JAIRO ANDRÉS FIERRO 202226326

MARCOS RODRIGO ESPAÑA 202124714

JUAN FELIPE PUIG PARDO

DOCUMENTO INFORME

Documentación de los RF:

RF1:

Este requerimiento funcional se encarga de hacer el login para los usuarios de la aplicación. Para hacer este requerimiento se hace una consulta a la base de datos. En esta consulta se obtienen los usuarios de clientes y empleados. Dependiendo del tipo de usuario es se asigna su interfaz. Para el login los únicos que pueden crear login es el administrador y el gerente.

RF2:

Para este requerimiento lo principal es enseñar al usuario un form donde pueda ingresar sus datos para el registro de la oficina. En este requerimiento también se pueden crear y borrar puntos de atención. Esta funcionalidad se encuentra en la interfaz del administrador del banco.

RF3:

Esta operación se inicia desde el usuario de Gerente de Oficina el cual se envía hacia Cuentas la cual contiene la información de todas las cuentas creadas, y existe un botón que permite crear cuentas el cual lleva a un Forms que está en Fragments el cual recibe los datos necesarios para la cuenta. Solo se modificó el controller para poner cuentasNew para que redirija hacia allá al hacer ese llamado.

RF4:

Para poder desactivar o cerrar una cuenta se requerían unas condiciones antes por tanto estas son validadas por tanto en el contoller de Cuentas " cuentas/{id}/edit" validando que su saldo sea 0 y que el estado actual de la cuenta sea activo, si estos se cumplían se redirige hacia un form diferente al anterior el cual tiene la opcionalidad de poder cerrar la cuenta o desactivarla.

Para la parte de agregar operaciones se tiene una página que muestra todas las operaciones y además hay un botón que permite agregar una operación, y acá se llama a un form que contiene toda la información para que el usuario pueda registrar su operación, para hacer esta operación efectiva y se modifique el saldo independiente de la operación se obtiene la cuenta desde OperacionesCuentasController y en la parte de crearla se verifica el tipo de operación que es y si es consignación se le agrega al saldo, si es retiro se le resto y si es transferencia se actualizan los 2 saldos de las 2 cuentas afectadas.

RF5:

Esta operación se inicia desde el usuario de Gerente de Oficina el cual se envía hacia Prestamos la cual contiene la información de todas las prestamos creados, y existe un botón que permite crear prestamos el cual lleva a un Forms que está en Fragments el cual recibe los datos necesarios para los préstamos. Solo se modificó el controller para poner prestamosNew para que redirija hacia allá al hacer ese llamado.

Para la parte de registro prestamos se tiene una página que muestra todas las operaciones y además hay un botón que permite agregar una operación, y acá se llama a un form que contiene toda la información para que el usuario pueda registrar su operación, para hacer esta operación efectiva y se modifique el saldo pendiente

RF6:

Para poder cerrar un préstamo se requerían unas condiciones antes por tanto estas son validadas por tanto en el contoller de Prestamo “ prestamo/{id}/edit" validando que el saldo pendiente sea 0, si este se cumple se redirige hacia un form diferente al anterior el cual le da la opción de cerrarlo.

Parte B:

Para hacer los requerimientos de la parte B fue necesario agregar una capa a la aplicación para los servicios donde se validan las reglas de negocio y se definen los niveles de aislamiento para las transacciones, en cada operación de los requerimientos. Los métodos de esta capa proveen funcionalidades de consulta y actualización para las transacciones.

Documentación RFC4:

En este requerimiento que trata de una consulta sobre una cuenta específica, esta consulta vendría siendo como un extracto bancario. En el método que se creó en el servicio de operaciones cuentas se agrega el nivel de aislamiento serializable, y se establece un rollback en caso de que haya un error. En este método se obtienen los datos de una consulta inicial y los datos de una consulta pasados 30 segundos, lo que permite estudiar el comportamiento de las transacciones y las consultas en los escenarios 1 y 2.

```java
@Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor = Exception.class)
public HashMap<String, Collection<OperacionCuenta>>  consultaOpCuentaUltimoMesSerializable(Integer numero_cuenta) throws InterruptedException {
    try{
            HashMap<String, Collection<OperacionCuenta>> map = new HashMap<String, Collection<OperacionCuenta> >();

            Date fecha = new Date(System.currentTimeMillis());

            Collection<OperacionCuenta>  operacionesCuentas = operacionesCuentasRepository.consultaOpCuentaUltimoMes(fecha,numero_cuenta);
            map.put(key:"operacion_cuenta1", operacionesCuentas);
            System.out.println(operacionesCuentas.size());

            Thread.sleep(millis:300);
            Collection<OperacionCuenta>  operacionesCuentas2 = operacionesCuentasRepository.consultaOpCuentaUltimoMes(fecha,numero_cuenta);
            map.put(key:"operacion_cuenta2", operacionesCuentas2);

            return map;

    }catch(InterruptedException e){
            throw new InterruptedException(s:"Error en la transaccion Serializable");
    }
}
```

