

Prueba Técnica/Práctica

Herramientas y tecnologías utilizadas

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

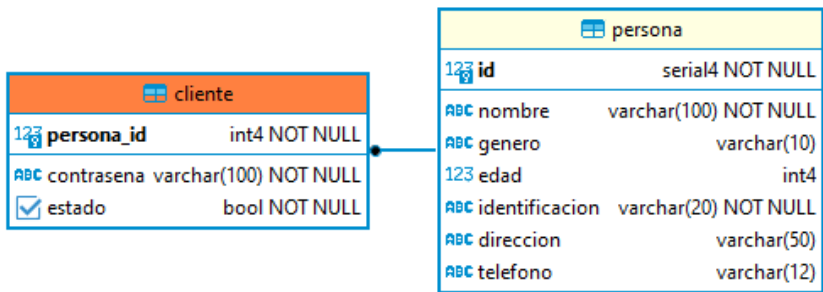
Requerimiento de Microservicios y Funcionalidades

1. Microservicio userService

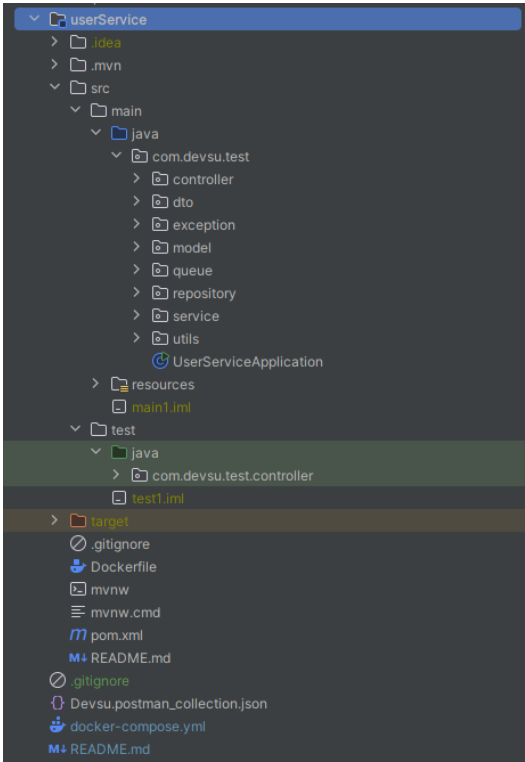
Este microservicio contiene la funcionalidad relacionada con los clientes. El servicio se enfoca a manejar toda las funcionalidades de clientes, a través de comunicación asíncrona mediante el uso de **rabbitMQ** el servicio publica en las colas de mensaje cuando una operación de inserción, actualización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



