

Tabla de contenidos

Architecture Design	1
Code design choices	2
Quality, maintainability and extensibility	3
API Design	3
Using semantic version to URI	3
Using entity-collection	3
Using master-slave design pattern accounts is slave of customers	3
Using command pattern	3
Problem understanding	4
Entities (E-R)	4
SonarQube report	5
API consume in postman	6
Gatling testing load and stress (performance in concurrent request)	12
PATH: http://localhost:8080/v1/customers/130303/retrieve-account GET	12
Load test 100 rps 1 node->	12
Stress test 3000 rps 1 node ->	14
PATH: http://localhost:8080/v1/transactions POST	18
Load test 100 tps 1 node ->	18
Stress test 1000 tps 1 node ->	19
Data in PostgreSQL	21
Data in h2 db file	24
TDD with BDD focus	26
Directory structure	27
Reactive programming	30

Architecture Design

The micro-services architecture is really important for applications, but I select a standalone application, because provides more facilities of run, test and show you the

requirements, with more than one components I need to share some code for the purpose of reduce duplicated code and more time to connect the services and the problem is easier than that.

Code design choices

I've use many tools and I'll describe it:

Java 11: this is the latest LTS, that means that the components are stable and long supported.

TDD with JUnit5: junit 5 is clear to use and has many features to allow better testing, with TDD I've used some tests to try quickly if my controllers and logic was working ok while I was programming.

WireMock: I've consumed an external service for currency rates information, this was mocked in tests for testing isolation.

Drools: the logic of the application can be change with configuration files that compiles at application startup, this a common feature to change the logic quickly even in prod environments

Change of currencies API: The API <https://api.exchangeratesapi.io/latest?base=USD> is obsolete and isn't working without api-key, so I've create one to use the newest, but the base option of the API is only available for paid users so, I don't use it, instead I've mock a similar one with postman.

WebFlux: Using a different approach to solve problems shows you my skill in learning, and this reactive programming is useful for applications with many concurrent requests with non-blocking connections. This is more difficult to implement because the information and the community can be larger but it's really profitable.

H2: this is require to execute in standalone mode

PostgreSQL: the application currently support postgres database with the same logic, it can be ran with **docker-compose**

Format google java code: I've use the formatter of google for java code, this is important in quality (on clean code terms) and readability

Logger: I've used the native logging tool and not the Apache Log4j only because are an included feature in spring boot

r2dbc: instead of jpa or jdbc, because they have support to webflux implementation

jacoco: to report coverage to sonarqube

I expose more resources and methods to the API and change one because on my design accounts with others Currencies are supported.

I've been using Custom error classes and I've a single class to handle it.

Quality, maintainability and extensibility

Quality was checked with sonarqube and IntelliJ Inspector tools, to prevent missing errors, the TDD testing using the behavior of interface is incredibly useful to ensure easy refactoring supporting the maintainability feature, and is so extensible because the Spring framework community has a lot of plugins to adapt the code to new challenges with few changes.

API Design

I've used some patterns to describe the work

Using semantic version to URI

Using base path indicating the major version, changing between APIs with URL using an API gateway:

/v1

Using entity-collection

Get Customer entity:

GET /v1/customers/{id} Customer

Get Customer collection:

GET /v1/customers

Create a transaction:

POST /v1/transactions

Using master-slave design pattern accounts is slave of customers

Get single account:

GET /v1/customers/{idCustomer}/accounts/{idAccount}

Get all accounts of customer:

