=forms.TextInput(attrs={'class': 'form-control form-control-lg', 'id': 'title'}))
    platform = forms.ModelChoiceField(queryset=Platform.objects.all().order_by('name'), widget=forms.Select(attrs={'class': 'form-control form-control-lg', 'id': 'platform'}))
    formato = forms.ModelChoiceField(queryset=Format.objects.all(), widget=forms.Select(attrs={'class': 'form-control form-control-lg', 'id': 'formato'}))
    release_date = forms.DateField(required=False, widget=forms.DateInput(attrs={'class': 'form-control form-control-lg text-center', 'id': 'release_date', 'type': 'date', 'format': '%Y-%m-%d'}))
    front_cover = forms.ImageField(required=False, widget=forms.FileInput(attrs={'class': 'form-control form-control-lg', 'id': 'front_cover'}))
    description = forms.CharField(required=False, widget=forms.Textarea(attrs={'class': 'form-control form-control-lg', 'id': 'formato', 'rows': 4}))

class Meta:
    model = Videogame
    fields = ('title', 'platform', 'formato', 'release_date', 'front_cover', 'description')
```


Desarrollo

En los requerimientos están explicados la mayoría de los conceptos y soluciones adoptadas para el desarrollo, aquí explicaré algunos detalles adicionales.

Estructura

 sergiosano Update README.md	0789c74 2 hours ago	 13 commits
 collection	Add owner to admin	16 hours ago
 media	Initial commit	yesterday
 pyGAMEcol	Add owner to admin	16 hours ago
 static	Initial commit	yesterday
 templates	Initial commit	yesterday
 users	Add owner to admin	16 hours ago
 videogames	Add owner to admin	16 hours ago
 .gitattributes	Initial commit	yesterday
 .gitignore	Initial commit	yesterday
 README.md	Update README.md	2 hours ago
 manage.py	Initial commit	yesterday
 requirements.txt	Initial commit	yesterday

El proyecto principal consta de las siguientes aplicaciones:

- **pyGAMEcol**: Aplicación principal que incluye la página de presentación
- **collection**: Aplicación para visualizar la colección de videojuegos de cada usuario.
- **users**: Aplicación para gestionar la creación y autenticación de usuarios.
- **videogames**: Aplicación para gestionar la creación, edición, eliminación y visualización de los videojuegos pertenecientes a una colección.

Slugs

```
def set_slug(sender, instance, *args, **kwargs):
    if instance.title and not instance.slug:
        slug = slugify(f"{str(uuid.uuid4())}")

        while Videogame.objects.filter(slug=slug).exists():
            slug = slugify(f"{str(uuid.uuid4())}")

        instance.slug = slug

pre_save.connect(set_slug, sender=Videogame)
```

Para gestionar la visualización, edición, actualización y eliminación de los videojuegos, el proyecto utiliza slugs generados al crear un videojuego, usando UUID.

Seguridad

```
if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=slug).owner:
    return redirect('index')
```

Cada vez que se intenta acceder a un videojuego, se comprueba que el usuario esté tanto autenticado como que ese videojuego pertenezca a su colección.

DRY

```
{% load static %}
<!DOCTYPE html>
<html lang="es" data-bs-theme="dark">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}{% endblock title %}</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GlhtQR8d2L1603ovWMSkQ66b771n1Z13/Jr59b6E6G011aFkw7cMDA6J6GD" crossorigin="anonymous">
    <link rel="stylesheet" type="text/css" href="{% static 'css/autofill.css' %}">
  </head>
  <body>
    <nav class="navbar navbar-expand fixed-top navbar-dark bg-dark">
      <div class="container-fluid">
        <a class="navbar-brand" href="{% url 'index' %}">
          
          pyGAMEcol
        </a>
        {% block navbar %}{% endblock navbar %}
      </div>
    </nav>
    {% block subnavbar %}{% endblock subnavbar %}
    {% block body %}{% endblock body %}
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPFO0MB0x38jS15ge76pr3+5MLQ17A0C+nu2B+EYdgR2giwhtBTKf7CkxW" crossorigin="anonymous"></script>
    <script src="https://kit.fontawesome.com//414548d6.js" crossorigin="anonymous"></script>
  </body>
</html>
```

En la medida de lo posible, se han utilizado plantillas con los template tags adecuados para extender el código y no repetirlo.

Datos iniciales

