

ENTIDAD COMPROBANTE-INGRESOS

```
package edu.mtisw.payrollbackend.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "comprobante_ingresos")
@Data
@NoArgsConstructor
@AllArgsConstructor
@NoArgsConstructor
public class ComprobanteIngresosEntity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true, nullable = false, name = "id")
    private Long id;

    @Column(name = "antiguedad_laboral")
    private int antiguedadLaboral; // años en el empleo actual

    @Column(name = "ingreso_mensual")
    private int ingresoMensual;

    @Column(name = "ingresos_ultimos_24_meses")
    private String ingresosUltimos24Meses;

    @Column(name = "saldo")
    private int saldo; // saldo en cuenta de ahorro o inversiones

    @Column(name = "deudas")
    private int deudas; // deudas actuales

    @Column(name = "cantidad_deudas_pendientes")
    private int cantidadDeudasPendientes; // Número de deudas pendientes

    @Column(name = "saldos_mensuales")
    private String saldosMensuales; // Saldos mensuales de los últimos 12 meses, separados por comas

    @Column(name = "depositos_ultimos_12_meses")
    private String depositosUltimos12Meses; // Montos de los depósitos, separados por comas

    @Column(name = "antiguedad_cuenta")
    private int antiguedadCuenta; // Años de antigüedad de la cuenta de ahorros

    @Column(name = "retiros_ultimos_6_meses")
    private String retirosUltimos6Meses; // Montos de los retiros en los últimos 6 meses, separados por comas
}
```

ENTIDAD PRESTAMO

```
package edu.mtisw.payrollbackend.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;

import java.io.Serializable;

@Entity
@Table(name = "prestamo")
@Data
@NoArgsConstructor
@AllArgsConstructor
@NoArgsConstructor
public class PrestamoEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true, nullable = false, name = "id")
    private Long id;

    @Column(name = "tipo") // entre "Primera vivienda", "Segunda vivienda", "Propiedades comerciales", "Remodelacion"
    private String tipo;

    @Column(name = "plazo")
    private int plazo;

    @Column(name = "tasa_interes")
    private double tasaInteres;

    @Column(name = "monto")
    private int monto;

    @Column(name = "estado") // entre "Aprobado", "Rechazado", "En proceso"
    private String estado;

    @Column(name = "valor_propiedad")
    private int valorPropiedad; // Valor de la propiedad
}
```

ENTIDAD USUARIO-COMPROBANTE-INGRESOS

```
package edu.mtisw.payrollbackend.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;

import jakarta.persistence.*;

@Entity
@Table(name = "usuario_comprobante_ingresos")
@Data
@NoArgsConstructor
@AllArgsConstructor

public class UsuarioComprobanteIngresosEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true, nullable = false, name = "id")
    private Long id;

    @Column(name = "id_usuario")
    private Long idUsuario;

    @Column(name = "id_comprobante_ingresos")
    private Long idComprobanteIngresos;
}
```

ENTIDAD USUARIO

```
package edu.mtisw.payrollbackend.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;

import jakarta.persistence.*;

@Entity
@Table(name = "usuario")
@Data
@NoArgsConstructor
@AllArgsConstructor

public class UsuarioEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true, nullable = false, name = "id")
    private Long id;

    @Column(name = "rut")
    private String rut;

    @Column(name = "nombre")
    private String nombre;

    @Column(name = "apellido")
    private String apellido;

    @Column(name = "edad")
    private int edad;

    @Column(name = "tipo_empleado")
    private String tipoEmpleado; // "Empleado" o "Independiente"
}
```

ENTIDAD USUARIO-PRESTAMO

```
package edu.mtisw.payrollbackend.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;

import jakarta.persistence.*;

@Entity
@Table(name = "usuario_prestamo")
@Data
@NoArgsConstructor
@AllArgsConstructor

public class UsuarioPrestamoEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true, nullable = false, name = "id")
    private Long id;

    @Column(name = "id_usuario")
    private Long idUsuario;

    @Column(name = "id_prestamo")
    private Long idPrestamo;
}
```

REPOSITORIO PRESTAMO

```
package edu.mtisw.payrollbackend.repositories;

import edu.mtisw.payrollbackend.entities.PrestamoEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface PrestamoRepository extends JpaRepository<PrestamoEntity, Long> {
}
```

REPOSITORIO COMPROBANTE-INGRESOS

```
package edu.mtisw.payrollbackend.repositories;

import edu.mtisw.payrollbackend.entities.ComprobanteIngresosEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ComprobanteIngresosRepository extends JpaRepository<ComprobanteIngresosEntity, Long>{
}
```

REPOSITORIO USUARIO-COMPROBANTE-INGRESOS

```
ackage edu.mtisw.payrollbackend.repositories;

import edu.mtisw.payrollbackend.entities.UsuarioComprobanteIngresosEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public interface UsuarioComprobanteIngresosRepository extends JpaRepository<UsuarioComprobanteIngresosEntity, Long> {
    Optional<UsuarioComprobanteIngresosEntity> findByIdUsuario(Long idUsuario);
}
```

REPOSITORIO USUARIO-PRESTAMO

```
package edu.mtisw.payrollbackend.repositories;

import edu.mtisw.payrollbackend.entities.UsuarioPrestamoEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public interface UsuarioPrestamoRepository extends JpaRepository<UsuarioPrestamoEntity, Long> {
    Optional<UsuarioPrestamoEntity> findByIdUsuario(Long idUsuario);
}
```

REPOSITORIO USUARIO

```
package edu.mtisw.payrollbackend.repositories;

import edu.mtisw.payrollbackend.entities.UsuarioEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UsuarioRepository extends JpaRepository<UsuarioEntity, Long>{
}
```

SERVICIO BANCO

```
package edu.mtisw.payrollbackend.services;

import edu.mtisw.payrollbackend.entities.*;
import edu.mtisw.payrollbackend.repositories.*;
import jakarta.persistence.Id;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.*;
```

```

@Service
public class BancoService {
    @Autowired
    UsuarioRepository usuarioRepository;
    @Autowired
    ComprobanteIngresosRepository comprobanteIngresosRepository;
    @Autowired
    PrestamoRepository prestamoRepository;
    @Autowired
    UsuarioPrestamoRepository usuarioPrestamoRepository;
    @Autowired
    UsuarioComprobanteIngresosRepository usuarioComprobanteIngresosRepository;
    @Autowired
    UsuarioService usuarioService;

    // -----CRUD-----
    // -----PRINCIPALES-----

    // Evaluar Relación Cuota/Ingreso (R1)
    public boolean evaluarRelacionCuotaIngreso(Long idUsuario) throws Exception {
        System.out.println("Evaluando relación cuota ingreso para usuario: " + idUsuario);

        UsuarioEntity usuario = usuarioRepository.findById(idUsuario)
            .orElseThrow(() -> new Exception("Usuario no encontrado"));
        System.out.println("Usuario encontrado: " + usuario);

        UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
            .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
        System.out.println("Comprobante de ingresos encontrado: " + usuarioComprobanteIngresos);

        ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
            .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));
        System.out.println("Ingreso mensual: " + comprobanteIngresos.getIngresoMensual());

        UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findById(idUsuario)
            .orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));

        PrestamoEntity prestamo = prestamoRepository.findById(usuarioPrestamo.getIdPrestamo())
            .orElseThrow(() -> new Exception("Préstamo no encontrado"));
        System.out.println("Préstamo encontrado: " + prestamo);

        // Cálculo de la relación cuota ingreso
        double tasaInteresMensual = (prestamo.getTasaInteres() / 12) / 100;
        int numeroPagos = prestamo.getPlazo() * 12;
        double pagoMensual = (prestamo.getMonto() * tasaInteresMensual * Math.pow(1 + tasaInteresMensual, numeroPagos)) /
            (Math.pow(1 + tasaInteresMensual, numeroPagos) - 1);
        double relacionCuotaIngreso = (pagoMensual / comprobanteIngresos.getIngresoMensual()) * 100;
        System.out.println("Relación cuota ingreso: " + relacionCuotaIngreso);

        return relacionCuotaIngreso <= 35;
    }

    // Evaluar Historial Crediticio del Cliente (R2)
    public boolean evaluarHistorialCrediticio(Long idUsuario) throws Exception {
        // Obtener el comprobante de ingresos asociado al usuario
        UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
            .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
        ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
            .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

        int cantidadDeudasPendientes = comprobanteIngresos.getCantidadDeudasPendientes(); // Número de deudas pendientes
        int montoTotalDeudasPendientes = comprobanteIngresos.getDeudas(); // Monto total de deudas pendientes

        // Obtener el ingreso mensual del cliente
        int ingresoMensual = comprobanteIngresos.getIngresoMensual();

        // Calcular el porcentaje de las deudas pendientes sobre los ingresos mensuales
        double porcentajeDeudasSobreIngresos = ((double) montoTotalDeudasPendientes / ingresoMensual) * 100;

        // Definir umbrales para rechazar la solicitud
        boolean demasiadasDeudasPendientes = cantidadDeudasPendientes > 3; // Umbral de 3 deudas pendientes
        boolean deudasExcesivas = porcentajeDeudasSobreIngresos > 30; // Umbral del 30% sobre ingresos

        if (demasiadasDeudasPendientes || deudasExcesivas) {
            // Si tiene demasiadas deudas pendientes o las deudas superan el 30% de los ingresos, se rechaza la solicitud
            return false;
        }
        // Si las deudas pendientes están dentro de un rango aceptable, se acepta
        return true;
    }

    // Evaluar Antigüedad Laboral y Estabilidad (R3)
    public boolean evaluarAntigüedad(Long idUsuario) throws Exception {
        // Obtener el usuario por ID
        UsuarioEntity usuario = usuarioRepository.findById(idUsuario)
            .orElseThrow(() -> new Exception("Usuario no encontrado"));

        // Obtener el comprobante de ingresos asociado al usuario
        UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
            .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
        ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
            .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

        if (usuario.getTipoEmpleado().equalsIgnoreCase("Empleado")) {
            // Si es empleado, verificar que tenga al menos 1 año de antigüedad laboral
            int antigüedadLaboral = comprobanteIngresos.getAntigüedadLaboral();
            if (antigüedadLaboral >= 1) {
                return true; // Cumple con la antigüedad requerida
            } else {
                return false; // No cumple con la antigüedad requerida
            }
        } else if (usuario.getTipoEmpleado().equalsIgnoreCase("Independiente")) {
            // Si es independiente, revisar ingresos de los últimos 2 años
            String ingresosUltimos24MesesStr = comprobanteIngresos.getIngresosUltimos24Meses().replace("[", "").replace("]", "");
            String[] ingresosArray = ingresosUltimos24MesesStr.split(",");
            if (ingresosArray.length >= 24) {