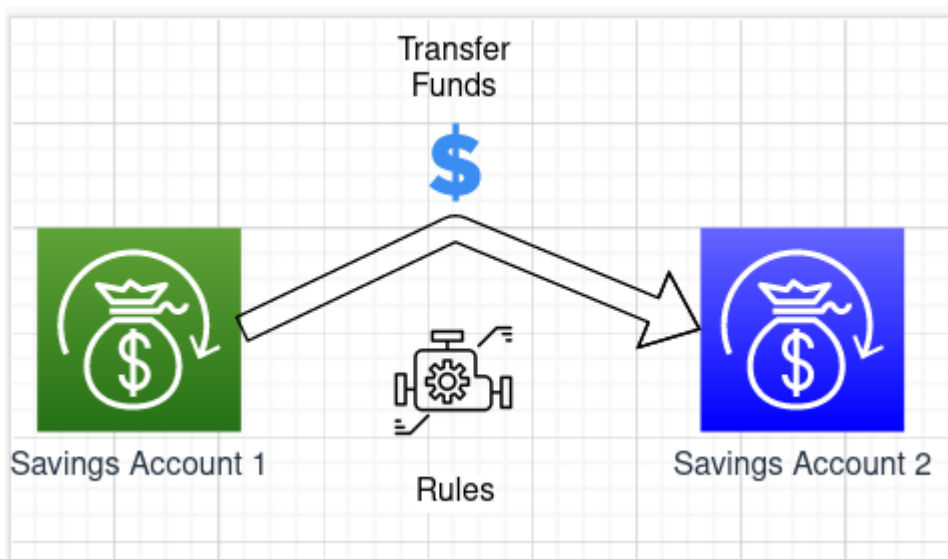
GET /v1/customers/{idCustomer}/accounts

Using command pattern

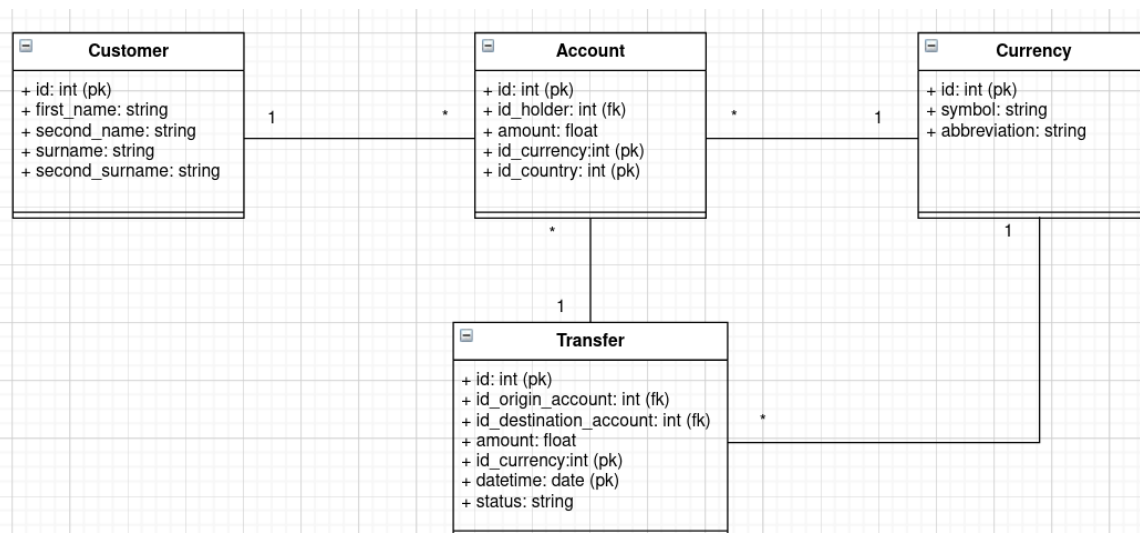
Get a single account with a specific message, the command pattern allows POST to retrieve information, this is used only because the challenge specifies this:

