Prueba Técnica/Práctica

Herramientas y tecnologías utilizadas

- Java
- Hibernate
- Spring Boot, Data
- JUnit, Mockito
- PostgreSQL
- RabbitMQ
- Docker, Docker Compose
- IntelliJ
- Postman

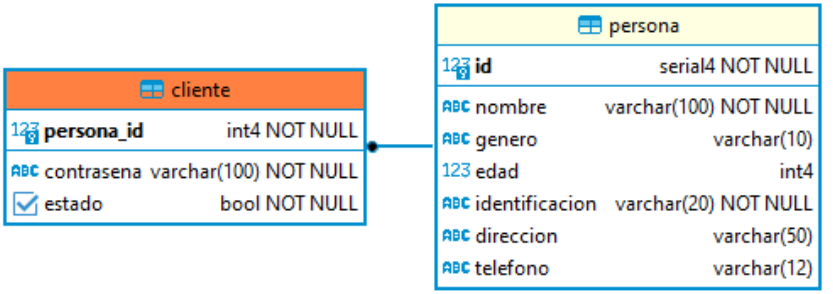
Requerimiento de Microservicios y Funcionalidades

1. Microservicio userService

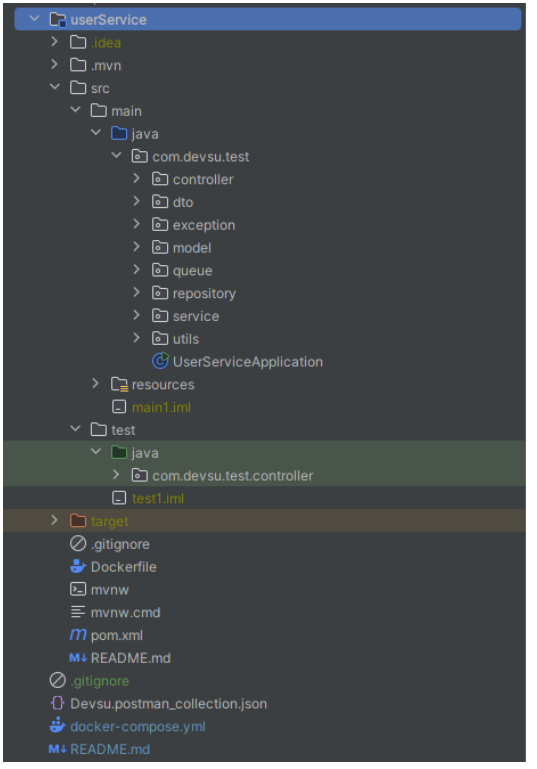
Este microservicio contiene la funcionalidad relacionada con los clientes. El servicio se enfoca a manejar toda las funcionalidades de clientes, a través de comunicación asincrónica mediante el uso de **rabbitMQ** el servicio publica en las colas de mensaje cuando una operación de inserción, actualización o borrado de clientes es realizada.

Para los dos microservicios se utilizó el patrón de base de datos por microservicio lo cual nos permite mejorar el rendimiento, escalabilidad y resiliencia del sistema.

Diagrama Entidad Relación



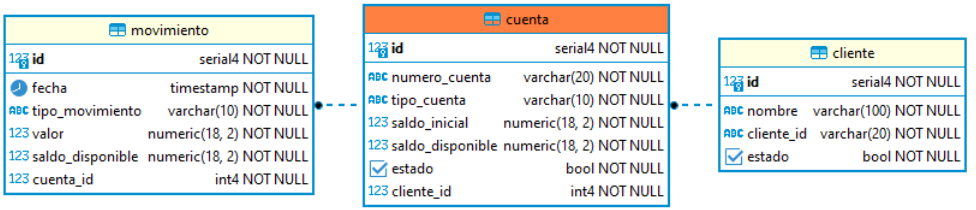
Estructura del proyecto



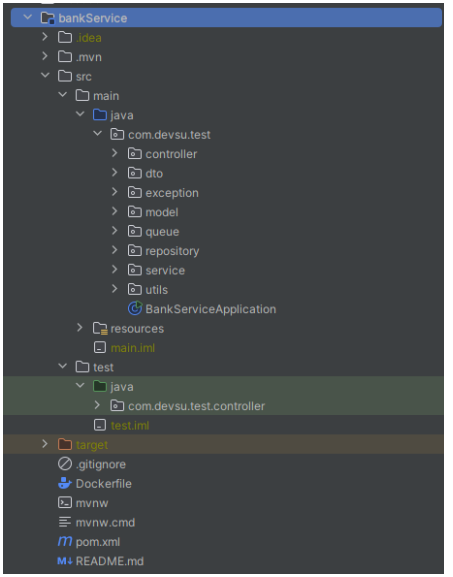
2. Microservicio bankService

Este microservicio contiene la funcionalidad relacionada con las cuentas y movimientos. El servicio cuenta con listeners para las colas 3 colas creadas (Creación, Actualización, Borrado). El servicio de forma asíncrona escucha cuando un nuevo cliente es creado actualizado o borrado en el microservicio 1, esto le permite actualizar la información su propia la tabla **cliente**, en la cual se almacena información relevante para el microservicio.