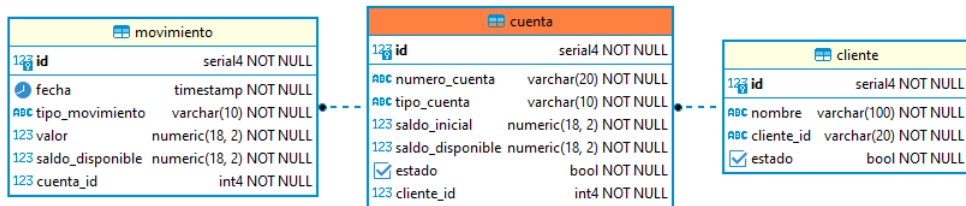
Estructura del proyecto



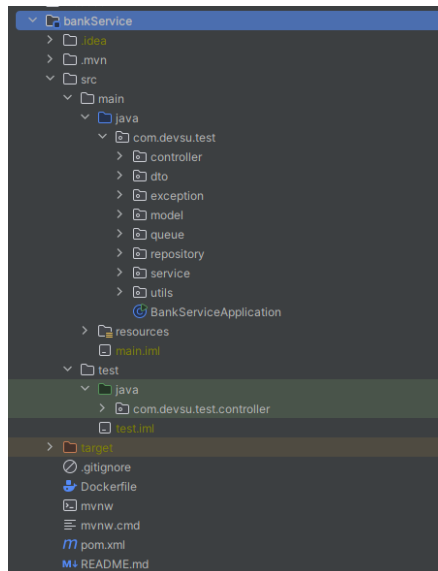
2. Microservicio bankService

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
1 package com.devsu.test.controller;
2
3 import ...
13
14 @RestController
15 @RequestMapping("/clientes")
16 @Validated
17 public class clienteController {
18
19     @Value("${message.cliente.delete}")
20     private String clienteBorrado;
21
22     @Value("${message.cliente.notfound}")
23     private String clienteNoEncontrado;
24
25     private final clienteService clienteService;
26
27     @Autowired
28     public clienteController(clienteService clienteService) { this.clienteService = clienteService; }
31
32     @GetMapping
33     public ResponseEntity<Iterable<cliente>> getAllClientes() {...}
36
37     @GetMapping("/{clienteId}")
38     public ResponseEntity<cliente> getClientesByClienteId(@PathVariable String clienteId) {...}
42
43     @PostMapping
44     public ResponseEntity<cliente> createCliente(@Valid @RequestBody cliente cliente) {...}
47
48     @PutMapping("/{id}")
49     public ResponseEntity<cliente> updateCliente(@PathVariable Long id, @RequestBody cliente clienteDetails) {...}
53
54     @PatchMapping("/{id}")
55     public ResponseEntity<cliente> updateEstadoCliente(@PathVariable Long id) {...}
59
60     @DeleteMapping("/{id}")
61     public ResponseEntity<ApiResponse> deleteCliente(@PathVariable Long id) {...}
```

1. Microservicio bankService

```
bankService\...\ClienteController.java x
1 package com.devsu.test.controller;
2
3 import ...
4
5
6
7
8
9
10
11 @RestController
12 @RequestMapping("/clientes")
13 @Validated
14 public class clienteController {
15
16     private final clienteService clienteService;
17
18     @Autowired
19     public clienteController(clienteService clienteService) { this.clienteService = clienteService; }
20
21
22
23     @GetMapping
24     public ResponseEntity<Iterable<cliente>> getAllClientes() {
25         return new ResponseEntity<>(clienteService.getAllClientes(), HttpStatus.OK);
26     }
27
28     @GetMapping("/{clienteId}")
29     public ResponseEntity<cliente> getClienteByClienteId(@PathVariable String clienteId) {
30         return clienteService.getClienteByClienteId(clienteId).map(ResponseEntity::ok)
31             .orElse(ResponseEntity.notFound().build());
32     }
33
34 }
```

```
bankService\...\ClienteController.java  bankService\...CuentaController.java x  bankService\...MovimientoController.java  bankService\...ReporteController.java
1 package com.devsu.test.controller;
2
3 import ...
4
5
6
7
8
9
10
11
12
13
14
15 @RestController
16 @RequestMapping("/cuentas")
17 @Validated
18 public class cuentaController {
19
20     private final cuentaService cuentaService;
21
22     @Autowired
23     public cuentaController(cuentaService cuentaService) { this.cuentaService = cuentaService; }
24
25
26
27     @GetMapping
28     public ResponseEntity<Iterable<cuenta>> getAllCuentas() {...}
29
30
31
32     @GetMapping("/{clienteId}")
33     public ResponseEntity<Iterable<cuenta>> getCuentasByClienteId(@PathVariable String clienteId) {...}
34
35
36
37     @PostMapping
38     public ResponseEntity<cuenta> crearCuenta(@Valid @RequestBody CuentaDTO cuentaDTO) {...}
39
40
41
42     @PatchMapping("/{id}")
43     public ResponseEntity<ApiResponse> cerrarCuenta(@PathVariable Long id) {...}
44
45
46
47
48
49
50 }
```

```
bankService... \ClienteController.java  CuentaController.java  MovimientoController.java  ReporteController.java
1 package com.devsu.test.controller;
2
3 import ...
12
13 @RestController
14 @RequestMapping("/movimientos")
15 @Validated
16 public class MovimientoController {
17
18     private final MovimientoService movimientoService;
19
20     @Autowired
21     public MovimientoController(MovimientoService movimientoService) { this.movimientoService = movimientoService; }
24
25     @PostMapping
26     public ResponseEntity<Movimiento> crearMovimiento(@Valid @RequestBody MovimientoDTO movimientoDTO) {
27         return new ResponseEntity<>(movimientoService.createMovimiento(movimientoDTO), HttpStatus.CREATED);
28     }
29
30 }
31
```