POST /v1/customers/130303/retrieve-account

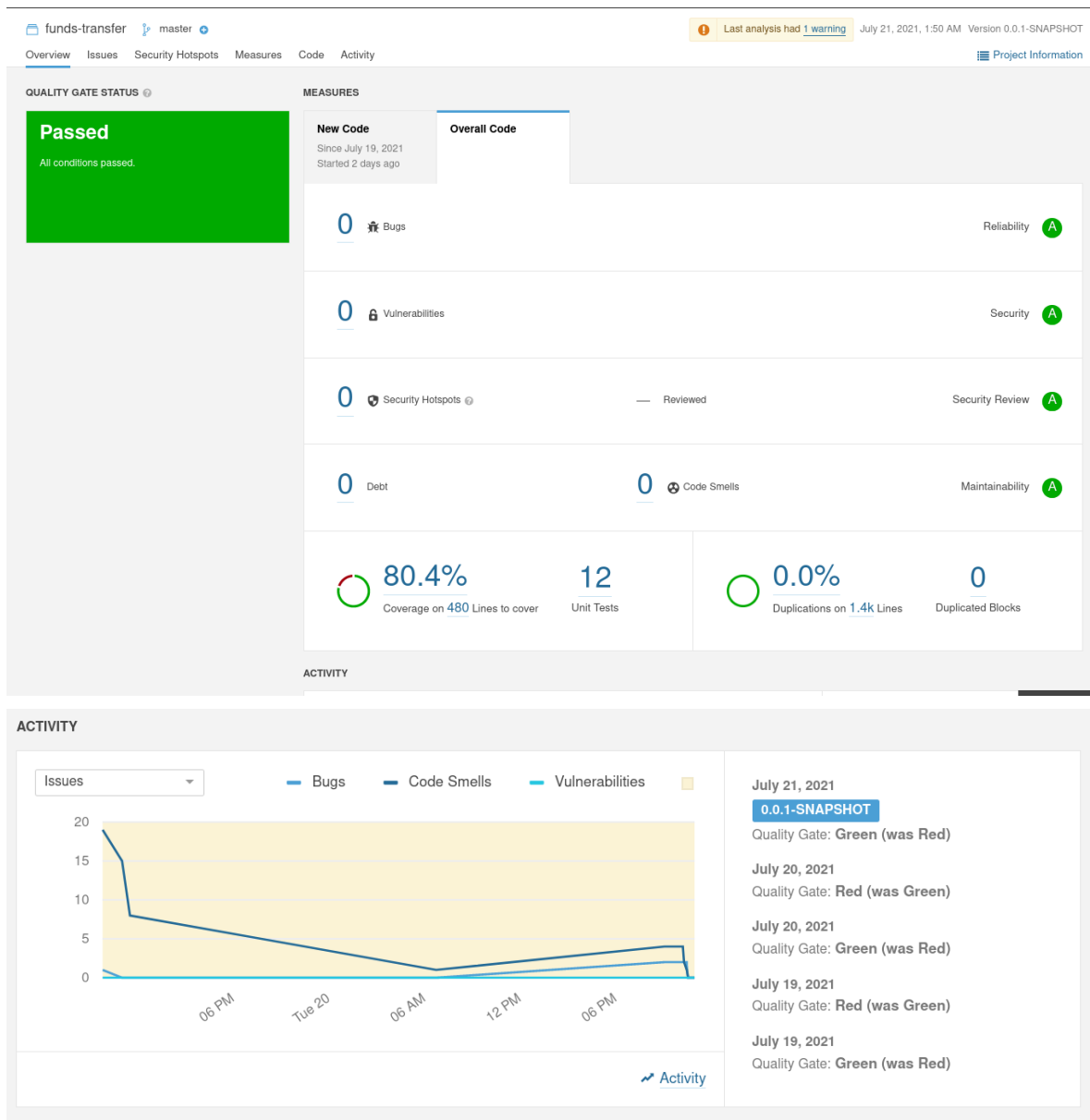
Problem understanding



Entities (E-R)



SonarQube report



API consume in postman

The screenshot displays the Postman application interface. At the top, a collection of requests is visible, including several 'GET ht...' and 'POST h...' requests. The active request is titled 'Untitled Request' and is a GET method to the URL 'http://localhost:8080/v1/customers/1'. The 'Send' button is highlighted in blue. Below the request details, the 'Params' tab is selected, showing a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The table contains one row with 'Key' and 'Value' in the first two columns, and 'Description' in the third. The 'Body' tab is also visible, showing a JSON response. The status bar at the bottom indicates a successful response with status '200 OK', time '279 ms', and size '262 B'. The response body is displayed in the 'Body' tab, showing a JSON object with fields 'id', 'surname', 'first_name', 'second_name', and 'second_surname'.

GET ht... GET ht... POST h... POST h... GET ht... GET C... D... GET ht... + ... Mi Mocksito

Untitled Request BUILD

GET http://localhost:8080/v1/customers/1 Send Save

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 279 ms Size: 262 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "surname": "Useche",
4   "first_name": "Jorge",
5   "second_name": "Ulises",
6   "second_surname": "Cuellar"
7 }
```


GET

{{url}}/currency-mock

Send

Save

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		


Pretty


Raw

Preview

Visualize

JSON





```
1 {
2   "success": true,
3   "timestamp": 1626582424,
4   "base": "USD",
5   "date": "2021-07-18",
6   "rates": {
7     "CAD": 1.26
8   }
9 }
```

POST

http://localhost:8080/v1/customers/130303/retrieve-account

Send

Save

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Code

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify


Pretty


Raw

Preview

Visualize

JSON





```
1 {
2   "account": "3"
3 }
```


Pretty


Raw

Preview

Visualize

JSON





```
1 {
2   "status": "OK",
3   "errors": [],
4   "currency": "CAD",
5   "account_balance": 2000
6 }
```


POST

http://localhost:8080/v1/customers/130303/retrieve-account

Send

Save

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Code

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

1 {

2 "account": "1"

3 }

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 14 ms

Size: 231 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1 {

2 "status": "OK",

3 "errors": [],

4 "currency": "USD",

5 "account_balance": 1000

6 }

POST http://localhost:8080/v1/transactions Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {
2   "amount": 100000,
3   "currency": "USD",
4   "origin_account": "5",
5   "destination_account": "6",
6   "description": "Hey dude! I am sending you the money you loaned to me lastweek."
7 }
```

Body Cookies Headers (5) Test Results Status: 412 Precondition Failed Time: 24 ms Size: 254 B Save Response

Pretty Raw Preview Visualize JSON Beautify

```
1 {
2   "status": "KO",
3   "errors": [
4     "insufficient-funds"
5   ],
6   "tax_collected": 0,
7   "CAD": 0
8 }
```

POST http://localhost:8080/v1/transactions Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {
2   "amount": 100,
3   "currency": "USD",
4   "origin_account": "1",
5   "destination_account": "2",
6   "description": "Hey dude! I am sending you the money you loaned to me lastweek."
7 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 35 ms Size: 236 B Save Response