Diagrama Entidad Relación



Estructura del proyecto



Funcionalidades del API

F1: Generación de CRUDS

Los dos microservicios cuentan con las operaciones CRUD sobre las entidades: Cliente, Cuenta y Movimiento.

1. **Microservicio userService**

ClienteController.java

```java
package com.devsu.test.controller;

import ...

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

    @Value("${messagge.cliente.delete}")
    private String clienteBorrado;

    @Value("${messagge.cliente.notfound}")
    private String clienteNoEncontrado;

    private final ClienteService clienteService;

    @Autowired
    public ClienteController(ClienteService clienteService) { this.clienteService = clienteService; }

    @GetMapping
    public ResponseEntity<Iterable<Cliente>> getAllClientes() {...}

    @GetMapping("/{clienteId}")
    public ResponseEntity<Cliente> getClienteByClienteId(@PathVariable String clienteId) {...}

    @PostMapping
    public ResponseEntity<Cliente> createCliente(@Valid @RequestBody Cliente cliente) {...}

    @PutMapping("/{id}")
    public ResponseEntity<Cliente> updateCliente(@PathVariable Long id, @RequestBody Cliente clienteDetails) {...}

    @PatchMapping("/{id}")
    public ResponseEntity<Cliente> updateEstadoCliente(@PathVariable Long id) {...}

    @DeleteMapping("/{id}")
    public ResponseEntity<ApiResponse> deleteCliente(@PathVariable Long id) {...}
```

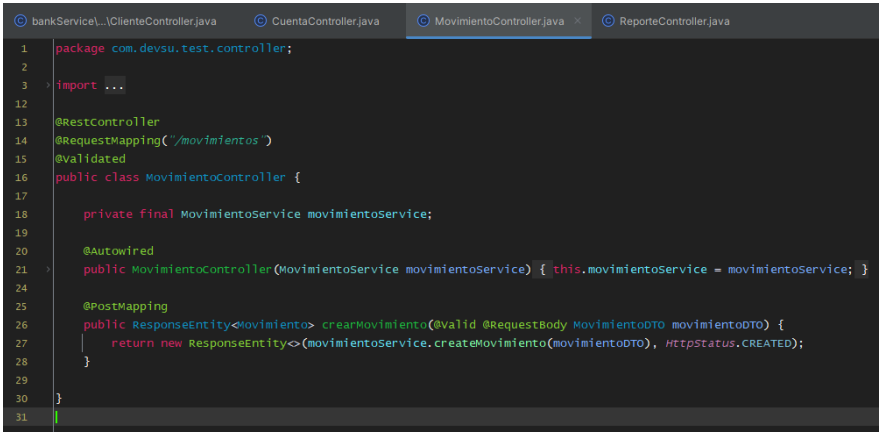
1. Microservicio bankService

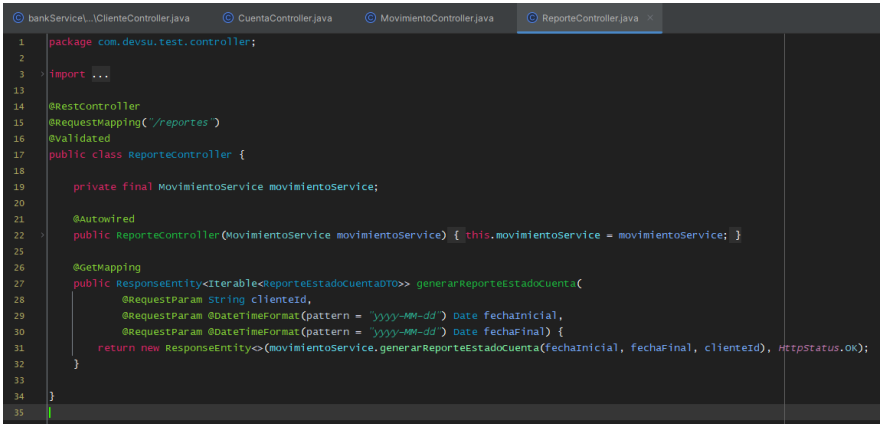
bankService\...\ClienteController.java

```java
package com.devsu.test.controller;

import ...

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

    private final ClienteService clienteService;

    @Autowired
    public ClienteController(ClienteService clienteService) { this.clienteService = clienteService; }

    @GetMapping
    public ResponseEntity<Iterable<Cliente>> getAllClientes() {
        return new ResponseEntity<>(clienteService.getAllClientes(), HttpStatus.OK);
    }

    @GetMapping("/{clienteId}")
    public ResponseEntity<Cliente> getClienteByClienteId(@PathVariable String clienteId) {
        return clienteService.getClienteByClienteId(clienteId).map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

}
```

bankService\...\ClienteController.java | CuentaController.java | MovimientoController.java | ReporteController.java

