Contenido

[ENTIDAD BANCO **¡Error! Marcador no definido.**](#_Toc180589427)

[ENTIDAD COMPROBANTE-INGRESOS 2](#_Toc180589428)

[ENTIDAD PRESTAMO 2](#_Toc180589429)

[ENTIDAD USUARIO-COMPROBANTE-INGRESOS 3](#_Toc180589430)

[ENTIDAD USUARIO 3](#_Toc180589431)

[ENTIDAD USUARIO-PRESTAMO 3](#_Toc180589432)

[REPOSITORIO BANCO **¡Error! Marcador no definido.**](#_Toc180589433)

[REPOSITORIO PRESTAMO 4](#_Toc180589434)

[REPOSITORIO COMPROBANTE-INGRESOS 4](#_Toc180589435)

[REPOSITORIO USUARIO-PRESTAMO 4](#_Toc180589436)

[REPOSITORIO USUARIO 4](#_Toc180589437)

[SERVICIO BANCO 5](#_Toc180589438)

[SERVICIO PRESTAMO 12](#_Toc180589439)

[SERVICIO COMPROBANTE-INGRESOS 13](#_Toc180589440)

[SERVICIO USUARIO-PRESTAMO 13](#_Toc180589441)

[SERVICIO USUARIO 13](#_Toc180589442)

[CONTROLADOR BANCO 14](#_Toc180589443)

[CONTROLADOR COMPROBANTE-INGRESOS 15](#_Toc180589444)

[CONTROLADOR PRESTAMO 18](#_Toc180589445)

[CONTROLADOR USUARIO-COMPROBANTE-INGRESOS 18](#_Toc180589446)

[CONTROLADOR USUARIOS 19](#_Toc180589447)

[CONTROLADOR USUARIO-PRESTAMOS 21](#_Toc180589448)

### ENTIDAD COMPROBANTE-INGRESOS

package edu.mtisw.payrollbackend.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
import java.util.ArrayList;  
import java.util.List;  
  
@Entity  
@Table(name = "comprobanteIngresos")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
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 lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "prestamo")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
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 jakarta.persistence.\*;  
  
@Entity  
@Table(name = "usuarioComprobanteIngresos")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
  
public class UsuarioComprobanteIngresosEntity {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(unique = true, nullable = false, name = "id")  
 private Long id;  
  
 @Column(name = "idUsuario")  
 private Long idUsuario;  
  
 @Column(name = "idComprobanteIngresos")  
 private Long idComprobanteIngresos;  
}

### ENTIDAD USUARIO

package edu.mtisw.payrollbackend.entities;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
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 jakarta.persistence.\*;  
  
@Entity  
@Table(name = "usuarioPrestamo")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
  
public class UsuarioPrestamoEntity {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(unique = true, nullable = false, name = "id")  
 private Long id;  
  
 @Column(name = "idUsuario")  
 private Long idUsuario;  
  
 @Column(name = "idPrestamo")  
 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.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;  
  
@Repository  
public interface UsuarioComprobanteIngresosRepository extends JpaRepository<UsuarioComprobanteIngresosEntity, Long > {  
  
}

### 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;  
  
@Repository  
public interface UsuarioPrestamoRepository extends JpaRepository<UsuarioPrestamoEntity, Long > {  
}

### 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>{  
  
}

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

### SERVICIO BANCO

package edu.mtisw.payrollbackend.services;  
  
import edu.mtisw.payrollbackend.entities.\*;  
import edu.mtisw.payrollbackend.repositories.\*;  
import edu.mtisw.payrollbackend.services.\*;  
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 BancoService {  
 @Autowired  
 UsuarioRepository usuarioRepository;  
 @Autowired  
 ComprobanteIngresosRepository comprobanteIngresosRepository;  
 @Autowired  
 PrestamoRepository prestamoRepository;  
 @Autowired  
 UsuarioPrestamoRepository usuarioPrestamoRepository;  
 @Autowired  
 UsuarioComprobanteIngresosRepository usuarioComprobanteIngresosRepository;  
 @Autowired  
 UsuarioService usuarioService;  
 //------------------------------------CRUD----------------------------------------------  
 //------------------------------------PRINCIPALES---------------------------------------  
 //evaluarRelacionCuotaIngreso()(R1)  
 public boolean evaluarRelacionCuotaIngreso(Long idUsuario, Long idPrestamo) throws Exception {  
 System.*out*.println("Evaluando relación cuota ingreso para usuario: " + idUsuario + " y préstamo: " + idPrestamo);  
  