```
def fill_initial_data():

    if len(Format.objects.all()) == 0:
        Format.objects.create(name='3DS Game Card')
        Format.objects.create(name='Blue-ray Disc')
        Format.objects.create(name='Cartucho ROM')
        Format.objects.create(name='Casete')
        Format.objects.create(name='CD-ROM')
        Format.objects.create(name='Digital')
        Format.objects.create(name='Disquete de 5¼")
        Format.objects.create(name='Disquete de 3½")
        Format.objects.create(name='DS Game Card')
        Format.objects.create(name='DVD')
        Format.objects.create(name='GameCube Game Disc')
        Format.objects.create(name='GD-ROM')
        Format.objects.create(name='PS Vita Card')
        Format.objects.create(name='Switch Game Card')
        Format.objects.create(name='Wii Optical Disc')
        Format.objects.create(name='Wii U Optical Disc')

    if len(Platform.objects.all()) == 0:
        Platform.objects.create(name='NES', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Game Boy', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Super Nintendo', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Virtual Boy', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Nintendo 64', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='GameBoy Color', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Nintendo GameCube', icon='platforms/icons/NintendoGameCube.svg')
        Platform.objects.create(name='GameBoy Advance', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Nintendo DS', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Wii', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Nintendo 3DS', icon='platforms/icons/Nintendo3DS.svg')
        Platform.objects.create(name='Wii U', icon='platforms/icons/Nintendo.svg')
        Platform.objects.create(name='Nintendo Switch', icon='platforms/icons/NintendoSwitch.svg')
        Platform.objects.create(name='Master System', icon='platforms/icons/Sega.svg')
        Platform.objects.create(name='Mega Drive', icon='platforms/icons/Sega.svg')
        Platform.objects.create(name='Game Gear', icon='platforms/icons/Sega.svg')
        Platform.objects.create(name='Saturn', icon='platforms/icons/Sega.svg')
        Platform.objects.create(name='Dreamcast', icon='platforms/icons/Sega.svg')
        Platform.objects.create(name='PC', icon='platforms/icons/PC.svg')
        Platform.objects.create(name='PlayStation', icon='platforms/icons/PlayStation.svg')
        Platform.objects.create(name='PlayStation 2', icon='platforms/icons/PlayStation2.svg')
        Platform.objects.create(name='PlayStation 3', icon='platforms/icons/PlayStation3.svg')
        Platform.objects.create(name='PlayStation 4', icon='platforms/icons/PlayStation4.svg')
        Platform.objects.create(name='PlayStation 5', icon='platforms/icons/PlayStation5.svg')
        Platform.objects.create(name='PSP', icon='platforms/icons/Sony.svg')
        Platform.objects.create(name='PS Vita', icon='platforms/icons/PSVita.svg')
        Platform.objects.create(name='XBOX', icon='platforms/icons/XBOX.svg')