```

```

        // Tiene registros de ingresos de los últimos 24 meses
        // Aquí podrías evaluar la estabilidad financiera según tus criterios
        // Por simplicidad, asumiremos que cumple si tiene ingresos en los últimos 24 meses
        return true;
    } else {
        return false; // No tiene suficientes datos de ingresos
    }
} else {
    throw new Exception("Tipo de empleado desconocido");
}
}

// Evaluar Relación Deuda/Ingreso (R4)
public boolean evaluarRelacionDeudaIngreso(Long idUsuario) throws Exception {
    // Obtener el préstamo asociado al usuario
    UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));
    Long idPrestamo = usuarioPrestamo.getIdPrestamo();

    // Obtener el usuario por ID
    UsuarioEntity usuario = usuarioRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Usuario no encontrado"));

    // Obtener el comprobante de ingresos asociado al usuario
    UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
    ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
        .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

    // Obtener el ingreso mensual del usuario
    int ingresoMensual = comprobanteIngresos.getIngresoMensual();

    // Obtener las deudas actuales del usuario
    int deudasActuales = comprobanteIngresos.getDeudas();

    // Obtener el préstamo por ID
    PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)
        .orElseThrow(() -> new Exception("Préstamo no encontrado"));

    // Calcular la cuota mensual del nuevo préstamo
    double tasaInteresAnual = prestamo.getTasaInteres();
    int plazoEnAnios = prestamo.getPlazo();
    int monto = prestamo.getMonto();

    // Calcular la tasa de interés mensual
    double tasaInteresMensual = (tasaInteresAnual / 12) / 100;

    // Número de pagos (meses)
    int numeroPagos = plazoEnAnios * 12;

    // Cálculo del pago mensual usando la fórmula de amortización
    double cuotaNueva = (monto * tasaInteresMensual * Math.pow(1 + tasaInteresMensual, numeroPagos)) /
        (Math.pow(1 + tasaInteresMensual, numeroPagos) - 1);

    // Sumar la cuota nueva a las deudas actuales
    double totalDeudas = deudasActuales + cuotaNueva;

    // Calcular la relación deuda/ingreso en porcentaje
    double relacionDeudaIngreso = (totalDeudas / ingresoMensual) * 100;

    // Verificar si la relación es mayor que el 50%
    if (relacionDeudaIngreso > 50) {
        // La solicitud debe ser rechazada
        return false;
    } else {
        // La solicitud puede continuar
        return true;
    }
}

// Evaluar Monto Máximo de Financiamiento (R5)
public boolean evaluarMontoMaximoFinanciamiento(Long idUsuario) throws Exception {
    // Obtener el préstamo asociado al usuario
    UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));
    Long idPrestamo = usuarioPrestamo.getIdPrestamo();

    // Obtener el préstamo por ID
    PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)
        .orElseThrow(() -> new Exception("Préstamo no encontrado"));
    String tipoPrestamo = prestamo.getTipo();
    int valorPropiedad = prestamo.getValorPropiedad();
    int montoSolicitado = prestamo.getMonto();

    // Definir el porcentaje máximo según el tipo de préstamo
    double porcentajeMaximo;
    switch (tipoPrestamo.toLowerCase()) {
        case "primera vivienda":
            porcentajeMaximo = 0.80; // 80%
            break;
        case "segunda vivienda":
            porcentajeMaximo = 0.70; // 70%
            break;
        case "propiedades comerciales":
            porcentajeMaximo = 0.60; // 60%
            break;
        case "remodelacion":
            porcentajeMaximo = 0.50; // 50%
            break;
        default:
            throw new Exception("Tipo de préstamo desconocido");
    }

    // Calcular el monto máximo permitido
    double montoMaximoPermitido = valorPropiedad * porcentajeMaximo;

```

```

// Verificar si el monto solicitado excede el máximo permitido
if (montoSolicitado <= montoMaximoPermitido) {
    // La solicitud puede continuar
    return true;
} else {
    // La solicitud debe ser rechazada
    return false;
}
}

// Evaluar Edad del Solicitante (R6)
public boolean evaluarEdad(Long idUsuario) throws Exception {
    // Obtener el préstamo asociado al usuario
    UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));
    Long idPrestamo = usuarioPrestamo.getIdPrestamo();

    // Obtener el usuario por ID
    UsuarioEntity usuario = usuarioRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Usuario no encontrado"));

    // Obtener el préstamo por ID
    PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)
        .orElseThrow(() -> new Exception("Préstamo no encontrado"));

    int edadActual = usuario.getEdad();
    int plazoPrestamo = prestamo.getPlazo(); // En años

    // Calcular la edad al finalizar el préstamo
    int edadAlFinalizarPrestamo = edadActual + plazoPrestamo;

    // Verificar si la edad al finalizar el préstamo excede el límite
    if (edadAlFinalizarPrestamo >= 70) {
        // La solicitud debe ser rechazada
        return false;
    } else {
        // La solicitud puede continuar
        return true;
    }
}

// Evaluar Saldo Mínimo Requerido (R71)
public boolean evaluarSaldoMinimo(Long idUsuario) throws Exception {
    // Obtener el préstamo asociado al usuario
    UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));
    Long idPrestamo = usuarioPrestamo.getIdPrestamo();

    // Obtener el comprobante de ingresos asociado al usuario
    UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
    ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
        .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

    // Obtener el saldo del cliente
    int saldoCliente = comprobanteIngresos.getSaldo();

    // Obtener el préstamo por ID
    PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)
        .orElseThrow(() -> new Exception("Préstamo no encontrado"));

    // Obtener el monto del préstamo solicitado
    int montoPrestamo = prestamo.getMonto();

    // Calcular el 10% del monto del préstamo
    double montoRequerido = montoPrestamo * 0.10;

    // Verificar si el saldo del cliente cumple con el mínimo requerido
    if (saldoCliente >= montoRequerido) {
        // El cliente cumple con el saldo mínimo requerido
        return true;
    } else {
        // El cliente no cumple con el saldo mínimo requerido
        return false;
    }
}

//evaluarHistorialAhorroConsistente(R72)
public boolean evaluarHistorialAhorroConsistente(Long idUsuario) throws Exception {
    // Obtener el comprobante de ingresos asociado al usuario
    UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
    ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
        .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

    // Obtener los saldos mensuales
    String saldosMensualesStr = comprobanteIngresos.getSaldosMensuales();
    String[] saldosArray = saldosMensualesStr.split(",");
    if (saldosArray.length < 12) {
        throw new Exception("No hay suficientes datos de saldos mensuales");
    }
    // Convertir a una lista de enteros
    List<Double> saldosMensuales = new ArrayList<>();
    for (String saldoStr : saldosArray) {
        saldosMensuales.add(Double.parseDouble(saldoStr));
    }
    // Verificar que el saldo haya sido positivo durante los últimos 12 meses
    for (Double saldo : saldosMensuales) {
        if (saldo <= 0) {
            // Se encontró un saldo no positivo
            return false;
        }
    }
    // No se encontraron saldos negativos en los últimos 12 meses
    return true;
}
}

