### **Documento de Planificación y Arquitectura**

#### **Portada**

ShieldBank

Sistema de Detección de Transacciones Anómalas

**Documento de Planificación y Arquitectura**

**Versión 1.0**

**Fecha:** 30 de agosto de 2025

**Elaborado por:**

Carlos Macias Leyton

Ronny Dario Torres

#### **1. Introducción al Proyecto**

##### **1.1. Problema**

Las transacciones fraudulentas son una amenaza constante para la seguridad bancaria, causando pérdidas financieras y erosionando la confianza de los clientes. Los sistemas de seguridad tradicionales a menudo no se adaptan a las nuevas tácticas de fraude, lo que hace crucial tener un sistema proactivo que identifique actividades sospechosas.

##### **1.2. Solución y Enfoque**

La solución es **ShieldBank**, un sistema que detecta anomalías en las transacciones bancarias. Nuestro enfoque se basa en el **análisis de datos** para identificar patrones de comportamiento inusuales, usando **Pandas** para procesar archivos CSV y una interfaz en **Django** para la visualización. El sistema se basa en reglas personalizables para filtrar transacciones inusuales (monto alto, país diferente, horario extraño) y almacenar datos en una base de datos **SQL**.

##### **1.3. Nombre del Proyecto: ShieldBank**

El nombre **ShieldBank** evoca protección y confianza. "Shield" (escudo) simboliza una capa de defensa contra el fraude, mientras que "Bank" lo asocia directamente a la industria financiera.

#### **2. Arquitectura del Sistema**

##### **2.1. Arquitectura de Capas**

**ShieldBank** sigue una arquitectura de tres capas, utilizando el patrón **MVT (Model-View-Template)** de Django para mantener la separación de responsabilidades.

* **Capa de Presentación:** La interfaz de usuario.
* **Capa de Lógica del Negocio:** El "cerebro" del sistema, donde reside toda la lógica de Python y Django.
* **Capa de Datos:** Almacenamiento persistente de la información.

##### **2.2. Componentes y Flujo de Trabajo**

El flujo de trabajo principal es el siguiente:

1. Un usuario sube un archivo CSV a través de la interfaz web.
2. El **CSVReader** lo procesa y limpia con Pandas.
3. El **FraudDetector** aplica reglas para identificar anomalías.
4. Los datos y las alertas se almacenan en la base de datos SQL.
5. El **StatisticsReporter** consulta la base de datos para generar gráficos y métricas que se muestran en el panel de control.

##### **2.3. Diagrama de Clases UML**

@startuml  
skinparam handwritten true  
skinparam style strict  
  
' Agrupación de Clases de Usuarios '  
package "Módulo de Usuarios" {  
 class User {  
 + username: string  
 + password: string  
 + email: string  
 + role: string  
 --  
 + login()  
 + logout()  
 + change\_password()  
 }  
  
 class Admin {  
 --  
 + create\_user(user\_data)  
 + delete\_user(user\_id)  
 + assign\_role(user\_id, new\_role)  
 }  
  
 class Analyst {  
 --  
 + review\_alerts()  
 + update\_alert\_status(alert\_id, new\_status)  
 }  
  
 User <|-- Admin  
 User <|-- Analyst  
}  
  
' Agrupación de Clases de Transacciones y Clientes '  
package "Módulo de Transacciones" {  
 class Client {  
 + client\_id: int  
 + name: string  
 + country: string  
 + registration\_date: date  
 }  
  
 class Transaction {  
 + transaction\_id: int  
 + client\_id: int  
 + amount: float  
 + date: date  
 + time: time  
 + location: string  
 }  
  
 Client "1" -- "1..\*" Transaction : tiene >  
}  
  
' Agrupación de Detección y Alertas '  
package "Módulo de Alertas" {  
 class Rule {  
 + rule\_id: int  
 + name: string  
 + threshold: float  
 --  
 + check\_transaction(transaction)  
 }  
  
 class Alert {  
 + alert\_id: int  
 + transaction\_id: int  
 + reason: string  
 + status: string  
 + date\_created: datetime  
 }  
}  
  
' Agrupación de Lógica y Estadísticas '  
package "Módulo de Lógica" {  
 class CSVReader {  
 --  
 + read\_csv(file\_path)  
 + clean\_data(dataframe)  
 }  
  
 class FraudDetector {  
 --  
 + analyze\_transactions(transactions)  
 + apply\_rule(rule)  
 }  
  
 class StatisticsReporter {  
 --  
 + get\_total\_transactions()  
 + get\_alert\_count\_by\_status()  
 + get\_top\_fraud\_locations()  
 + get\_average\_fraud\_amount()  
 }  
}  
  
