Tienda Online

Llacma Quispe Kevin Andree
Universidad Nacional de San Agustín
Arequipa, Perú
Email: kllacma@unsa.edu.pe

***Abstract*—Keywords—Django, React, API Rest, SQLite, Heroku, Frontend, Backend, Despliegue, HTTPS**

I. Introducción

Este proyecto consiste en el desarrollo de una aplicación web que integra un backend creado con Django y un frontend con React. El objetivo principal es proporcionar una plataforma en línea para gestionar y visualizar productos implementando una arquitectura de API RESTful para la comunicación entre el frontend y el backend. La aplicación también se despliega en Heroku utilizando tecnologías como HTTPS para la seguridad y SQLite como base de datos.

II. Marco Teórico

A. Django

Un framework de desarrollo web de alto nivel en Python que promueve el desarrollo rápido y el diseño limpio y pragmático. Incluye un ORM para la gestión de bases de datos y una estructura modular que facilita la creación de aplicaciones web escalables [1].

B. React

Una biblioteca de JavaScript para construir interfaces de usuario desarrollada por Facebook. Utiliza un modelo de componentes y un DOM virtual para ofrecer una experiencia de usuario rápida y dinámica [2].

C. API Rest

Un estilo de arquitectura de software para sistemas distribuidos que utilizan HTTP para realizar operaciones CRUD (Create, Read, Update, Delete) en recursos web [3].

D. SQLite

Una base de datos ligera y embebida que se utiliza como base de datos de desarrollo por defecto en Django debido a su simplicidad y portabilidad [4].

E. Heroku

Una plataforma como servicio (PaaS) que permite desplegar y gestionar aplicaciones web en la nube de manera sencilla [5].

III. Desarrollo

Para el siguiente proyecto se dividió en 3 partes:

A. Configuración del Backend

Se definieron los modelos de datos en Django y se implementaron serializers y vistas API utilizando Django Rest Framework. Las rutas y endpoints se configuraron en `urls.py` para manejar la lógica de la aplicación.

Los `models.py`, al ser de una tienda online, tienen tablas de `User`, `Category`, `Product`, `Order`, `OrderDetail`, `ShoppingCart`, `CartDetail`, `Address`, `Payment`.

```python
class User(Time):

    username = models.CharField(max_length=50,unique=True)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=100)

    def __str__(self):
        return self.username
```

Fig. 1. UserModel

- Tabla `User` atributos: `username`, `email` y `password`

Se realizaron serializers para trabajar con las API que sirven para instanciar modelos.

```python
class UserSerializer(serializers.ModelSerializer):
    You, 3 weeks ago | 1 author (You)
    class Meta:
        model = User
        fields = ['id', 'username', 'email', 'password']
        extra_kwargs = {'password': {'write_only': True}}

    def create(self, validated_data):
        user = User.objects.create_user(**validated_data)
        return user
```

Fig. 2. UserSerializers

Agregamos las rutas `urls.py` para los endpoints, todos estos pasos se realizaron para manejar la lógica del proyecto. Agregamos archivos estáticos y configuramos `settings.py` para que pueda utilizarlos.

B. Desarrollo del Frontend

Para la realización del frontend se usó React para una mayor manejabilidad. Además de esto, se utilizó componentes esenciales como `ProductList` y `CreateProduct`.

```
import React, { useEffect, useState } from 'react'
```

```python
router = routers.DefaultRouter()
router.register(r'user', views.UserListView)
router.register(r'products', views.ProductListView)
router.register(r'categories', views.CategoryListView)
router.register(r'orders', views.OrderListView)
router.register(r'ordersDetail', views.OrderDetailView)
router.register(r'shoppingCarts', views.ShoppingCartView)
router.register(r'cartsDetail', views.CartDetailView)
router.register(r'address', views.AddressView)
router.register(r'payments', views.PaymentView)

urlpatterns = [
    path('app/', include(router.urls)),
    path('login/', views.login, name='login'),
    path('register/', views.register, name='register'),
]
```

Fig. 3. Urls

```
const ProductList = () => {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch('/api/products/')
      .then(response => {
        if (!response.ok) {
          throw new Error('Network response
        }
        return response.json();
      })
      .then(data => {
        setProducts(data);
        setLoading(false);
      })
      .catch(error => {
        console.error('There was a problem with the fetch operation:', error);
        setError(error);
        setLoading(false);
      });
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error.message}</div>;

  return (
    <div>
      <h1>Product List</h1>
      <ul>
        {products.map(product => (
          <li key={product.id}>{product.name} - ${product.price}</li>
        ))}
      </ul>
    </div>
  );
};

export default ProductList;
```

Luego, hacemos uso del comando `npm run build`. Esto nos crea una carpeta `build`. Se trató de utilizar React Router para la navegación.