```

```

// Evaluar Depósitos Periódicos (R73)
public boolean evaluarDepositosPeriodicos(Long idUsuario) throws Exception {
    // Obtener el comprobante de ingresos asociado al usuario
    UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
    ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
        .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

    // Obtener los depósitos de los últimos 12 meses y limpiar la cadena
    String depositosStr = comprobanteIngresos.getDepositosUltimos12Meses().replace("[", "").replace("]", "");
    String[] depositosArray = depositosStr.split(",");
    if (depositosArray.length < 12) {
        throw new Exception("No hay suficientes datos de depósitos");
    }

    // Convertir a una lista de Double
    List<Double> depositosMensuales = new ArrayList<>();
    for (String depositoStr : depositosArray) {
        depositoStr = depositoStr.trim(); // Eliminar espacios en blanco
        depositosMensuales.add(Double.parseDouble(depositoStr));
    }

    // Obtener el ingreso mensual
    int ingresoMensual = comprobanteIngresos.getIngresoMensual();
    double montoMinimoDeposito = ingresoMensual * 0.05;

    // Variables para contar depósitos regulares
    int depositosMensualesCount = 0;
    int depositosTrimestralesCount = 0;

    // Verificar depósitos mensuales
    for (Double deposito : depositosMensuales) {
        if (deposito >= montoMinimoDeposito) {
            depositosMensualesCount++;
        }
    }

    // Verificar si hay al menos 12 depósitos mensuales
    if (depositosMensualesCount >= 12) {
        return true; // Cumple con depósitos mensuales regulares
    }

    // Verificar depósitos trimestrales
    for (int i = 0; i < depositosMensuales.size(); i += 3) {
        double sumaTrimestre = depositosMensuales.get(i);
        if (i + 1 < depositosMensuales.size()) sumaTrimestre += depositosMensuales.get(i + 1);
        if (i + 2 < depositosMensuales.size()) sumaTrimestre += depositosMensuales.get(i + 2);
        if (sumaTrimestre >= montoMinimoDeposito * 3) {
            depositosTrimestralesCount++;
        }
    }

    // Verificar si hay al menos 4 depósitos trimestrales
    if (depositosTrimestralesCount >= 4) {
        return true; // Cumple con depósitos trimestrales regulares
    }

    // No cumple con la regularidad o monto mínimo
    return false;
}

// Evaluar Relación Saldo/Antigüedad (R74)
public boolean evaluarRelacionSaldoAntigüedad(Long idUsuario) throws Exception {
    // Obtener el préstamo asociado al usuario
    UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));
    Long idPrestamo = usuarioPrestamo.getIdPrestamo();

    // Obtener el comprobante de ingresos asociado al usuario
    UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));
    ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
        .orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

    // Obtener la antigüedad de la cuenta y el saldo acumulado
    int antigüedadCuenta = comprobanteIngresos.getAntigüedadCuenta();
    int saldoAcumulado = comprobanteIngresos.getSaldo();

    // Obtener el préstamo por ID
    PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)
        .orElseThrow(() -> new Exception("Préstamo no encontrado"));

    // Obtener el monto del préstamo solicitado
    int montoPrestamo = prestamo.getMonto();

    // Determinar el porcentaje requerido según la antigüedad
    double porcentajeRequerido = (antigüedadCuenta < 2) ? 0.20 : 0.10;

    // Calcular el monto requerido
    double montoRequerido = montoPrestamo * porcentajeRequerido;

    // Verificar si el saldo acumulado cumple con el monto requerido
    if (saldoAcumulado >= montoRequerido) {
        // El cliente cumple con la relación saldo/antigüedad
        return true;
    } else {
        // No cumple con los porcentajes requeridos
        return false;
    }
}

// Evaluar Retiros Recientes (R75)
public boolean evaluarRetirosRecientes(Long idUsuario) throws Exception {
    // Obtener el comprobante de ingresos asociado al usuario
    UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Comprobante de ingresos no asociado al usuario"));

```

```

ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosRepository.findById(usuarioComprobanteIngresos.getIdComprobanteIngresos())
.orElseThrow(() -> new Exception("Comprobante de ingresos no encontrado"));

```

```

// Obtener los retiros de los últimos 6 meses y limpiar la cadena
String retirosStr = comprobanteIngresos.getRetirosUltimos6Meses().replace("[", "").replace("]", "");
String[] retirosArray = retirosStr.split("");
if (retirosArray.length < 6) {
    throw new Exception("No hay suficientes datos de retiros");
}

// Obtener los saldos mensuales y limpiar la cadena
String saldosMensualesStr = comprobanteIngresos.getSalDOSmensuales().replace("[", "").replace("]", "");
String[] saldosArray = saldosMensualesStr.split("");
if (saldosArray.length < 6) {
    throw new Exception("No hay suficientes datos de saldos mensuales");
}

```

```

// Tomar los últimos 6 saldos y retiros
List<Double> retirosMensuales = new ArrayList<>();
List<Double> saldosMensuales = new ArrayList<>();
for (int i = saldosArray.length - 6; i < saldosArray.length; i++) {
    String saldoStr = saldosArray[i].trim(); // Eliminar espacios en blanco
    saldosMensuales.add(Double.parseDouble(saldoStr));
}
for (int i = retirosArray.length - 6; i < retirosArray.length; i++) {
    String retiroStr = retirosArray[i].trim(); // Eliminar espacios en blanco
    retirosMensuales.add(Double.parseDouble(retiroStr));
}

```

```

// Verificar si hay algún retiro superior al 30% del saldo correspondiente
for (int i = 0; i < 6; i++) {
    double saldo = saldosMensuales.get(i);
    double retiro = retirosMensuales.get(i);
    if (saldo == 0) {
        // Evitar división por cero
        continue;
    }
    double porcentajeRetiro = (retiro / saldo) * 100;
    if (porcentajeRetiro > 30) {
        // Se ha realizado un retiro superior al 30% del saldo
        return false;
    }
}

```

```

// No se encontraron retiros superiores al 30% del saldo en los últimos 6 meses
return true;
}

```

```

// Evaluar Capacidad de Ahorro (R7)
public Map<String, Object> evaluarCapacidadAhorro(Long idUsuario) throws Exception {
    Map<String, Object> resultado = new HashMap<>();
    int reglasCumplidas = 0;

```

```

// Evaluar R71: Saldo Mínimo Requerido
boolean r71 = evaluarSaldoMinimo(idUsuario);
if (r71) reglasCumplidas++;

```

```

// Evaluar R72: Historial de Ahorro Consistente
boolean r72 = evaluarHistorialAhorroConsistente(idUsuario);
if (r72) reglasCumplidas++;

```

```

// Evaluar R73: Depósitos Periódicos
boolean r73 = evaluarDepositosPeriodicos(idUsuario);
if (r73) reglasCumplidas++;

```

```

// Evaluar R74: Relación Saldo/Años de Antigüedad
boolean r74 = evaluarRelacionSaldoAntigüedad(idUsuario);
if (r74) reglasCumplidas++;

```

```

// Evaluar R75: Retiros Recientes
boolean r75 = evaluarRetirosRecientes(idUsuario);
if (r75) reglasCumplidas++;

```

```

String capacidadAhorro;
if (reglasCumplidas == 5) {
    capacidadAhorro = "sólida";
} else if (reglasCumplidas >= 3) {
    capacidadAhorro = "moderada";
} else {
    capacidadAhorro = "insuficiente";
}

```

```

resultado.put("capacidadAhorro", capacidadAhorro);
resultado.put("reglasCumplidas", reglasCumplidas);
resultado.put("detalles", Map.of(
    "R71", r71,
    "R72", r72,
    "R73", r73,
    "R74", r74,
    "R75", r75
));
return resultado;
}

```

```

// Evaluar Crédito (P4)
public Map<String, Object> evaluarCredito(Long idUsuario) throws Exception {
    Map<String, Object> resultado = new HashMap<>();
    Map<String, Boolean> reglasCumplidas = new HashMap<>();
    boolean aprobado = true;

```

```

// Obtener el préstamo asociado al usuario
UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findByUsuario(idUsuario)
.orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));
Long idPrestamo = usuarioPrestamo.getIdPrestamo();

