structura del proyecto para cumplir con los requisitos. Esto incluirá:

- **Proyecto base en Java Spring Boot:** Configuración inicial del proyecto.
- **Entidades y repositorios:** Creación de las clases Persona, Cliente, Cuenta y Movimiento.
- **Controladores y servicios:** Implementación de CRUDs y registro de movimientos.
- **Gestión de excepciones:** Manejo de errores, incluyendo la validación de saldo insuficiente.
- **Pruebas unitarias e integración**.
- **Despliegue con Docker.**
- **Script de base de datos y colección Postman.**

código base.

```
import javax.persistence.*;
import java.util.List;

@Entity
public class Persona {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nombre;
    private String genero;
    private int edad;
    private String identificacion;
    private String direccion;
    private String telefono;

    // Getters y Setters
    // ...
```

```
}

@Entity
public class Cliente extends Persona {
    @Column(unique = true)
    private Long clienteId;
    private String contrasena;
    private boolean estado;

    // Getters y Setters
    // ...
}

@Entity
public class Cuenta {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long numeroCuenta;
    private String tipoCuenta;
    private double saldoInicial;
    private boolean estado;

    @ManyToOne
    private Cliente cliente;

    // Getters y Setters
    // ...
}
```

```
@Entity
public class Movimiento {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String fecha;
    private String tipoMovimiento;
    private double valor;
    private double saldo;



  @ManyToOne
    private Cuenta cuenta;

    // Getters y Setters
    // ...
}

import org.springframework.data.jpa.repository.JpaRepository;

public interface PersonaRepository extends JpaRepository<Persona, Long> {
}

public interface ClienteRepository extends JpaRepository<Cliente, Long> {
}

public interface CuentaRepository extends JpaRepository<Cuenta, Long> {
}
```

```
public interface MovimientoRepository extends JpaRepository<Movimiento, Long> {
}

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/clientes")
public class ClienteController {



@Autowired
   private ClienteRepository clienteRepository;

   @GetMapping
   public List<Cliente> getAllClientes() {
      return clienteRepository.findAll();
   }

   @PostMapping
   public Cliente createCliente(@RequestBody Cliente cliente) {
      return clienteRepository.save(cliente);
   }

   @PutMapping("/{id}")
   public Cliente updateCliente(@PathVariable Long id, @RequestBody Cliente clienteDetails) {
      Cliente cliente = clienteRepository.findById(id).orElseThrow(() -> new
ResourceNotFoundException("Cliente no encontrado"));
```

```
        cliente.setContrasena(clienteDetails.getContrasena());
        cliente.setEstado(clienteDetails.isEstado());
        return clienteRepository.save(cliente);
    }

    @DeleteMapping("/{id}")
    public void deleteCliente(@PathVariable Long id) {
        clienteRepository.deleteById(id);
    }
}

@RestController
@RequestMapping("/cuentas")
public class CuentaController {

    @Autowired
    private CuentaRepository cuentaRepository;

    @GetMapping
    public List<Cuenta> getAllCuentas() {
        return cuentaRepository.findAll();
    }

    @PostMapping
    public Cuenta createCuenta(@RequestBody Cuenta cuenta) {
        return cuentaRepository.save(cuenta);
    }

    @PutMapping("/{id}")
```

```
    public Cuenta updateCuenta(@PathVariable Long id, @RequestBody Cuenta cuentaDetails) {
        Cuenta cuenta = cuentaRepository.findById(id).orElseThrow(() -> new
ResourceNotFoundException("Cuenta no encontrada"));
        cuenta.setTipoCuenta(cuentaDetails.getTipoCuenta());
        cuenta.setSaldoInicial(cuentaDetails.getSaldoInicial());
        cuenta.setEstado(cuentaDetails.isEstado());
        return cuentaRepository.save(cuenta);
    }

    @DeleteMapping("/{id}")
    public void deleteCuenta(@PathVariable Long id) {
        cuentaRepository.deleteById(id);
    }
}

@RestController
@RequestMapping("/movimientos")
public class MovimientoController {

    @Autowired
    private MovimientoRepository movimientoRepository;
    @Autowired
    private CuentaRepository cuentaRepository;

    @GetMapping
    public List<Movimiento> getAllMovimientos() {
        return movimientoRepository.findAll();
    }
```

```
    @PostMapping
    public Movimiento createMovimiento(@RequestBody Movimiento movimiento) {
        Cuenta cuenta = cuentaRepository.findById(movimiento.getCuenta().getNumeroCuenta())
                .orElseThrow(() -> new ResourceNotFoundException("Cuenta no encontrada"));

        double nuevoSaldo = cuenta.getSaldoInicial() + movimiento.getValor();
        if (nuevoSaldo < 0) {
            throw new InsufficientBalanceException("Saldo no disponible");
        }

        movimiento.setSaldo(nuevoSaldo);
        cuenta.setSaldoInicial(nuevoSaldo);
        cuentaRepository.save(cuenta);

        return movimientoRepository.save(movimiento);
    }

    @PutMapping("/{id}")
    public Movimiento updateMovimiento(@PathVariable Long id, @RequestBody Movimiento
movimientoDetails) {
        Movimiento movimiento = movimientoRepository.findById(id).orElseThrow(() -> new
ResourceNotFoundException("Movimiento no encontrado"));
        movimiento.setTipoMovimiento(movimientoDetails.getTipoMovimiento());
        movimiento.setValor(movimientoDetails.getValor());
        movimiento.setSaldo(movimientoDetails.getSaldo());
        return movimientoRepository.save(movimiento);
    }

    @DeleteMapping("/{id}")
```

```
    public void deleteMovimiento(@PathVariable Long id) {
        movimientoRepository.deleteById(id);
    }
}

@ResponseStatus(code = org.springframework.http.HttpStatus.NOT_FOUND)
class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }
}

@ResponseStatus(code = org.springframework.http.HttpStatus.BAD_REQUEST)
class InsufficientBalanceException extends RuntimeException {
    public InsufficientBalanceException(String message) {
        super(message);
    }
}
```