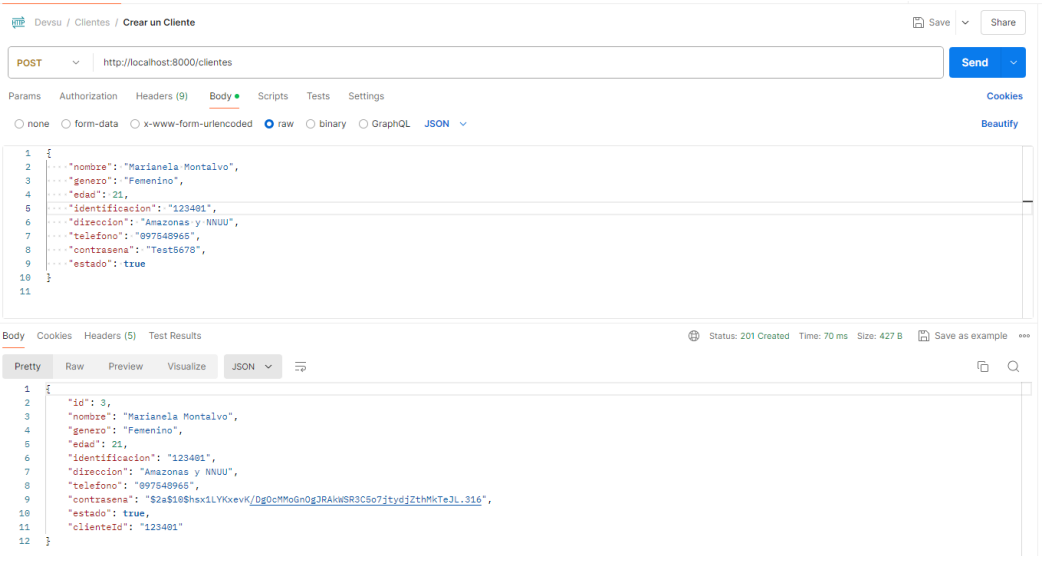
```java
package com.devsu.test.controller;

import ...

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

    private final CuentaService cuentaService;

    @Autowired
    public CuentaController(CuentaService cuentaService) { this.cuentaService = cuentaService; }

    @GetMapping
    public ResponseEntity<Iterable<Cuenta>> getAllCuentas() {...}

    @GetMapping("/{clienteId}")
    public ResponseEntity<Iterable<Cuenta>> getCuentasByClienteId(@PathVariable String clienteId) {...}

    @PostMapping
    public ResponseEntity<Cuenta> crearCuenta(@Valid @RequestBody CuentaDTO cuentaDTO) {...}

    @PatchMapping("/{id}")
    public ResponseEntity<ApiResponse> cerrarCuenta(@PathVariable Long id) {...}

}
```