```

```

// Evaluar R1: Relación Cuota/Ingreso

```



```

boolean r1 = evaluarRelacionCuotaIngreso(idUsuario);
reglasCumplidas.put("Relación Cuota/Ingreso Aprobada", r1);
if (!r1) aprobado = false;

// Evaluar R2: Historial Crediticio del Cliente
boolean r2 = evaluarHistorialCrediticio(idUsuario);
reglasCumplidas.put("Historial Crediticio del Cliente Aprobado", r2);
if (!r2) aprobado = false;

// Evaluar R3: Antigüedad Laboral y Estabilidad
boolean r3 = evaluarAntigüedad(idUsuario);
reglasCumplidas.put("Antigüedad Laboral y Estabilidad Aprobada", r3);
if (!r3) aprobado = false;

// Evaluar R4: Relación Deuda/Ingreso
boolean r4 = evaluarRelacionDeudaIngreso(idUsuario);
reglasCumplidas.put("Relación Deuda/Ingreso Aprobada", r4);
if (!r4) aprobado = false;

// Evaluar R5: Monto Máximo de Financiamiento
boolean r5 = evaluarMontoMaximoFinanciamiento(idUsuario);
reglasCumplidas.put("Monto Máximo de Financiamiento Aprobado", r5);
if (!r5) aprobado = false;

// Evaluar R6: Edad del Solicitante
boolean r6 = evaluarEdad(idUsuario);
reglasCumplidas.put("Edad del Solicitante Aprobada", r6);
if (!r6) aprobado = false;

// Evaluar R7: Capacidad de Ahorro
Map<String, Object> evaluacionAhorro = evaluarCapacidadAhorro(idUsuario);
String capacidadAhorro = (String) evaluacionAhorro.get("capacidadAhorro");
int reglasAhorroCumplidas = (int) evaluacionAhorro.get("reglasCumplidas");

// Si la capacidad de ahorro es "insuficiente", se rechaza la solicitud
if ("insuficiente".equals(capacidadAhorro)) {
    aprobado = false;
}

// Agregar los resultados al mapa de reglas cumplidas
reglasCumplidas.put("Capacidad de Ahorro Aprobada", !"insuficiente".equals(capacidadAhorro));

// Agregar toda la información al resultado
resultado.put("aprobado", aprobado);
resultado.put("reglasCumplidas", reglasCumplidas);
resultado.put("capacidadAhorro", capacidadAhorro);
resultado.put("detallesAhorro", evaluacionAhorro.get("detalles"));
return resultado;
}

// Calcular Costos Totales (P6)
public Map<String, Object> calcularCostoTotalPrestamo(Long idUsuario) throws Exception {
    Map<String, Object> resultado = new HashMap<>();

    // Obtener el préstamo asociado al usuario
    UsuarioPrestamoEntity usuarioPrestamo = usuarioPrestamoRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Préstamo no asociado al usuario"));
    PrestamoEntity prestamo = prestamoRepository.findById(usuarioPrestamo.getIdPrestamo())
        .orElseThrow(() -> new Exception("Préstamo no encontrado"));

    double montoPrestamo = prestamo.getMonto();
    int plazoAnios = prestamo.getPlazo();
    double tasaInteresAnual = prestamo.getTasaInteres();

    // Paso 1: Cálculo de la Cuota Mensual del Préstamo
    double tasaInteresMensual = tasaInteresAnual / 12 / 100; // Convertir a decimal
    int numeroPagos = plazoAnios * 12;

    // Fórmula de amortización
    double cuotaMensual = (montoPrestamo * tasaInteresMensual * Math.pow(1 + tasaInteresMensual, numeroPagos)) /
        (Math.pow(1 + tasaInteresMensual, numeroPagos) - 1);

    // Paso 2: Cálculo de los Seguros
    // Seguro de desgravamen: 0.03% del monto del préstamo por mes
    double seguroDesgravamenMensual = montoPrestamo * 0.0003; // 0.03% = 0.0003

    // Seguro de incendio: $20,000 mensuales
    double seguroIncendioMensual = 20000;

    // Paso 3: Cálculo de la Comisión por Administración
    // Comisión por administración: 1% del monto del préstamo
    double comisionAdministracion = montoPrestamo * 0.01; // 1% = 0.01

    // Paso 4: Cálculo del Costo Total del Préstamo
    // Costo mensual total: Cuota mensual + Seguros
    double costoMensualTotal = cuotaMensual + seguroDesgravamenMensual + seguroIncendioMensual;

    // Costo total durante la vida del préstamo
    double costoTotal = (costoMensualTotal * numeroPagos) + comisionAdministracion;

    // Redondear los valores a dos decimales
    cuotaMensual = Math.round(cuotaMensual * 100.0) / 100.0;
    seguroDesgravamenMensual = Math.round(seguroDesgravamenMensual * 100.0) / 100.0;
    costoMensualTotal = Math.round(costoMensualTotal * 100.0) / 100.0;
    costoTotal = Math.round(costoTotal * 100.0) / 100.0;
    comisionAdministracion = Math.round(comisionAdministracion * 100.0) / 100.0;

    // Paso 5: Preparar el resultado
    resultado.put("montoPrestamo", montoPrestamo);
    resultado.put("plazoAnios", plazoAnios);
    resultado.put("tasaInteresAnual", tasaInteresAnual);
    resultado.put("tasaInteresMensual", tasaInteresMensual * 100); // Convertir a porcentaje
    resultado.put("cuotaMensual", cuotaMensual);
    resultado.put("seguroDesgravamenMensual", seguroDesgravamenMensual);
    resultado.put("seguroIncendioMensual", seguroIncendioMensual);
    resultado.put("comisionAdministracion", comisionAdministracion);

```

```

        resultado.put("costoMensualTotal", costoMensualTotal);
        resultado.put("costoTotal", costoTotal);
        resultado.put("numeroPagos", numeroPagos);
        return resultado;
    }
}

```

SERVICIO COMPROBANTE-INGRESOS

```

package edu.mtisw.payrollbackend.services;

import edu.mtisw.payrollbackend.entities.ComprobanteIngresosEntity;
import edu.mtisw.payrollbackend.entities.UsuarioEntity;
import edu.mtisw.payrollbackend.repositories.ComprobanteIngresosRepository;
import edu.mtisw.payrollbackend.repositories.UsuarioRepository;
import jakarta.persistence.Id;
import org.hibernate.type.TrueFalseConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Service;

import java.util.ArrayList;

@Service
public class ComprobanteIngresosService {
    @Autowired
    ComprobanteIngresosRepository comprobanteIngresosRepository;

    //-----CRUD-----
    // Obtener todos los comprobantes de ingresos
    public ArrayList<ComprobanteIngresosEntity> getComprobanteIngresos(){
        return (ArrayList<ComprobanteIngresosEntity>) comprobanteIngresosRepository.findAll();
    }

    // Obtener un comprobante de ingresos por id
    public ComprobanteIngresosEntity getComprobanteIngresosById(Long id){
        return comprobanteIngresosRepository.findById(id).get();
    }

    // Guardar un comprobante de ingresos
    public ComprobanteIngresosEntity saveComprobanteIngresos(ComprobanteIngresosEntity comprobanteIngresos){
        return comprobanteIngresosRepository.save(comprobanteIngresos);
    }

    // Actualizar un comprobante de ingresos
    public ComprobanteIngresosEntity updateComprobanteIngresos(ComprobanteIngresosEntity comprobanteIngresos){
        return comprobanteIngresosRepository.save(comprobanteIngresos);
    }

    // Eliminar un comprobante de ingresos
    public boolean deleteComprobanteIngresos(Long id) throws Exception {
        try{
            comprobanteIngresosRepository.deleteById(id);
            return true;
        } catch (Exception e) {
            throw new Exception(e.getMessage());
        }
    }

    //-----PRINCIPALES-----

}

```

SERVICIO PRESTAMO

```

package edu.mtisw.payrollbackend.services;

import edu.mtisw.payrollbackend.entities.PrestamoEntity;
import edu.mtisw.payrollbackend.repositories.PrestamoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;

@Service
public class PrestamoService {
    @Autowired
    PrestamoRepository prestamoRepository;

    //-----CRUD-----
    // Obtener todos los prestamos
    public ArrayList<PrestamoEntity> getPrestamos(){
        return (ArrayList<PrestamoEntity>) prestamoRepository.findAll();
    }

    // Obtener un prestamo por id
    public PrestamoEntity getPrestamoById(Long id){
        return prestamoRepository.findById(id).get();
    }

    // Guardar un prestamo
    public PrestamoEntity savePrestamo(PrestamoEntity prestamo){
        return prestamoRepository.save(prestamo);
    }

    // Actualizar un prestamo
    public PrestamoEntity updatePrestamo(PrestamoEntity prestamo){
        return prestamoRepository.save(prestamo);
    }
}

```

```

    }

    // Eliminar un prestamo
    public boolean deletePrestamo(Long id) throws Exception {
        try{
            prestamoRepository.deleteByld(id);
            return true;
        } catch (Exception e) {
            throw new Exception(e.getMessage());
        }
    }
}

//-----PRINCIPALES-----
}

