



**Universidad
Tecnológica
del Perú**

FACULTAD DE INGENIERÍA

TÍTULO DEL PROYECTO:

Plataforma de Inclusión Financiera
(ODS 8: Trabajo Decente y Crecimiento
Económico)

CURSO:

Diseño de Patrones

DOCENTE:

Jorge Alfredo Guevara Jimenez

SECCIÓN:

21646

INTEGRANTES:

Alvarez Samaniego, Luis Carlos Jaren U23218265

Vidal Alamo, Frank Edu U22304185

PolliCo (Asistente Virtual)

2024

Índice

Capítulo 1: Introducción	3
1.1 Objetivo de Desarrollo Sostenible (ODS 8).....	4
1.2 Problema Identificado.....	4
1.3 Solución Propuesta.....	4
1.4 Objetivos del Proyecto	4
Objetivo general	4
Objetivos específicos	5
1.5 Marco Teórico	5
Unidad I: Principios de Diseño y Patrones de Software	5
Unidad II: Patrones de Diseño Creacionales.....	6
Unidad III: Patrones de Diseño Estructurales	6
Unidad IV: Patrones de Comportamiento y Antipatrones.....	6
1.6 Antecedentes de Investigación.....	7
Capítulo 2: Desarrollo.....	8
2.1 Diagrama de Descomposición Modular	8
2.2 Cuadro de Resultados por Versiones y Semana	9
2.2.1 Versión 1: Capacitación	9
2.2.1.1 Usuario.....	9
2.2.1.2 Curso	9
2.2.2 Versión 2: Préstamo	9
2.2.2.1 Usuario.....	9
2.2.2.2 Préstamo.....	9
2.2.3 Versión 3: Seguimiento.....	10
2.2.3.1 Usuario.....	10
2.2.3.2 Seguimiento Financiero	10
2.3 Diseño de Arquitectura de Software a 4 Capas.....	10
2.3.1 Diagrama de Arquitectura de Software	10
2.3.2 Diagrama Base de Datos	11
Modelo Lógico.....	11
Modelo Físico	11
2.3.3 Modelo MVC (Modelo – Vista – Controlador).....	12
Diagrama de Modelo.....	12
Diagrama de Vista.....	13
Diagrama de Controlador.....	13
2.5 Programación Java en NetBeans	14
2.5.1 Capa Modelo (Patrones Creacionales)	16

Singleton	16
Prototype	17
2.5.2 Capa Controlador (Patrones Creacionales)	18
Factory	18
Facade	19
Proxy	20
2.5.3 Capa Vista	21
Decorator.....	21
Observer.....	23
COMMAND	24
COMPOSITE.....	25
2.5.4 Estructura del Proyecto en NetBeans	26
2.6. Programación de Base de Datos, Tablas y Store-Procedure	27
2.6.1 Módulo Capacitación	27
2.6.2 Módulo Préstamo	28
2.6.3 Módulo Seguimiento.....	28
2.6.4 Procedimientos Almacenados	29
2.7 Definición de roles y accesos.....	30
2.8 Interfaz Visual	31
2.8.1 Wireframes	31
2.8.2 Mockup Visual con Paleta de Colores	35
2.8.3 Aplicación de colores	39
2.8.4 Aplicación de los 10 principios de usabilidad.....	39
2.9 Pruebas del software	40
2.9.1 Prueba en consola del Módulo Capacitación	40
2.9.2 Prueba de interfaz.....	41
2.9.3 Prueba del antes y después	¡Error! Marcador no definido.
2.10 Antipatrones	43
2.10.1 Concepto y propósito	43
2.10.2 Antipatrones de desarrollo de software	43
2.10.3 Antipatrones organizacionales	43
2.10.4 Hoja de verificación	¡Error! Marcador no definido.
Capítulo 3: Conclusiones	44
3.1 Conclusiones por cada capa de arquitectura.....	45
3.2 Conclusiones personales	45
Referencias Bibliográficas	46
Anexos	46

GitHub.....	46
Drive.....	46
Imágenes	¡Error! Marcador no definido.

Índice Tablas

Tabla 1 <i>Tabla de Roles de la Plataforma Financiera</i>	30
Tabla 2 <i>Tabla de Psicología de Colores</i>	39
Tabla 3 <i>Tabla de Antipatrones Encontrados en el Sistema.....</i>	43
Tabla 4 <i>Tabla de Antipatrones Organizacionales Aplicados</i>	43

Índice Figuras

Figura 1 <i>Diagrama de Descomposición Modular de FinNexus.....</i>	8
Figura 2 <i>Diagrama de Arquitectura de Software de FinNexus.....</i>	10
Figura 3 <i>Diagrama Lógico de la Base de Datos</i>	11
Figura 4 <i>Diagrama Físico de la Base de Datos</i>	11
Figura 5 <i>Diagrama UML del Paquete Modelo</i>	12
Figura 6 <i>Diagrama UML de la Capa Modelo.Procedimientos.....</i>	12
Figura 8 <i>Diagrama UML del Paquete Vista</i>	13
Figura 7 <i>Diagrama UML del Paquete PersonalizacionVista</i>	13
Figura 9 <i>Diagrama UML del Paquete Controlador.....</i>	13

Capítulo 1: Introducción

1.1 Objetivo de Desarrollo Sostenible (ODS 8)

El proyecto se enmarca en el Objetivo de Desarrollo Sostenible (ODS) N° 8: Trabajo decente y crecimiento económico, establecido por la Organización de las Naciones Unidas. Este ODS busca promover el crecimiento económico sostenido, inclusivo y sostenible, el empleo pleno y productivo, y el trabajo decente para todos. Entre sus metas se incluye el apoyo a la productividad a través de la diversificación, la innovación y el acceso a servicios financieros, en especial para pequeñas y medianas empresas, así como para personas en situaciones vulnerables.

1.2 Problema Identificado

En la actualidad, muchas personas especialmente en contextos de vulnerabilidad económica o educativa carecen de acceso a conocimientos básicos de educación financiera, lo que afecta negativamente en su capacidad para administrar sus recursos, tomar decisiones económicas informadas o iniciar emprendimientos que sean sostenibles. Esta falta de capacitación financiera, combinada con la dificultad de acceder a microcréditos o herramientas de gestión personal, limita sus posibilidades de generar ingresos estables o mejorar su calidad de vida.

1.3 Solución Propuesta

Como respuesta a este problema, se propone el desarrollo de una plataforma digital de inclusión financiera que brinde a los usuarios acceso a educación financiera, servicios de microcrédito y herramientas para la administración de sus finanzas personales. La solución contempla tres módulos principales:

- **Módulo de Capacitación Financiera:** Proporciona cursos y recursos sobre gestión financiera personal y empresarial.
- **Módulo de Préstamos y Microcréditos:** Permite solicitar pequeños préstamos destinados a necesidades personales o iniciativas de negocio.
- **Módulo de Seguimiento de Finanzas:** Facilita el registro de ingresos y egresos, así como la generación de reportes financieros.

El proyecto se desarrolla aplicando los principios de diseño de software y patrones de diseño creacionales y estructurales, dentro de una arquitectura de cuatro capas, utilizando Java y el entorno NetBeans.

1.4 Objetivos del Proyecto

Objetivo general

- Desarrollar una plataforma digital en Java (NetBeans) que contribuya al cumplimiento del ODS 8 mediante la educación financiera y la gestión económica personal.

Objetivos específicos

1. Implementar una arquitectura a cuatro capas que permita la separación lógica de las funcionalidades del sistema.
2. Creación de una base de datos para manejar datos de los usuarios, préstamos y cursos financieros. Además del manejo de procedimientos almacenados mediante los patrones de diseño.
3. Aplicar patrones de diseño creacionales (Singleton, Prototype, Factory), estructurales (Facade, Decorator, Composite) y de comportamiento (Command, Observer) en el desarrollo de los módulos del sistema.
4. Desarrollar una interfaz de usuario clara y funcional para cada uno de los módulos principales: capacitación, préstamos y finanzas personales.

1.5 Marco Teórico

Unidad I: Principios de Diseño y Patrones de Software

Los principios de diseño y los patrones de software sirven como base para desarrollar sistemas bien estructurados, mantenibles y escalables. Entre ellos destaca el conjunto SOLID, propuesto por Robert C. Martin en 2003, el cual promueve prácticas orientadas a la modularidad y coherencia en el diseño. Este conjunto está compuesto por cinco principios fundamentales: Responsabilidad Única, Abierto/Cerrado, Sustitución de Liskov, Segregación de Interfaces e Inversión de Dependencias. De estos, los más utilizados en el desarrollo del proyecto fueron:

- **Responsabilidad Única (SRP):** Cada clase debe encargarse de una sola funcionalidad dentro del sistema. Este principio favorece la claridad del código y su mantenibilidad.
- **Abierto/Cerrado (OCP):** Las clases deben estar abiertas a la extensión, pero cerradas a la modificación. Esto se logra mediante el uso de interfaces, herencia o composición.
- **Inversión de Dependencias (DIP):** Los módulos de alto nivel no deben depender de módulos de bajo nivel, sino de abstracciones. Este principio se refleja en el uso de interfaces en la lógica del proyecto, lo que permite desacoplar la ejecución del procedimiento de su definición concreta.
- **Segregación de Interfaces (ISP):** Este principio indica que ninguna clase debe estar obligada a implementar interfaces que no utiliza. Promueve la creación de interfaces específicas y pequeñas, en lugar de interfaces generales y grandes.

Por otro lado, los patrones de diseño descritos por Gamma et al. (1995), conocidos como el grupo **Gang of Four** (GoF), ofrecen soluciones reutilizables a problemas recurrentes de diseño de software. Estos patrones se clasifican según su propósito en tres categorías: creacionales (controlan la instanciación de objetos), estructurales (organizan relaciones entre clases y objetos) y de comportamiento (gestionan la comunicación y responsabilidades entre objetos). Estos patrones han sido aplicados tanto en su forma

creacional como estructural y de comportamiento, como parte de una arquitectura en capas que promueve el cumplimiento de los principios SOLID.

Unidad II: Patrones de Diseño Creacionales

Los patrones creacionales permiten abstraer el proceso de instanciación de objetos, buscando separar la lógica de creación del resto del sistema. Esto mejora la reutilización, el desacoplamiento y la flexibilidad de las aplicaciones (Gamma et al., 1995). A continuación, se describen los principales patrones creacionales aplicados:

- **Singleton:** Permite que una clase tenga una única instancia en todo el sistema, controlando su acceso global.
- **Prototype:** Facilita la creación de nuevos objetos clonando instancias existentes, lo que evita depender de constructores complejos.
- **Factory Method:** Define una interfaz para la creación de objetos, delegando a subclases la decisión de qué instancia devolver.

Unidad III: Patrones de Diseño Estructurales

Los patrones estructurales permiten establecer relaciones entre clases y objetos para formar estructuras más complejas y robustas, manteniendo la flexibilidad del sistema (Gamma et al., 1995). A continuación, se detallan los patrones estructurales utilizados:

- **Facade:** Este patrón proporciona una interfaz simplificada a un conjunto de interfaces dentro de un subsistema.
- **Decorator:** Permite añadir responsabilidades adicionales a un objeto de forma dinámica, sin modificar su clase base.
- **Proxy:** Este patrón actúa como intermediario para controlar el acceso a ciertos objetos, añadiendo funcionalidades como control de acceso, logs o diferimiento de carga.
- **Composite:** Este patrón permite tratar objetos individuales y composiciones de objetos de manera uniforme. Esto facilita operaciones recursivas sobre la estructura sin necesidad de distinguir entre elementos simples o compuestos.