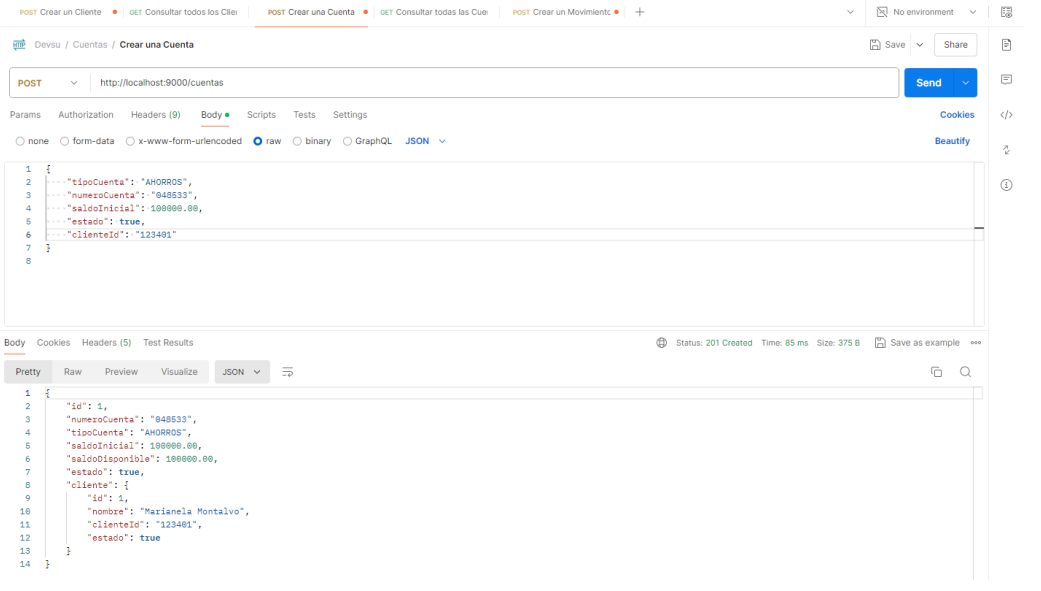


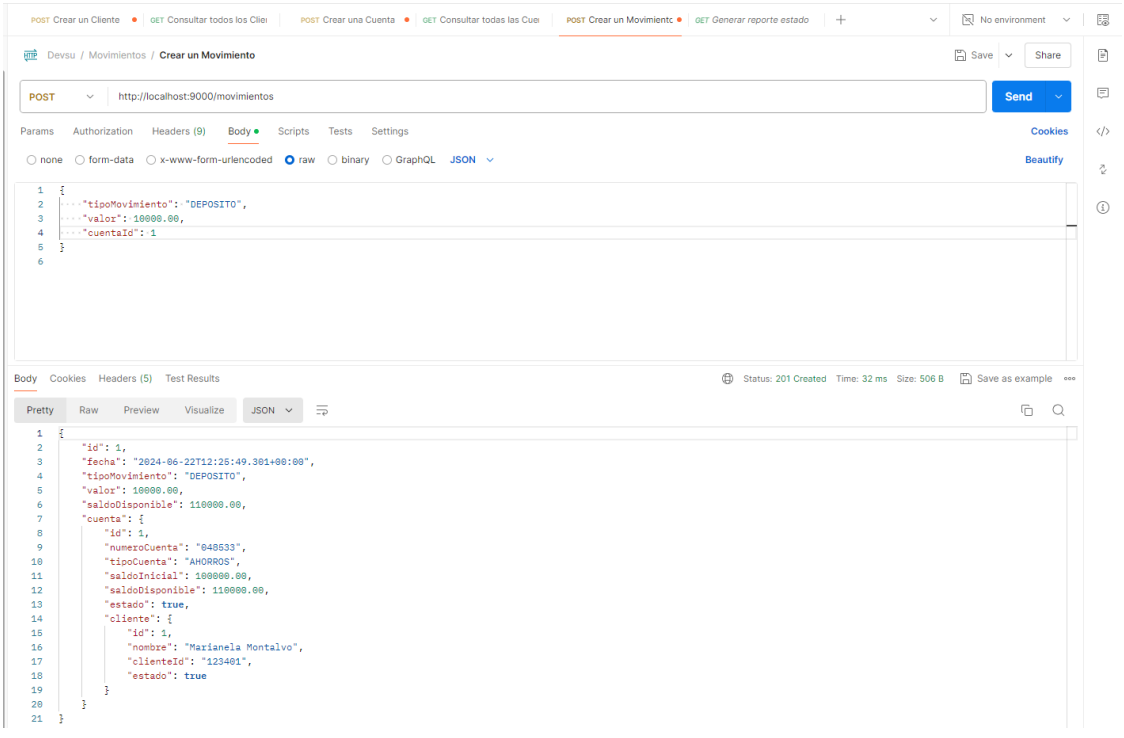


Los nombres de los endpoints a generar son:

- /clientes
- /cuentas
- /movimientos


Devsu / Clientes / Crear un Cliente
Save
Share
POST
http://localhost:8000/clientes
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nombre": "Marianela Montalvo",
"genero": "Femenino",
"edad": 21,
"identificacion": "123401",
"direccion": "Amazonas y NNUU",
"telefono": "097548965",
"contrasena": "Test6678",
"estado": true
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 70 ms
Size: 427 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": 3,
"nombre": "Marianela Montalvo",
"genero": "Femenino",
"edad": 21,
"identificacion": "123401",
"direccion": "Amazonas y NNUU",
"telefono": "097548965",
"contrasena": "$2a$10$hsx1LYKxevK/DgOcMMoGnOgJRAkWSR3C5o7jtydjZthMkTeJL.316",
"estado": true,
"clienteId": "123401"
}


POST Crear un Cliente
GET Consultar todos los Clie
POST Crear una Cuenta
GET Consultar todas las Cue
POST Crear un Movimiento
No environment
Devsu / Cuentas / Crear una Cuenta
Save
Share
POST
http://localhost:9000/cuentas
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"tipoCuenta": "AHORROS",
"numeroCuenta": "048533",
"saldoInicial": 100000.00,
"estado": true,
"clienteId": "123401"
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 85 ms
Size: 375 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"numeroCuenta": "048533",
"tipoCuenta": "AHORROS",
"saldoInicial": 100000.00,
"saldoDisponible": 100000.00,
"estado": true,
"cliente": {
"id": 1,
"nombre": "Marianela Montalvo",
"clienteId": "123401",
"estado": true
}
}