Pretty Raw Preview Visualize JSON Beautify

```
1 {
2   "status": "OK",
3   "errors": [],
4   "tax_collected": 0.2,
5   "CAD": 0.7936507936507936
6 }
```

POST http://localhost:8080/v1/transactions Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {
2   "amount": 1,
3   "currency": "USD",
4   "origin_account": "5",
5   "destination_account": "6",
6   "description": "Hey dude! I am sending you the money you loaned to me lastweek."
7 }
```

Body Cookies Headers (5) Test Results 🌐 Status: 412 Precondition Failed Time: 49 ms Size: 250 B Save Response

Pretty Raw Preview Visualize JSON ↺ 📄 🔍

```
1 {
2   "status": "KO",
3   "errors": [
4     "limit_exceeded"
5   ],
6   "tax_collected": 0,
7   "CAD": 0
8 }
```

POST http://localhost:8080/v1/customers/130303/retrieve-account Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON Beautify

```
1 {
2   "account": "100"
3 }
```

Body Cookies Headers (5) Test Results 🌐 Status: 404 Not Found Time: 11 ms Size: 281 B Save Response

Pretty Raw Preview Visualize JSON ↺ 📄 🔍

```
1 {
2   "status": "KO",
3   "errors": [
4     "Account with id 100 not found in the database"
5   ],
6   "currency": null,
7   "account_balance": 0