Unidad IV: Patrones de Comportamiento y Antipatrones

Los patrones de comportamiento describen cómo los objetos interactúan y se comunican entre sí, distribuyendo las responsabilidades de forma eficiente dentro del sistema. (Gamma et al., 1995). A continuación, se presentan los patrones de comportamiento aplicados en el proyecto:

- **Observer:** Permite que múltiples objetos (observadores) estén atentos a los cambios de un objeto central (sujeto), reaccionando automáticamente cuando este se actualiza.
- **Command:** Este patrón encapsula una solicitud como un objeto, lo que permite parametrizar acciones, registrar comandos o implementar funcionalidades.

Además de las buenas prácticas, es importante reconocer los errores comunes de diseño conocidos como antipatrones. Estos surgen como soluciones aparentemente válidas, pero que en la práctica deterioran la calidad del software, dificultando su mantenimiento, escalabilidad o comprensión (Brown et al., 1998). Entre los más frecuentes se encuentran el God Object, que centra demasiadas responsabilidades en una sola clase; los Hard-coded Values, que dificultan la reutilización del código; y la rigidez por dependencia excesiva, donde las clases están tan acopladas que no pueden modificarse sin alterar múltiples partes del sistema.

1.6 Antecedentes de Investigación

A continuación, se presentan tres trabajos previos similares que abordan problemáticas relacionadas con el desarrollo de sistemas de inclusión financiera, el uso de patrones de diseño en entornos Java, y el impacto de la digitalización en el acceso a servicios financieros:

Estudio de patrones de diseño en plataforma Java Enterprise Edition versión 6 para el desarrollo de aplicaciones web

La tesis desarrollada por Acosta Yerovi (2014) aborda de manera detallada el uso de patrones de diseño en el desarrollo de aplicaciones empresariales utilizando Java EE. En su investigación se aplican patrones como DAO, Singleton, Facade y Decorator para construir un sistema web modular y escalable. A través de ejemplos prácticos, la autora demuestra cómo estos patrones permiten mejorar la calidad del código, reducir el acoplamiento entre clases y facilitar el mantenimiento. Su estudio respalda la importancia de aplicar patrones desde la etapa de diseño, resaltando que el uso del patrón Singleton para el control de conexiones y el patrón Facade para simplificar interacciones entre subsistemas resultan particularmente efectivos. Estos fundamentos refuerzan la arquitectura implementada en FinNexus, la cual se basa en principios similares aplicados en un entorno académico de desarrollo modular.

Plataforma Fintech y la inclusión financiera en una empresa del distrito de Miraflores – Lima, 2024

La investigación realizada por López Ortega y Mayta Alfaro (2024) tiene como objetivo analizar el impacto de una plataforma fintech en los niveles de inclusión financiera de una empresa ubicada en el distrito de Miraflores, Lima. A través de un enfoque cuantitativo con encuestas aplicadas a 92 usuarios y el uso de herramientas estadísticas, la tesis evidencia una percepción positiva respecto al uso de tecnología financiera. Sin embargo, también se concluye que los niveles de inclusión real alcanzados son moderados, debido a factores como brechas educativas, falta de cultura financiera y acceso desigual a servicios digitales. Este trabajo constituye un referente importante para el desarrollo de FinNexus, ya que valida que las plataformas tecnológicas orientadas a la educación financiera y la gestión de microcréditos pueden ser bien recibidas por el público objetivo, aunque deben ir acompañadas de componentes formativos sólidos y un diseño centrado en el usuario. Así, se refuerza la pertinencia de integrar módulos de

capacitación, préstamos y seguimiento económico como lo hace FinNexus, alineándose al enfoque del ODS 8.

Digitalización e inclusión financiera en el Perú

El artículo de Tambini, Paliza y Ramírez (2024), publicado por el Banco Central de Reserva del Perú en la revista Moneda, examina cómo la digitalización ha impulsado la inclusión financiera en el país entre 2014 y 2022. Se destacan herramientas como los pagos digitales, el internet móvil, los canales digitales de los bancos y el desarrollo de nuevas tecnologías como billeteras electrónicas y apps bancarias. El estudio también subraya que la infraestructura tecnológica no es suficiente si no va acompañada de programas de educación financiera, protección al consumidor y adecuación normativa. Esta perspectiva coincide plenamente con el enfoque de la plataforma FinNexus, la cual no solo ofrece servicios digitales, sino también contenidos de capacitación estructurados para fortalecer las competencias financieras de sus usuarios. El artículo proporciona un marco contextual y estadístico sólido que justifica el diseño integral de FinNexus, donde la tecnología se combina con educación para fomentar el empoderamiento económico, la toma de decisiones responsables y el desarrollo sostenible, especialmente en poblaciones con menor acceso al sistema financiero formal.

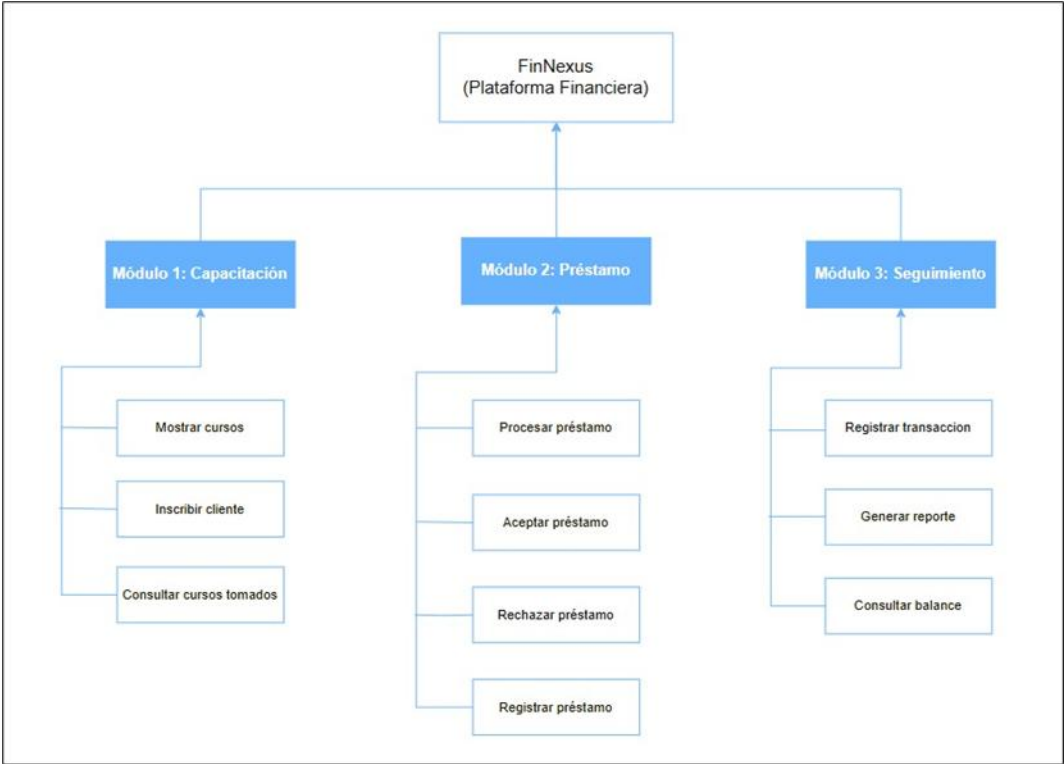
Capítulo 2: Desarrollo

2.1 Diagrama de Descomposición Modular

Como se muestra en la Figura 1, la plataforma financiera está compuesta por 3 módulos principales.

Figura 1

Diagrama de Descomposición Modular de FinNexus



2.2 Cuadro de Resultados por Versiones y Semana

2.2.1 Versión 1: Capacitación

Este módulo tiene como objetivo brindar a los usuarios/clientes acceso a diversos cursos de educación financiera, promoviendo su formación y el desarrollo de competencias económicas. Está compuesto por las siguientes entidades y funciones:

2.2.1.1 Usuario

El usuario en este módulo representa al cliente que accede a la plataforma con el propósito de mejorar sus conocimientos a través de cursos relacionados con la educación financiera. Su información está registrada en la tabla Clientes, que almacena datos como el nombre, el correo electrónico y el número de teléfono. En este contexto, el usuario cumple el rol de participante activo que puede consultar la oferta de cursos y realizar inscripciones, lo que permite personalizar su experiencia formativa y llevar un control individual del progreso educativo.

2.2.1.2 Curso

El curso es el eje formativo del módulo de capacitación. Cada curso está registrado en la tabla Cursos con información relevante como el nombre, la descripción, la duración en horas y la fecha de inicio. Los cursos disponibles se ofrecen como una opción educativa para los usuarios registrados, permitiendo que estos se inscriban mediante procedimientos almacenados. Además, se lleva un historial de los cursos tomados por cada usuario, lo cual es fundamental para el seguimiento académico y la mejora continua del sistema de formación.

2.2.2 Versión 2: Préstamo

Este módulo tiene como objetivo ofrecer microcréditos a los usuarios, evaluando previamente si cumplen ciertos requisitos básicos como haber completado cursos o contar con un historial adecuado.

2.2.2.1 Usuario

En este módulo, el usuario mantiene su rol como cliente del sistema, pero ahora enfocado en el acceso a productos financieros. La información del cliente sigue siendo fundamental para identificarlo, validar su historial y permitirle realizar solicitudes de préstamo. A través de procedimientos específicos, se gestiona la interacción del cliente con el sistema financiero, permitiéndole consultar sus préstamos activos, historial de pagos y nuevos montos solicitados.

2.2.2.2 Préstamo

El préstamo es el recurso financiero gestionado dentro del módulo. Este componente implica el registro y control de las solicitudes realizadas por los usuarios, incluyendo detalles como el monto solicitado, la tasa de interés, el plazo, la fecha de emisión y el estado del préstamo. Estos datos son almacenados y gestionados por medio de tablas específicas y procedimientos que aseguran una trazabilidad clara y una correcta evaluación de cada caso. La finalidad es

facilitar el acceso responsable a microcréditos y apoyar el crecimiento económico de los clientes.

2.2.3 Versión 3: Seguimiento

Este módulo tiene como finalidad brindar acompañamiento al usuario tras haber recibido un préstamo, permitiéndole llevar un control de sus finanzas, evaluar su progreso y detectar posibles alertas de riesgo financiero.

2.2.3.1 Usuario

El usuario, en este contexto, pasa a ser un agente que monitorea y gestiona su propia situación financiera a través de la plataforma. Su participación es clave para revisar información consolidada sobre sus capacitaciones, préstamos y alertas del sistema. Aquí, el usuario accede a un panel de control que le permite tener una visión global de su progreso, facilitando así la toma de decisiones informadas y promoviendo hábitos financieros responsables.

2.2.3.2 Seguimiento Financiero

El seguimiento financiero representa una herramienta de apoyo que recopila y presenta información relevante del usuario a lo largo del tiempo. Este componente puede incluir indicadores de avance, gráficos de pagos, estadísticas de asistencia a cursos y alertas relacionadas con fechas importantes. Su propósito es brindar retroalimentación continua al usuario y reforzar el impacto del proyecto en términos de inclusión financiera y educación personalizada. Mediante este sistema, se busca que el usuario no solo acceda a recursos, sino que también comprenda su evolución económica dentro del programa.

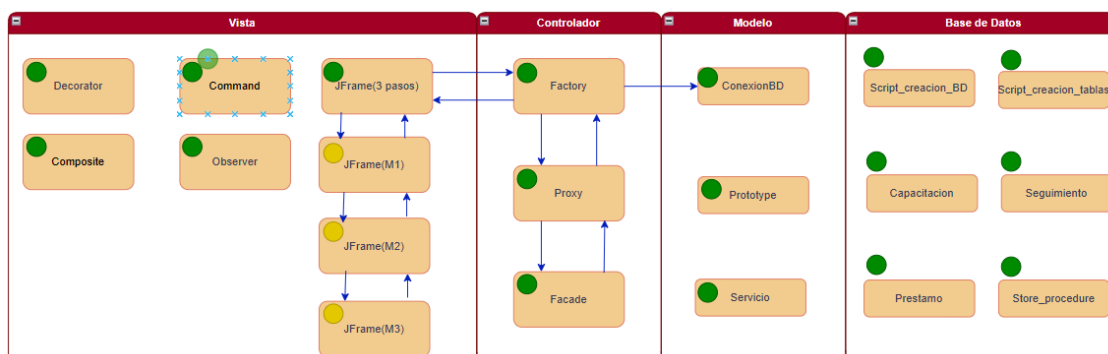
2.3 Diseño de Arquitectura de Software a 4 Capas

2.3.1 Diagrama de Arquitectura de Software

En esta sección, la Figura 2 muestra el diagrama de Arquitectura de Software, el cual está compuesto por 4 capas (Modelo – Vista – Controlador - Base de Datos) y también muestra el mapa de calor que representa el estado de implementación del software.