```

SERVICIO USUARIO-COMPROBANTE-INGRESOS

```

package edu.mtisw.payrollbackend.services;

import edu.mtisw.payrollbackend.entities.UsuarioComprobanteIngresosEntity;
import edu.mtisw.payrollbackend.entities.UsuarioEntity;
import edu.mtisw.payrollbackend.repositories.UsuarioComprobanteIngresosRepository;
import edu.mtisw.payrollbackend.repositories.UsuarioRepository;
import jakarta.persistence.Id;
import org.hibernate.type.TrueFalseConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Service;

import java.util.ArrayList;

@Service
public class UsuarioComprobanteIngresosService {
    @Autowired
    UsuarioComprobanteIngresosRepository usuarioComprobanteIngresosRepository;
    //-----CRUD-----
    // Obtener todos los usuarios comprobante de ingresos
    public ArrayList<UsuarioComprobanteIngresosEntity> getUsuariosComprobanteIngresos(){
        return (ArrayList<UsuarioComprobanteIngresosEntity>) usuarioComprobanteIngresosRepository.findAll();
    }

    // Obtener un usuario comprobante de ingresos por id
    public UsuarioComprobanteIngresosEntity getUsuarioComprobanteIngresosByld(Long id){
        return usuarioComprobanteIngresosRepository.findByld(id).get();
    }

    // Guardar un usuario comprobante de ingresos
    public UsuarioComprobanteIngresosEntity saveUsuarioComprobanteIngresos(UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos){
        return usuarioComprobanteIngresosRepository.save(usuarioComprobanteIngresos);
    }

    // Actualizar un usuario comprobante de ingresos
    public UsuarioComprobanteIngresosEntity updateUsuarioComprobanteIngresos(UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos){
        return usuarioComprobanteIngresosRepository.save(usuarioComprobanteIngresos);
    }

    // Eliminar un usuario comprobante de ingresos
    public boolean deleteUsuarioComprobanteIngresos(Long id) throws Exception {
        try{
            usuarioComprobanteIngresosRepository.deleteByld(id);
            return true;
        } catch (Exception e) {
            throw new Exception(e.getMessage());
        }
    }
}

//-----PRINCIPALES-----
}

```

SERVICIO USUARIO-PRESTAMO

```

package edu.mtisw.payrollbackend.services;

import edu.mtisw.payrollbackend.entities.UsuarioPrestamoEntity;
import edu.mtisw.payrollbackend.repositories.UsuarioPrestamoRepository;
import edu.mtisw.payrollbackend.repositories.UsuarioRepository;
import jakarta.persistence.Id;
import org.hibernate.type.TrueFalseConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Service;

import java.util.ArrayList;

@Service
public class UsuarioPrestamoService {
    @Autowired
    UsuarioPrestamoRepository usuarioPrestamoRepository;
    //-----CRUD-----
    // Obtener todos los usuarios-prestamos
    public ArrayList<UsuarioPrestamoEntity> getUsuariosPrestamos(){
        return (ArrayList<UsuarioPrestamoEntity>) usuarioPrestamoRepository.findAll();
    }
}

```

```

// Obtener un usuario-prestamo por id
public UsuarioPrestamoEntity getUsuarioPrestamoById(Long id){
    return usuarioPrestamoRepository.findById(id).get();
}

// Guardar un usuario-prestamo
public UsuarioPrestamoEntity saveUsuarioPrestamo(UsuarioPrestamoEntity usuarioPrestamo){
    return usuarioPrestamoRepository.save(usuarioPrestamo);
}

// Actualizar un usuario-prestamo
public UsuarioPrestamoEntity updateUsuarioPrestamo(UsuarioPrestamoEntity usuarioPrestamo){
    return usuarioPrestamoRepository.save(usuarioPrestamo);
}

// Eliminar un usuario-prestamo
public boolean deleteUsuarioPrestamo(Long id) throws Exception {
    try{
        usuarioPrestamoRepository.deleteById(id);
        return true;
    } catch (Exception e) {
        throw new Exception(e.getMessage());
    }
}

//-----PRINCIPALES-----
}

```

SERVICIO USUARIO

```

package edu.mt.isw.payrollbackend.services;

import edu.mt.isw.payrollbackend.entities.*;
import edu.mt.isw.payrollbackend.repositories.*;
import edu.mt.isw.payrollbackend.repositories.UsuarioRepository;
import jakarta.persistence.Id;
import org.hibernate.type.TrueFalseConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Service
public class UsuarioService {
    @Autowired
    UsuarioRepository usuarioRepository;
    @Autowired
    ComprobanteIngresosRepository comprobanteIngresosRepository;
    @Autowired
    PrestamoRepository prestamoRepository;
    @Autowired
    UsuarioPrestamoRepository usuarioPrestamoRepository;
    @Autowired
    UsuarioComprobanteIngresosRepository usuarioComprobanteIngresosRepository;
    //-----CRUD-----
    // Obtener todos los usuarios
    public ArrayList<UsuarioEntity> getUsuarios(){
        return (ArrayList<UsuarioEntity>) usuarioRepository.findAll();
    }
    // Obtener un usuario por id
    public UsuarioEntity getUsuarioById(Long id){
        return usuarioRepository.findById(id).get();
    }
    // Guardar un usuario
    public UsuarioEntity saveUsuario(UsuarioEntity usuario){
        return usuarioRepository.save(usuario);
    }
    // Actualizar un usuario
    public UsuarioEntity updateUsuario(UsuarioEntity usuario){
        return usuarioRepository.save(usuario);
    }
    // Eliminar un usuario
    public boolean deleteUsuario(Long id) throws Exception {
        try{
            usuarioRepository.deleteById(id);
            return true;
        } catch (Exception e) {
            throw new Exception(e.getMessage());
        }
    }
    //-----PRINCIPALES-----
    //simularCredito()(P1)
    public Map<String, Object> simularCredito(Long idUsuario, Long idPrestamo) throws Exception {
        // Buscar el usuario por ID
        UsuarioEntity usuario = usuarioRepository.findById(idUsuario)
            .orElseThrow(() -> new Exception("Usuario no encontrado"));

        // Buscar el prestamo por ID
        PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)

```

```

        .orElseThrow(() -> new Exception("Préstamo no encontrado"));

// Datos del préstamo para la simulación
double tasaInteresAnual = prestamo.getTasaInteres();
int plazoEnAnios = prestamo.getPlazo();
int monto = prestamo.getMonto();

// Calcular la tasa de interés mensual
double tasaInteresMensual = (tasaInteresAnual / 12) / 100;

// Número de pagos (meses)
int numeroPagos = plazoEnAnios * 12;

// Cálculo del pago mensual usando la fórmula de amortización
double pagoMensual = (monto * tasaInteresMensual * Math.pow(1 + tasaInteresMensual, numeroPagos)) /
    (Math.pow(1 + tasaInteresMensual, numeroPagos) - 1);

// Cálculo del interés total a pagar
double totalPagos = pagoMensual * numeroPagos;
double interesesTotales = totalPagos - monto;

// Crear un mapa con los resultados de la simulación
Map<String, Object> simulacionResultado = new HashMap<>();
simulacionResultado.put("monto", monto);
simulacionResultado.put("plazo", plazoEnAnios);
simulacionResultado.put("tasaInteres", tasaInteresAnual);
simulacionResultado.put("pagoMensual", pagoMensual);
simulacionResultado.put("interesesTotales", interesesTotales);
simulacionResultado.put("totalPagos", totalPagos);

return simulacionResultado;
}

// Registrar usuario(implementado en el CRUD)

// Solicitar Credito (P3) (por implementar)
// Consiste en aplicar las funciones saveComprobanteIngresos de la entidad
// y sus id en la tabla intermedia usuario_comprobante_ingresos
// ComprobanteIngresos savePrestamo de la entidad Prestamo
// y sus id en la tabla intermedia usuario_prestamo
public PrestamoEntity solicitarCredito(Long idUsuario, PrestamoEntity prestamo, ComprobanteIngresosEntity comprobanteIngresos) throws Exception {
    // Obtener el usuario por ID
    UsuarioEntity usuario = usuarioRepository.findById(idUsuario).orElseThrow(() -> new Exception("Usuario no encontrado"));

    // Guardar el comprobante de ingresos
    ComprobanteIngresosEntity comprobanteIngresosGuardado = comprobanteIngresosRepository.save(comprobanteIngresos);

    // Guardar el prestamo
    prestamo.setEstado("En proceso"); // Establecer el estado inicial como "En proceso"
    PrestamoEntity prestamoGuardado = prestamoRepository.save(prestamo);

    // Crear y guardar la relación en UsuarioPrestamo
    UsuarioPrestamoEntity usuarioPrestamo = new UsuarioPrestamoEntity();
    usuarioPrestamo.setIdUsuario(usuario.getId());
    usuarioPrestamo.setIdPrestamo(prestamoGuardado.getId());
    usuarioPrestamoRepository.save(usuarioPrestamo);

    // También podríamos relacionar el comprobante de ingresos si es necesario.
    UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = new UsuarioComprobanteIngresosEntity();
    usuarioComprobanteIngresos.setIdUsuario(usuario.getId());
    usuarioComprobanteIngresos.setIdComprobanteIngresos(comprobanteIngresosGuardado.getId());
    // Guardar en la tabla intermedia
    usuarioComprobanteIngresosRepository.save(usuarioComprobanteIngresos);

    // Retornar el prestamo guardado
    return prestamoGuardado;
}

// Obtener estado solicitud (P5) (por implementar)
// Método para obtener los estados de los préstamos de un usuario
public List<PrestamoEntity> obtenerEstadoSolicitudes(Long idUsuario) throws Exception {
    // Verificar si el usuario existe
    UsuarioEntity usuario = usuarioRepository.findById(idUsuario)
        .orElseThrow(() -> new Exception("Usuario no encontrado"));

    // Buscar todas las relaciones usuario-prestamo
    List<UsuarioPrestamoEntity> usuarioPrestamos = usuarioPrestamoRepository.findByIdUsuario(idUsuario);

    // Extraer todos los préstamos asociados
    List<PrestamoEntity> prestamos = new ArrayList<>();
    for (UsuarioPrestamoEntity usuarioPrestamo : usuarioPrestamos) {
        PrestamoEntity prestamo = prestamoRepository.findById(usuarioPrestamo.getIdPrestamo())
            .orElseThrow(() -> new Exception("Préstamo no encontrado"));
        prestamos.add(prestamo);
    }

    return prestamos;
}
}