POST Crear un Cliente
GET Consultar todos los Clie
POST Crear una Cuenta
GET Consultar todas las Cue
POST Crear un Movimiento
GET Generar reporte estado
No environment
Devsu / Movimientos / Crear un Movimiento
Save
Share
POST
http://localhost:9000/movimientos
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"tipoMovimiento": "DEPOSITO",
"valor": 10000.00,
"cuentaId": 1
}
Body
Cookies
Headers (5)
Test Results
Status: 201 Created
Time: 32 ms
Size: 506 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"id": 1,
"fecha": "2024-06-22T12:25:49.301+00:00",
"tipoMovimiento": "DEPOSITO",
"valor": 10000.00,
"saldoDisponible": 110000.00,
"cuenta": {
"id": 1,
"numeroCuenta": "648533",
"tipoCuenta": "AHORROS",
"saldoInicial": 100000.00,
"saldoDisponible": 110000.00,
"estado": true,
"cliente": {
"id": 1,
"nombre": "Marianela Montalvo",
"clienteId": "123401",
"estado": true
}
}
}

F2: Registro de movimientos:

- Un movimiento se pueden tener valores positivos o negativos.
- Al realizar un movimiento se actualiza el saldo disponible.
- Se lleva el registro de las transacciones realizadas.

F3: Registro de movimientos: Al realizar un movimiento el cual no cuente con saldo, se alerta mediante el siguiente mensaje:

```
messagge.saldo.insuficiente = Saldo insuficiente para realizar el movimiento
```

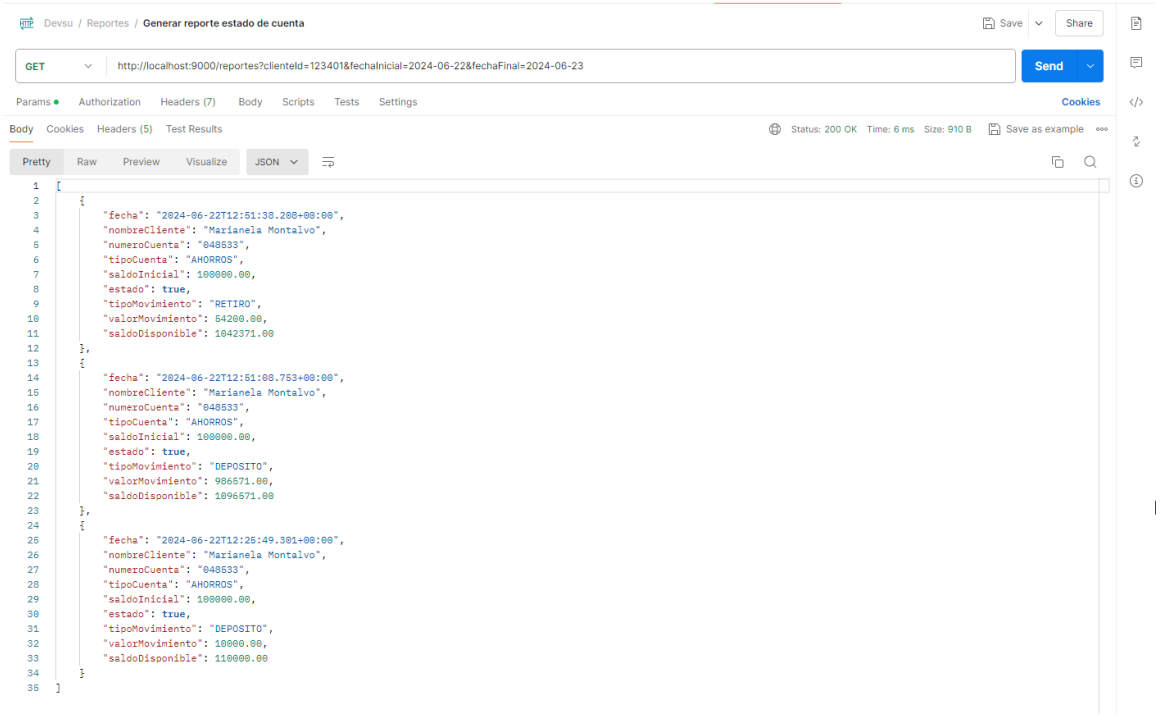
MovimientoServiceImpl.java

```
19  public class MovimientoServiceImpl implements MovimientoService {
34
35      @Override
36      public Movimiento createMovimiento(MovimientoDTO movimientoDTO) {
37          Cuenta cuenta = cuentaRepository.findById(movimientoDTO.getCuentaId())
38                  .orElseThrow(() -> new CuentaNotFoundException());
39
40          if (cuenta.getEstado()) {
41              if (cuenta.getSaldoDisponible().compareTo(BigDecimal.ZERO) > 0 ||
42                      TipoMovimiento.DEPOSITO.equals(TipoMovimiento.valueOf(movimientoDTO.getTipoMovimiento().toUpperCase()))) {
43                  Movimiento movimiento = new Movimiento();
44                  movimiento.setFecha(new Date());
45                  movimiento.setTipoMovimiento(TipoMovimiento.valueOf(movimientoDTO.getTipoMovimiento().toUpperCase()));
46                  movimiento.setValor(movimientoDTO.getValor());
47
48                  BigDecimal nuevoSaldo = cuenta.getSaldoDisponible();
49                  if (TipoMovimiento.DEPOSITO.equals(TipoMovimiento.valueOf(movimientoDTO.getTipoMovimiento().toUpperCase()))) {
50                      nuevoSaldo = nuevoSaldo.add(movimientoDTO.getValor());
51                  } else if (TipoMovimiento.RETIRO.equals(TipoMovimiento.valueOf(movimientoDTO.getTipoMovimiento().toUpperCase()))) {
52                      nuevoSaldo = nuevoSaldo.subtract(movimientoDTO.getValor());
53                  }
54
55                  cuenta.setSaldoDisponible(nuevoSaldo);
56                  cuenta = cuentaRepository.save(cuenta);
57
58                  movimiento.setSaldoDisponible(nuevoSaldo);
59                  movimiento.setCuenta(cuenta);
60                  return movimientoRepository.save(movimiento);
61              }
62              throw new SaldoInsuficienteException();
63          } else {
64              throw new CuentaDesabilitadaException();
65          }
66      }
67
68  }
69
```