// Pruebas unitarias e integración

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
```

```
@SpringBootTest
@AutoConfigureMockMvc
public class ClienteControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testCreateCliente() throws Exception {
        String clienteJson = "{\"nombre\":\"Jose
Lema\",\"clienteId\":1,\"contrasena\":\"1234\",\"estado\":true}";
        mockMvc.perform(post("/clientes")
                .contentType(MediaType.APPLICATION_JSON)
                .content(clienteJson))
                .andExpect(status().isOk());
    }

    @Test
    public void testGetAllClientes() throws Exception {
        mockMvc.perform(get("/clientes"))
                .andExpect(status().isOk())
                .andExpect(content().contentType(MediaType.APPLICATION_JSON));
    }
}

@SpringBootTest
@AutoConfigureMockMvc
public class MovimientoControllerIntegrationTest {
```

```
    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testMovimientoWithInsufficientBalance() throws Exception {
        String movimientoJson = "{\"cuenta\":{\"numeroCuenta\":1},\"valor\":-5000}";

        mockMvc.perform(post("/movimientos")
                .contentType(MediaType.APPLICATION_JSON)
                .content(movimientoJson))
                .andExpect(status().isBadRequest())
                .andExpect(content().string("Saldo no disponible"));
    }
}
```

-- BaseDatos.sql --

```
CREATE TABLE persona (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255),
    genero VARCHAR(50),
    edad INT,
    identificacion VARCHAR(255),
    direccion VARCHAR(255),
    telefono VARCHAR(50)
);

CREATE TABLE cliente (
```

```
    cliente_id BIGINT PRIMARY KEY,
    contrasena VARCHAR(255),
    estado BOOLEAN,
    id BIGINT,
    FOREIGN KEY (id) REFERENCES persona(id)
);

CREATE TABLE cuenta (
    numero_cuenta BIGINT AUTO_INCREMENT PRIMARY KEY,
    tipo_cuenta VARCHAR(50),
    saldo_inicial DOUBLE,
    estado BOOLEAN,
    cliente_id BIGINT,
    FOREIGN KEY (cliente_id) REFERENCES cliente(cliente_id)
);

CREATE TABLE movimiento (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    fecha VARCHAR(50),
    tipo_movimiento VARCHAR(50),
    valor DOUBLE,
    saldo DOUBLE,
    cuenta_id BIGINT,
    FOREIGN KEY (cuenta_id) REFERENCES cuenta(numero_cuenta)
);

-- Dockerfile --

FROM openjdk:17-jdk-slim
```

```
VOLUME /tmp
ARG JAR_FILE=target/backend-application.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]

-- docker-compose.yml --

version: '3.8'
services:
  backend:
    build: .
    ports:
      - "8080:8080"
    depends_on:
      - db
  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: testdb
    ports:
      - "3306:3306"
```