Figura 2

Diagrama de Arquitectura de Software de FinNexus



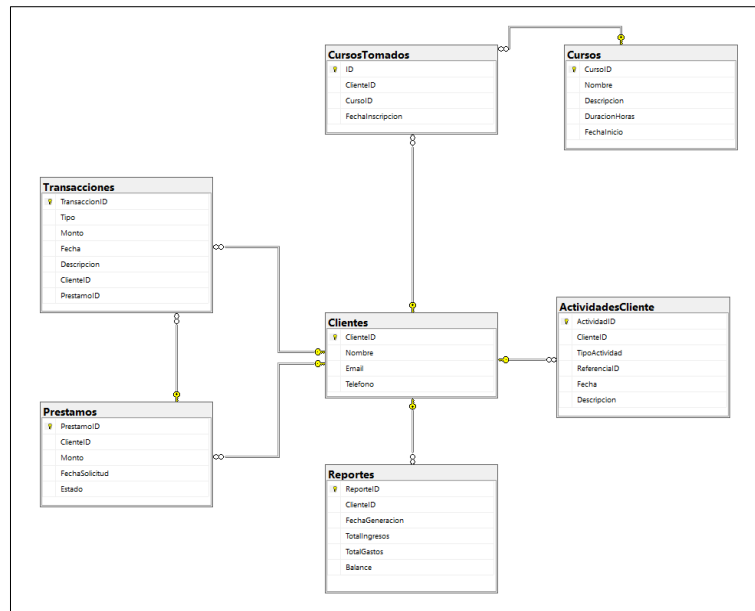
2.3.2 Diagrama Base de Datos

Modelo Lógico

Este modelo representa la estructura conceptual de la base de datos, mostrando las entidades principales, sus atributos y las relaciones entre ellas

Figura 3

Diagrama Lógico de la Base de Datos

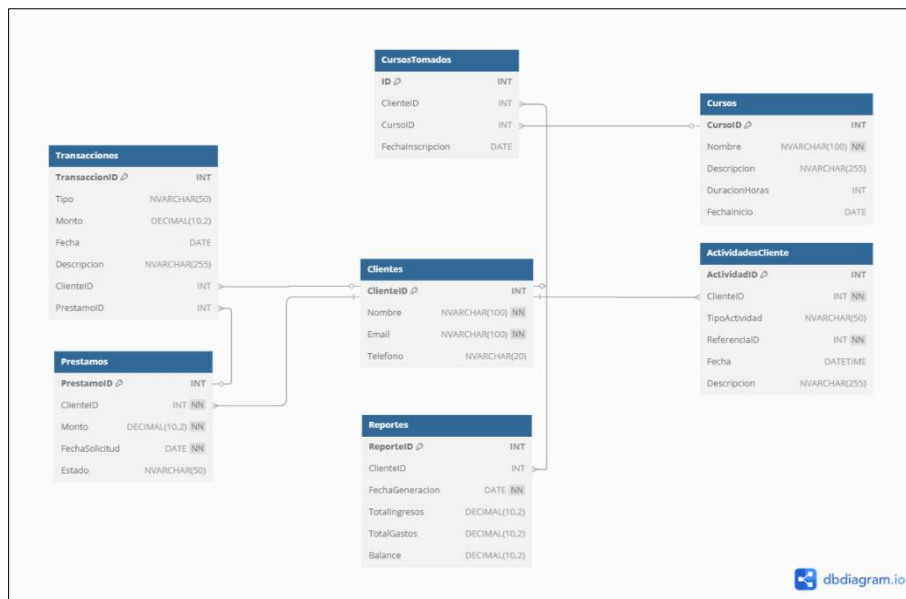


Modelo Físico

Este modelo detalla la implementación concreta de la base de datos en un sistema gestor, incluyendo tipos de datos, claves primarias y foráneas, así como índices y otras optimizaciones técnicas.

Figura 4

Diagrama Físico de la Base de Datos



2.3.3 Modelo MVC (Modelo – Vista – Controlador)

Diagrama de Modelo

En este apartado, la Figura 5 y 6 muestran el diagrama UML del paquete Modelo y Modelo.Procedimientos, conteniendo sus clases y atributos correspondientes.

Figura 5

Diagrama UML del Paquete Modelo



Figura 6

Diagrama UML de la Capa Modelo.Procedimientos

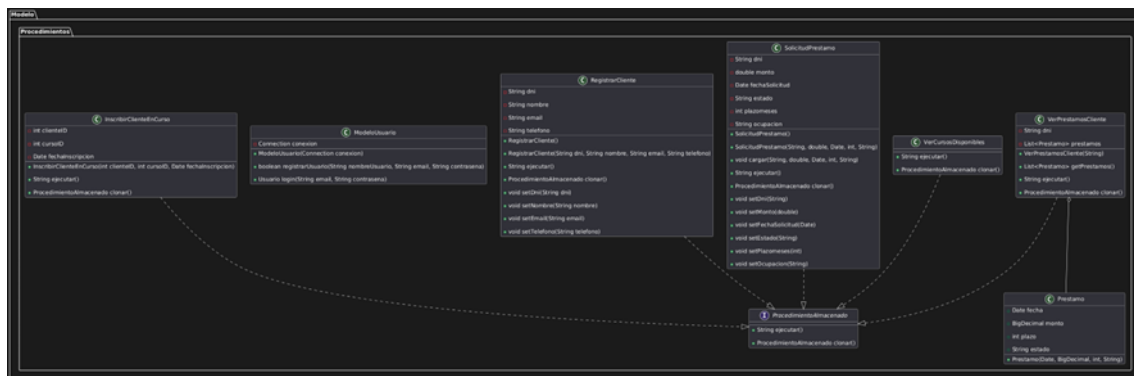


Diagrama de Vista

En este apartado, la Figura 7 y 8 muestran el diagrama UML de la capa Vista, esta capa cuenta con 2 paquetes que son PersonalizacionVista y Vista, los cuales manejan el diseño de los JFrame y lo que verá el usuario.

Figura 8

Diagrama UML del Paquete PersonalizacionVista

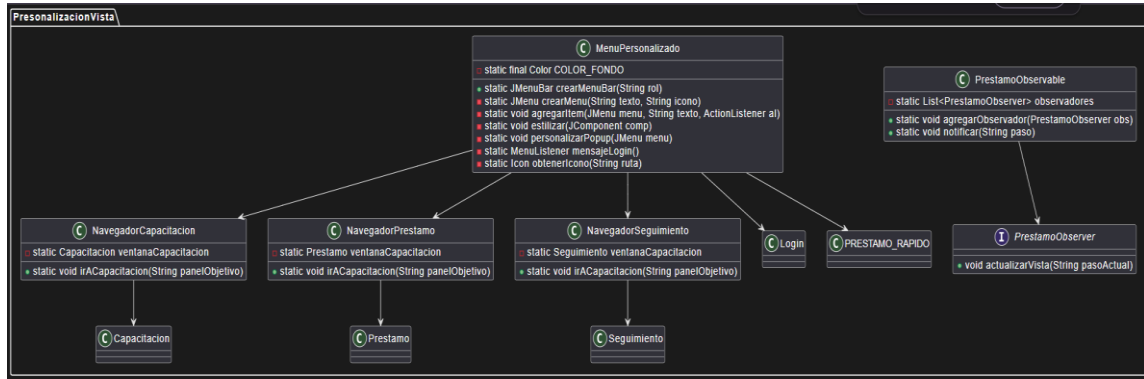


Figura 7

Diagrama UML del Paquete Vista

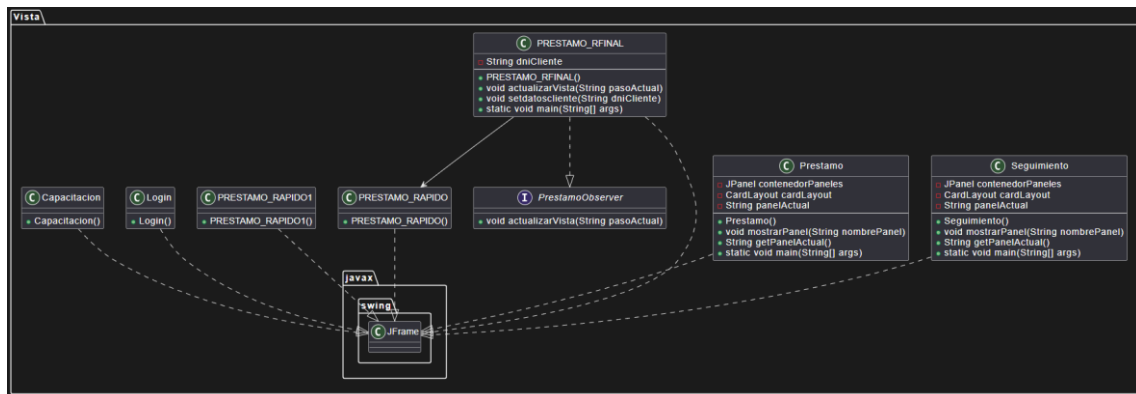
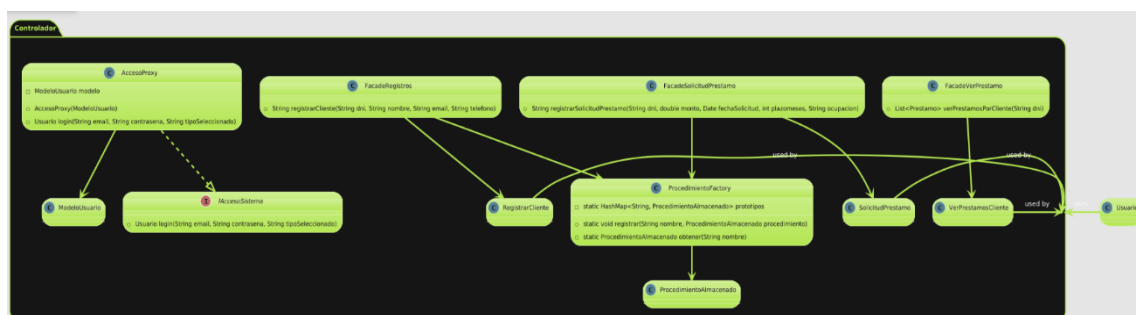


Diagrama de Controlador

En este apartado, la Figura 9 muestra el diagrama UML del paquete Controlador, el cual se encarga de manejar la lógica del sistema.