```
bankService... \ClienteController.java  CuentaController.java  MovimientoController.java  ReporteController.java
1 package com.devsu.test.controller;
2
3 import ...
13
14 @RestController
15 @RequestMapping("/reportes")
16 @Validated
17 public class ReporteController {
18
19     private final MovimientoService movimientoService;
20
21     @Autowired
22     public ReporteController(MovimientoService movimientoService) { this.movimientoService = movimientoService; }
25
26     @GetMapping
27     public ResponseEntity<Iterable<ReporteEstadoCuentaDTO>> generarReporteEstadoCuenta(
28         @RequestParam String clienteId,
29         @RequestParam @DateTimeFormat(pattern = "yyyy-MM-dd") Date fechaInicial,
30         @RequestParam @DateTimeFormat(pattern = "yyyy-MM-dd") Date fechaFinal) {
31         return new ResponseEntity<>(movimientoService.generarReporteEstadoCuenta(fechaInicial, fechaFinal, clienteId), HttpStatus.OK);
32     }
33
34 }
35
```

Los nombres de los endpoints a generar son:

- /clientes
- /cuentas
- /movimientos

Devsu / Clientes / Crear un Cliente

SaveShare

POSThttp://localhost:8000/clientes

Send

ParamsAuthorizationHeaders (9)BodyScriptsTestsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

Beautify

```
1 {
2   <!--"nombre": "Marianela Montalvo",
3   <!--"genero": "Femenino",
4   <!--"edad": 21,
5   <!--"identificacion": "123491",
6   <!--"direccion": "Amazonas y NNUU",
7   <!--"telefono": "997548965",
8   <!--"contrasena": "Test5678",
9   <!--"estado": true
10 }
11 }
```

BodyCookiesHeaders (5)Test Results

Status: 201 CreatedTime: 70 msSize: 427 BSave as example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "id": 3,
3   "nombre": "Marianela Montalvo",
4   "genero": "Femenino",
5   "edad": 21,
6   "identificacion": "123491",
7   "direccion": "Amazonas y NNUU",
8   "telefono": "997548965",
9   "contrasena": "$2a$10$heX1YkxevK/Dg0cMMoGn0gJRakwSR3C5o7jtydJzthMkTeJL316",
10  "estado": true,
11  "clienteId": "123491"
12 }
```

POST Crear un ClienteGET Consultar todos los ClientesPOST Crear una CuentaGET Consultar todas las CuentasPOST Crear un Movimiento

No environment

Devsu / Cuentas / Crear una Cuenta

SaveShare

POSThttp://localhost:9000/cuentas

Send

ParamsAuthorizationHeaders (9)BodyScriptsTestsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

Beautify

```
1 {
2   <!--"tipoCuenta": "AHORROS",
3   <!--"numeroCuenta": "948533",
4   <!--"saldoInicial": 100000.00,
5   <!--"estado": true,
6   <!--"clienteId": "123491"
7 }
8 }
```

BodyCookiesHeaders (5)Test Results

Status: 201 CreatedTime: 85 msSize: 375 BSave as example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "id": 1,
3   "numeroCuenta": "948533",
4   "tipoCuenta": "AHORROS",
5   "saldoInicial": 100000.00,
6   "saldoDisponible": 100000.00,
7   "estado": true,
8   "cliente": {
9     "id": 1,
10    "nombre": "Marianela Montalvo",
11    "clienteId": "123491",
12    "estado": true
13  }
14 }
```

POST Crear un ClienteGET Consultar todos los ClienPOST Crear una CuentaGET Consultar todas las CuePOST Crear un MovimientoGET Generar reporte estado

No environment

Devsu / Movimientos / Crear un Movimiento

SaveShare

POSThttp://localhost:9000/movimientosSend

ParamsAuthorizationHeaders (9)BodyScriptsTestsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

Beautify