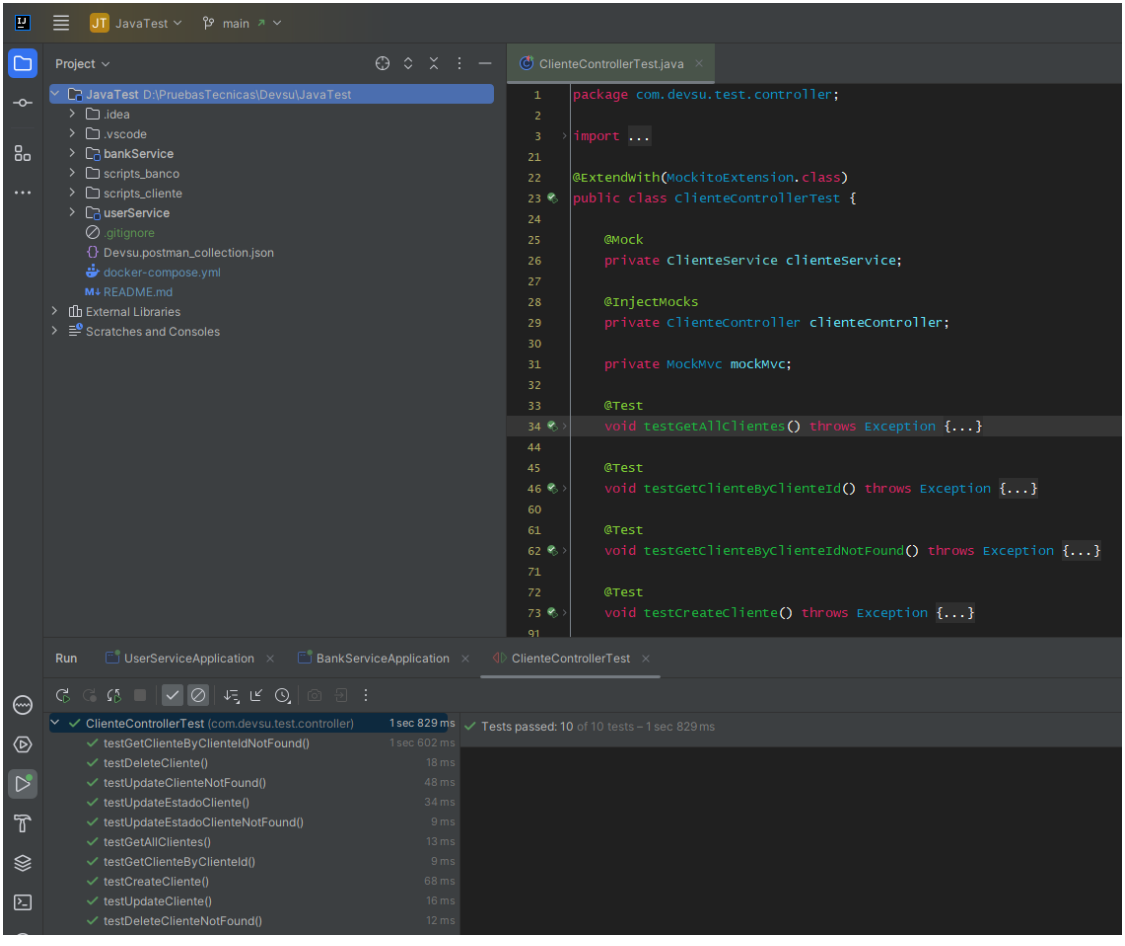
F4: Reportes: Generar un reporte de "Estado de cuenta" especificando un rango de fechas y cliente.

