

**FUNDACIÓN UNIVERSITARIA DEL ÁREA ANDINA**

**Implementación de Framework de Desarrollo**

**Presentado por:**

Karina Quiroga

María Amaya

Laura Ramirez

Diego Murillo

**Presentado a:**

Deivys Morales

**ARQUITECTURA DE SOFTWARE Y SQA - IS - 202510-1A - 11**

**30 de Marzo del 2025**

**Bogotá Colombia**

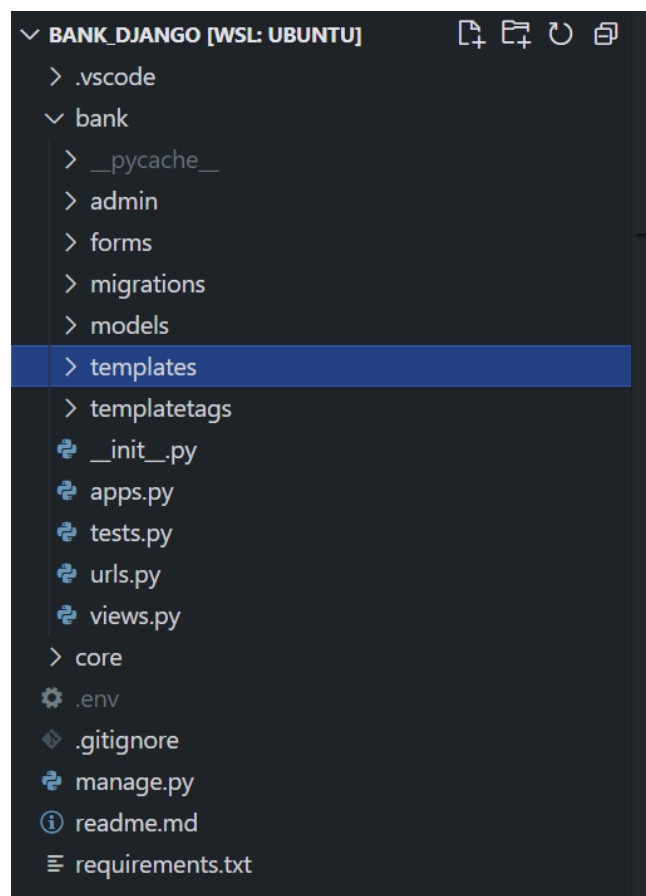
# Sistema Bancario Django

## Introducción

El presente documento tiene como objetivo describir la estructura y funcionalidad de un sistema bancario desarrollado utilizando el framework Django. Este sistema está diseñado para proporcionar una plataforma web donde los usuarios pueden gestionar sus cuentas bancarias y realizar diversas transacciones financieras de manera segura y eficiente.

## Arquitectura del Proyecto

La estructura del proyecto se organiza de la siguiente manera:



## Componentes Principales

**Modelos:** Los encargados de transformar las tablas de DB en objetos de programación

1. **Account (Cuenta):** Gestiona la información de las cuentas bancarias, incluyendo usuario, saldo y número de cuenta. Tiene relaciones con las transacciones.
2. **Transaction (Transacción):** Registra todas las operaciones bancarias, como monto, descripción y cuentas de origen/destino.

**Ejemplo de un modelo creado en Django:**

```
from django.db import models
from bank.models.account import Account
from bank.models.transaction_type import TransactionType

class Transaction(models.Model):
    id = models.AutoField(primary_key=True)
    amount_in_transaction = models.DecimalField(max_digits=10, decimal_places=2)
    description = models.TextField()
    id_from_account = models.ForeignKey(Account, related_name='sent_transactions', on_delete=models.CASCADE)
    id_to_account = models.ForeignKey(Account, related_name='received_transactions', on_delete=models.CASCADE)
    id_transaction_type = models.ForeignKey('TransactionType', on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def save(self, *args, **kwargs):
        """ Actualiza los saldos de las cuentas (variable) amount_in_transaction: Decimal
        if self.id_from_account.amount >= self.amount_in_transaction:
            self.id_from_account.amount -= self.amount_in_transaction
            self.id_to_account.amount += self.amount_in_transaction
            self.id_from_account.save()
            self.id_to_account.save()
            super().save(*args, **kwargs)
        else:
            raise ValueError("Saldo insuficiente en la cuenta de origen")

    class Meta:
        db_table = 'bank_transaction'

    def __str__(self):
        return f"Transaction {self.id} - {self.amount_in_transaction}"
```

**Formularios:** Los encargados de hacer las validaciones de un formulario antes de enviarlo.

1. **MakeTransactionForm:** Maneja la creación de nuevas transacciones, validando montos y cuentas, y filtrando cuentas por usuario.
2. **TopAccountForm:** Gestiona las recargas de cuenta, validando montos mínimos y manteniendo el campo de cuenta no editable.