C. Integración y Despliegue

Para el despliegue en Heroku se usaron archivos estáticos, además de `whitenoise` y `gunicorn`. Además, se creó un archivo `Procfile`, que es lo que Heroku va a leer, por lo que es importante tenerlo en el archivo. Debe ir esto:

```
web: gunicorn tienda_online.wsgi --log-file -
```

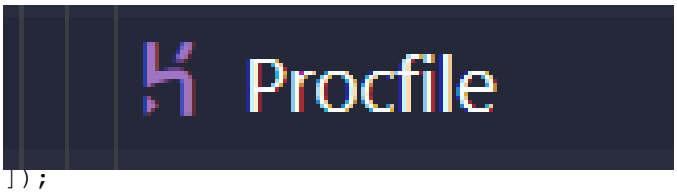

Fig. 4. Procfile

Adicionalmente, se tuvo que crear una cuenta en Heroku para el uso. Posteriormente, Heroku te guía cómo conectarse a su servicio, que puede ser a través de `git`.

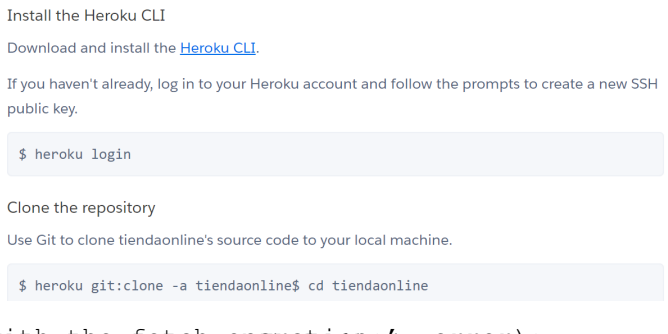

Fig. 5. githeroku

Como observamos en la imagen, se indican los pasos para conectarse usando Heroku con comandos `git`. Una vez corriendo el proyecto, podemos colocar en la consola `heroku open` y automáticamente se abrirá la aplicación en nuestro navegador:

https://tiendaonline-9c83954f4ca8.herokuapp.com/admin/

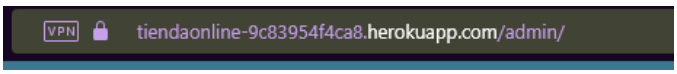

Fig. 6. link

Podemos ver en django Admin (admin : 1234) como podemos ver nuestro datos y cambiarlos (agregando o eliminando)

IV. Recomendaciones

Este trabajo tiene mucho rango de mejora ya que no se exploró de manera muy profunda el desarrollo de estos. Pero satisfecho con el resultado propongo las siguientes mejoras:

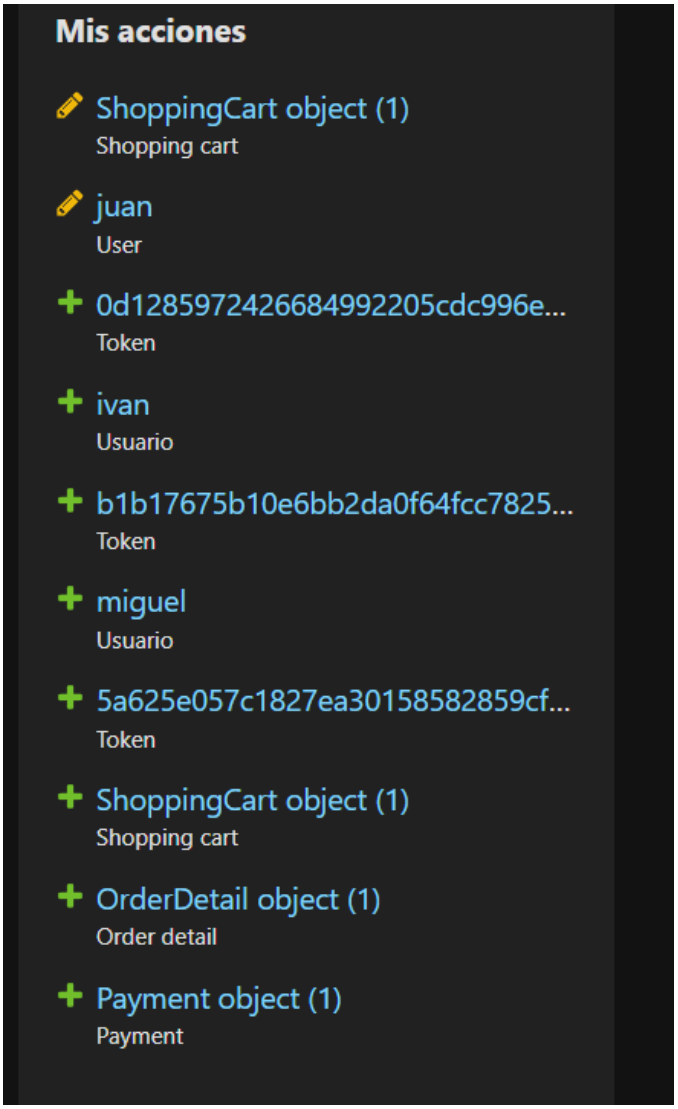

Fig. 7. admin registros

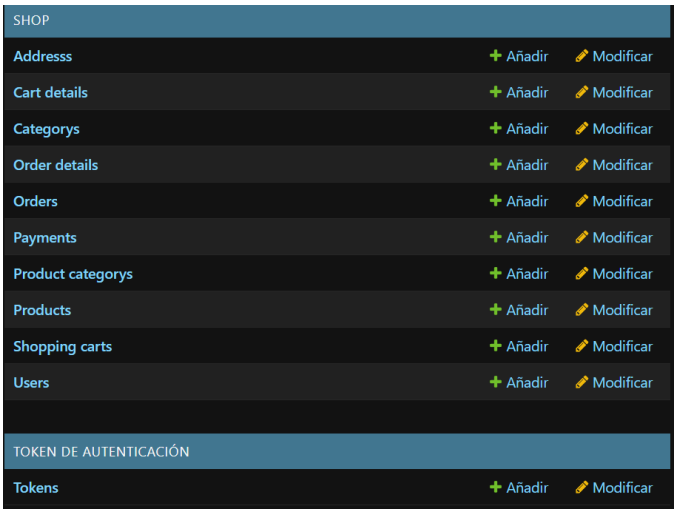

Fig. 8. admin tablas

- **Escalabilidad**: Migrar a una base de datos más robusta como PostgreSQL o MySQL para manejar mayores volúmenes de datos.
- **Seguridad**: Implementar autenticación y autorización para proteger los datos y las operaciones del sistema.
- **Optimización de rendimiento**: Considerar el uso de técnicas como la compresión de imágenes y el uso de CDN para mejorar los tiempos de carga.

V. Conclusiones

El proyecto permitió explorar la integración de tecnologías frontend y backend, demostrando la eficacia de Django y React en la construcción de una aplicación. El despliegue en Heroku proporcionó una plataforma accesible, aunque se identificaron áreas para mejorar, especialmente en términos de seguridad y rendimiento. Así mismo, gracias a este trabajo, se pudo mejorar las habilidades con esta tecnología, solucionando problemas que se tuvieron al momento del desarrollo.

References

[1] Tarantino, Q., Foxx, J., Waltz, C., & DiCaprio, L. (2013). *Django unchained*. Sony Pictures Home Entertainment.

[2] Gackenheimer, C. (2015). *Introduction to React*. Apress.

[3] Arsaute, A., Zorzán, F. A., Daniele, M., González, A., & Frutos, M. (2018). Generación automática de API REST a partir de API Java basada en transformación de Modelos (MDD). In *XX Workshop de Investigadores en Ciencias de la Computación (WICC 2018, Universidad Nacional del Nordeste)*.

[4] Owens, M., & Allen, G. (2010). *SQLite*. New York: Apress LP.

[5] Lee, B. H., Dewi, E. K., & Wajdi, M. F. (2018, April). Data security in cloud computing using AES under HEROKU cloud. In *2018 27th wireless and optical communication conference (WOCC)* (pp. 1-5). IEEE.