8 }
```

Gatling testing load and stress (performance in concurrent request)

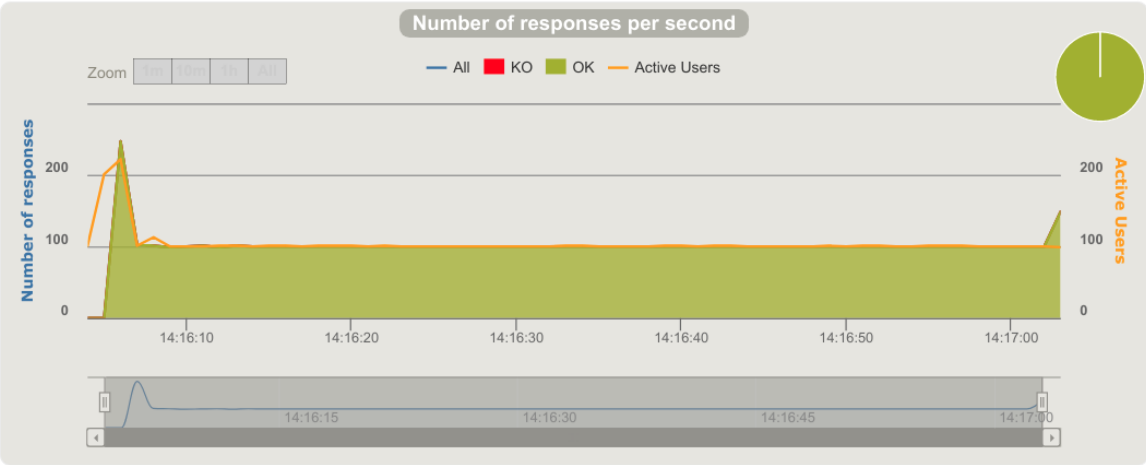
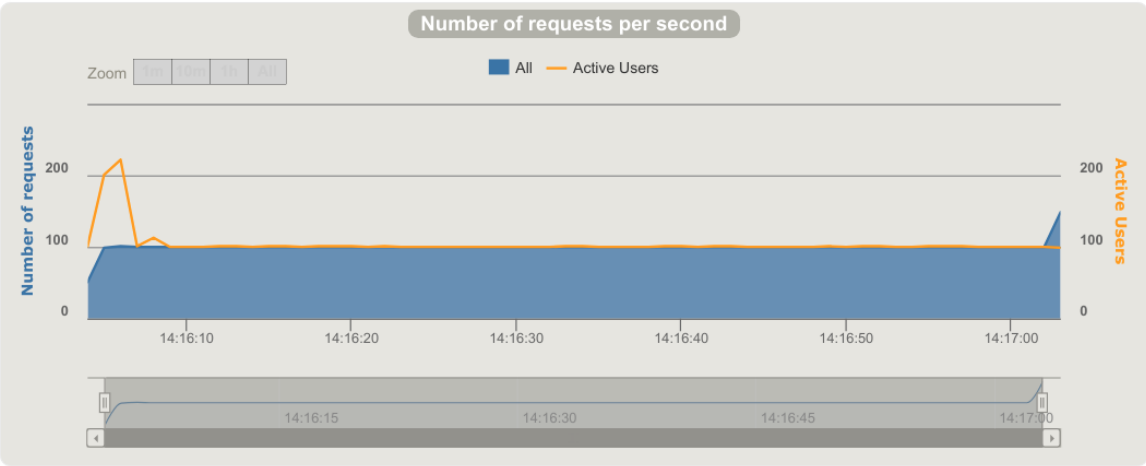
PATH:

<http://localhost:8080/v1/customers/130303/retrieve-account>

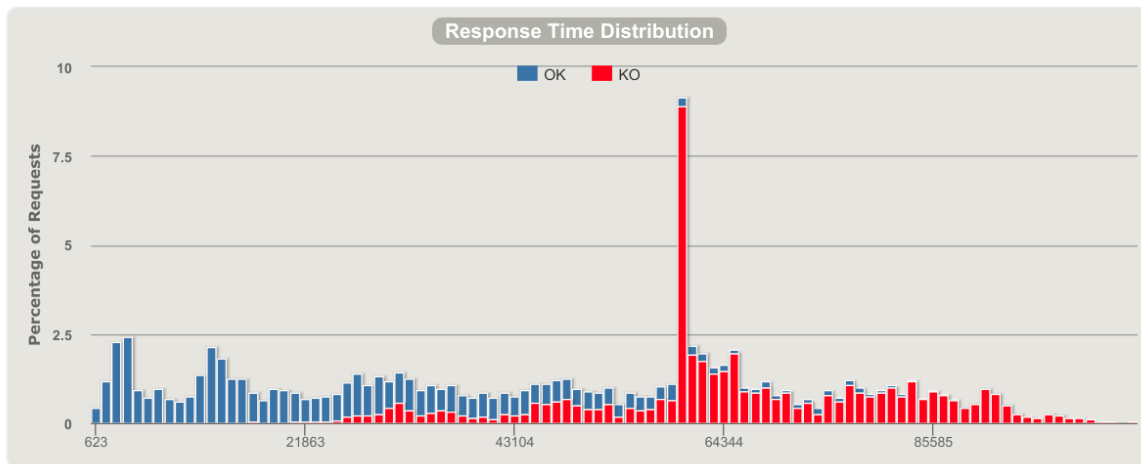
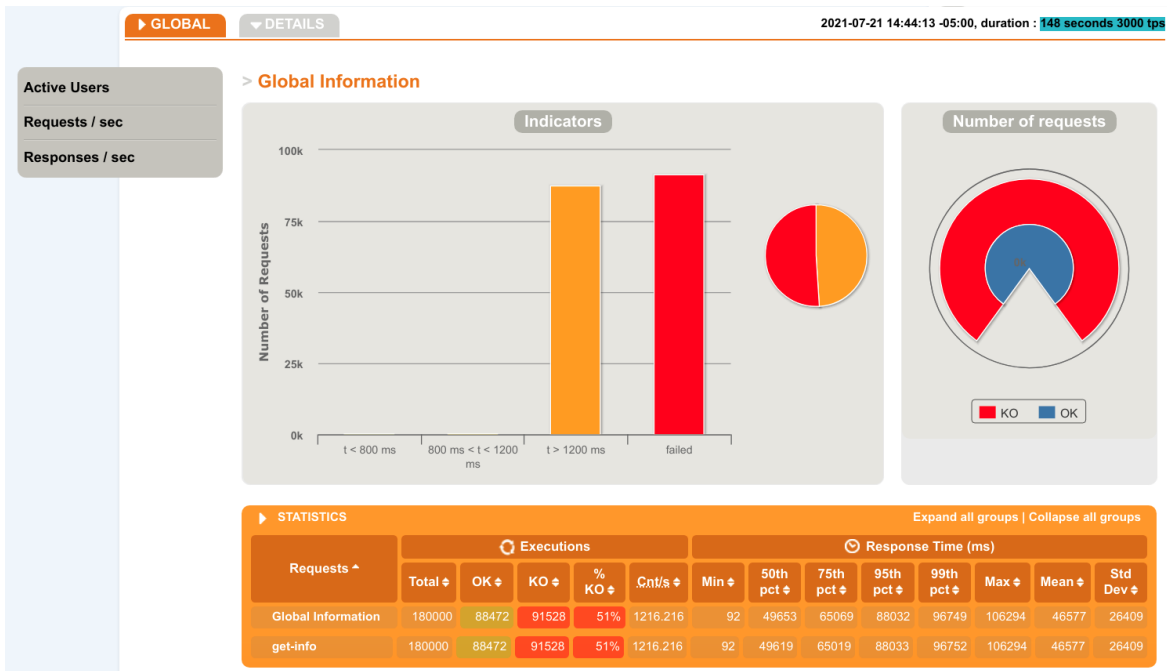
GET

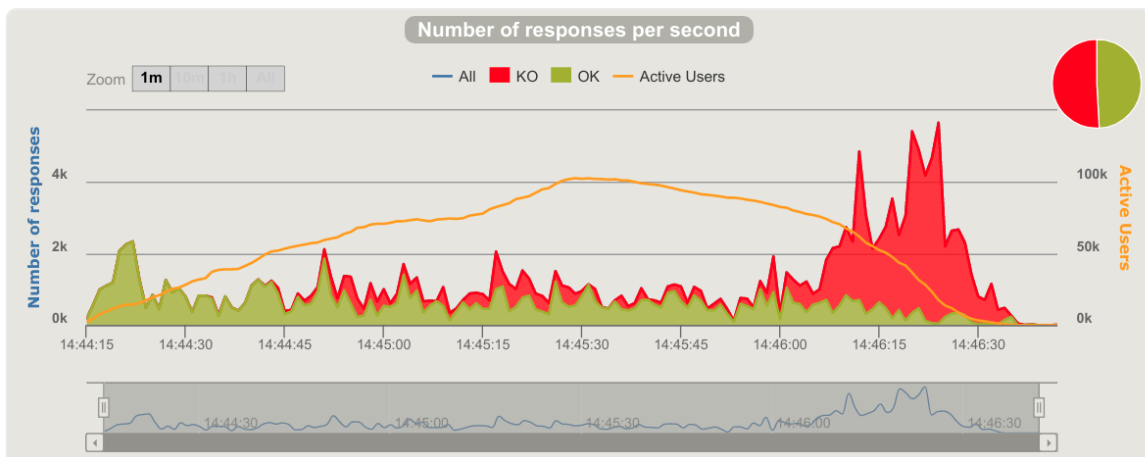
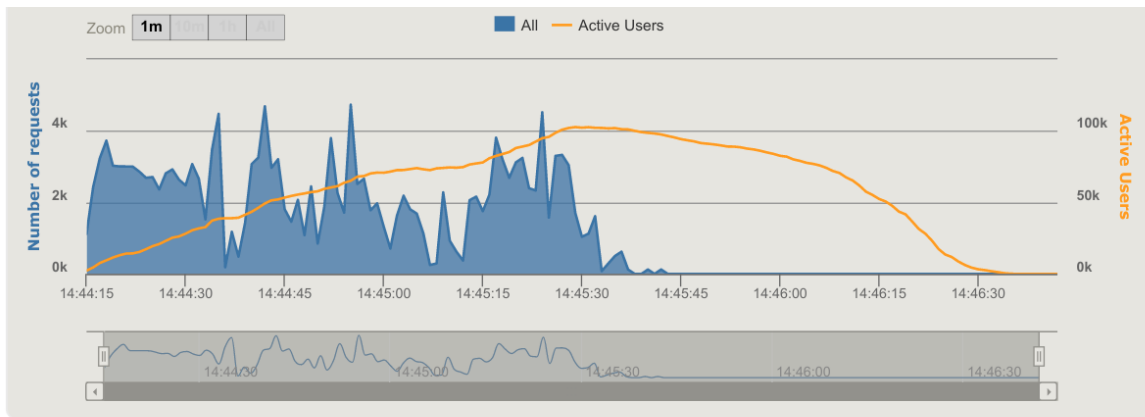
Load test 100 rps 1 node->

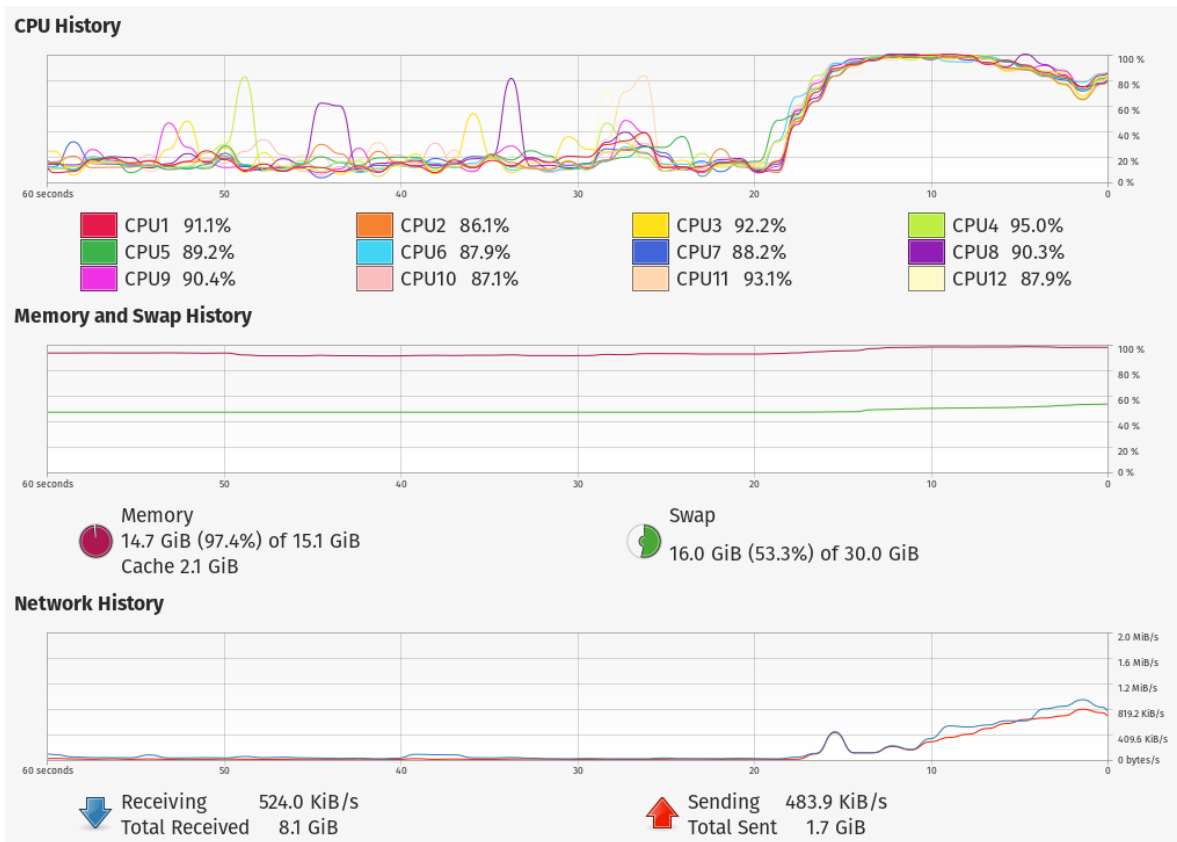




Stress test 3000 rps 1 node ->







More than request brokes the computer OS limits with Java problems like this