        Platform.objects.create(name='XBOX 360', icon='platforms/icons/XBOX.svg')
        Platform.objects.create(name='XBOX One', icon='platforms/icons/XBOX.svg')
        Platform.objects.create(name='XBOX One X', icon='platforms/icons/XBOX.svg')
        Platform.objects.create(name='XBOX Series S', icon='platforms/icons/XBOX.svg')
```

Para facilitar su uso, se incluye una función que se ejecuta al visitar la página de inicio y crea un usuario de demostración (democol:democol23) y los formatos y plataformas más utilizados si estos no existen.

Uso

El uso de del sitio web es realmente sencillo, básicamente se compone de una figura de administrador que gestiona los usuarios, formatos y plataformas a través del panel de administración de Django y los usuarios, que pueden registrarse, iniciar sesión, y gestionar su colección de videojuegos pudiendo crearlos, eliminarlos y actualizarlos en cualquier momento.

Página de inicio



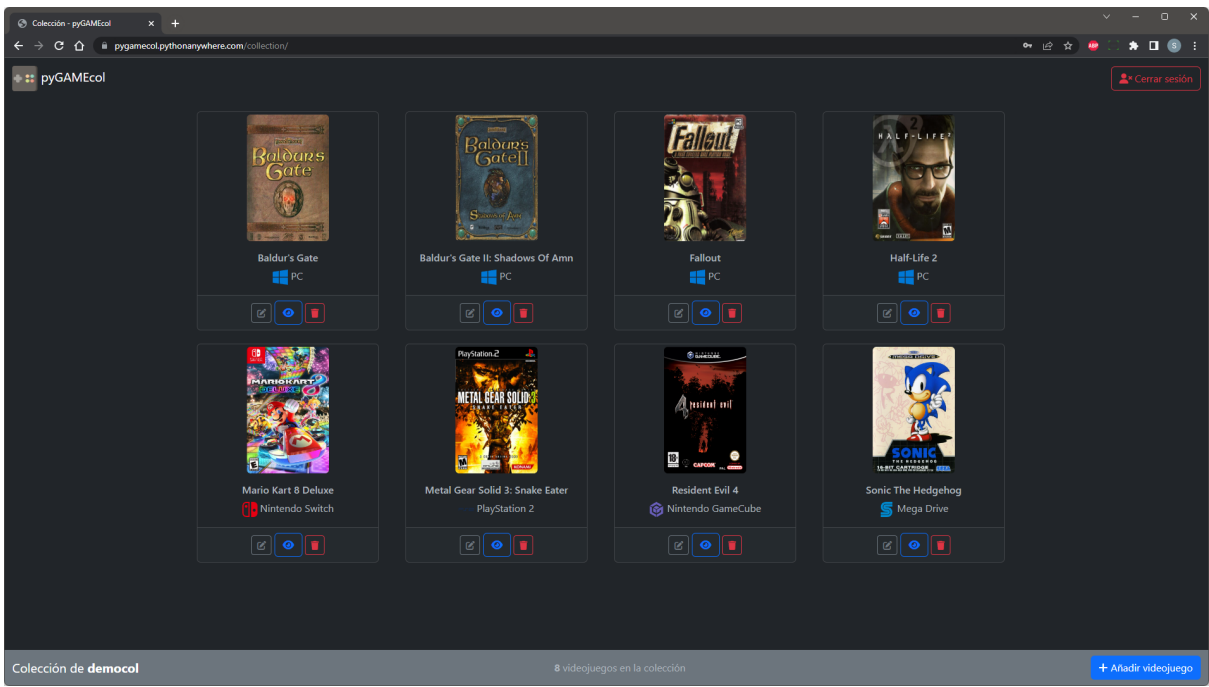
Registro

The screenshot shows a web browser window with the title "Registro - pyGAMEcol". The address bar displays "pygamecol.pythonanywhere.com/users/register/?". The page features a dark theme with a central registration form. The form is titled "REGISTRO" with the subtitle "Introduce tus datos". It contains four input fields: "Usuario", "Correo electrónico", "Contraseña", and "Confirmar contraseña". Below these fields is a blue button labeled "Registro". At the bottom of the form, there is a link that says "¿Dispones de una cuenta?" followed by a green button labeled "Iniciar sesión". The pyGAMEcol logo is visible in the top left corner, and a green "Iniciar sesión" button is in the top right corner.