### Ejemplo de un formulario creado en Django:

```
class RegistrationForm(UserCreationForm):
    email = forms.EmailField(required=True)
    first_name = forms.CharField(max_length=30, required=False)
    last_name = forms.CharField(max_length=30, required=False)
    identificacion = forms.CharField(max_length=20, required=True)

    class Meta:
        model = User
        fields = ("username", "email", "first_name", "last_name", "password1", "password2")

    def save(self, commit=True):
        user = super().save(commit=False)
        user.email = self.cleaned_data["email"]
        user.first_name = self.cleaned_data["first_name"]
        user.last_name = self.cleaned_data["last_name"]

        # Verificar si la identificación ya existe
        identificacion = self.cleaned_data["identificacion"]
        if UserProfile.objects.filter(identificacion=identificacion).exists():
            raise ValidationError("Ya existe un usuario con esta identificación")

        if commit:
            user.save()
            # Crear el perfil de usuario con la identificación
            UserProfile.objects.create(
                user=user,
                identificacion=identificacion
            )
        return user
```

**Vistas:** *El apartado que nos ayuda a hacer peticiones del front hacia el back*

1. **bank:** Vista principal del dashboard, mostrando el listado de cuentas del usuario e integrando el formulario de transacciones.
2. **make\_transaction:** Procesa nuevas transacciones, validando fondos y permisos, y mostrando mensajes de confirmación.
3. **top\_account:** Maneja recargas de cuenta, validando montos y actualizando saldos.

### Ejemplo del archivo views.py:

```
1 # Create your views here.
2 @login_required
3 def bank(request):
4     accounts = Account.objects.filter(user_id=request.user)
5     form = MakeTransactionForm(request.user)
6     context = {
7         'accounts': accounts,
8         'form': form,
9     }
10    return render(request, 'bank/bank.html', context)
11
12 def logout_view(request):
13     logout(request)
14     return redirect('bank:index')
15
16 def register_user(request):
17     if request.method == 'POST': (function) POST: Any
18         form = RegistrationForm(request.POST)
19         if form.is_valid():
20             try:
21                 form.save()
22                 return redirect('bank')
23             except ValidationError as e:
24                 form.add_error("identificacion", e.message)
25         else:
26             form = RegistrationForm()
27     return render(request, 'bank/register.html', {'form': form})
```

### *Templates: Las visuales que tendremos a nivel de frontend*

1. **bank.html:** Dashboard principal con listado de cuentas e integración de formularios.
2. **make\_transaction.html:** Formulario de transacciones con selección de cuentas y mensajes de error/éxito.
3. **top\_account.html:** Formulario de recarga con campo de cuenta bloqueado y diseño responsivo con Bootstrap.

## Flujos de Trabajo

### *Realizar una Transacción*

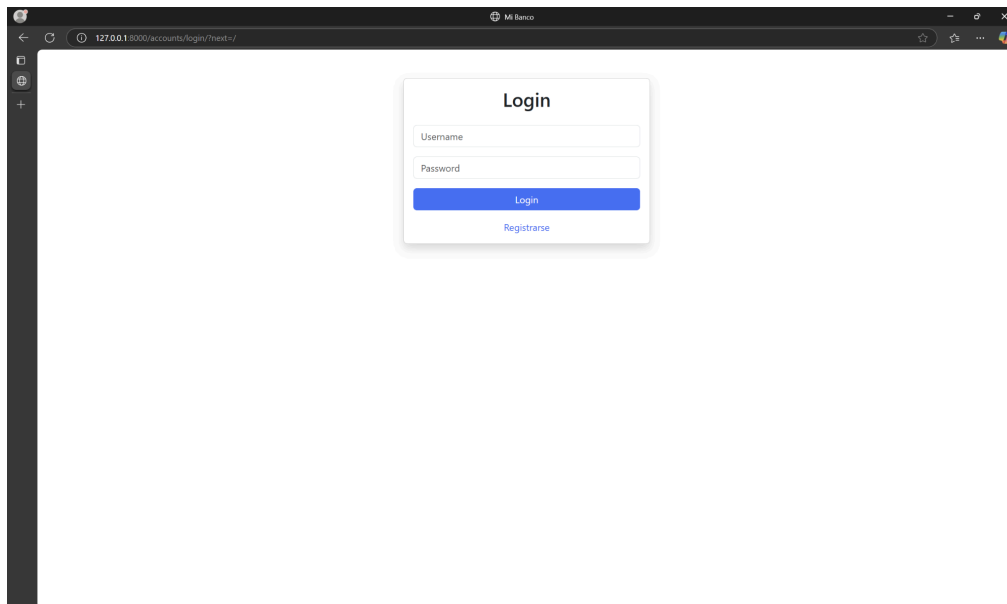
1. El usuario accede al dashboard.
2. Selecciona cuentas de origen y destino.
3. Ingresa monto y descripción.
4. El sistema valida la operación.
5. Actualiza saldos y registra la transacción.
6. Muestra mensaje de confirmación.