Este reporte debe contener:

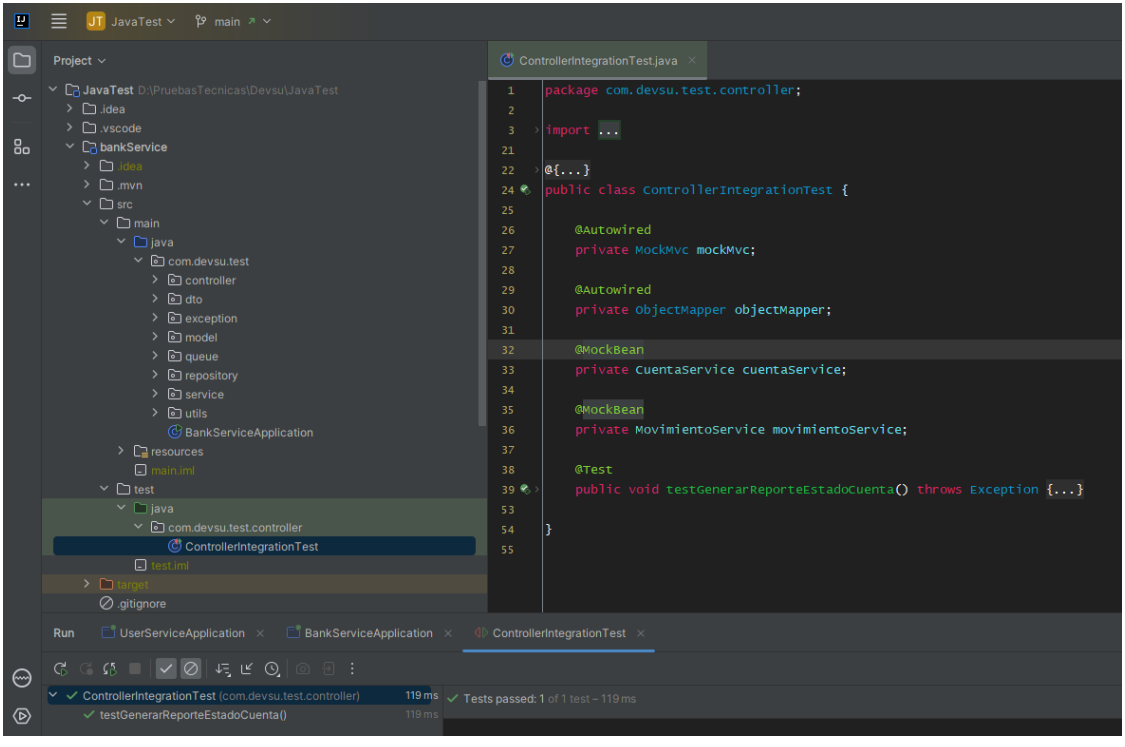
- Cuentas asociadas con sus respectivos saldos
- Detalle de movimientos de las cuentas
    - El endpoint que se debe utilizar para esto debe ser el siguiente: (/reportes?fecha=rango fechas & cliente)
    - El servicio del reporte retorna la información en formato JSON