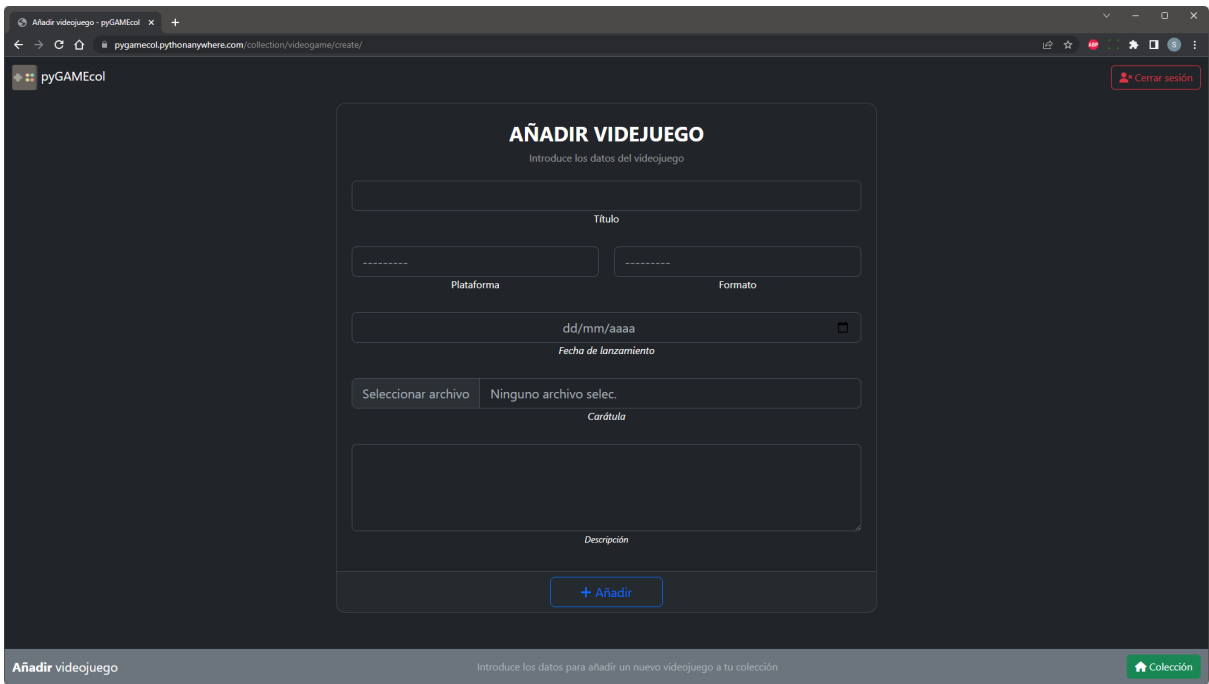
Inicio de sesión

The screenshot shows a web browser window with the title "Iniciar sesión - pyGAMEcol". The address bar displays "pygamecol.pythonanywhere.com/users/login/?". The page features a dark theme with a central login form. The form is titled "INICIAR SESIÓN" with the subtitle "Introduce tu usuario y contraseña". It contains two input fields: "Usuario" and "Contraseña". Below these fields is a green button labeled "Iniciar sesión". At the bottom of the form, there is a link that says "¿No dispones de una cuenta?" followed by a blue button labeled "Registro". The pyGAMEcol logo is visible in the top left corner, and a blue "Registro" button is in the top right corner.

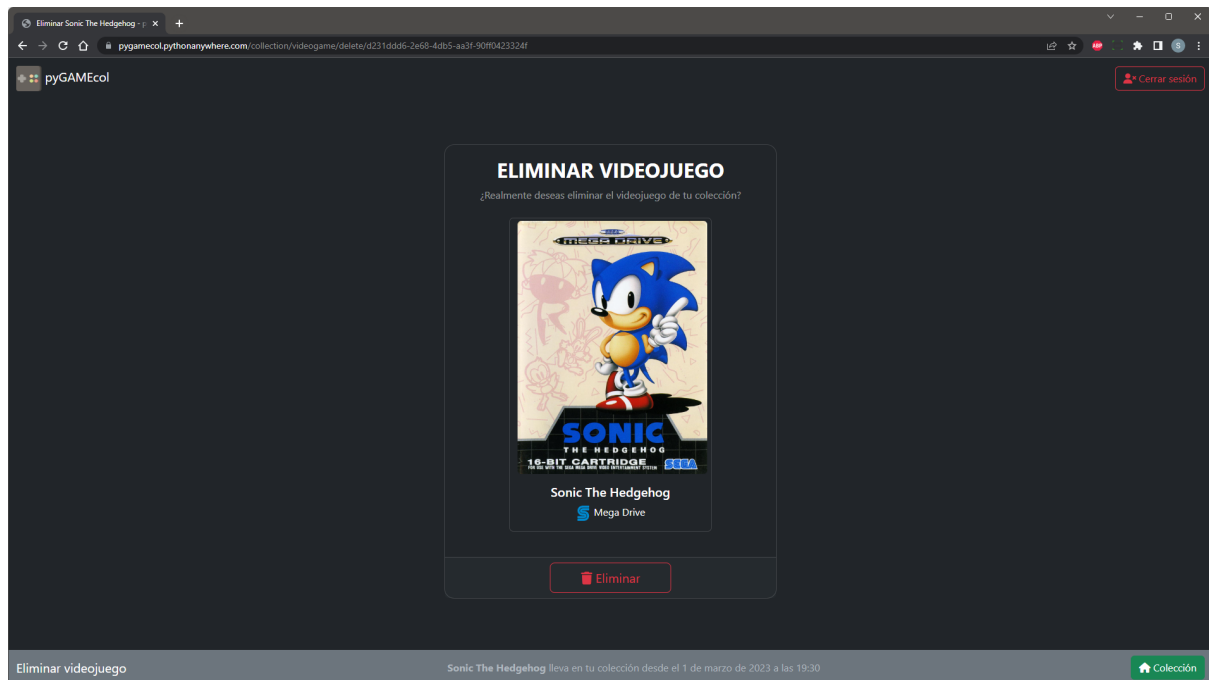
Colección



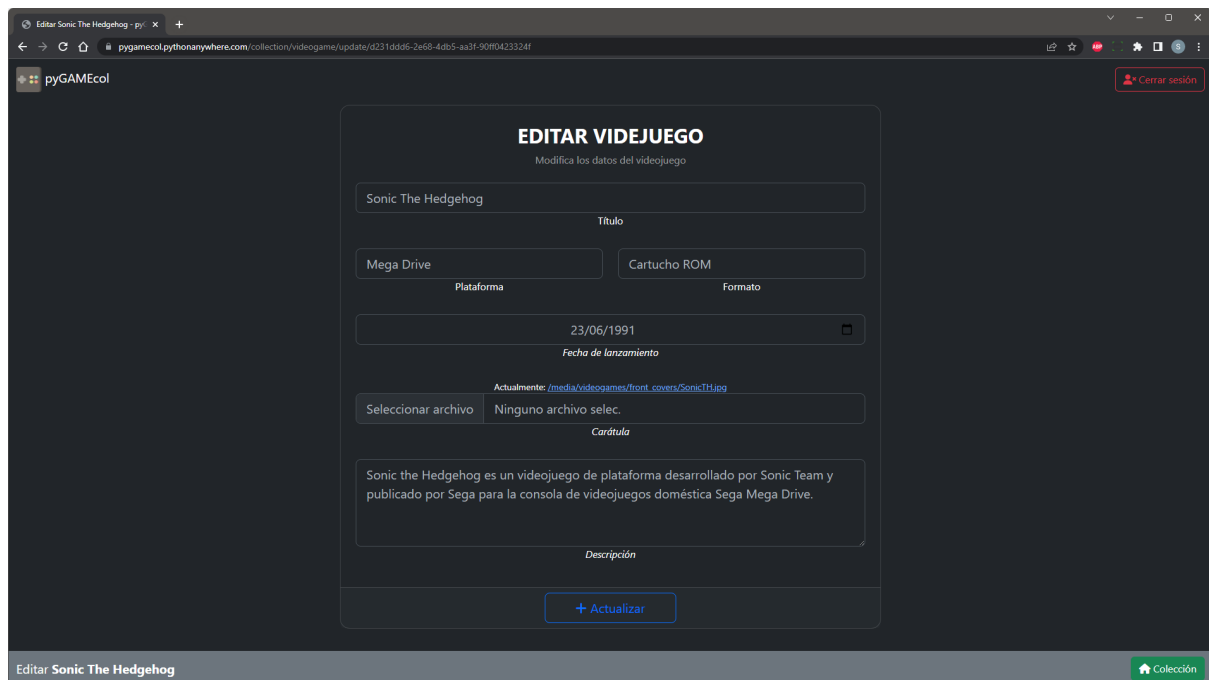
Añadir videojuego



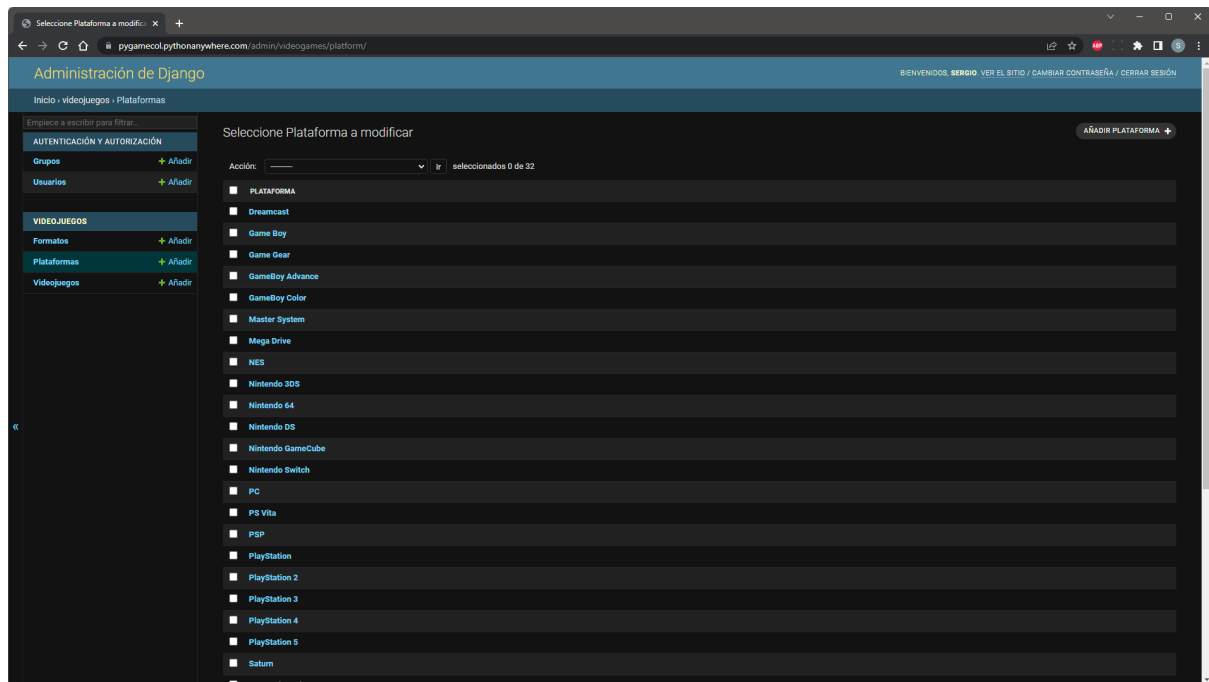
Eliminar videojuego



Editar videojuego



Edición de plataformas desde el panel de administración de Django



Dificultades

Más allá de las dificultades intrínsecas de enfrentarse a un proyecto sin conocimientos sólidos y del poco tiempo disponible, estas son algunas dificultades que me he encontrado y su solución.

Problema

Cualquier usuario autenticado podía acceder a un videojuego de otro usuario conociendo previamente el slug, teniendo la opción de editarlo o incluso eliminarlo.

Solución