 UsuarioEntity usuario = usuarioRepository.findById(idUsuario)  
 .orElseThrow(() -> new Exception("Usuario no encontrado"));  
 System.*out*.println("Usuario encontrado: " + usuario);  
  
 UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findByIdUsuario(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());  
  
 PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)  
 .orElseThrow(() -> new Exception("Préstamo no encontrado"));  
 System.*out*.println("Préstamo encontrado: " + prestamo);  
  
 // Calculo 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;  
 }  
  
  
 //evaluarDeudas()(R2)  
 public boolean evaluarHistorialCrediticio(Long idUsuario) throws Exception {  
 // Obtener el comprobante de ingresos asociado al usuario  
 UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findByIdUsuario(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(); // Usamos el atributo 'deudas' existente  
  
 // 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;  
 }  
  
 //evaluarAntiguedad()(R3)  
 // Evaluar Antigüedad Laboral y Estabilidad (R3)  
 public boolean evaluarAntiguedad(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.findByIdUsuario(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 antiguedadLaboral = comprobanteIngresos.getAntiguedadLaboral();  
 if (antiguedadLaboral >= 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();  
 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");  
 }  
 }  
  
 //evaluarRelacionDeudaIngreso()(R4)  
 public boolean evaluarRelacionDeudaIngreso(Long idUsuario, Long idPrestamo) 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.findByIdUsuario(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;  
 }  
 }  
  
 //evaluarMontoMaximoFinanciamiento()(R5)  
 public boolean evaluarMontoMaximoFinanciamiento(Long idPrestamo) throws Exception {  
 // 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;  
 }  
 }  
  
 //evaluarEdad()(R6)  
 // Evaluar Edad del Solicitante (R6)  
 public boolean evaluarEdad(Long idUsuario, Long idPrestamo) throws Exception {  
 // 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;  
 }  
 }  
  
 //evaluarSaldoMinimo()(R71)  
 public boolean evaluarSaldoMinimo(Long idUsuario, Long idPrestamo) throws Exception {  
 // Obtener el comprobante de ingresos asociado al usuario  
 UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findByIdUsuario(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.findByIdUsuario(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;  
 }  
 }  
  
 // Verificar si hubo retiros significativos (> 50% del saldo)  
 for (int i = 1; i < saldosMensuales.size(); i++) {  
 double saldoAnterior = saldosMensuales.get(i - 1);  
 double saldoActual = saldosMensuales.get(i);  
  
 double disminucion = saldoAnterior - saldoActual;  
  
 if (disminucion > (saldoAnterior \* 0.50)) {  
 // Se detectó un retiro significativo  
 return false;  
 }  
 }  
  
 // Cumple con el historial de ahorro consistente  
 return true;  
 }  
  
 //evaluarDepositosPeriodicos(R73)  
 public boolean evaluarDepositosPeriodicos(Long idUsuario) throws Exception {  
 // Obtener el comprobante de ingresos asociado al usuario  
 UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findByIdUsuario(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  
 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();  
 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;  
 }  
  
 //evaluarRelacionSaldoAntiguedad(R74)  
 public boolean evaluarRelacionSaldoAntiguedad(Long idUsuario, Long idPrestamo) throws Exception {  
 // Obtener el comprobante de ingresos asociado al usuario  
 UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findByIdUsuario(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 antiguedadCuenta = comprobanteIngresos.getAntiguedadCuenta();  
 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 = (antiguedadCuenta < 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;  
 }  
 }  
  
 //evaluarRetiroReciente(R75)  
 public boolean evaluarRetirosRecientes(Long idUsuario) throws Exception {  
 // Obtener el comprobante de ingresos asociado al usuario  
 UsuarioComprobanteIngresosEntity usuarioComprobanteIngresos = usuarioComprobanteIngresosRepository.findByIdUsuario(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();  
 saldosMensuales.add(Double.*parseDouble*(saldoStr));  
 }  
  
 for (int i = retirosArray.length - 6; i < retirosArray.length; i++) {  
 String retiroStr = retirosArray[i].trim();  
 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;  
 }  
  
 //evaluarCapacidadAhorro()(R7)  
 public Map<String, Object> evaluarCapacidadAhorro(Long idUsuario, Long idPrestamo) throws Exception {  
 Map<String, Object> resultado = new HashMap<>();  
  
 int reglasCumplidas = 0;  
  
 // Evaluar R71: Saldo Mínimo Requerido  
 boolean r71 = evaluarSaldoMinimo(idUsuario, idPrestamo);  
 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 = evaluarRelacionSaldoAntiguedad(idUsuario, idPrestamo);  
 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;  
 }  
  