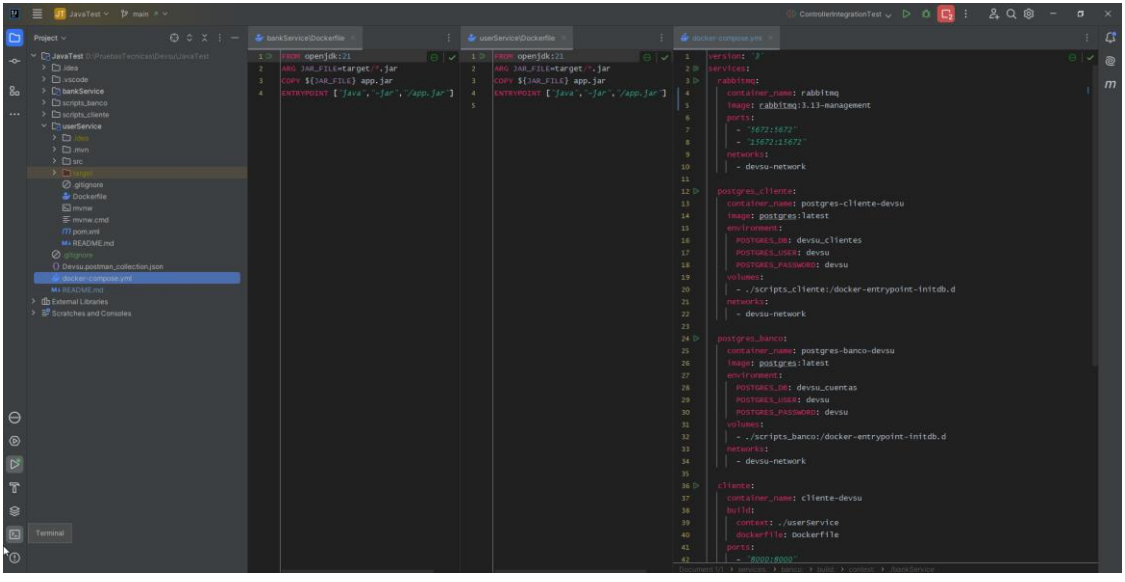
F5: Pruebas unitarias: Implementar 1 prueba unitaria para la entidad de dominio Cliente.

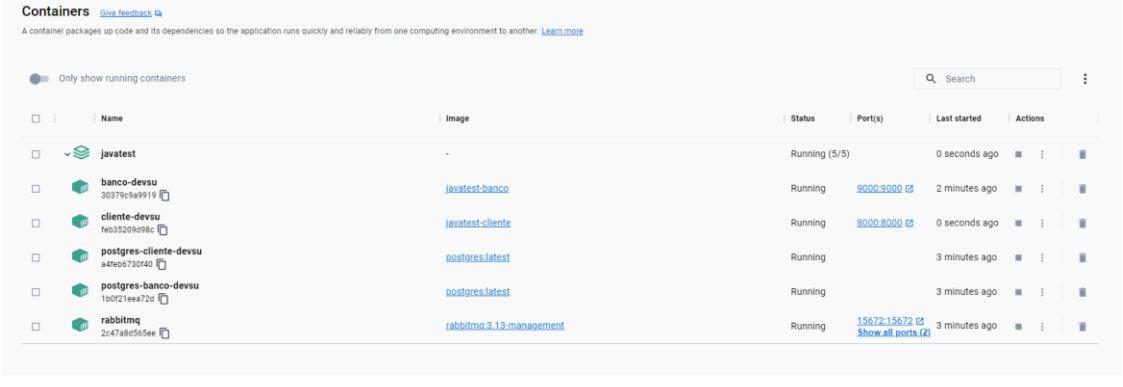


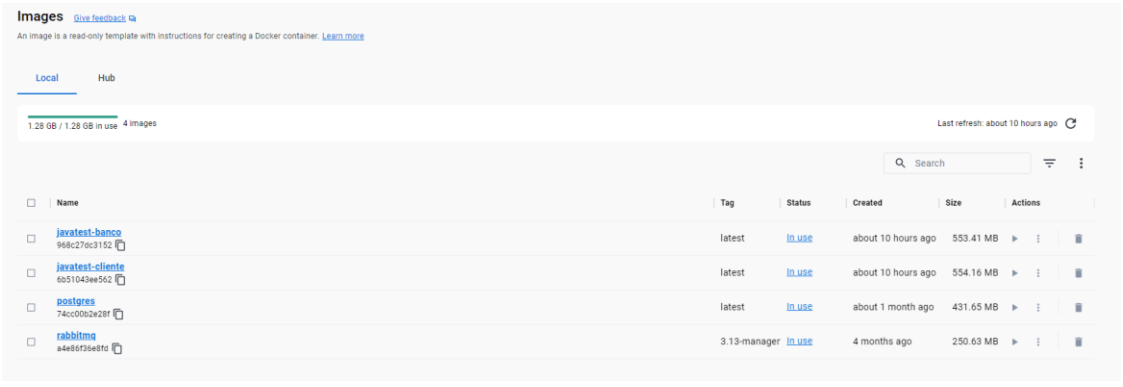
F6: Pruebas de Integración: Implementar 1 prueba de integración.



F7: Despliegue de la solución en contenedores.






Images
Give feedback
An image is a read-only template with instructions for creating a Docker container. Learn more
Local
Hub
1.28 GB / 1.28 GB in use
4 images
Last refresh: about 10 hours ago
Search
Name
Tag
Status
Created
Size
Actions
javatest-banco
968c27dc3152
latest
In use
about 10 hours ago
553.41 MB
javatest-cliente
6b51043ee562
latest
In use
about 10 hours ago
554.16 MB
postgres
74cc00b2e28f
latest
In use
about 1 month ago
431.65 MB
rabbitmq
a4e86f36e8fd
3.13-manager
In use
4 months ago
250.63 MB