### *Recargar Cuenta*

1. El usuario selecciona la cuenta a recargar.
2. El sistema muestra el formulario con la cuenta bloqueada.
3. El usuario ingresa el monto.
4. El sistema valida el monto mínimo.
5. Actualiza el saldo.
6. Confirma la operación.

## Vistas en HTML. ¿Cómo quedaron?

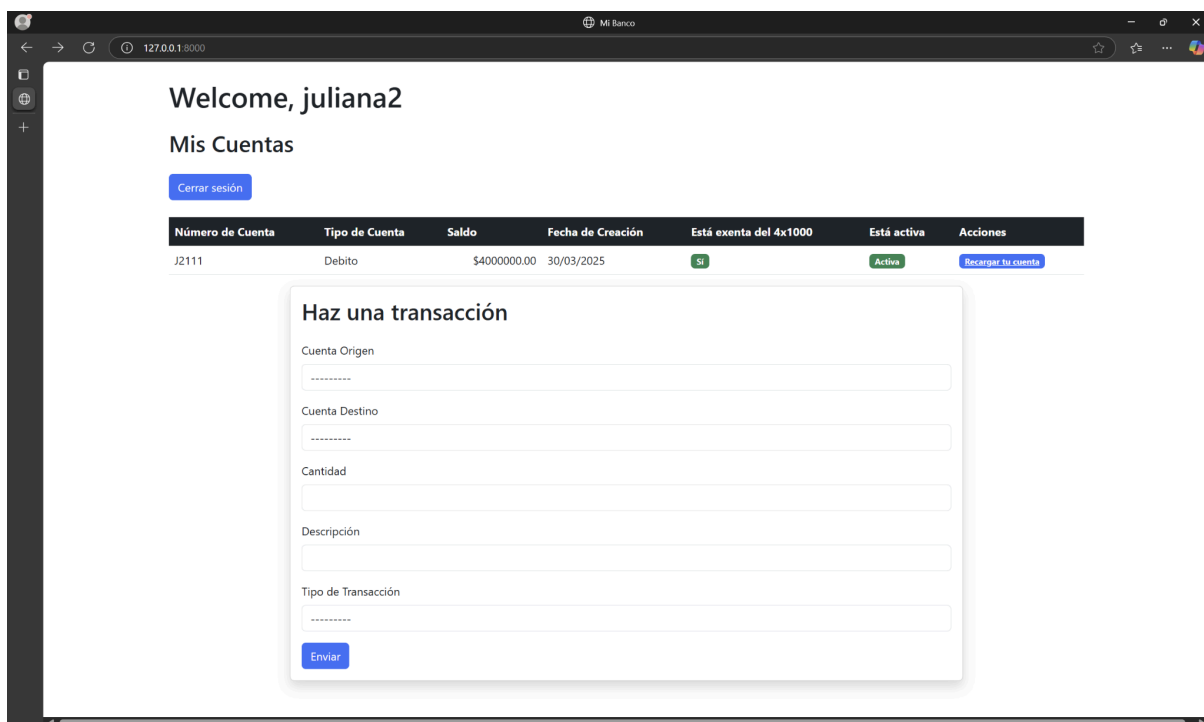
### *Login:*



A screenshot of a web browser window displaying a login form. The browser's address bar shows the URL "127.0.0.1:8000/accounts/login/?next=/". The login form is centered on the page and contains the following elements:

- A title "Login" in bold.
- A text input field labeled "Username".
- A text input field labeled "Password".
- A blue button labeled "Login".
- A blue link labeled "Regístrate" below the login button.

### *Dashboard del cliente:*



A screenshot of a web browser window displaying a client dashboard. The browser's address bar shows the URL "127.0.0.1:8000". The dashboard content includes:

- A greeting "Welcome, juliana2".
- A section titled "Mis Cuentas" with a blue button "Cerrar sesión".
- A table with the following columns: "Número de Cuenta", "Tipo de Cuenta", "Saldo", "Fecha de Creación", "Está exenta del 4x1000", "Está activa", and "Acciones".
- A single row of data: "J2111", "Debito", "\$4000000.00", "30/03/2025", "Si", "Activa", and a blue button "Recargar tu cuenta".
- A modal form titled "Haz una transacción" with the following fields: "Cuenta Origen", "Cuenta Destino", "Cantidad", "Descripción", and "Tipo de Transacción".
- A blue button "Enviar" at the bottom of the modal form.