```
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid725443.hprof ...
Heap dump file created [1769915655 bytes in 76.692 secs]
```

Or this:


```

527
---- Response Time Distribution -----
> t < 800 ms                                494 (  0%)
> 800 ms < t < 1200 ms                      324 (  0%)
> t > 1200 ms                              87654 ( 49%)
> failed                                    91528 ( 51%)
---- Errors -----
> i.n.c.ConnectTimeoutException: connection timed out: localhost 31708 (34.64%)
/0:0:0:0:0:0:1:8080
> i.g.h.c.i.RequestTimeoutException: Request timeout after 60000 30015 (32.79%)
ms
> i.g.h.c.i.RequestTimeoutException: Request timeout to localhos 16341 (17.85%)
t/0:0:0:0:0:0:1:8080 after 60000 ms
> i.g.h.c.i.RequestTimeoutException: Request timeout to localhos 12829 (14.02%)
t/127.0.0.1:8080 after 60000 ms
> j.i.IOException: Premature close                                635 ( 0.69%)
=====

```

Or this:

```

as thrown by a user handler's exceptionCaught() method while handling the follow
ing exception:
java.lang.OutOfMemoryError: Java heap space
14:35:19.780 [WARN ] i.n.u.c.SingleThreadEventExecutor - An event executor termi
nated with non-empty task queue (33276)
14:35:19.780 [WARN ] i.n.u.c.SingleThreadEventExecutor - An event executor termi
nated with non-empty task queue (32613)
14:35:19.784 [WARN ] i.n.c.AbstractChannelHandlerContext - An exception 'java.la
ng.OutOfMemoryError: Java heap space' [enable DEBUG level for full stacktrace] w
as thrown by a user handler's exceptionCaught() method while handling the follow
ing exception:
java.lang.OutOfMemoryError: Java heap space
14:35:19.783 [WARN ] i.n.c.AbstractChannelHandlerContext - Failed to mark a prom
ise as failure because it has succeeded already: DefaultChannelPromise@3253ef2a(s
uccess)
java.lang.OutOfMemoryError: Java heap space
Exception in thread "gatling-1-14" java.lang.OutOfMemoryError: Java heap space
Exception in thread "gatling-1-24" java.lang.OutOfMemoryError: Java heap space