```

CONTROLADOR BANCO

```
package edu.mtisiw.payrollbackend.controllers;
```

```

import edu.mt.isw.payrollbackend.services.BancoService;
import edu.mt.isw.payrollbackend.services.UsuarioService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.HashMap;
import java.util.Map;

@RestController
@RequestMapping("/api/v1/bancos")
@CrossOrigin("*")
public class BancoController {
    @Autowired
    UsuarioService usuarioService;
    @Autowired
    BancoService bancoService;

    // Evaluar Crédito (P4)
    @GetMapping("/evaluar-credito/{idUser}")
    public ResponseEntity<Map<String, Object>> evaluarCredito(
        @PathVariable Long idUsuario) {
        try {
            Map<String, Object> resultado = bancoService.evaluarCredito(idUsuario);
            return ResponseEntity.ok(resultado);
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> errorResponse = new HashMap<>();
            errorResponse.put("error", e.getMessage());
            return ResponseEntity.badRequest().body(errorResponse);
        }
    }

    // Evaluar Relación Cuota/Ingreso (R1)
    @GetMapping("/evaluar-relacion-cuota-ingreso/{idUser}")
    public ResponseEntity<Map<String, Object>> evaluarRelacionCuotaIngreso(
        @PathVariable Long idUsuario) {
        try {
            boolean resultado = bancoService.evaluarRelacionCuotaIngreso(idUsuario);
            Map<String, Object> response = new HashMap<>();
            response.put("resultado", resultado);
            return ResponseEntity.ok(response);
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> errorResponse = new HashMap<>();
            errorResponse.put("error", e.getMessage());
            return ResponseEntity.badRequest().body(errorResponse);
        }
    }

    // Evaluar Historial Crediticio del Cliente (R2)
    @GetMapping("/evaluar-historial-credicio/{idUser}")
    public ResponseEntity<Map<String, Object>> evaluarHistorialCrediticio(
        @PathVariable Long idUsuario) {
        try {
            boolean resultado = bancoService.evaluarHistorialCrediticio(idUsuario);
            Map<String, Object> response = new HashMap<>();
            response.put("resultado", resultado);
            return ResponseEntity.ok(response);
        } catch (Exception e) {
            e.printStackTrace();
            Map<String, Object> errorResponse = new HashMap<>();
            errorResponse.put("error", e.getMessage());
            return ResponseEntity.badRequest().body(errorResponse);
        }
    }

    // Evaluar Antigüedad Laboral y Estabilidad (R3)
    @GetMapping("/evaluar-antiguedad/{idUser}")
    public ResponseEntity<Map<String, Object>> evaluarAntiguedad(
        @PathVariable Long idUsuario) {
        try {
            boolean resultado = bancoService.evaluarAntiguedad(idUsuario);
            Map<String, Object> response = new HashMap<>();
            response.put("resultado", resultado);
            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> errorResponse = new HashMap<>();
            errorResponse.put("error", e.getMessage());
            return ResponseEntity.badRequest().body(errorResponse);
        }
    }

    // Evaluar Relación Deuda/Ingreso (R4)
    @GetMapping("/evaluar-relacion-deuda-ingreso/{idUser}")
    public ResponseEntity<Map<String, Object>> evaluarRelacionDeudaIngreso(
        @PathVariable Long idUsuario) {
        try {
            boolean resultado = bancoService.evaluarRelacionDeudaIngreso(idUsuario);
            Map<String, Object> response = new HashMap<>();
            response.put("resultado", resultado);
            return ResponseEntity.ok(response);
        } catch (Exception e) {
            Map<String, Object> errorResponse = new HashMap<>();
            errorResponse.put("error", e.getMessage());
            return ResponseEntity.badRequest().body(errorResponse);
        }
    }
}

```

```

// Evaluar Monto Máximo de Financiamiento (R5)
@GetMapping("/evaluar-monto-maximo/{idUser}")
public ResponseEntity<Map<String, Object>> evaluarMontoMaximoFinanciamiento(
    @PathVariable Long idUsuario){
    try {
        boolean resultado = bancoService.evaluarMontoMaximoFinanciamiento(idUsuario);
        Map<String, Object> response = new HashMap<>();
        response.put("resultado", resultado);
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

// Evaluar Edad del Solicitante (R6)
@GetMapping("/evaluar-edad/{idUser}")
public ResponseEntity<Map<String, Object>> evaluarEdad(
    @PathVariable Long idUsuario){
    try {
        boolean resultado = bancoService.evaluarEdad(idUsuario);
        Map<String, Object> response = new HashMap<>();
        response.put("resultado", resultado);
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

// Evaluar Saldo Mínimo Requerido (R71)
@GetMapping("/evaluar-saldo-minimo/{idUser}")
public ResponseEntity<Map<String, Object>> evaluarSaldoMinimo(
    @PathVariable Long idUsuario){
    try {
        boolean resultado = bancoService.evaluarSaldoMinimo(idUsuario);
        Map<String, Object> response = new HashMap<>();
        response.put("resultado", resultado);
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

// Evaluar Historial de Ahorro Consistente (R72)
@GetMapping("/evaluar-historial-ahorro/{idUser}")
public ResponseEntity<Map<String, Object>> evaluarHistorialAhorroConsistente(
    @PathVariable Long idUsuario){
    try {
        boolean resultado = bancoService.evaluarHistorialAhorroConsistente(idUsuario);
        Map<String, Object> response = new HashMap<>();
        response.put("resultado", resultado);
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

// Evaluar Depósitos Periódicos (R73)
@GetMapping("/evaluar-depositos-periodicos/{idUser}")
public ResponseEntity<Map<String, Object>> evaluarDepositosPeriodicos(
    @PathVariable Long idUsuario){
    try {
        boolean resultado = bancoService.evaluarDepositosPeriodicos(idUsuario);
        Map<String, Object> response = new HashMap<>();
        response.put("resultado", resultado);
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        e.printStackTrace();
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

// Evaluar Relación Saldo/Antigüedad (R74)
@GetMapping("/evaluar-relacion-saldo-antiguedad/{idUser}")
public ResponseEntity<Map<String, Object>> evaluarRelacionSaldoAntiguedad(
    @PathVariable Long idUsuario){
    try {
        boolean resultado = bancoService.evaluarRelacionSaldoAntiguedad(idUsuario);
        Map<String, Object> response = new HashMap<>();
        response.put("resultado", resultado);
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        e.printStackTrace();
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

```

```

// Evaluar Retiros Recientes (R75)
@GetMapping("/evaluar-retiros-recientes/{idUsuario}")
public ResponseEntity<Map<String, Object>> evaluarRetirosRecientes(
    @PathVariable Long idUsuario){
    try {
        boolean resultado = bancoService.evaluarRetirosRecientes(idUsuario);
        Map<String, Object> response = new HashMap<>();
        response.put("resultado", resultado);
        return ResponseEntity.ok(response);
    } catch (Exception e) {
        e.printStackTrace();
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

// Evaluar Capacidad de Ahorro (R7)
@GetMapping("/evaluar-capacidad-ahorro/{idUsuario}")
public ResponseEntity<Map<String, Object>> evaluarCapacidadAhorro(
    @PathVariable Long idUsuario){
    try {
        Map<String, Object> resultado = bancoService.evaluarCapacidadAhorro(idUsuario);
        return ResponseEntity.ok(resultado);
    } catch (Exception e) {
        e.printStackTrace();
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}

// Calcular Costos Totales (P6)
@GetMapping("/calcular-costo-total/{idUsuario}")
public ResponseEntity<Map<String, Object>> calcularCostoTotalPrestamo(
    @PathVariable Long idUsuario){
    try {
        Map<String, Object> resultado = bancoService.calcularCostoTotalPrestamo(idUsuario);
        return ResponseEntity.ok(resultado);
    } catch (Exception e) {
        e.printStackTrace();
        Map<String, Object> errorResponse = new HashMap<>();
        errorResponse.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorResponse);
    }
}
}