Figura 9

Diagrama UML del Paquete Controlador



2.4 Aplicación de Principios SOLID

2.4.1 Responsabilidad Única (SRP)

Clase: ConexionBD.java

Justificación: Esta clase cumple el principio SRP al encargarse únicamente de la conexión a base de datos. Ya que, no contiene lógica de negocio, presentación ni controladores, lo que refleja una única responsabilidad bien definida.

```
public class ConexionBD {
    private static ConexionBD instancia;
    private Connection conexion;

    private final String URL = "jdbc:sqlserver://localhost:1433;databaseName=FinNexus;"
        + "encrypt=true;trustServerCertificate=true;user=Godsher;password=PolliCol23;";
    private final String Usu = "Godsher";
    private final String Contra = "PolliCol23";

    private ConexionBD() {
        try {
            conexion = DriverManager.getConnection(URL, Usu, Contra);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static ConexionBD getInstancia() {
        if (instancia == null) {
            instancia = new ConexionBD();
        }
        return instancia;
    }

    public Connection getConexion() {
        return conexion;
    }
}
```

2.4.2 Abierto/Cerrado (OCP)

Clase: FacadeRegistros.java

Justificación: Esta clase cumple el principio OCP al permitir la extensión de nuevas funcionalidades sin modificar su código base. Gracias a su estructura, se puede incorporar cambios en los subsistemas sin alterar la lógica central.

```
public class FacadeRegistros {

    public String registrarCliente(String dni, String nombre, String email, String telefono) {
        ProcedimientoAlmacenado procedimiento = ProcedimientoFactory.obtener("RegistrarCliente");

        // Validaciones
        if (dni.isEmpty() || nombre.isEmpty() || email.isEmpty() || telefono.isEmpty()) {
            return "Todos los campos son obligatorios.";
        }

        if (!dni.matches("\\d{8}")) {
            return "DNI debe tener 8 números.";
        }

        if (!telefono.matches("\\d{6,}")) {
            return "Teléfono debe tener al menos 6 números.";
        }

        // Registrar el prototipo si no está
        if (ProcedimientoFactory.obtener("registrarCliente") == null) {
            ProcedimientoFactory.registrar("registrarCliente", new RegistrarCliente());
        }

        // Obtener y configurar datos
        RegistrarCliente proc = (RegistrarCliente) ProcedimientoFactory.obtener("registrarCliente");
        proc.setDni(dni);
        proc.setNombre(nombre);
        proc.setEmail(email);
        proc.setTelefono(telefono);
        // Ejecutar
        return proc.ejecutar();
    }
}
```

2.4.3 Inversión de Dependencias (DIP)

Clase: IAccesoSistema.java y AccesoProxy.java

Justificación: Este principio se cumple ya que el controlador o clase que utiliza AccesoProxy depende de la abstracción IAccesoSistema y no de una clase concreta. Así, se reduce el acoplamiento entre componentes de alto y bajo nivel, promoviendo una arquitectura flexible y fácilmente modificable.

```
public interface IAccesoSistema {
    Usuario login(String email, String contrasena, String tipoSeleccionado);
}
```

```
public class AccesoProxy implements IAccesoSistema {
    private final ModeloUsuario modelo;

    public AccesoProxy(ModeloUsuario modelo) {
        this.modelo = modelo;
    }

    @Override
    public Usuario login(String email, String contrasena, String tipoSeleccionado) {
        Usuario usuario = modelo.login(email, contrasena);

        if (usuario == null) {
            return null; // Credenciales inválidas
        }

        if (tipoSeleccionado.equalsIgnoreCase("Cliente") && usuario.getRol().equalsIgnoreCase("Cliente")) {
            return usuario;
        } else if (tipoSeleccionado.equalsIgnoreCase("Empleado") &&
            (usuario.getRol().equalsIgnoreCase("Empleado") || usuario.getRol().equalsIgnoreCase("Admin"))) {
            return usuario;
        }

        return null; // Intentó entrar como tipo incorrecto
    }
}
```

2.4.4 Segregación de Interfaces (ISP)

Clase: IAccesoSistema.java

Justificación: Esta interfaz está diseñada con métodos específicos relacionados únicamente al acceso al sistema. No obliga a las clases que la implementan a definir métodos innecesarios, lo cual cumple el principio ISP.

```
public interface IAccesoSistema {
    Usuario login(String email, String contrasena, String tipoSeleccionado);
}
```


2.5 Programación Java en NetBeans

2.5.1 Capa Modelo (Patrones Creacionales)

Singleton

En este proyecto, se aplica en la clase ConexionBD, que administra la conexión con la base de datos. De esta manera, se asegura que todas las operaciones del sistema compartan una misma conexión eficiente y centralizada.

Clase ConexionBD

```
package Modelo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/**
 *
 * @author Luis Carlos y Frank Vidal
 */
public class ConexionBD {
    private static ConexionBD instancia;
    private Connection conexion;

    private final String URL = "jdbc:sqlserver://localhost:1433;databaseName=FinNexus;"
        + "encrypt=true;trustServerCertificate=true;user=Godsher;password=PolliCol23;";
    private final String Usu = "Godsher";
    private final String Contra = "PolliCol23";

    private ConexionBD() {
        try {
            conexion = DriverManager.getConnection(URL, Usu, Contra);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static ConexionBD getInstancia() {
        if (instancia == null) {
            instancia = new ConexionBD();
        }
        return instancia;
    }

    public Connection getConexion() {
        return conexion;
    }
}
```

Prototype

En este proyecto, cada procedimiento almacenado implementa la interfaz `ProcedimientoAlmacenado`, la cual incluye el método `clonar()`. Esto facilita la duplicación de procedimientos ya registrados en memoria, mejorando el rendimiento y reduciendo el acoplamiento entre clases.

Interfaz `ProcedimientoAlmacenado`

```
package Modelo.Procedimientos;

/**
 *
 * @author Luis Carlos y Frank Vidal
 */
public interface ProcedimientoAlmacenado extends Cloneable {
    void ejecutar();
    ProcedimientoAlmacenado clonar();
}
```

Clase `InscribirClienteEnCurso` (Procedimiento Almacenado)

```
package Modelo.Procedimientos;
import java.util.Date;
import Modelo.ConexionBD;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
/**
 *
 * @author Asus
 */
public class InscribirClienteEnCurso implements ProcedimientoAlmacenado{

    private int clienteID;
    private int cursoID;
    private Date fechaInscripcion;

    public InscribirClienteEnCurso(int clienteID, int cursoID, Date fechaInscripcion) {
        this.clienteID = clienteID;
        this.cursoID = cursoID;
        this.fechaInscripcion = fechaInscripcion;
    }
    @Override
    public void ejecutar() {
        try {
            Connection conexion = ConexionBD.getInstance().getConexion();
            try (CallableStatement stmt = conexion.prepareCall("{call InscribirClienteEnCurso(?, ?, ?)}")) {
                stmt.setInt(1, clienteID);
                stmt.setInt(2, cursoID);
                stmt.setDate(3, new java.sql.Date(fechaInscripcion.getTime()));

                boolean tieneResultado = stmt.execute();

                if (tieneResultado) {
                    ResultSet rs = stmt.getResultSet();
                    while (rs.next()) {
                        System.out.println("Resultado: " + rs.getString(1));
                    }
                    rs.close();
                } else {
                    System.out.println("✔ Procedimiento ejecutado correctamente.");
                }
            }
        } catch (Exception e) {
            System.out.println("✗ Error al ejecutar el procedimiento: " + e.getMessage());
        }
    }
    @Override
    public ProcedimientoAlmacenado clonar() {
        try {
            return (ProcedimientoAlmacenado) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException("No se pudo clonar el procedimiento");
        }
    }
}
```

Clase VerCursosDisponibles (Procedimiento Almacenado)

```
package Modelo.Procedimientos;
import Modelo.ConexionBD;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
/**
 *
 * @author Luis Carlos
 */
public class VerCursosDisponibles implements ProcedimientoAlmacenado{
    @Override
    public void ejecutar() {
        try {
            Connection conexion = ConexionBD.getInstancia().getConexion();
            CallableStatement stmt = conexion.prepareCall("{call VerCursosDisponibles}");

            ResultSet rs = stmt.executeQuery();
            System.out.println("Cursos Disponibles:");
            while (rs.next()) {
                System.out.println("- ID: " + rs.getInt("CursoID") +
                    ", Nombre: " + rs.getString("Nombre") +
                    ", Descripción: " + rs.getString("Descripcion") +
                    ", Duración: " + rs.getInt("DuracionHoras") +
                    ", Fecha Inicio: " + rs.getDate("FechaInicio"));
            }
            rs.close();
            stmt.close();
        } catch (Exception e) {
            System.out.println("X Error al ejecutar el procedimiento: " + e.getMessage());
        }
    }

    @Override
    public ProcedimientoAlmacenado clonar() {
        try {
            return (ProcedimientoAlmacenado) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException("No se pudo clonar el procedimiento");
        }
    }
}
```

2.5.2 Capa Controlador (Patrones Creacionales)

Factory

El patrón Factory permite encapsular la lógica de creación de objetos. En este proyecto, la clase **ProcedimientoFactory** actúa como fábrica de procedimientos almacenados registrados previamente. Cuando se requiere ejecutar uno, se obtiene una copia (Prototipe) del procedimiento solicitado.

Clase ProcedimientoFactory

```
package Modelo.Procedimientos;
import java.util.HashMap;
/**
 *
 * @author Luis Carlos, Frank Vidal y PolliCo
 */
public class ProcedimientoFactory {
    private static final HashMap<String, ProcedimientoAlmacenado> prototipos = new HashMap<>();

    public static void registrar(String nombre, ProcedimientoAlmacenado procedimiento) {
        prototipos.put(nombre, procedimiento);
    }

    public static ProcedimientoAlmacenado obtener(String nombre) {
        ProcedimientoAlmacenado prototipo = prototipos.get(nombre);
        return prototipo != null ? prototipo.clonar() : null;
    }
}
```

Facade

El patrón Facade simplifica la interacción con subsistemas complejos al ofrecer una interfaz única y sencilla. En este proyecto, la clase **FacadeRegistros** centraliza las operaciones de registro y gestión de usuarios y cursos, la clase **FacadeVerPrestamo** facilita la consulta y visualización de los préstamos, y la clase **FacadeSolicitudPrestamo** coordina el proceso completo de solicitud y validación de préstamos, ocultando la complejidad interna y mejorando la usabilidad del sistema.

Clase FacadeRegistros

```
public class FacadeRegistros {

    public String registrarCliente(String dni, String nombre, String email, String telefono) {
        ProcedimientoAlmacenado procedimiento = ProcedimientoFactory.obtener("RegistrarCliente");

        // Validaciones
        if (dni.isEmpty() || nombre.isEmpty() || email.isEmpty() || telefono.isEmpty()) {
            return "Todos los campos son obligatorios.";
        }

        if (!dni.matches("\\d{8}")) {
            return "DNI debe tener 8 números.";
        }

        if (!telefono.matches("\\d{6,}")) {
            return "Teléfono debe tener al menos 6 números.";
        }

        // Registrar el prototipo si no está
        if (ProcedimientoFactory.obtener("registrarCliente") == null) {
            ProcedimientoFactory.registrar("registrarCliente", new RegistrarCliente());
        }

        // Obtener y configurar datos
        RegistrarCliente proc = (RegistrarCliente) ProcedimientoFactory.obtener("registrarCliente");
        proc.setDni(dni);
        proc.setNombre(nombre);
        proc.setEmail(email);
        proc.setTelefono(telefono);

        // Ejecutar
        return proc.ejecutar();
    }

}
```

Clase FacadeVerPrestamo

```
public class FacadeVerPrestamo {

    public List<VerPrestamosCliente.Prestamo> verPrestamosPorCliente(String dni) {
        ProcedimientoAlmacenado procedimiento = ProcedimientoFactory.obtener("VerPrestamosCliente");

        if (procedimiento instanceof VerPrestamosCliente) {
            VerPrestamosCliente verPrestamos = (VerPrestamosCliente) procedimiento;
            // Asignamos el DNI
            verPrestamos = new VerPrestamosCliente(dni);
            String mensaje = verPrestamos.ejecutar(); // Ejecuta el procedimiento
            System.out.println(mensaje);
            return verPrestamos.getPrestamos(); // Retorna la lista
        }

        System.out.println("X No se pudo ejecutar el procedimiento VerPrestamosCliente.");
        return Collections.emptyList(); // Retorna una lista vacía si falla
    }

}
```

Clase FacadeSolicitudPrestamo

```
public class FacadeSolicitudPrestamo {  
    public String registrarSolicitudPrestamo(String dni, double monto, Date fechaSolicitud, int plasomeses, String ocupacion) {  
        ProcedimientoAlmacenado procedimiento = ProcedimientoFactory.obtener("SolicitudPrestamo");  
  
        if (procedimiento instanceof SolicitudPrestamo) {  
            SolicitudPrestamo solicitud = (SolicitudPrestamo) procedimiento;  
            solicitud.setDni(dni);  
            solicitud.setMonto(monto);  
            solicitud.setFechaSolicitud(fechaSolicitud);  
            solicitud.setPlasomeses(plasomeses);  
            solicitud.setOcupacion(ocupacion);  
  
            return solicitud.ejecutar();  
        }  
  
        return "No se pudo ejecutar el procedimiento SolicitudPrestamo.";  
    }  
}
```

Proxy

El patrón Proxy controla el acceso a un objeto real mediante un intermediario que puede añadir funcionalidades adicionales como control de permisos, carga diferida o registro de accesos. En este proyecto, la clase **CrearMenuPersonalizado** gestiona la personalización dinámica del menú según el usuario, asegurando que solo se muestren las opciones autorizadas y optimizando la carga de elementos en la interfaz.