' Componente de Interfaz de Usuario (Vistas de Django) '  
component "Vistas (Django)" as Views {  
 ' Las vistas orquestan la interacción con todos los módulos  
}  
  
' Relaciones entre componentes '  
Transaction "1" -- "0..1" Alert : genera >  
Alert "1" o-- "1" Rule : activada por >  
  
' Relaciones clave que demuestran el acceso del Admin '  
Admin .> Views : interactúa para\ngestionar usuarios y todo el sistema >  
Analyst .> Views : interactúa para\nrevisar alertas y reportes >  
  
' Las Vistas son el punto central de control '  
Views ..> CSVReader : usa para subir y procesar datos  
Views ..> FraudDetector : orquesta el análisis de fraude  
Views ..> StatisticsReporter : solicita informes y métricas  
Views ..> Transaction : interactúa con el modelo  
Views ..> Alert : interactúa con el modelo  
  
CSVReader .> FraudDetector : provee datos  
FraudDetector "1" o-- "1..\*" Rule : usa >  
StatisticsReporter .> Transaction : consulta datos  
StatisticsReporter .> Alert : consulta datos  
  
@enduml

#### **3. Desglose de Clases y Módulos**

Esta sección detalla cada clase con sus atributos, métodos y las relaciones que mantiene con otras clases, reflejando el diseño del diagrama UML.

##### **3.1. Módulo de Usuarios**

* **User**: Clase base para cualquier usuario que interactúe con el sistema. Proporciona la funcionalidad básica de autenticación.
* **Atributos:**
  + username (string): Identificador único para el inicio de sesión.
  + password (string): Credencial de seguridad (hash).
  + email (string): Correo electrónico del usuario.
  + first\_name (string): Nombre de pila.
  + last\_name (string): Apellido.
  + role (string): Rol asignado al usuario (e.g., 'Admin', 'Analyst').
* **Métodos:**
  + login(): Autentica al usuario en el sistema.
  + logout(): Cierra la sesión activa del usuario.
  + change\_password(): Permite al usuario modificar su contraseña.
* **Relaciones:** Es la clase de la que heredan Admin y Analyst.

##### **3.1.2. Clase: Admin**

* **Descripción:** Representa a un usuario con privilegios administrativos totales, incluyendo la gestión de otros usuarios y el acceso a todas las funcionalidades del sistema a través de las Vistas.
* **Relación:** Hereda de la clase User.
* **Métodos:**
  + create\_user(user\_data): Crea un nuevo usuario en el sistema.
  + delete\_user(user\_id): Elimina un usuario existente por su ID.
  + assign\_role(user\_id, new\_role): Modifica el rol y, por ende, los permisos de un usuario.

##### **3.1.3. Clase: Analyst**

* **Descripción:** Representa a un usuario con rol de analista, encargado de revisar, clasificar y gestionar las alertas de fraude generadas por el sistema.
* **Relación:** Hereda de la clase User.
* **Métodos:**
  + review\_alerts(): Accede al panel de alertas para su revisión.
  + update\_alert\_status(alert\_id, new\_status): Cambia el estado de una alerta (e.g., 'Pending' a 'Reviewed').

##### **3.2. Módulo de Transacciones**

* **Client**: Entidad que representa a un cliente del banco, almacenando su información básica.
* **Atributos:**
  + client\_id (int): Identificador único del cliente.
  + name (string): Nombre completo del cliente.
  + country (string): País de residencia del cliente.
  + registration\_date (date): Fecha de registro del cliente en el banco.
* **Relación:** Tiene una relación de uno a muchos (1..\*) con la clase Transaction. Un Client puede tener múltiples Transaction.

##### **3.2.2. Clase: Transaction**

* **Descripción:** Representa una transacción bancaria individual con todos sus detalles relevantes para el análisis.
* **Atributos:**
  + transaction\_id (int): Identificador único de la transacción.
  + client\_id (int): Clave foránea que enlaza la transacción con el cliente al que pertenece.
  + amount (float): Monto de la transacción.
  + date (date): Fecha en que se realizó la transacción.
  + time (time): Hora en que se realizó la transacción.
  + location (string): País o ciudad donde se originó la transacción.
* **Relación:** Puede tener una relación de uno a cero o uno (0..1) con la clase Alert. Una Transaction puede generar una o ninguna Alert.

##### **3.3. Módulo de Detección y Alertas**

* **Rule**: Clase abstracta o base para definir las reglas de negocio utilizadas en la detección de fraude.
* **Atributos:**
  + rule\_id (int): Identificador único de la regla.
  + name (string): Nombre descriptivo de la regla (e.g., 'Monto Alto', 'Ubicación Inusual').
  + threshold (float): Valor límite o umbral que activa la regla.
* **Métodos:**
  + check\_transaction(transaction): Método abstracto que evalúa si una Transaction cumple con los criterios de la regla. Las reglas específicas (e.g., HighAmountRule) heredarán y implementarán este método.
* **Relación:** Es utilizada por la clase FraudDetector (relación de agregación o--).