```
if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=slug).owner:  
    return redirect('index')
```

Comprobar que el videojuego accedido pertenece realmente al usuario autenticado en la vista adecuada.

Problema

Vale pero, tengo algunas vistas basadas en clases, ¿cómo lo soluciono en estas?

Solución

```
def dispatch(self, request, *args, **kwargs):  
    if not request.user.is_authenticated or request.user != Videogame.objects.get(slug=kwargs['slug']).owner:  
        return redirect('index')  
  
    return super().dispatch(request, *args, **kwargs)
```

Sobreescribiendo el método *dispatch*.

Problema

Mis escasos conocimientos de HTML implican que la vista de la colección principal se muestre con márgenes incorrectos que cortan la visualización impidiendo el normal funcionamiento.

Solución

```
{% endfor %}
{% with 'center:4 as range %}
{% for _ in range %}
    <div></div>
{% endfor %}
{% endwith %}
```

Buscar una forma original de agregar márgenes dinámicos utilizando *template tags*.

Problema

Al editar un videojuego, si este disponía de fecha de lanzamiento y/o carátula previamente, no se mostraban en el formulario de edición

Solución

```
release_date = forms.DateField(required=False, widget=forms.DateInput(attrs={'class': 'form-control form-control-lg text-center', 'id': 'release_date', 'type': 'date', 'format': '%Y-%m-%d'}))
```

Añadir el formato de la fecha en el atributo del formulario.

```
<div class="form-outline form-white mb-4">
    {% if form.front_cover.errors %}
        {% for error in form.front_cover.errors %}
            <p class="small text-danger">{{ error }}</p>
        {% endfor %}
    {% endif %}
    <p class="small m-0">Actualmente: <a href="{{ videogame.front_cover.url }}" target="_blank">{{ videogame.front_cover.url }}</a></p>
    {{ form.front_cover }}
    <label class="form-label" for="front_cover"><em>Carátula</em></label>
</div>
```

Incluir el enlace a la carátula actual del videojuego encima del campo de selección de carátula en el formulario de actualización.

Problema

Los campos desplegables de *Plataforma* y *Formato* aparecen vacíos en los formularios de creación y edición del videojuego

Solución

```
platform = forms.ModelChoiceField(queryset=Platform.objects.all().order_by('name'), widget=forms.Select(attrs={'class': 'form-control form-control-lg', 'id': 'platform'}))
formato = forms.ModelChoiceField(queryset=Format.objects.all(), widget=forms.Select(attrs={'class': 'form-control form-control-lg', 'id': 'formato'}))
```

Utilizar una clase que permita pasar como argumento un *QuerySet*.