Clase AccesoProxy

```
import Modelo.Procedimientos.ModeloUsuario;  
import Modelo.Usuario;  
  
public class AccesoProxy implements IAccesoSistema {  
    private final ModeloUsuario modelo;  
  
    public AccesoProxy(ModeloUsuario modelo) {  
        this.modelo = modelo;  
    }  
  
    @Override  
    public Usuario login(String email, String contrasena, String tipoSeleccionado) {  
        Usuario usuario = modelo.login(email, contrasena);  
  
        if (usuario == null) {  
            return null; // Credenciales inválidas  
        }  
  
        if (tipoSeleccionado.equalsIgnoreCase("Cliente") && usuario.getRol().equalsIgnoreCase("Cliente")) {  
            return usuario;  
        } else if (tipoSeleccionado.equalsIgnoreCase("Empleado") &&  
            (usuario.getRol().equalsIgnoreCase("Empleado") || usuario.getRol().equalsIgnoreCase("Admin"))) {  
            return usuario;  
        }  
  
        return null; // Intentó entrar como tipo incorrecto  
    }  
}
```

2.5.3 Capa Vista

Decorator

El patrón Decorator permite extender el comportamiento de un objeto sin modificar su código original, añadiendo funcionalidades de forma flexible. En este sistema, se utiliza en la clase `MenuPersonalizado`, que actúa como un decorador de `JMenuBar`, generando un menú dinámico y adaptado al tipo de usuario (por ejemplo, "Invitado", "Cliente", etc.), agregando elementos específicos según el rol sin alterar la lógica principal de la ventana que lo utiliza. Esto mejora la reutilización del código y la personalización visual.

Clase MenuPersonalizado

```
import Vista.Login;
import Vista.PRESTAMO_RAPIDO;
import javax.swing.*;
import java.awt.*;
import javax.swing.SwingUtilities;
import javax.swing.JPanel;
import javax.swing.event.MenuEvent;
import javax.swing.event.MenuListener;

/**
 *
 * @author Alvarez y Vidal
 */
public class MenuPersonalizado {
    private static final Color COLOR_FONDO = new Color(26, 35, 126);

    public static JMenuBar crearMenuBar(String rol) {
        if (rol == null) rol = "Invitado"; // Seguridad ante nulos

        boolean esAdmin = "Admin".equalsIgnoreCase(rol);
        boolean esEmpleado = "Empleado".equalsIgnoreCase(rol);
        boolean esInvitado = "Invitado".equalsIgnoreCase(rol);
        // Cliente: no es admin, ni empleado, ni invitado
        boolean esCliente = !esAdmin && !esEmpleado && !esInvitado;

        JMenuBar barra = new JMenuBar();
        barra.setBackground(COLOR_FONDO);
        barra.setPreferredSize(new Dimension(700, 50));

        /* ==== Menú Capacitación ==== */
        JMenu mCap = crearMenu("Capacitación", "/icons/inventario.png");
        if (esInvitado) {
            mCap.addMenuListener(mensajeLogin());
        } else {
            agregarItem(mCap, "Mostrar Cursos disponibles",
                e -> NavegadorCapacitacion.irACapacitacion("mostrar"));

            if (esEmpleado || esAdmin) {
                agregarItem(mCap, "Inscribir Cliente",
                    e -> NavegadorCapacitacion.irACapacitacion("inscribir"));
                agregarItem(mCap, "Consulta de Cursos",
                    e -> NavegadorCapacitacion.irACapacitacion("consultar"));
            }
        }
        barra.add(mCap);
    }
}
```

```

/* ==== Menú Préstamo ==== */
JMenu mPrest = crearMenu("Préstamo", "/icons/saldo.png");
if (esAdmin) {
    agregarItem(mPrest, "Registrar Préstamo",
        e -> NavegadorPrestamo.irACapacitacion("Registrar"));
    agregarItem(mPrest, "Ver Préstamos",
        e -> NavegadorPrestamo.irACapacitacion("Ver"));
} else {
    mPrest.addMenuListener(mensajeLogin());
}
barra.add(mPrest);

/* ==== Menú Seguimiento ==== */
JMenu mSeg = crearMenu("Seguimiento", "/icons/busqueda.png");
if (esAdmin) {
    agregarItem(mSeg, "Registrar Transacción",
        e -> NavegadorSeguimiento.irACapacitacion("RegistrarT"));
    agregarItem(mSeg, "Generar Reporte",
        e -> NavegadorSeguimiento.irACapacitacion("Reporte"));
    agregarItem(mSeg, "Consultar Balance",
        e -> NavegadorSeguimiento.irACapacitacion("Balance"));
} else {
    mSeg.addMenuListener(mensajeLogin());
}
barra.add(mSeg);

JMenu mPrestamoRapido = crearMenu("Préstamo Rápido", "/icons/money.png");
agregarItem(mPrestamoRapido, "Ingresar", e -> {
    JFrame actual = (JFrame) SwingUtilities.getWindowAncestor(barra);
    if (actual != null) {
        actual.dispose(); // Cierra el frame actual
    }
    new PRESTAMO_RAPIDO().setVisible(true); // Abre el nuevo frame
});

barra.add(mPrestamoRapido);

/* Panel de relleno — mantiene color de fondo */
JPanel filler = new JPanel();
filler.setPreferredSize(new Dimension(2000, 40));
filler.setBackground(COLOR_FONDO);
barra.add(filler);

```

```

/* ---- separador que empuja lo que sigue hacia la derecha ---- */
barra.add(Box.createHorizontalGlue());
/* ==== Menú / Label Iniciar Sesión ==== */
JMenu mLogin = crearMenu("Iniciar Sesión", "/icons/user.png"); // usa tu propio icono
mLogin.addMenuListener(new MenuListener() { // disponible para todos
    public void menuSelected(MenuEvent e) {
        new Login().setVisible(true); // abre tu ventana de login
        mLogin.getPopupMenu().setVisible(false); // evita desplegar popup vacío
    }
    public void menuDeselected(MenuEvent e) {}
    public void menuCanceled(MenuEvent e) {}
});
barra.add(mLogin);
return barra;
}

private static JMenu crearMenu(String texto, String icono) {
    JMenu menu = new JMenu(texto);
    menu.setIcon(icono != null ? obtenerIcono(icono) : null);
    estilizar(menu);
    personalizarPopup(menu);
    return menu;
}

private static void agregarItem(JMenu menu, String texto, java.awt.event.ActionListener al) {
    JMenuItem item = new JMenuItem(texto);
    item.addActionListener(al);
    estilizar(item);
    menu.add(item);
}

private static void estilizar(JComponent comp) {
    comp.setOpaque(true);
    comp.setBackground(COLOR_FONDO);
    comp.setForeground(COLOR_WHITE);
    comp.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    comp.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseEntered(java.awt.event.MouseEvent e) { comp.setBackground(new Color(63,81,181)); }
        public void mouseExited(java.awt.event.MouseEvent e) { comp.setBackground(COLOR_FONDO); }
    });
}

private static void personalizarPopup(JMenu menu) {
    JPopupMenu popup = menu.getPopupMenu();
    popup.setBackground(COLOR_FONDO);
    popup.setBorder(BorderFactory.createEmptyBorder());
}

```

```

private static MenuListener mensajeLogin() {
    return new MenuListener() {
        public void menuSelected(MenuEvent e) {
            JOptionPane.showMessageDialog(null,
                "Debe iniciar sesión o registrarse para acceder a esta opción.",
                "Acceso restringido", JOptionPane.WARNING_MESSAGE);
        }
        public void menuDeselected(MenuEvent e) {}
        public void menuCanceled(MenuEvent e) {}
    };
}

private static Icon obtenerIcono(String ruta) {
    return new ImageIcon(MenuPersonalizado.class.getResource(ruta))
        .getImage().getScaledInstance(30, 30, Image.SCALE_SMOOTH);
}
}

```

Observer

El patrón Observer permite establecer una relación de suscripción entre un objeto central (sujeto) y múltiples objetos dependientes (observadores), de manera que cuando el sujeto cambia su estado, todos los observadores son notificados automáticamente.

Requerimiento funcional aplicado

En esta aplicación, se implementa este patrón para controlar el flujo del proceso de préstamo paso a paso. La clase PrestamoObservable actúa como el sujeto que gestiona los cambios de estado del proceso (por ejemplo, "Paso1", "Paso2", "Paso3").

A su vez, las clases PRESTAMO_RAPIDO, PRESTAMO_RAPIDO1 y PRESTAMO_RFINAL implementan la interfaz PrestamoObserver, reaccionando de forma automática cuando ocurre un cambio en el estado del flujo. Esto permite sincronizar la activación o cierre de formularios, asegurando una navegación controlada, progresiva y coherente para el usuario dentro del módulo de préstamos.

PrestamoObserver

```

public interface PrestamoObserver {
    void actualizarVista(String pasoActual);
}

```

PrestamoObservable

```

import java.util.ArrayList;
import java.util.List;

public class PrestamoObservable {
    private static final List<PrestamoObserver> observadores = new ArrayList<>();

    public static void agregarObservador(PrestamoObserver obs) {
        observadores.add(obs);
    }

    public static void notificar(String paso) {
        for (PrestamoObserver obs : observadores) {
            obs.actualizarVista(paso);
        }
    }
}

```


Command

El patrón Command encapsula una acción y su ejecución dentro de un objeto independiente, permitiendo ejecutar, deshacer o rehacer acciones sin que quien las invoque conozca los detalles de su implementación.

Requerimiento funcional aplicado

En esta aplicación se implementa este patrón para manejar la navegación entre formularios del flujo de préstamo rápido. La clase ControladorPrestamo actúa como invocador, ejecutando comandos concretos como ComandoAbrirFrame, el cual encapsula la lógica para avanzar o retroceder entre los pasos PRESTAMO_RAPIDO, PRESTAMO_RAPIDO1 y PRESTAMO_RFINAL. De esta manera, al hacer clic en "Siguiente" o "Volver", no se llama directamente al siguiente formulario, sino que se crea un comando que realiza la acción y se guarda en una pila para poder deshacerla si el usuario desea retroceder. Esto permite un control más flexible y reversible del flujo de la interfaz sin acoplar los formularios entre sí.

Comando

```
public interface Comando {  
    void ejecutar();  
    void deshacer();  
}
```

ComandoAbrirFrame

```
public class ComandoAbrirFrame implements Comando {  
    private JFrame actual;  
    private JFrame destino;  
  
    public ComandoAbrirFrame(JFrame actual, JFrame destino) {  
        this.actual = actual;  
        this.destino = destino;  
    }  
  
    @Override  
    public void ejecutar() {  
        destino.setVisible(true);  
        destino.setLocationRelativeTo(null);  
        actual.dispose();  
    }  
  
    @Override  
    public void deshacer() {  
        actual.setVisible(true);  
        actual.setLocationRelativeTo(null);  
        destino.dispose();  
    }  
}
```

Composite

El patrón Composite permite organizar objetos en estructuras jerárquicas de tipo árbol, donde los objetos individuales y los compuestos se tratan de manera uniforme.