*Apartado para recargar la cuenta del usuario:*

Recarga tu cuenta

Cantidad

Cuenta

DP102

Recargar cuenta

*Dashboard del administrador:*

Django administration

Site administration

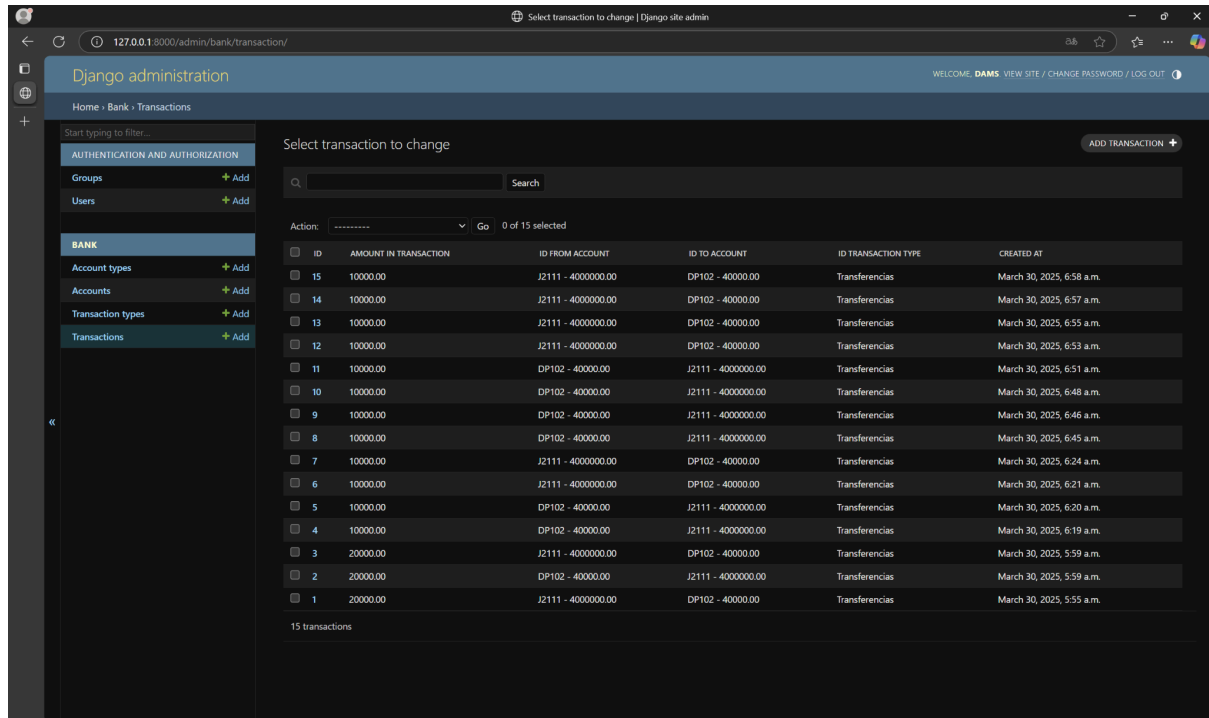
WELCOME, **DAMS** VIEW SITE / CHANGE PASSWORD / LOG OUT

Recent actions

My actions

- + Transferencias
  - Transaction type
  - DP102 - 0.00
    - Account
- + DP102 - 0.0
  - Account
- J2111 - 20000.00
  - Account
- J2111 - 20000.00
  - Account
- J2111 - 20000.00
  - Account
- J2111 - 10000.00
  - Account
- + J2111 - 0.0
  - Account
- ✖ J2111 - 0.00
  - Account
- + J2111 - 0.0
  - Account

## Seguimiento de transacciones por parte del administrador:



## Tecnologías Utilizadas

- Django
- Bootstrap
- PostgreSQL
- JavaScript (mensajes temporales)

## Conclusión

Este sistema bancario desarrollado con Django ofrece una solución robusta y segura para la gestión de cuentas y transacciones bancarias. La arquitectura modular y las características implementadas facilitan la administración y el uso del sistema, proporcionando una experiencia de usuario eficiente y confiable.

## Referencias

- Django Documentation. <https://docs.djangoproject.com/en/5.1/>
- Bootstrap Documentation. <https://getbootstrap.com/docs/4.1/getting-started/introduction/>
- Repositorio del proyecto: [https://github.com/DiegoMurillo7536/bank\\_with\\_django](https://github.com/DiegoMurillo7536/bank_with_django)