```
1 {
2   ...."tipoMovimiento": "DEPOSITO",
3   ...."valor": 10000.00,
4   ...."cuentaId": 1
5 }
6
```

BodyCookiesHeaders (5)Test Results

Status: 201 CreatedTime: 32 msSize: 506 BSave as example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "id": 1,
3   "fecha": "2024-06-22T12:25:40.301+00:00",
4   "tipoMovimiento": "DEPOSITO",
5   "valor": 10000.00,
6   "saldoDisponible": 110000.00,
7   "cuenta": {
8     "id": 1,
9     "numeroCuenta": "0489533",
10    "tipoCuenta": "AHORROS",
11    "saldoInicial": 100000.00,
12    "saldoDisponible": 110000.00,
13    "estado": true,
14    "cliente": {
15      "id": 1,
16      "nombre": "Mariana Montalvo",
17      "clienteId": "123401",
18      "estado": true
19    }
20  }
21 }
```


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:

```
message.saldo.insuficiente = saldo insuficiente para realizar el movimiento
```

```
MovimientoServiceImpl.java x
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

Devsu / Reportes / Generar reporte estado de cuenta

SaveShare

GEThttp://localhost:9000/reportes?clienteld=123401&fechaInicial=2024-06-22&fechaFinal=2024-06-23Send

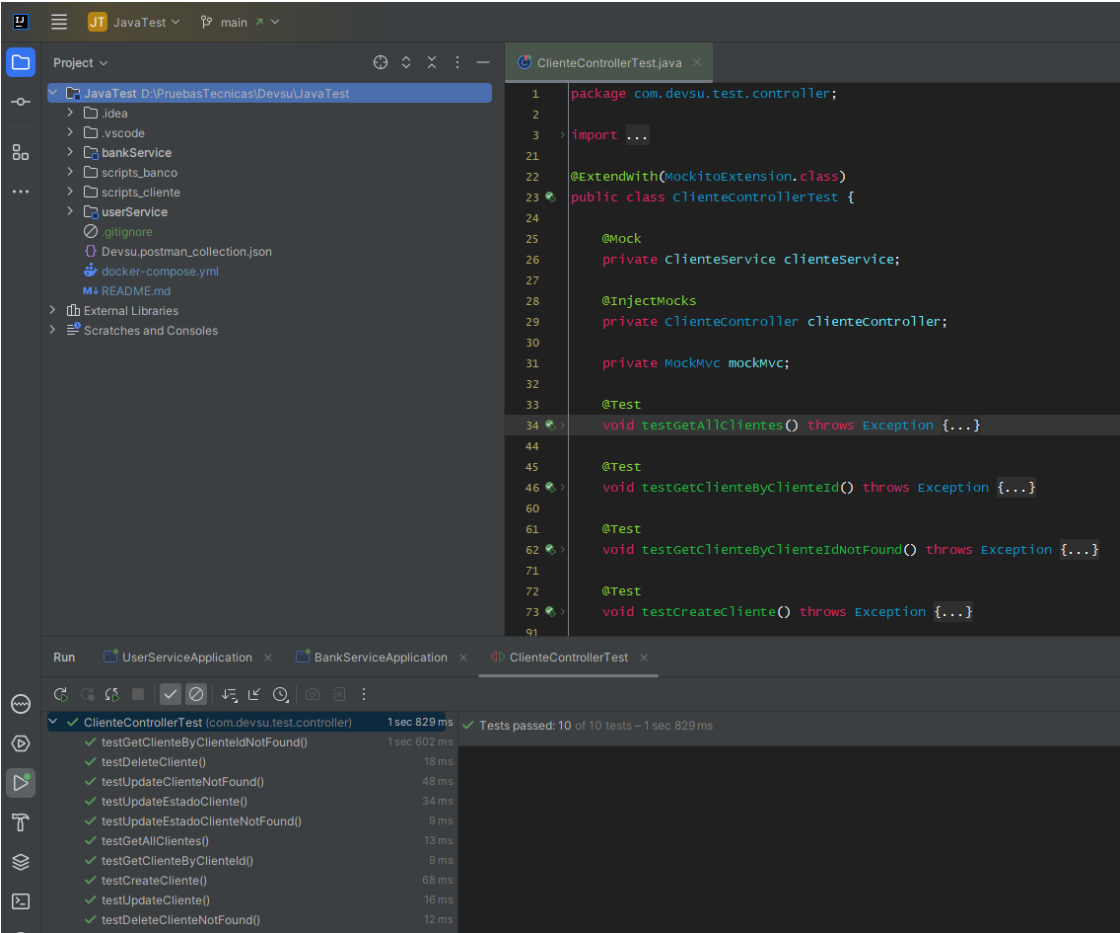
ParamsAuthorizationHeaders (7)BodyScriptsTestsSettingsCookies

BodyCookiesHeaders (5)Test ResultsStatus: 200 OKTime: 6 msSize: 910 BSave as example

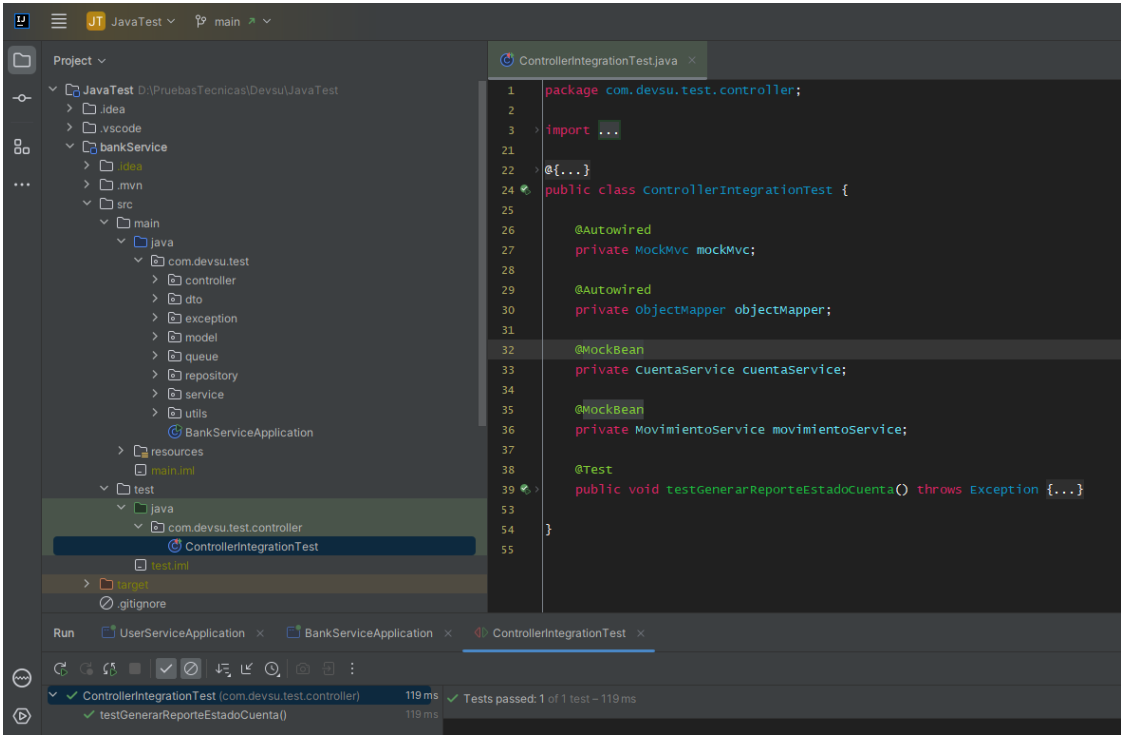
PrettyRawPreviewVisualizeJSON