Requerimiento funcional aplicado

En esta aplicación se aplica este patrón al sistema de menús. La clase MenuGrupo representa un grupo de opciones (como “Capacitación” o “Préstamo”) que puede contener subopciones representadas con el método agregarOpcion. Cada grupo se construye dinámicamente y se añade al JMenuBar mediante construirSobre. Según el rol del usuario (Admin, Empleado, Cliente o Invitado), se agregan diferentes grupos al menú. Esto permite estructurar el menú como una jerarquía flexible y fácilmente extensible, manteniendo una construcción limpia, organizada y centralizada para todas las vistas de la aplicación.

MenuGrupo

```
public class MenuGrupo {
    private final JMenu menu;

    public MenuGrupo(String titulo, String icono) {
        menu = new JMenu(titulo);
        menu.setIcon(obtenerIcono(icono));
        estilizar(menu);
        personalizarPopup(menu);
    }

    public void agregarOpcion(String texto, ActionListener accion, String icono) {
        JMenuItem item = new JMenuItem(texto, obtenerIcono(icono));
        item.addActionListener(accion);
        estilizar(item);
        menu.add(item);
    }

    private static void estilizar(JComponent comp) {
        comp.setOpaque(true);
        comp.setBackground(new Color(175, 26, 70)); // Puedes parametrizarlo si deseas
        comp.setForeground(Color.WHITE);
        comp.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        comp.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent e) {
                comp.setBackground(new Color(63, 81, 181));
            }

            public void mouseExited(java.awt.event.MouseEvent e) {
                comp.setBackground(new Color(175, 26, 70));
            }
        });
    }

    private static void personalizarPopup(JMenu menu) {
        JPopupMenu popup = menu.getPopupMenu();
        popup.setBackground(new Color(175, 26, 70));
        popup.setBorder(BorderFactory.createEmptyBorder());
    }

    private static Icon obtenerIcono(String ruta) {
        if (ruta == null || ruta.isBlank()) {
            return null; // o puedes retornar un ícono por defecto si lo prefieres
        }

        URL recurso = MenuGrupo.class.getResource(ruta);
        if (recurso == null) {
            System.err.println("Ícono no encontrado: " + ruta);
            return null;
        }

        ImageIcon original = new ImageIcon(recurso);
        Image imagenEscalada = original.getImage().getScaledInstance(30, 30, Image.SCALE_SMOOTH);
        return new ImageIcon(imagenEscalada);
    }

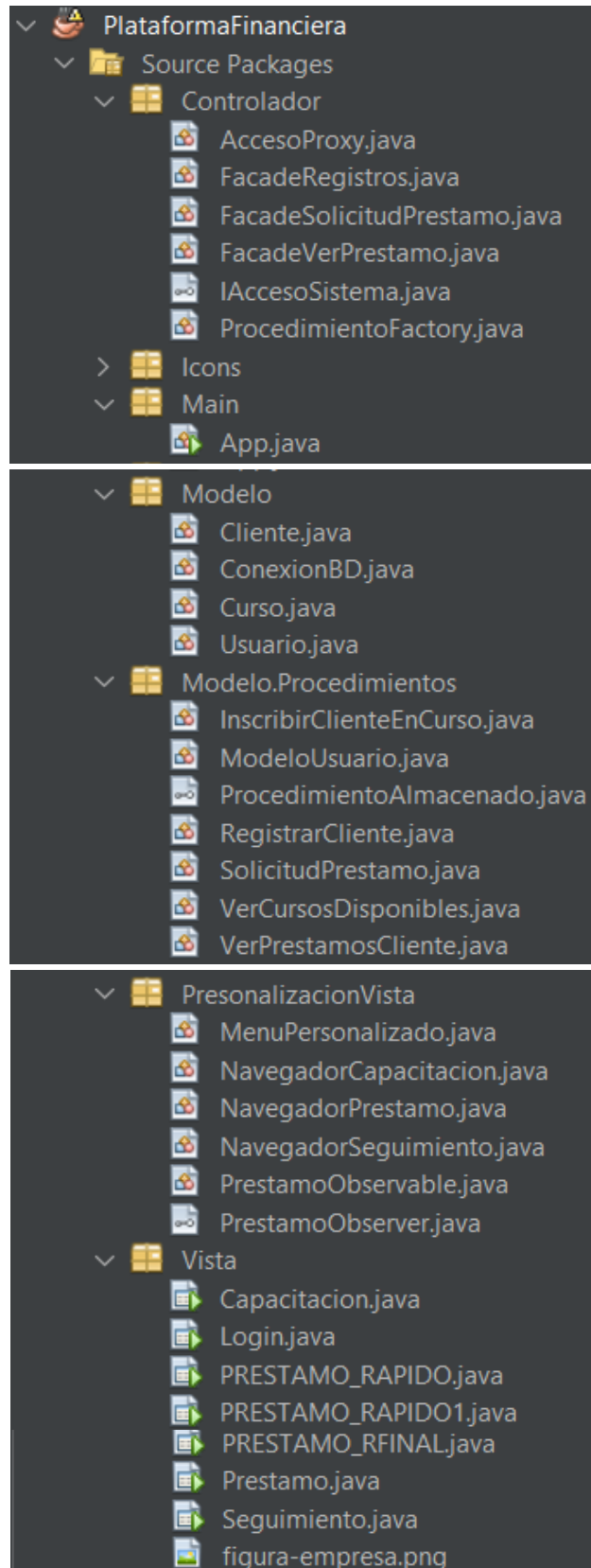
    public static JMenu crearMenu(String texto, String icono) {
        JMenu menu = new JMenu(texto);
        if (icono != null) {
            menu.setIcon(obtenerIcono(icono));
        }
        estilizar(menu);
        personalizarPopup(menu);
        return menu;
    }

    public void construirSobre(JMenuBar barra) {
        barra.add(menu); // inserta este sub-menú en la barra
    }

    public JMenu obtenerMenu() {
        return menu;
    }
}
```

2.5.4 Estructura del Proyecto en NetBeans

La imagen muestra la organización interna del proyecto FinNexus implementado en NetBeans, donde se aprecian los paquetes los cuales están alineados con la arquitectura en capas y la aplicación de patrones de diseño.



2.6. Programación de Base de Datos, Tablas y Store-Procedure

2.6.1 Módulo Capacitación

```
-- Tabla Clientes
CREATE TABLE Clientes (
    ClienteID INT PRIMARY KEY IDENTITY,
    DNI NVARCHAR(20) NOT NULL UNIQUE,
    Nombre NVARCHAR(100) NOT NULL,
    Email NVARCHAR(100) UNIQUE NOT NULL,
    Telefono NVARCHAR(20)
);
GO

-- Tabla Cursos
CREATE TABLE Cursos (
    CursoID INT PRIMARY KEY IDENTITY,
    Nombre NVARCHAR(100) NOT NULL,
    Descripcion NVARCHAR(255),
    DuracionHoras INT,
    FechaInicio DATE
);
GO

-- Tabla CursosTomados
CREATE TABLE CursosTomados (
    ID INT PRIMARY KEY IDENTITY,
    ClienteID INT,
    CursoID INT,
    FechaInscripcion DATE,
    CONSTRAINT FK_CursosTomados_Clientes FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID),
    CONSTRAINT FK_CursosTomados_Cursos FOREIGN KEY (CursoID) REFERENCES Cursos(CursoID)
);
GO
```

```
-- Insertar Clientes
INSERT INTO Clientes (Nombre, Email, Telefono)
VALUES
    ('Juan Pérez', 'juan.perez@ejemplo.com', '123456789'),
    ('Ana Gómez', 'ana.gomez@ejemplo.com', '987654321'),
    ('Carlos Ruiz', 'carlos.ruiz@ejemplo.com', '112233445');
GO

-- Insertar Cursos
INSERT INTO Cursos (Nombre, Descripcion, DuracionHoras, FechaInicio)
VALUES
    ('Educación Financiera', 'Curso de finanzas personales.', 20, '2025-06-01'),
    ('Desarrollo de Software', 'Curso para aprender desarrollo web.', 30, '2025-07-01'),
    ('Marketing Digital', 'Curso de marketing online y redes sociales.', 25, '2025-08-01');
GO

-- Insertar Cursos Tomados
INSERT INTO CursosTomados (ClienteID, CursoID, FechaInscripcion)
VALUES
    (1, 1, '2025-06-01'),
    (2, 2, '2025-07-01'),
    (3, 3, '2025-08-01');
GO
```

2.6.2 Módulo Préstamo

```
-- Tabla Prestamos
CREATE TABLE Prestamos (
    PrestamoID INT PRIMARY KEY IDENTITY,
    ClienteID INT NOT NULL,
    Monto DECIMAL(10,2) NOT NULL,
    FechaSolicitud DATE NOT NULL,
    Estado NVARCHAR(50) DEFAULT 'Pendiente',
    PlazoMeses INT,
    Ocupacion NVARCHAR(100),
    CONSTRAINT FK_Prestamos_Clientes FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);
GO
```

```
-- Tabla Transacciones
CREATE TABLE Transacciones (
    TransaccionID INT PRIMARY KEY IDENTITY,
    Tipo NVARCHAR(50), -- 'Ingreso' o 'Gasto'
    Monto DECIMAL(10,2),
    Fecha DATE,
    Descripcion NVARCHAR(255),
    ClienteID INT,
    PrestamoID INT NULL,
    CONSTRAINT FK_Transacciones_Clientes FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID),
    CONSTRAINT FK_Transacciones_Prestamos FOREIGN KEY (PrestamoID) REFERENCES Prestamos(PrestamoID)
);
GO
```

```
-- Insertar Prestamos
INSERT INTO Prestamos (ClienteID, Monto, FechaSolicitud, Estado)
VALUES
(1, 2000.00, '2025-05-01', 'Pendiente'),
(2, 1500.00, '2025-04-20', 'Aprobado'),
(3, 3000.00, '2025-03-15', 'Rechazado');

GO

-- Insertar Transacciones
INSERT INTO Transacciones (Tipo, Monto, Fecha, Descripcion, ClienteID, PrestamoID)
VALUES
('Ingreso', 2500.00, '2025-05-01', 'Depósito mensual', 1, NULL),
('Gasto', 1000.00, '2025-05-02', 'Pago de servicios', 1, NULL),
('Ingreso', 1500.00, '2025-04-25', 'Transferencia externa', 2, 2),
('Gasto', 500.00, '2025-04-26', 'Compra de materiales', 2, 2),
('Ingreso', 3500.00, '2025-03-10', 'Pago extraordinario', 3, NULL);

GO
```

2.6.3 Módulo Seguimiento

```
-- Tabla ActividadesCliente (registro de acciones por módulo)
CREATE TABLE ActividadesCliente (
    ActividadID INT PRIMARY KEY IDENTITY,
    ClienteID INT NOT NULL,
    TipoActividad NVARCHAR(50), -- 'Curso', 'Préstamo', 'Transaccion', 'Reporte', etc.
    ReferenciaID INT NOT NULL, -- ID relacionado al objeto (CursoID, PrestamoID, etc.)
    Fecha DATETIME DEFAULT GETDATE(),
    Descripcion NVARCHAR(255),
    CONSTRAINT FK_Actividad_Cliente FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);
GO

-- Tabla Reportes
CREATE TABLE Reportes (
    ReporteID INT PRIMARY KEY IDENTITY,
    ClienteID INT, -- Asociamos el reporte con un cliente
    FechaGeneracion DATE NOT NULL,
    TotalIngresos DECIMAL(10,2),
    TotalGastos DECIMAL(10,2),
    Balance DECIMAL(10,2),
    CONSTRAINT FK_Reportes_Clientes FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);
GO
```

```

-- Insertar Reportes
INSERT INTO Reportes (ClienteID, FechaGeneracion, TotalIngresos, TotalGastos, Balance)
VALUES
(1, '2025-05-03', 2500.00, 1000.00, 1500.00),
(2, '2025-04-30', 3000.00, 500.00, 2500.00),
(3, '2025-03-15', 3500.00, 0.00, 3500.00);

GO

-- Insertar ActividadesCliente
INSERT INTO ActividadesCliente (ClienteID, TipoActividad, ReferenciaID, Descripcion)
VALUES
(1, 'Curso', 1, 'Inscripción en Educación Financiera'),
(1, 'Transaccion', 1, 'Depósito mensual'),
(2, 'Prestamo', 2, 'Solicitud de préstamo aprobada'),
(2, 'Transaccion', 2, 'Transferencia externa'),
(3, 'Curso', 3, 'Inscripción en Marketing Digital');

GO