```

CONTROLADOR COMPROBANTE-INGRESOS

```
package edu.mtisiw.payrollbackend.controllers;
```

```

import edu.mtisiw.payrollbackend.entities.ComprobanteIngresosEntity;
import edu.mtisiw.payrollbackend.services.ComprobanteIngresosService;
import edu.mtisiw.payrollbackend.services.UsuarioService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

```

```
import java.util.List;
```

```

@RestController
@RequestMapping("/api/v1/comprobante-ingresos")
@CrossOrigin("**")
public class ComprobanteIngresosController {
    @Autowired
    ComprobanteIngresosService comprobanteIngresosService;

    //-----CRUD-----
    // Obtener comprobantes de ingresos
    @GetMapping("/")
    public ResponseEntity<List<ComprobanteIngresosEntity>> listComprobanteIngresos(){
        List<ComprobanteIngresosEntity> comprobanteIngresos = comprobanteIngresosService.getComprobanteIngresos();
        return ResponseEntity.ok(comprobanteIngresos);
    }

    // Obtener comprobante de ingreso
    @GetMapping("/{id}")
    public ResponseEntity<ComprobanteIngresosEntity> getComprobanteIngresosById(@PathVariable Long id){
        ComprobanteIngresosEntity comprobanteIngresos = comprobanteIngresosService.getComprobanteIngresosById(id);
        return ResponseEntity.ok(comprobanteIngresos);
    }

    // Guardar comprobante de ingreso
    @PostMapping("/")
    public ResponseEntity<ComprobanteIngresosEntity> saveComprobanteIngresos(@RequestBody ComprobanteIngresosEntity comprobanteIngresos){
        ComprobanteIngresosEntity comprobanteIngresosNuevo = comprobanteIngresosService.saveComprobanteIngresos(comprobanteIngresos);
        return ResponseEntity.ok(comprobanteIngresosNuevo);
    }

    // Actualizar comprobante de ingreso
    @PutMapping("/")
    public ResponseEntity<ComprobanteIngresosEntity> updateComprobanteIngresos(@RequestBody ComprobanteIngresosEntity comprobanteIngresos){
        ComprobanteIngresosEntity comprobanteIngresosActualizado = comprobanteIngresosService.updateComprobanteIngresos(comprobanteIngresos);
    }
}

```



```

        return ResponseEntity.ok(comprobanteIngresosActualizado);
    }

    // Eliminar comprobante de ingreso
    @DeleteMapping("/{id}")
    public ResponseEntity<Boolean> deleteComprobanteIngresosById(@PathVariable Long id) throws Exception {
        var isDeleted = comprobanteIngresosService.deleteComprobanteIngresos(id);
        return ResponseEntity.noContent().build();
    }

    //-----PRINCIPALES-----
}

```

CONTROLADOR PRESTAMO

```

package edu.mtisiw.payrollbackend.controllers;

import edu.mtisiw.payrollbackend.entities.PrestamoEntity;
import edu.mtisiw.payrollbackend.entities.UsuarioEntity;
import edu.mtisiw.payrollbackend.services.PrestamoService;
import edu.mtisiw.payrollbackend.services.UsuarioService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/v1/prestamos")
@CrossOrigin("*")
public class PrestamoController {
    @Autowired
    PrestamoService prestamoService;

    //-----CRUD-----

    // Obtener prestamos
    @GetMapping("/")
    public ResponseEntity<List<PrestamoEntity>> listPrestamos(){
        List<PrestamoEntity> prestamos = prestamoService.getPrestamos();
        return ResponseEntity.ok(prestamos);
    }

    // Obtener prestamo
    @GetMapping("/{id}")
    public ResponseEntity<PrestamoEntity> getPrestamoById(@PathVariable Long id){
        PrestamoEntity prestamo = prestamoService.getPrestamoById(id);
        return ResponseEntity.ok(prestamo);
    }

    // Guardar prestamo
    @PostMapping("/")
    public ResponseEntity<PrestamoEntity> savePrestamo(@RequestBody PrestamoEntity prestamo){
        PrestamoEntity prestamoNuevo = prestamoService.savePrestamo(prestamo);
        return ResponseEntity.ok(prestamoNuevo);
    }

    // Actualizar prestamo
    @PutMapping("/")
    public ResponseEntity<PrestamoEntity> updatePrestamo(@RequestBody PrestamoEntity prestamo){
        PrestamoEntity prestamoActualizado = prestamoService.updatePrestamo(prestamo);
        return ResponseEntity.ok(prestamoActualizado);
    }

    // Eliminar prestamo
    @DeleteMapping("/{id}")
    public ResponseEntity<Boolean> deletePrestamoById(@PathVariable Long id) throws Exception {
        var isDeleted = prestamoService.deletePrestamo(id);
        return ResponseEntity.noContent().build();
    }

    //-----PRINCIPALES-----
}

```

CONTROLADOR USUARIO-COMPROBANTE-INGRESOS

```

package edu.mtisiw.payrollbackend.controllers;

import edu.mtisiw.payrollbackend.entities.UsuarioComprobanteIngresosEntity;
import edu.mtisiw.payrollbackend.entities.UsuarioEntity;
import edu.mtisiw.payrollbackend.services.UsuarioComprobanteIngresosService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/v1/usuarios-comprobantes-ingresos")

```

```

@CrossOrigin("**")
public class UsuarioComprobanteIngresosController {
    @Autowired
    UsuarioComprobanteIngresosService usuarioComprobanteIngresosService;
    //-----CRUD-----
    // Obtener usuarios comprobantes ingresos
    @GetMapping("/")
    public ResponseEntity<List<UsuarioComprobanteIngresosEntity>> listUsuariosComprobantesIngresos(){
        List<UsuarioComprobanteIngresosEntity> usuariosComprobantesIngresos = usuarioComprobanteIngresosService.getUsuariosComprobanteIngresos();
        return ResponseEntity.ok(usuariosComprobantesIngresos);
    }

    // Obtener un usuario comprobante de ingresos por id
    @GetMapping("/{id}")
    public ResponseEntity<UsuarioComprobanteIngresosEntity> getUsuarioComprobanteIngresosById(@PathVariable Long id){
        UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosService.getUsuarioComprobanteIngresosById(id);
        return ResponseEntity.ok(usuarioComprobanteIngresos);
    }

    // Guardar un usuario comprobante de ingresos
    @PostMapping("/")
    public ResponseEntity<UsuarioComprobanteIngresosEntity> saveUsuarioComprobanteIngresos(@RequestBody UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos){
        UsuarioComprobanteIngresosEntity usuarioComprobanteIngresosSaved = usuarioComprobanteIngresosService.saveUsuarioComprobanteIngresos(usuarioComprobanteIngresos);
        return ResponseEntity.ok(usuarioComprobanteIngresosSaved);
    }

    // Actualizar un usuario comprobante de ingresos
    @PutMapping("/")
    public ResponseEntity<UsuarioComprobanteIngresosEntity> updateUsuarioComprobanteIngresos(@RequestBody UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos){
        UsuarioComprobanteIngresosEntity usuarioComprobanteIngresosUpdated = usuarioComprobanteIngresosService.updateUsuarioComprobanteIngresos(usuarioComprobanteIngresos);
        return ResponseEntity.ok(usuarioComprobanteIngresosUpdated);
    }

    // Eliminar un usuario comprobante de ingresos
    @DeleteMapping("/{id}")
    public ResponseEntity<Boolean> deleteUsuarioComprobanteIngresos(@PathVariable Long id){
        try{
            boolean isDeleted = usuarioComprobanteIngresosService.deleteUsuarioComprobanteIngresos(id);
            return ResponseEntity.ok(isDeleted);
        } catch (Exception e) {
            return ResponseEntity.badRequest().build();
        }
    }
}

//-----PRINCIPALES-----
}

```

CONTROLADOR USUARIOS

```
package edu.mtisiw.payrollbackend.controllers;
```

```

import edu.mtisiw.payrollbackend.entities.ComprobanteIngresosEntity;
import edu.mtisiw.payrollbackend.entities.PrestamoEntity;
import edu.mtisiw.payrollbackend.entities.UsuarioEntity;
import edu.mtisiw.payrollbackend.services.UsuarioService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