 //evaluarCredito()(P4)  
 public Map<String, Object> evaluarCredito(Long idUsuario, Long idPrestamo) throws Exception {  
 Map<String, Object> resultado = new HashMap<>();  
 Map<String, Boolean> reglasCumplidas = new HashMap<>();  
  
 boolean aprobado = true;  
  
 // Evaluar R1: Relación Cuota/Ingreso  
 boolean r1 = evaluarRelacionCuotaIngreso(idUsuario, idPrestamo);  
 reglasCumplidas.put("R1", r1);  
 if (!r1) aprobado = false;  
  
 // Evaluar R2: Historial Crediticio del Cliente  
 boolean r2 = evaluarHistorialCrediticio(idUsuario);  
 reglasCumplidas.put("R2", r2);  
 if (!r2) aprobado = false;  
  
 // Evaluar R3: Antigüedad Laboral y Estabilidad  
 boolean r3 = evaluarAntiguedad(idUsuario);  
 reglasCumplidas.put("R3", r3);  
 if (!r3) aprobado = false;  
  
 // Evaluar R4: Relación Deuda/Ingreso  
 boolean r4 = evaluarRelacionDeudaIngreso(idUsuario, idPrestamo);  
 reglasCumplidas.put("R4", r4);  
 if (!r4) aprobado = false;  
  
 // Evaluar R5: Monto Máximo de Financiamiento  
 boolean r5 = evaluarMontoMaximoFinanciamiento(idPrestamo);  
 reglasCumplidas.put("R5", r5);  
 if (!r5) aprobado = false;  
  
 // Evaluar R6: Edad del Solicitante  
 boolean r6 = evaluarEdad(idUsuario, idPrestamo);  
 reglasCumplidas.put("R6", r6);  
 if (!r6) aprobado = false;  
  
 // Evaluar R7: Capacidad de Ahorro  
 Map<String, Object> evaluacionAhorro = evaluarCapacidadAhorro(idUsuario, idPrestamo);  
 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("R7\_CapacidadAhorro", !"insuficiente".equals(capacidadAhorro));  
  
 resultado.put("aprobado", aprobado);  
 resultado.put("reglasCumplidas", reglasCumplidas);  
 resultado.put("capacidadAhorro", capacidadAhorro);  
 resultado.put("detallesAhorro", evaluacionAhorro.get("detalles"));  
  
 return resultado;  
 }  
  
 //calcularCostoTotales()(P6)  
 public Map<String, Object> calcularCostoTotalPrestamo(Long idPrestamo) throws Exception {  
 Map<String, Object> resultado = new HashMap<>();  
  
 // Obtener el préstamo por ID  
 PrestamoEntity prestamo = prestamoRepository.findById(idPrestamo)  
 .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 edu.mtisw.payrollbackend.repositories.ExtraHoursRepository;  
import edu.mtisw.payrollbackend.repositories.PrestamoRepository;  
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 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.deleteById(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 getUsuarioComprobanteIngresosById(Long id){  
 return usuarioComprobanteIngresosRepository.findById(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.deleteById(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.mtisw.payrollbackend.services;  
  
import edu.mtisw.payrollbackend.entities.\*;  
import edu.mtisw.payrollbackend.repositories.\*;  
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;  
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.mtisw.payrollbackend.controllers;  
  
import edu.mtisw.payrollbackend.services.BancoService;  
import edu.mtisw.payrollbackend.services.UsuarioService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Map;  
  