```
1 {
2   {
3     "fecha": "2024-06-22T12:51:38.288+08:00",
4     "nombreCliente": "Marianela Montalvo",
5     "numeroCuenta": "048533",
6     "tipoCuenta": "AHORROS",
7     "saldoInicial": 100000.00,
8     "estado": true,
9     "tipoMovimiento": "RETIRO",
10    "valorMovimiento": 64200.00,
11    "saldoDisponible": 1042371.00
12  },
13  {
14    "fecha": "2024-06-22T12:51:08.763+08:00",
15    "nombreCliente": "Marianela Montalvo",
16    "numeroCuenta": "048533",
17    "tipoCuenta": "AHORROS",
18    "saldoInicial": 100000.00,
19    "estado": true,
20    "tipoMovimiento": "DEPOSITO",
21    "valorMovimiento": 986571.00,
22    "saldoDisponible": 1096571.00
23  },
24  {
25    "fecha": "2024-06-22T12:25:49.381+08:00",
26    "nombreCliente": "Marianela Montalvo",
27    "numeroCuenta": "048533",
28    "tipoCuenta": "AHORROS",
29    "saldoInicial": 100000.00,
30    "estado": true,
31    "tipoMovimiento": "DEPOSITO",
32    "valorMovimiento": 10000.00,
33    "saldoDisponible": 110000.00
34  }
35 }
```

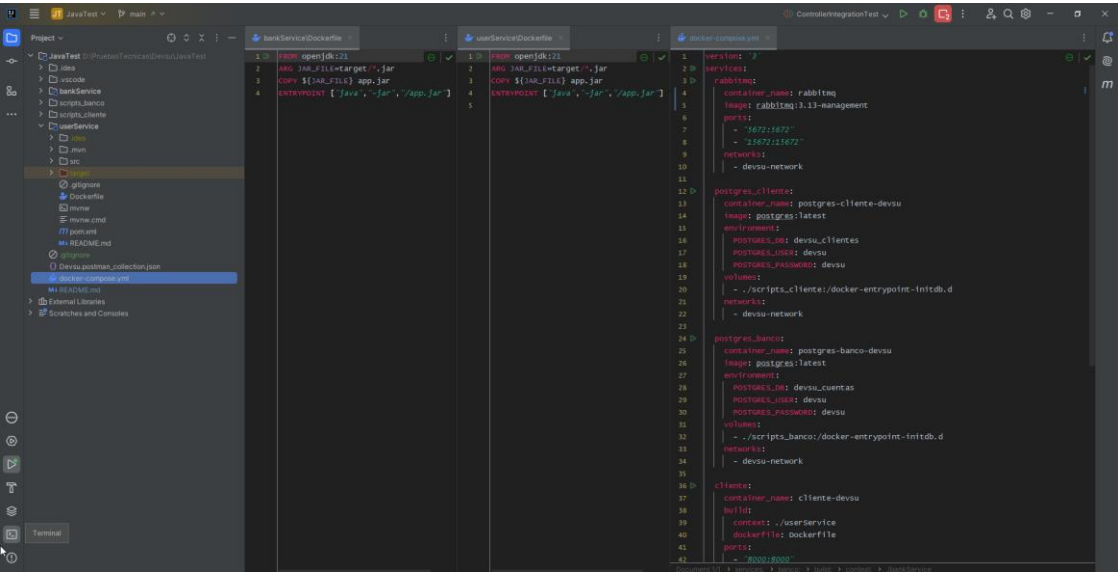
F5: Pruebas unitarias: Implementar 1 prueba unitaria para la entidad de dominio Cliente.



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



F7: Despliegue de la solución en contenedores.



Containers

[View feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

☒ Only show running containers

Search

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	Actions
<input checked="" type="checkbox"/>	javatest	-	Running (5/5)		0 seconds ago	
<input type="checkbox"/>	banco-devsu 30379c9a9919	javatest:banco	Running	9000:9000	2 minutes ago	
<input type="checkbox"/>	cliente-devsu feb33209096c	javatest:cliente	Running	8000:8000	0 seconds ago	
<input type="checkbox"/>	postgres-cliente-devsu a49e06730f40	postgres:latest	Running		3 minutes ago	
<input type="checkbox"/>	postgres-banco-devsu 1b0f21ee972d	postgres:latest	Running		3 minutes ago	
<input type="checkbox"/>	rabbitmq 2c47a8d555ee	rabbitmq:3.13-management	Running	15672:15672 Show all ports (2)	3 minutes ago	

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



<input type="checkbox"/>	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	jxratest-banco 968c270c3152	latest	in use	about 10 hours ago	553.41 MB	
<input type="checkbox"/>	jxratest-cliente 6051043ee562	latest	in use	about 10 hours ago	554.16 MB	
<input type="checkbox"/>	postgres 74cc00b2e28f	latest	in use	about 1 month ago	431.65 MB	
<input type="checkbox"/>	rabbitmq a4e8f39e8f6	3.13-manager	in use	4 months ago	250.63 MB	