```

```

@RestController
@RequestMapping("/api/v1/usuarios")
@CrossOrigin("**")
public class UsuarioController {
    @Autowired
    UsuarioService usuarioService;

    //-----CRUD-----
    // Obtener usuarios
    @GetMapping("/")
    public ResponseEntity<List<UsuarioEntity>> listUsuarios(){
        List<UsuarioEntity> usuarios = usuarioService.getUsuarios();
        return ResponseEntity.ok(usuarios);
    }

    // Obtener usuario
    @GetMapping("/{id}")
    public ResponseEntity<UsuarioEntity> getUsuarioById(@PathVariable Long id){
        UsuarioEntity usuario = usuarioService.getUsuarioById(id);
        return ResponseEntity.ok(usuario);
    }

    // Guardar usuario
    @PostMapping("/")
    public ResponseEntity<UsuarioEntity> saveUsuario(@RequestBody UsuarioEntity usuario){
        UsuarioEntity usuarioNuevo = usuarioService.saveUsuario(usuario);
        return ResponseEntity.ok(usuarioNuevo);
    }

    // Actualizar usuario
    @PutMapping("/")
    public ResponseEntity<UsuarioEntity> updateUsuario(@RequestBody UsuarioEntity usuario){
        UsuarioEntity usuarioActualizado = usuarioService.updateUsuario(usuario);
        return ResponseEntity.ok(usuarioActualizado);
    }
}

```

```

// Eliminar usuario
@DeleteMapping("/{id}")
public ResponseEntity<Boolean> deleteUsuarioById(@PathVariable Long id) throws Exception {
    var isDeleted = usuarioService.deleteUsuario(id);
    return ResponseEntity.noContent().build();
}
//-----PRINCIPALES-----

// Simular Crédito (P1)
@GetMapping("/{id}/simular-credito")
public ResponseEntity<Map<String, Object>> simularCredito(
    @PathVariable Long id) {
    try {
        Map<String, Object> resultadoSimulacion = usuarioService.simularCredito(id);
        return ResponseEntity.ok(resultadoSimulacion);
    } catch (Exception e) {
        return ResponseEntity.badRequest().body(null);
    }
}

// Solicitar Crédito (P3)
@PostMapping("/{id}/solicitar-credito")
public ResponseEntity<PrestamoEntity> solicitarCredito(
    @PathVariable Long id,
    @RequestBody Map<String, Object> requestBody) {
    try {
        // Extracción de datos para PrestamoEntity
        String tipo = (String) requestBody.get("tipo");
        int plazo = ((Number) requestBody.get("plazo")).intValue();
        double tasaInteres = ((Number) requestBody.get("tasaInteres")).doubleValue();
        int monto = ((Number) requestBody.get("monto")).intValue();
        int valorPropiedad = ((Number) requestBody.get("valorPropiedad")).intValue();

        // Crear PrestamoEntity
        PrestamoEntity prestamo = new PrestamoEntity();
        prestamo.setTipo(tipo);
        prestamo.setPlazo(plazo);
        prestamo.setTasaInteres(tasaInteres);
        prestamo.setMonto(monto);
        prestamo.setEstado("En proceso");
        prestamo.setValorPropiedad(valorPropiedad);

        // Extracción de datos para ComprobanteIngresosEntity
        int antiguedadLaboral = ((Number) requestBody.get("antiguedadLaboral")).intValue();
        int ingresoMensual = ((Number) requestBody.get("ingresoMensual")).intValue();
        int saldo = ((Number) requestBody.get("saldo")).intValue();
        int deudas = ((Number) requestBody.get("deudas")).intValue();
        int cantidadDeudasPendientes = ((Number) requestBody.get("cantidadDeudasPendientes")).intValue();
        int antiguedadCuenta = ((Number) requestBody.get("antiguedadCuenta")).intValue();

        // Extraer listas y convertir a cadenas
        List<Number> ingresosNumbers = (List<Number>) requestBody.get("ingresosUltimos24Meses");
        String ingresosUltimos24Meses = ingresosNumbers.stream()
            .map(String::valueOf)
            .collect(Collectors.joining(""));

        List<Number> saldosMensualesNumbers = (List<Number>) requestBody.get("saldosMensuales");
        String saldosMensuales = saldosMensualesNumbers.stream()
            .map(String::valueOf)
            .collect(Collectors.joining(""));

        List<Number> depositosNumbers = (List<Number>) requestBody.get("depositosUltimos12Meses");
        String depositosUltimos12Meses = depositosNumbers.stream()
            .map(String::valueOf)
            .collect(Collectors.joining(""));

        List<Number> retirosNumbers = (List<Number>) requestBody.get("retirosUltimos6Meses");
        String retirosUltimos6Meses = retirosNumbers.stream()
            .map(String::valueOf)
            .collect(Collectors.joining(""));

        // Crear ComprobanteIngresosEntity
        ComprobanteIngresosEntity comprobanteIngresos = new ComprobanteIngresosEntity();
        comprobanteIngresos.setAntiguedadLaboral(antiguedadLaboral);
        comprobanteIngresos.setIngresoMensual(ingresoMensual);
        comprobanteIngresos.setIngresosUltimos24Meses(ingresosUltimos24Meses);
        comprobanteIngresos.setSaldo(saldo);
        comprobanteIngresos.setDeudas(deudas);
        comprobanteIngresos.setCantidadDeudasPendientes(cantidadDeudasPendientes);
        comprobanteIngresos.setSaldosMensuales(saldosMensuales);
        comprobanteIngresos.setDepositosUltimos12Meses(depositosUltimos12Meses);
        comprobanteIngresos.setAntiguedadCuenta(antiguedadCuenta);
        comprobanteIngresos.setRetirosUltimos6Meses(retirosUltimos6Meses);

        // Llamar al servicio para procesar la solicitud de crédito
        PrestamoEntity prestamoSolicitado = usuarioService.solicitarCredito(id, prestamo, comprobanteIngresos);
        return ResponseEntity.ok(prestamoSolicitado);
    } catch (Exception e) {
        e.printStackTrace();
        return ResponseEntity.badRequest().body(null);
    }
}

// Obtener estado de la solicitud (P5)
@GetMapping("/{id}/estado-solicitud")
public ResponseEntity<PrestamoEntity> obtenerEstadoSolicitud(@PathVariable Long id) {
    try {
        PrestamoEntity prestamo = usuarioService.obtenerEstadoSolicitud(id);
    }
}

```

```

        return ResponseEntity.ok(prestamo);
    } catch (Exception e) {
        return ResponseEntity.badRequest().body(null);
    }
}
}

```

CONTROLADOR USUARIO-PRESTAMOS

```

package edu.mtisiw.payrollbackend.controllers;

import edu.mtisiw.payrollbackend.entities.UsuarioPrestamoEntity;
import edu.mtisiw.payrollbackend.services.UsuarioPrestamoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/v1/usuarios-prestamos")
@CrossOrigin("*")
public class UsuarioPrestamoController {
    @Autowired
    UsuarioPrestamoService usuarioPrestamoService;
    //-----CRUD-----
    // Obtener usuarios prestamos
    @GetMapping("/")
    public ResponseEntity<List<UsuarioPrestamoEntity>> listUsuarios(){
        List<UsuarioPrestamoEntity> usuarios = usuarioPrestamoService.getUsuariosPrestamos();
        return ResponseEntity.ok(usuarios);
    }

    // Obtener usuario prestamo
    @GetMapping("/{id}")
    public ResponseEntity<UsuarioPrestamoEntity> getUsuarioById(@PathVariable Long id){
        UsuarioPrestamoEntity usuario = usuarioPrestamoService.getUsuarioPrestamoById(id);
        return ResponseEntity.ok(usuario);
    }

    // Guardar usuario prestamo
    @PostMapping("/")
    public ResponseEntity<UsuarioPrestamoEntity> saveUsuario(@RequestBody UsuarioPrestamoEntity usuario){
        UsuarioPrestamoEntity usuarioNuevo = usuarioPrestamoService.saveUsuarioPrestamo(usuario);
        return ResponseEntity.ok(usuarioNuevo);
    }

    // Actualizar usuario prestamo
    @PutMapping("/")
    public ResponseEntity<UsuarioPrestamoEntity> updateUsuario(@RequestBody UsuarioPrestamoEntity usuario){
        UsuarioPrestamoEntity usuarioActualizado = usuarioPrestamoService.updateUsuarioPrestamo(usuario);
        return ResponseEntity.ok(usuarioActualizado);
    }

    // Eliminar usuario prestamo
    @DeleteMapping("/{id}")
    public ResponseEntity<Boolean> deleteUsuarioById(@PathVariable Long id) throws Exception {
        var isDeleted = usuarioPrestamoService.deleteUsuarioPrestamo(id);
        return ResponseEntity.noContent().build();
    }

    //-----PRINCIPALES-----
}

```