```

2.6.4 Procedimientos Almacenados

```

--procedimiento almacenado para registrar clientes

CREATE PROCEDURE RegistrarCliente
    @DNI NVARCHAR(20),
    @Nombre NVARCHAR(100),
    @Email NVARCHAR(100),
    @Telefono NVARCHAR(20),
    @MensajeSalida NVARCHAR(255) OUTPUT
AS
BEGIN
    INSERT INTO Clientes (DNI, Nombre, Email, Telefono)
    VALUES (@DNI, @Nombre, @Email, @Telefono);

    SET @MensajeSalida = 'Cliente registrado exitosamente.';
END;

GO

```

```

-- Procedimiento para Solicitar y Procesar un Préstamo

ALTER PROCEDURE SolicitarPrestamo
    @DNI NVARCHAR(8),
    @Monto DECIMAL(10, 2),
    @FechaSolicitud DATE,
    @Estado NVARCHAR(50),
    @PlazoMeses INT,
    @Ocupacion NVARCHAR(100),
    @MensajeSalida NVARCHAR(255) OUTPUT
AS
BEGIN
    DECLARE @ClienteID INT;

    -- Buscar ClienteID por DNI
    SELECT @ClienteID = ClienteID FROM Clientes WHERE DNI = @DNI;

    IF @ClienteID IS NOT NULL
    BEGIN
        -- Insertar nuevo préstamo
        INSERT INTO Prestamos (ClienteID, Monto, FechaSolicitud, Estado, PlazoMeses, Ocupacion)
        VALUES (@ClienteID, @Monto, @FechaSolicitud, @Estado, @PlazoMeses, @Ocupacion);

        SET @MensajeSalida = 'Solicitud de préstamo registrada con éxito.';
    END
    ELSE
    BEGIN
        SET @MensajeSalida = '✗ No se encontró un cliente con el DNI proporcionado.';
    END
END;

GO

```

```
--procedimiento almacenado para ver el estado del prestamo

ALTER PROCEDURE VerPrestamosCliente
    @DNI NVARCHAR(15)
AS
BEGIN
    SELECT
        p.FechaSolicitud AS Fecha,
        p.Monto,
        p.PlazoMeses AS Plazo,
        p.Estado
    FROM Prestamos p
    INNER JOIN Clientes c ON p.ClienteID = c.ClienteID
    WHERE c.DNI = @DNI;
END;
GO
```

2.7 Definición de roles y accesos

El sistema implementa un esquema de control de acceso basado en roles, con el fin de garantizar la seguridad y la correcta asignación de responsabilidades dentro de la aplicación. A continuación, se describen los roles definidos y los permisos asociados a cada uno.

Tabla 1

Tabla de Roles de la Plataforma Financiera

Rol	Descripción
Admin	Gestiona todo el sistema: cursos, clientes, reportes
Empleado	Registra préstamos, inscribe clientes, consulta datos
Cliente	Se registra solo en cursos y puede ver consultas relacionadas

2.8 Interfaz Visual

2.8.1 Wireframes

Préstamo Rápido (Paso 1)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

Paso 1

Paso 2

Paso 3

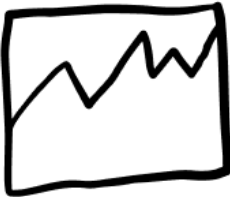
DNI

NOMBRE

CORREO

TELEFONO

MONTO



ACEPTAR

SIGUIENTE

Préstamo Rápido (Paso 2)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

Paso 1

Paso 2


Paso 3

MONTO

OCUPACION

MESES A PAGAR

CANTIDAD A PAGAR



ACEPTAR

SIGUIENTE

SALIR

Préstamo Rápido (Paso 3)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

Paso 1

Paso 2

Paso 3

FECHA	MONTO	PLAZO	ESTADO DEL PRESTAMO

TERMINOS Y CONDICIONES

ACEPTAR

SALIR

Inicio de Sesión (Roles)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

INICIAR SESION COMO



EMPLEADO



USUARIO


CONTINUAR

REGISTRARSE

Inicio de Sesión (Login)

Capacitación	Préstamo	Seguimiento	Préstamo Rápido	Inicia Sesión
--------------	----------	-------------	-----------------	---------------

BIENVENIDO



CORREO

CONTRASEÑA

INICIAR

VOLVER

Módulo Capacitación (Ver Cursos)

Capacitación	Préstamo	Seguimiento	Préstamo Rápido	Inicia Sesión
--------------	----------	-------------	-----------------	---------------

Ver Cursos

Inscribir Cursos

Consultar Cursos

VER CURSOS DISPONIBLES

CODIGO

NOMBRE DE CURSOS

FECHA



INICIAR

VOLVER

Módulo Préstamo (Registrar Préstamo)

Capacitación	Préstamo	Seguimiento	Préstamo Rápido	Inicia Sesión
--------------	----------	-------------	-----------------	---------------

Registrar Préstamo

Ver Préstamo

REGISTRAR PRESTAMO

MONTO

OCUPACION

MESES A PAGAR

CANTIDAD A PAGAR

INICIAR

VOLVER

Módulo Seguimiento (Registrar Transacción)

Capacitación	Préstamo	Seguimiento	Préstamo Rápido	Inicia Sesión
--------------	----------	-------------	-----------------	---------------

Registrar Transacción

Generar Reporte

Consultar Balance

REGISTRAR TRANSACCION

MONTO

DESCRIPCION

FECHA

CANTIDAD RESTANTE

INICIAR

VOLVER

2.8.2 Mockup Visual con Paleta de Colores

Préstamo Rápido (Paso 1)

Capacitación Préstamo Seguimiento Préstamo Rápido Inicia Sesión

Paso 1 Paso 2 Paso 3

DNI

NOMBRE

CORREO

TELEFONO

MONTO

ACEPTAR SIGUIENTE

A line graph icon is displayed on the right side of the form.

Préstamo Rápido (Paso 2)

Capacitación Préstamo Seguimiento Préstamo Rápido Inicia Sesión

Paso 1 Paso 2 Paso 3

MONTO

OCUPACION

MESES A PAGAR

CANTIDAD A PAGAR

ACEPTAR SIGUIENTE SALIR

A line graph icon is displayed on the right side of the form.

Préstamo Rápido (Paso 3)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

Paso 1

Paso 2

Paso 3

FECHA	MONTO	PLAZO	ESTADO DEL PRESTAMO

TERMINOS Y CONDICIONES

ACEPTAR

SALIR

Inicio de Sesión (Roles)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

INICIAR SESION COMO



EMPLEADO

CONTINUAR



USUARIO

REGISTRARSE

Inicio de Sesión (Login)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

BIENVENIDO



CORREO

CONTRASEÑA

INICIAR

VOLVER

Módulo Capacitación (Ver Cursos)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

Ver Cursos

Inscribir Cursos

Consultar Cursos

VER CURSOS DISPONIBLES

CODIGO

NOMBRE DE CURSOS

FECHA



INICIAR

VOLVER

Módulo Préstamo (Registrar Préstamo)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

Registrar Préstamo

Ver Préstamo

REGISTRAR PRESTAMO

MONTO

OCUPACION

MESES A PAGAR

CANTIDAD A PAGAR

INICIAR

VOLVER

Módulo Seguimiento (Registrar Transacción)

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Inicia Sesión

Registrar Transacción

Generar Reporte

Consultar Balance

REGISTRAR TRANSACCION

MONTO

DESCRIPCION

FECHA

CANTIDAD RESTANTE

INICIAR

VOLVER

2.8.3 Aplicación de colores

En esta sección se especifica la paleta de colores aplicada en la plataforma, considerando la psicología de colores y la identidad visual del ODS 8.

Tabla 2

Tabla de Psicología de Colores

Color	Hex	Significado / Psicología	Aplicación en la plataforma
Rojo burdeos	#A21942	Impulso, energía, compromiso social, dinamismo. Representa el objetivo del ODS 8.	Barra de navegación, botones principales (ej. "Aceptar", "Capacitación")
Azul profesional oscuro	#003B73	Confianza, seriedad, seguridad financiera, autoridad. Muy común en bancos y fintech.	Botón secundario (ej. "Siguiendo"), encabezados secundarios, texto de pasos
Gris muy claro (blanco sutil)	#FAFAFA	Limpieza, minimalismo, claridad. Da sensación de orden y frescura visual.	Fondo general para las secciones y formularios
Azul grisáceo moderno	#E5ECEF	Tecnología, modernidad, calma profesional. Ideal para productos digitales.	Fondo general de la interfaz
Gris oscuro	#1E1E1E	Profesionalismo, madurez, formalidad y alto contraste.	Texto principal, títulos, texto del botón "Salir" y "Volver"
Gris claro	#B0BEC5	Neutralidad, estructura. Suaviza los límites sin ser muy marcado.	Bordes de secciones, fondo de los botones "Salir" y "Volver"
Verde validación / éxito	#4CAF50	Crecimiento, aceptación, bienestar. Refuerza mensajes positivos y aprobaciones.	Mensajes de éxito: "Gracias por registrarte", validaciones correctas
Rojo advertencia	#D32F2F	Alerta, error, atención inmediata. Ideal para advertencias o validaciones negativas.	Mensajes de error: nombre corto, correo inválido, edad baja

2.8.4 Aplicación de los 10 principios de usabilidad

La experiencia del usuario es un factor esencial en el diseño y desarrollo de sistemas interactivos. Para asegurar que la plataforma de inclusión financiera sea intuitiva, eficiente y accesible, se han considerado los 10 principios de usabilidad de Jakob Nielsen, los cuales sirven como guía para evaluar y mejorar la interacción entre el usuario y el sistema.

A continuación, se detalla cómo estos principios han sido implementados en los distintos módulos de la aplicación:

1. Visibilidad del estado del sistema

El sistema informa al usuario sobre lo que está ocurriendo mediante mensajes como “Cargando datos...”, “Curso agregado con éxito”, o al mostrar el progreso del préstamo solicitado.

2. Correspondencia entre el sistema y el mundo real

Se emplean un lenguaje familiar para el usuario, utilizando términos como “usuario”, “préstamo” o “curso” lo que facilita la comprensión y reduce la curva de aprendizaje.

3. Control y libertad del usuario

Se permite cancelar acciones como el registro de préstamos o capacitaciones mediante botones “Cancelar” o “Atrás”, evitando acciones accidentales.

4. Consistencia y estándares

Los elementos visuales y de navegación mantienen un diseño uniforme entre módulos, con iconos y colores consistentes para acciones similares (como editar o eliminar).

5. Prevención de errores

El sistema valida los datos ingresados antes de procesarlos. Por ejemplo, evita que se puedan ingresar letras en campos numéricos como “monto del préstamo”, reduciendo errores comunes.

6. Reconocer antes que recordar

Se utilizan menús desplegables, autocompletado y campos precargados para facilitar la interacción sin exigir que el usuario recuerde información compleja.