@RestController  
@RequestMapping("/api/v1/bancos")  
@CrossOrigin("\*")  
public class BancoController {  
 @Autowired  
 UsuarioService usuarioService;  
 @Autowired  
 BancoService bancoService;  
 //------------------------------------CRUD----------------------------------------------  
 //------------------------------------PRINCIPALES---------------------------------------  
 //evaluarCredito()(P4)  
 @GetMapping("/evaluar-credito/{idUsuario}/{idPrestamo}")  
 public ResponseEntity<Map<String, Object>> evaluarCredito(  
 @PathVariable Long idUsuario,  
 @PathVariable Long idPrestamo) {  
 try {  
 Map<String, Object> resultado = bancoService.evaluarCredito(idUsuario, idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarRelacionCuotaIngreso()(R1)  
 @GetMapping("/evaluar-relacion-cuota-ingreso/{idUsuario}/{idPrestamo}")  
 public ResponseEntity<Boolean> evaluarRelacionCuotaIngreso(  
 @PathVariable Long idUsuario,  
 @PathVariable Long idPrestamo) {  
 try {  
 boolean resultado = bancoService.evaluarRelacionCuotaIngreso(idUsuario, idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarDeudas()(R2)  
 @GetMapping("/evaluar-historial-crediticio/{idUsuario}")  
 public ResponseEntity<Boolean> evaluarHistorialCrediticio(  
 @PathVariable Long idUsuario) {  
 try {  
 boolean resultado = bancoService.evaluarHistorialCrediticio(idUsuario);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarAntiguedad()(R3)  
 @GetMapping("/evaluar-antiguedad/{idUsuario}")  
 public ResponseEntity<Boolean> evaluarAntiguedad(  
 @PathVariable Long idUsuario) {  
 try {  
 boolean resultado = bancoService.evaluarAntiguedad(idUsuario);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarRelacionDeudaIngreso()(R4)  
 @GetMapping("/evaluar-relacion-deuda-ingreso/{idUsuario}/{idPrestamo}")  
 public ResponseEntity<Boolean> evaluarRelacionDeudaIngreso(  
 @PathVariable Long idUsuario,  
 @PathVariable Long idPrestamo) {  
 try {  
 boolean resultado = bancoService.evaluarRelacionDeudaIngreso(idUsuario, idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarMontoMaximoFinanciamiento()(R5)  
 @GetMapping("/evaluar-monto-maximo/{idPrestamo}")  
 public ResponseEntity<Boolean> evaluarMontoMaximoFinanciamiento(  
 @PathVariable Long idPrestamo) {  
 try {  
 boolean resultado = bancoService.evaluarMontoMaximoFinanciamiento(idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarEdad()(R6)  
 @GetMapping("/evaluar-edad/{idUsuario}/{idPrestamo}")  
 public ResponseEntity<Boolean> evaluarEdad(  
 @PathVariable Long idUsuario,  
 @PathVariable Long idPrestamo) {  
 try {  
 boolean resultado = bancoService.evaluarEdad(idUsuario, idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarSaldoMinimo()(R71)  
 @GetMapping("/evaluar-saldo-minimo/{idUsuario}/{idPrestamo}")  
 public ResponseEntity<Boolean> evaluarSaldoMinimo(  
 @PathVariable Long idUsuario,  
 @PathVariable Long idPrestamo) {  
 try {  
 boolean resultado = bancoService.evaluarSaldoMinimo(idUsuario, idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarHistorialAhorroConsistente(R72)  
 @GetMapping("/evaluar-historial-ahorro/{idUsuario}")  
 public ResponseEntity<Boolean> evaluarHistorialAhorroConsistente(  
 @PathVariable Long idUsuario) {  
 try {  
 boolean resultado = bancoService.evaluarHistorialAhorroConsistente(idUsuario);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarDepositosPeriodicos(R73)  
 @GetMapping("/evaluar-depositos-periodicos/{idUsuario}")  
 public ResponseEntity<Boolean> evaluarDepositosPeriodicos(  
 @PathVariable Long idUsuario) {  
 try {  
 boolean resultado = bancoService.evaluarDepositosPeriodicos(idUsuario);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarRelacionSaldoAntiguedad(R74)  
 @GetMapping("/evaluar-relacion-saldo-antiguedad/{idUsuario}/{idPrestamo}")  
 public ResponseEntity<Boolean> evaluarRelacionSaldoAntiguedad(  
 @PathVariable Long idUsuario,  
 @PathVariable Long idPrestamo) {  
 try {  
 boolean resultado = bancoService.evaluarRelacionSaldoAntiguedad(idUsuario, idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarRetiroReciente(R75)  
 @GetMapping("/evaluar-retiros-recientes/{idUsuario}")  
 public ResponseEntity<Boolean> evaluarRetirosRecientes(  
 @PathVariable Long idUsuario) {  
 try {  
 boolean resultado = bancoService.evaluarRetirosRecientes(idUsuario);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //evaluarCapacidadAhorro()(R7)  
 @GetMapping("/evaluar-capacidad-ahorro/{idUsuario}/{idPrestamo}")  
 public ResponseEntity<Map<String, Object>> evaluarCapacidadAhorro(  
 @PathVariable Long idUsuario,  
 @PathVariable Long idPrestamo) {  
 try {  
 Map<String, Object> resultado = bancoService.evaluarCapacidadAhorro(idUsuario, idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 //calcularCostoTotales()(P6)  
 @GetMapping("/calcular-costo-total/{idPrestamo}")  
 public ResponseEntity<Map<String, Object>> calcularCostoTotalPrestamo(  
 @PathVariable Long idPrestamo) {  
 try {  
 Map<String, Object> resultado = bancoService.calcularCostoTotalPrestamo(idPrestamo);  
 return ResponseEntity.*ok*(resultado);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
}