##### **3.3.2. Clase: Alert**

* **Descripción:** Registra una notificación generada por el sistema cuando una Transaction se clasifica como sospechosa.
* **Atributos:**
  + alert\_id (int): Identificador único de la alerta.
  + transaction\_id (int): Clave foránea que enlaza la alerta con la Transaction que la originó.
  + reason (string): Descripción del motivo por el cual se generó la alerta.
  + status (string): Estado actual de la alerta (e.g., 'Pending', 'Reviewed', 'Confirmed Fraud', 'False Positive').
  + date\_created (datetime): Fecha y hora de la creación de la alerta.
* **Relación:** Se asocia a una Transaction y también a la Rule que la activó (relación de asociación o--).

##### **3.4. Módulo de Lógica y Procesamiento**

* **FraudDetector**: Componente central que orquesta la aplicación de las reglas de detección de fraude sobre un conjunto de transacciones.
* **Métodos:**
  + analyze\_transactions(transactions): Recibe un DataFrame de transacciones y las procesa para identificar anomalías.
  + apply\_rule(rule): Aplica una regla específica al conjunto de transacciones.
* **Relación:** Agrega (o--) múltiples clases Rule para realizar su análisis.

##### **3.4.2. Clase: CSVReader**

* **Descripción:** Encargado de la lectura y el preprocesamiento inicial de los archivos CSV que contienen los datos transaccionales.
* **Métodos:**
  + read\_csv(file\_path): Lee un archivo CSV desde una ruta específica y lo carga en un DataFrame de Pandas.
  + clean\_data(dataframe): Realiza tareas de limpieza de datos, como manejo de valores nulos, corrección de tipos de datos y estandarización.
* **Relación:** Tiene una relación de dependencia (..>) con FraudDetector, ya que provee los datos que el detector analizará.

##### **3.5. Módulo de Estadísticas**

* **StatisticsReporter**: Clase dedicada a la generación de informes, métricas y estadísticas relevantes del sistema, útiles para la supervisión y toma de decisiones.
* **Métodos:**
  + get\_total\_transactions(start\_date, end\_date): Calcula el número total de transacciones en un periodo dado.
  + get\_alert\_count\_by\_status(): Devuelve un recuento de alertas agrupadas por su estado (e.g., 'Pending': 50, 'Reviewed': 20).
  + get\_top\_fraud\_locations(): Identifica las ubicaciones geográficas con mayor incidencia de alertas.
  + get\_average\_fraud\_amount(): Calcula el monto promedio de las transacciones que generaron alertas.
  + generate\_report(report\_type): Genera un informe estructurado en un formato específico (e.g., JSON, PDF).
* **Relación:** Tiene una relación de dependencia (..>) con las clases Transaction y Alert, ya que consulta sus datos para generar las estadísticas.

##### **3.6. Componente de Interfaz de Usuario (Vistas Django)**

* **Descripción:** Este componente actúa como el punto de interacción entre los usuarios (Admin, Analyst) y la lógica de negocio. Las vistas de Django procesan las solicitudes HTTP, controlan el flujo de la aplicación, gestionan la autenticación y autorización, e interactúan con todos los módulos de lógica y la base de datos para preparar los datos que se mostrarán en las plantillas HTML.
* **Relaciones:**
  + Admin y Analyst interactúan (.>) directamente con las Vistas.
  + Las Vistas usan (..>) o dependen (..>) de CSVReader, FraudDetector, StatisticsReporter, y los modelos de Transaction y Alert para operar.

#### **4. Mockups (Diseño Conceptual)**

* **Página de Inicio (Login)**: Un diseño simple y profesional con los campos de inicio de sesión y el logo de ShieldBank.
* **Panel de Control (Dashboard)**: La página principal que muestra indicadores clave de rendimiento (KPIs), gráficos de tendencias de alertas y resúmenes de transacciones recientes.
* **Gestión de Transacciones**: Una tabla interactiva y paginada con opciones de filtrado y búsqueda. Las transacciones sospechosas están resaltadas.
* **Página de Detalles de Alerta**: Una vista dedicada a una alerta específica, mostrando todos los datos de la transacción, el motivo de la alerta y opciones para que el analista actualice el estado.
* **Gestión de Usuarios (Solo para Administradores)**: Una tabla con la lista de usuarios, sus roles y botones para crear, editar o eliminar usuarios.