7. Flexibilidad y eficiencia de uso

Se ha implementado una interfaz intuitiva que permite a usuarios expertos interactuar rápidamente con accesos directos, sin entorpecer a usuarios nuevos.

8. Diseño estético y minimalista

Se emplea una paleta de colores sobria (azul profesional, rojo burdeos, gris suave), acompañada de una estructura limpia que evita la sobrecarga visual y mantiene el enfoque en las acciones principales.

9. Ayudar a reconocer, diagnosticar y recuperarse de errores

Los mensajes de error son claros, como “El monto debe ser mayor a cero” o “Usuario no encontrado”, permitiendo una fácil corrección.

10. Ayuda y documentación

Cada módulo incluye una sección de ayuda básica para orientar al usuario sobre el uso del sistema.

2.9 Pruebas del software

2.9.1 Prueba en consola del Módulo Capacitación

```
Output - PlataformaFinanciera (run)

run:
Conexión exitosa a la base de datos.
Cursos Disponibles:
- ID: 1, Nombre: Educación Financiera, Descripción: Curso de finanzas personales., Duración: 20, Fecha Inicio: 2025-06-01
- ID: 2, Nombre: Desarrollo de Software, Descripción: Curso para aprender desarrollo web., Duración: 30, Fecha Inicio: 2025-07-01
- ID: 3, Nombre: Marketing Digital, Descripción: Curso de marketing online y redes sociales., Duración: 25, Fecha Inicio: 2025-08-01
Mostrando cursos disponibles:
Educación Financiera
Desarrollo de Software
Marketing Digital
El cliente ya está inscrito en este curso.
Cursos inscritos por el cliente:
- Educación Financiera | Inscrito el: 2025-06-01
- Desarrollo de Software | Inscrito el: 2025-07-01
BUILD SUCCESSFUL (total time: 2 seconds)
```

2.9.2 Prueba de interfaz

The screenshot shows the login interface of the FinNexus application. The top navigation bar is dark red and contains icons and labels for 'Capacitación', 'Préstamo', 'Seguimiento', 'Préstamo Rápido', and a user profile icon labeled 'Iniciar Sesión'. The main area is split into two panels. The left panel features a blue background with a business person in a suit interacting with a glowing line graph and bar chart, with a small box labeled 'COMUNICACION Y +AS'. The right panel is white and titled 'INICIAR SESION'. It contains two input fields: 'Correo Electronico' and 'Contraseña'. Below these fields are two buttons: 'INICIAR' and 'Volver'.

The screenshot shows the loan application interface titled 'Préstamos Fáciles con FinNexus'. The top navigation bar is identical to the previous screen. The main area is white and contains a form with several input fields. On the left, there are four fields labeled 'DNI:', 'NOMBRE:', 'EMAIL:', and 'TELÉFO...'. The values entered are '43567809', 'Marta Fernandez', 'Martina@gmail.com', and '915234567' respectively. To the right of these fields is a section titled '¿Cuánto necesitas?' (How much do you need?) with a text input field containing '1000' and a slider control below it. To the right of the slider is a label 'jLabel6'. At the bottom of the form are two buttons: 'Aceptar' (Accept) and 'Siguiente' (Next).

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Iniciar Sesión

Préstamos Fáciles con FinNexus

Ocupación

Administrador

Cuotas que necesitará pagar durante los meses

Monto Solicitado:

1000

Meses a pagar

3

Haz crecer tu negocio con nuestros préstamos

Aceptar

Siguiente

Cancelar

Capacitación

Préstamo

Seguimiento

Préstamo Rápido

Iniciar Sesión

Préstamos Fáciles con FinNexus

Fecha	Monto	Plazo	Estado
30/06/2025	1000	3	Pendiente

Términos y condiciones

Intereses aplicables

Requisitos de pago puntual por los meses que solicita

Uso de datos personales

Aceptación de contacto por medios digitales

Debe aceptar todos los términos para poder continuar con el préstamo solicitado

☐

Acepta los términos y condiciones

Aceptar

Cancelar

2.10 Antipatrones

2.10.1 Concepto y propósito

Los antipatrones representan soluciones aparentemente correctas, pero que conllevan consecuencias negativas en el diseño, mantenimiento o escalabilidad de un sistema.

A diferencia de los patrones, los antipatrones deben ser identificados para corregirse o evitarse a tiempo, promoviendo así buenas prácticas tanto a nivel técnico como organizacional.

2.10.2 Antipatrones de desarrollo de software

Tabla 3

Tabla de Antipatrones Encontrados en el Sistema

Antipatrón	Descripción	Problema	Solución Alternativa
Big Ball of Mud (Bola de Barro Gigante)	Código sin estructura clara, mezclando múltiples responsabilidades	Dificulta el mantenimiento, aumenta errores al modificar, y complica escalar o testear	Separar responsabilidades en capas (MVC), aplicar principios SOLID, y modularizar lógicamente el código
Magic Numbers (Números Mágicos)	Números fijos o valores literales duros en el código	Reduce legibilidad, dificulta cambios globales y puede inducir errores por valores “mágicos” mal entendidos	Usar constantes con nombres descriptivos, o enums cuando aplica
Golden Hammer (Martillo de Oro)	Aplicación del mismo patrón en todo, incluso donde no conviene	Puede llevar a sobreingeniería, ocultar lógica que debería ser explícita, o dificultar pruebas	Evaluar si realmente se necesita el patrón; en su lugar, aplicar el principio KISS (Keep It Simple)

2.10.3 Antipatrones organizacionales

Tabla 4

Tabla de Antipatrones Organizacionales Aplicados

Antipatrón	Descripción	Problema	Solución Alternativa
Blame Culture (Cultura de Culpar)	Se busca culpables en lugar de soluciones cuando algo falla	Inhibe la innovación, crea miedo al error, desmotiva	Promover cultura de aprendizaje, retroalimentación constructiva y mejora continua

Capítulo 3: Conclusiones

3.1 Conclusiones por unidad

Unidad I: Principios de Diseño y Patrones de Software

En esta primera unidad se abordaron los principios SOLID, lo que representó una nueva forma de entender cómo se debe estructurar un sistema para mejorar su mantenibilidad y escalabilidad. Este enfoque nos permitió repensar el diseño del código no solo como una serie de instrucciones que “funcionan”, sino como una arquitectura que puede crecer y adaptarse a futuros cambios. Comprender estos principios fue clave para evitar errores comunes y tener una base sólida al momento de implementar patrones más complejos.

Unidad II: Patrones de Diseño Creacionales

Durante esta unidad se profundizó en los patrones creacionales como Singleton, Prototype y Factory Method. Aunque ya conocíamos Singleton de forma superficial, en esta unidad se comprendió mejor su utilidad práctica al gestionar una única instancia en todo el sistema. Asimismo, descubrir la utilidad de Prototype y Factory Method, especialmente en contextos como el uso de procedimientos almacenados en bases de datos, representó un aprendizaje muy significativo. Estos patrones nos facilitaron tener modularidad en el sistema y nos ayudaron a optimizar procesos que antes se resolvían con técnicas menos eficientes.

Unidad III: Patrones de Diseño Estructurales

El estudio de los patrones estructurales permitió entender cómo organizar y simplificar la interacción entre clases y objetos. Patrones como Decorator demostraron ser una forma útil de extender funcionalidades sin modificar clases existentes, mientras que el patrón Proxy nos permitió controlar el acceso a recursos sensibles. También aprendimos el valor del patrón Facade para centralizar llamadas complejas en una única interfaz, así como de Composite que fue útil para estructurar jerárquicamente elementos del sistema. En conjunto, estos patrones nos facilitaron tener una codificación más limpia y una comprensión más clara del flujo del sistema.

Unidad IV: Patrones de Comportamiento y Antipatrones

En esta unidad se exploraron los patrones de comportamiento como Observer y Command. El patrón Observer resultó muy útil para coordinar la reacción de múltiples objetos ante algún cambio de estado en la plataforma, mientras que Command nos permitió encapsular solicitudes como objetos, facilitando la programación de acciones más flexibles y reutilizables.

Por otra parte, al estudiar los antipatrones pudimos tomar consciencia sobre los errores comunes de diseño al momento de desarrollarr un sistema,. Se reconocieron casos como el Big Ball of Mud o el God Object, esto nos sirvió para reflexionar sobre algunas prácticas erróneas que se cometieron en proyectos pasados, las cuales comprometían la calidad del software. También se aprendió sobre los antipatrones organizacionales, lo que permitió identificar qué prácticas evitar al momento de trabajar en equipo o distribuir las responsabilidades de forma poco eficiente.

3.2 Conclusiones por cada capa de arquitectura

Capa Vista:

Se aprendió a separar correctamente la interfaz gráfica del resto de la lógica del sistema. Se implementaron vistas independientes que interactúan mediante eventos y se reforzó el uso de patrones como Observer y Command para controlar el flujo entre formularios de forma dinámica y ordenada.

Capa Controlador:

El controlador centralizó la lógica de conexión entre la vista y el modelo. Aquí se integraron patrones como Facade y Command para gestionar acciones de negocio de manera más limpia, facilitando la extensión del sistema y reduciendo el acoplamiento entre capas.

Capa Modelo:

Se organizó correctamente la lógica relacionada con la base de datos, incluyendo procedimientos almacenados. Se aprendió a encapsular operaciones específicas en clases dedicadas y se comprendió la importancia de mantener esta capa libre de detalles de interfaz, enfocándose solo en los datos y la lógica de negocio.

3.2 Conclusiones personales

Este proyecto fue fundamental para comprender cómo trabajar de forma estructurada y profesional, aplicando buenas prácticas de desarrollo, como el uso de la arquitectura en capas y el patrón MVC, lo que permitió organizar correctamente la lógica, la presentación y la interacción con la base de datos. La implementación y estudio de los patrones de diseño nos permitió abordar problemas comunes al momento de desarrollar software. Patrones como Singleton, Factory, Facade, Command u Observer resultaron ser clave para construir un sistema robusto, escalable y mantenible. Esto nos ayudó a mejorar nuestro conocimiento, comprensión y habilidades en Java, especialmente en temas como la programación orientada a objetos (POO), el diseño modular y el desarrollo de evento en JFrame.

Uno de los aspectos más valiosos fue el estudio de los antipatrones, ya que nos permitió identificar errores comunes en el desarrollo de software y reflexionar sobre prácticas inadecuadas que posiblemente se aplicaron en proyectos anteriores. Todo ello contribuyó a desarrollar una visión más crítica y responsable sobre cómo escribir código limpio, comprensible y sostenible en el tiempo.

Referencias Bibliográficas

- Acosta Yerovi, M. L. (2014). *Estudio de patrones de diseño en plataforma Java Enterprise Edition versión 6 para el desarrollo de aplicaciones web* [Tesis de pregrado]. Universidad Técnica del Norte.
- Brown, W. J., Malveau, R. C., McCormick III, H. W., & Mowbray, T. J. (1998). *AntiPatterns: Refactoring software, architectures, and projects in crisis*. John Wiley & Sons.
- Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head first design patterns*. O'Reilly Media.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
- López Ortega, J. E., & Mayta Alfaro, M. A. (2024). *Plataforma Fintech y la inclusión financiera en una empresa del distrito de Miraflores – Lima* [Tesis de maestría]. Universidad de San Martín de Porres.
- Martin, R. C. (2003). *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall.
- Shalloway, A., & Trott, J. R. (2004). *Design patterns explained: A new perspective on object-oriented design* (2nd ed.). Addison-Wesley.
- Tambini, J., Paliza, M., & Ramírez, D. (2024). Digitalización e inclusión financiera en el Perú. *Moneda*, (197). Banco Central de Reserva del Perú.

Anexos

GitHub

Link: <https://github.com/LuisAS7/FinNexus.git>

Drive

Link:

<https://drive.google.com/file/d/1QZyji1EdhqSF88S7oKAA7G2bMWtOqliP/view?usp=sharing>

Excel (Antipatrones)



S14_Antipatrones_Revision.xlsx