CONTROLADOR COMPROBANTE-INGRESOS

package edu.mtisw.payrollbackend.controllers;  
  
import edu.mtisw.payrollbackend.entities.ComprobanteIngresosEntity;  
import edu.mtisw.payrollbackend.services.ComprobanteIngresosService;  
import edu.mtisw.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.mtisw.payrollbackend.controllers;  
  
import edu.mtisw.payrollbackend.entities.PrestamoEntity;  
import edu.mtisw.payrollbackend.entities.UsuarioEntity;  
import edu.mtisw.payrollbackend.services.PrestamoService;  
import edu.mtisw.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.mtisw.payrollbackend.controllers;  
  
import edu.mtisw.payrollbackend.entities.UsuarioComprobanteIngresosEntity;  
import edu.mtisw.payrollbackend.entities.UsuarioEntity;  
import edu.mtisw.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.mtisw.payrollbackend.controllers;  
  
import edu.mtisw.payrollbackend.entities.ComprobanteIngresosEntity;  
import edu.mtisw.payrollbackend.entities.PrestamoEntity;  
import edu.mtisw.payrollbackend.entities.UsuarioEntity;  
import edu.mtisw.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---------------------------------------  
 //simularCredito()(P1)  
 @GetMapping("/{id}/simular-credito/{idPrestamo}")  
 public ResponseEntity<Map<String, Object>> simularCredito(  
 @PathVariable Long id,  
 @PathVariable Long idPrestamo) {  
 try {  
 // Llamar al servicio para simular el crédito  
 Map<String, Object> resultadoSimulacion = usuarioService.simularCredito(id, idPrestamo);  
 return ResponseEntity.*ok*(resultadoSimulacion);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
  
 // Registrar usuario(implementado en el CRUD)  
  
 // Solicitar Credito (P3) (por implementar)  
 // Registrar usuario (P2) (implementado en el CRUD)  
 @PostMapping("/{id}/solicitar-credito")  
 public ResponseEntity<PrestamoEntity> solicitarCredito(  
 @PathVariable Long id,  
 @RequestBody Map<String, Object> requestBody) {  
  
 try {  
 // Extracción segura de datos  
 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(); // Nuevo código  
  
 int antiguedadLaboral = ((Number) requestBody.get("antiguedadLaboral")).intValue();  
 int ingresoMensual = ((Number) requestBody.get("ingresoMensual")).intValue();  
 int saldo = ((Number) requestBody.get("saldo")).intValue();  
  
 // Manejo de listas (solo ingresos)  
 List<Number> ingresosNumbers = (List<Number>) requestBody.get("ingresosUltimos24Meses");  
 List<Integer> ingresosUltimos24Meses = ingresosNumbers.stream()  
 .map(Number::intValue)  
 .collect(Collectors.*toList*());  
  
 // Crear los objetos PrestamoEntity y ComprobanteIngresosEntity  
 PrestamoEntity prestamo = new PrestamoEntity();  
 prestamo.setTipo(tipo);  
 prestamo.setPlazo(plazo);  
 prestamo.setTasaInteres(tasaInteres);  
 prestamo.setMonto(monto);  
 prestamo.setEstado("En proceso");  
 prestamo.setValorPropiedad(valorPropiedad); // Nuevo código  
  
 ComprobanteIngresosEntity comprobanteIngresos = new ComprobanteIngresosEntity();  
 comprobanteIngresos.setAntiguedadLaboral(antiguedadLaboral);  
 comprobanteIngresos.setIngresoMensual(ingresoMensual);  
 comprobanteIngresos.setSaldo(saldo);  
  
 // Convertir lista de ingresos a cadena y asignar  
 String ingresosString = ingresosUltimos24Meses.stream()  
 .map(String::*valueOf*)  
 .collect(Collectors.*joining*(","));  
 comprobanteIngresos.setIngresosUltimos24Meses(ingresosString);  
  
 // 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 solicitud (P5) (por implementar)  
 @GetMapping("/{id}/estado-solicitudes")  
 public ResponseEntity<List<PrestamoEntity>> obtenerEstadoSolicitudes(@PathVariable Long id) {  
 try {  
 List<PrestamoEntity> estadosPrestamos = usuarioService.obtenerEstadoSolicitudes(id);  
 return ResponseEntity.*ok*(estadosPrestamos);  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body(null);  
 }  
 }  
}

### CONTROLADOR USUARIO-PRESTAMOS

package edu.mtisw.payrollbackend.controllers;  
  
import edu.mtisw.payrollbackend.entities.UsuarioPrestamoEntity;  
import edu.mtisw.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---------------------------------------  
}