Documentación RFC4:

La implementación de este requerimiento es similar al anterior, en este caso se hace la misma consulta con la diferencia de que el nivel de aislamiento es Read commited. Aquí también se hacen dos consultas para comparar una consulta anterior y una consulta posterior.

```java
@Transactional(isolation = Isolation.READ_COMMITTED, rollbackFor = Exception.class)
public HashMap<String, Collection<OperacionCuenta>> consultaOpCuentaUltimoMesReadCommited(Integer numero_cuenta) throws InterruptedException {
    try{
            HashMap<String, Collection<OperacionCuenta>> map = new HashMap<String, Collection<OperacionCuenta> >();

            Date fecha = new Date(System.currentTimeMillis());

            Collection<OperacionCuenta>  operacionesCuentas = operacionesCuentasRepository.consultaOpCuentaUltimoMes(fecha,numero_cuenta);
            map.put(key:"operacion_cuenta1", operacionesCuentas);
            System.out.println(operacionesCuentas.size());

            Thread.sleep(millis:300);
            Collection<OperacionCuenta>  operacionesCuentas2 = operacionesCuentasRepository.consultaOpCuentaUltimoMes(fecha,numero_cuenta);
            map.put(key:"operacion_cuenta2", operacionesCuentas2);

            return map;

    }catch(InterruptedException e){
            throw new InterruptedException(s:"Error en la transaccion Read Committed");
    }

}
```

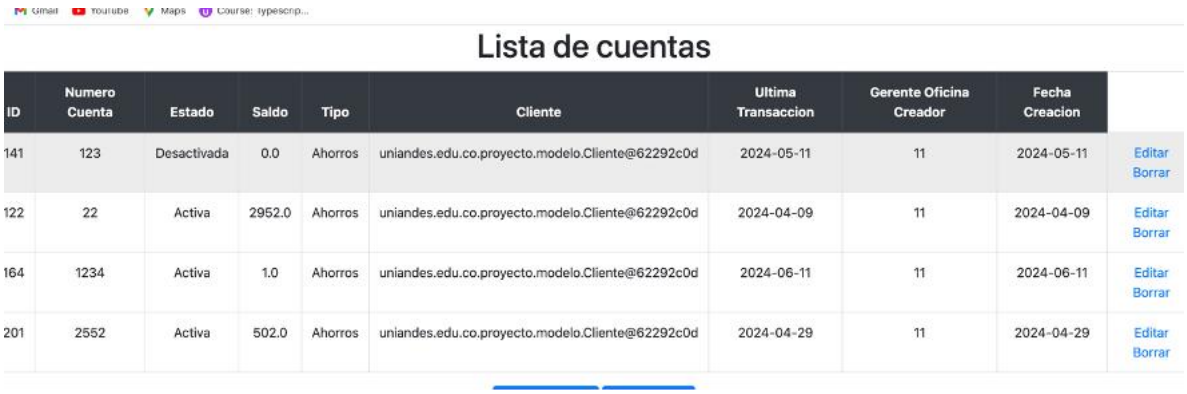
Documentación RFC5:

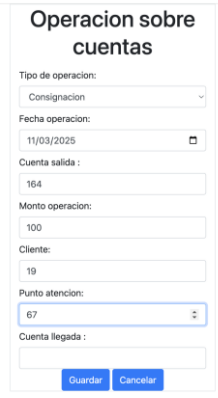
```java
@PostMapping("/operacionCuenta/new/save")
@Transactional
public String transaccionGuardar (@ModelAttribute OperacionCuenta operacionCuenta, Model model) {
  try {
  Cuenta cuentaSalida=cuentaRepository.darCuenta(operacionCuenta.getCuenta_salida());

  Float valorOperacion=operacionCuenta.getMonto_operacion();
  Float saldo=cuentaSalida.getSaldo();
  valorOperacion=valorOperacion+saldo;
  cuentaRepository.actualizarCuenta(cuentaSalida.getId(), cuentaSalida.getNumero_cuenta(), cuentaSalida.getEstado(), valorOperacion, cuentaSalida.getTipo(),
  cuentaSalida.getCliente().getId(), cuentaSalida.getUltima_transaccion(), cuentaSalida.getGerente_oficina_creador(), cuentaSalida.getFecha_creacion());
int rowsAffectedConsignar = cuentaRepository.actualizarSaldoConsignar (cuentaSalida.getId(), operacionCuenta.getMonto_operacion());
int rowsAffectedRetirar = cuentaRepository.actualizarSaldoRetiro (cuentaSalida.getId(), operacionCuenta.getMonto_operacion());
if (rowsAffectedConsignar > 0 && rowsAffectedRetirar > 0) {
  OperacionCuentaRepository.insertarOperacioneCuenta (cuentaSalida.getNumero_cuenta(), operacionCuenta.getFecha(), rowsAffectedRetirar,
  operacionCuenta.getMonto_operacion(), cuentaSalida.getId(), operacionCuenta.getCuenta_llegada(), operacionCuenta.getPunto_atencion().getId());
  return "redirect:/cuentas";
} else {

throw new RuntimeException("Error al hacer la transaccion: No se pudieron completar las actualizaciones de saldo");
}
} catch (Exception e) {
TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
model.addAttribute(attributeName:"errorMessage", e.getMessage());
System.out.println(e.getMessage());
return "error";
}
}



@PostMapping("/retirar/new/save")
@Transactional
public String retirarDinero (@ModelAttribute OperacionCuenta operacion_cuenta, Model model) {
try {
  int rowsAffectedRetirar = cuentaRepository.actualizarSaldoRetiro (operacion_cuenta.getId(), operacion_cuenta.getMonto_operacion());
  if (rowsAffectedRetirar > 0) {
    OperacionCuentaRepository.insertarOperacioneCuenta(operacion_cuenta.getTipo_operacion(), operacion_cuenta.getFecha(),
    rowsAffectedRetirar, operacion_cuenta.getMonto_operacion(), operacion_cuenta.getPunto_atencion().getId(), rowsAffectedRetirar, rowsAffectedRetirar);
return "redirect:/cuentas";
  }
else {
throw new RuntimeException("Error al hacer el retiro: No se pudieron completar las actualizaciones de saldo");
} }catch (Exception e) {
TransactionAspectSupport.currentTransactionStatus().setRollbackOnly();
model.addAttribute(attributeName:"errorMessage", "Error al retirar dinero: " + e.getMessage());
System.out.println(e.getMessage());
return "redirect:/error";
}
}
}
```

este requerimiento funcional involucra 3 operaciones: retirar dinero de una cuenta, consignar dinero de una cuenta y transferir dinero de una cuenta a otra del mismo banco. En este requerimiento se establecen las transacciones para que se registren las operaciones de cuenta al momento en que se haga una transacción como un retiro o una consignación. El objetivo de esta es que al momento en que se haga una transaccion de consignacion, por ejemplo, simultaneamente se haga el registro de dicha transaccion en operacionesCuenta con el fin de que quede guardada la informacion de este proceso

6.)

Antes de realizar la transacción y la consulta la cuenta afectada será la de ID=164 la cual cuenta con saldo 1, cliente=19



RFC4		RF6
 Se empieza la consulta sobre la cuenta recién creada para este ejemplo	T1	
 La operación empieza a cargar	T2	
	T3	En la otra sesión se registra una operación de consignación 
	T4	Se hace efectiva en el registro de Operaciones cuentas
Como la transacción es serializable los datos no se verán reflejados 	T4	

Lo sucedido se debe al uso de la operación RFC4 con aislamiento serializable, el nivel más alto de aislamiento la cual mantiene los datos leídos de manera consistente e intacta, a la vez que aísla las

modificaciones realizadas por otras transacciones concurrentes para asegurar la consistencia de la base de datos.

Cuando se realiza la solicitud de información sobre la cuenta ID=164 en la transacción T1 mediante RFC4, esta transacción opera sobre los últimos datos consistentes disponibles en ese momento. Mientras tanto, en la transacción T3, se realiza una inserción de operación con RFC6 en la misma cuenta ID=164. Debido al aislamiento serializable, la inserción de la operación en T3 no se reflejará en la sesión de RFC4 Esto se debe a que RFC4 se ejecuta sobre un estado consistente de la base de datos que existía en el inicio de la transacción y no es consciente de las modificaciones realizadas por otras transacciones concurrentes, como la inserción realizada por RFC6. Para que RFC4 refleje de manera correcta la nueva operación realizada por RFC6, la transacción de RFC6 (T3) debería haber esperado a que RFC4 (T1) completara su lectura de los datos antes de realizar la inserción. Esto permitiría que RFC4 viera un estado actualizado de la cuenta que incluyera la nueva consignación.

7.)

Para esta parte el RFC5 tiene tipo de serializacion Read commited el cual permite leer datos los cuales ya están confirmados es decir ya están cargados en la BD, por tanto, en este caso el RFC5 al empezar la consulta, y después haber tenido una inserción de datos mediante RF6 se insertan los datos en esta sesión ya que estos si son confirmados por tanto se produce un fantasma. El componente RF6 debió esperar a que este termine y así no generar fantasmas los cuales los cuales violan el aislamiento y consistencia de la base de datos.