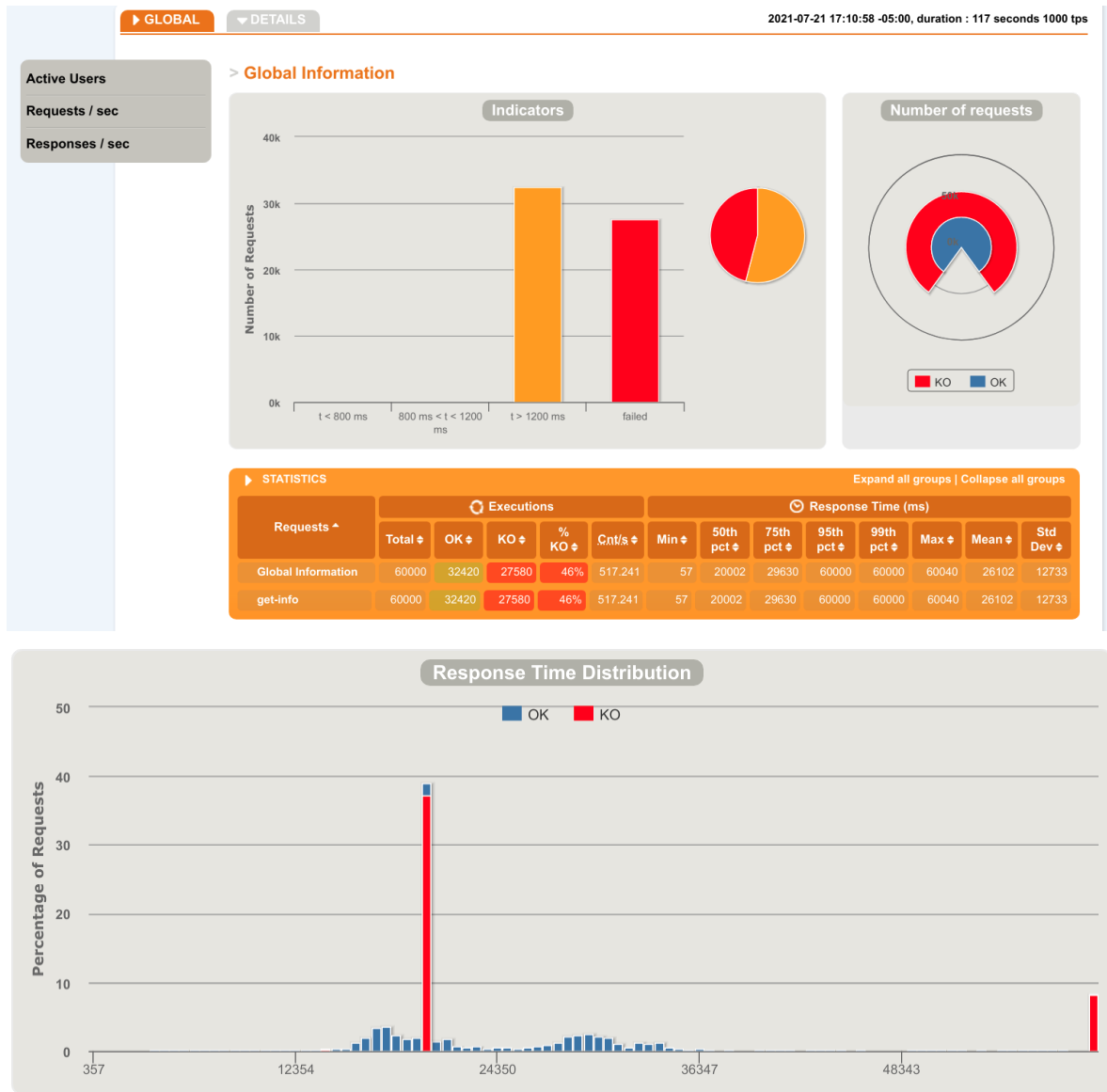
```

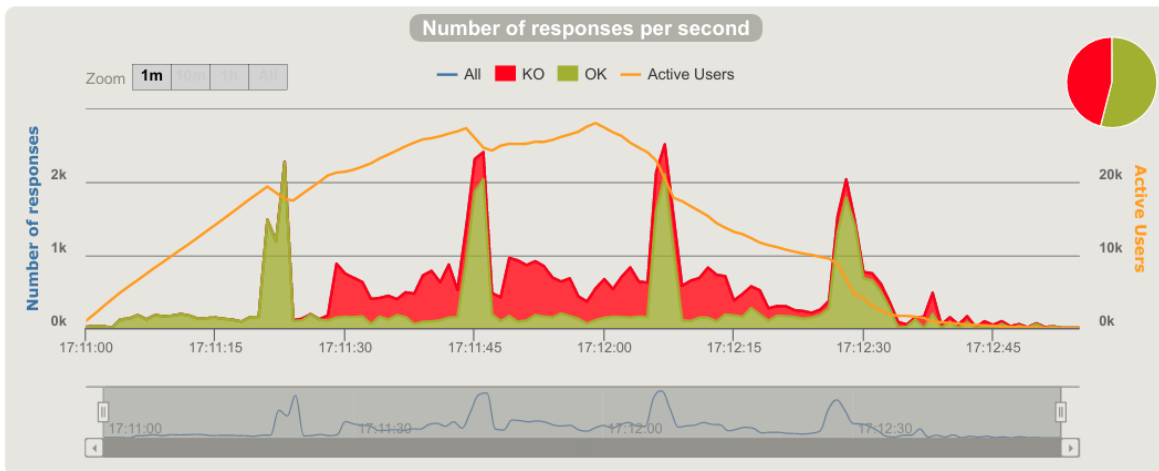
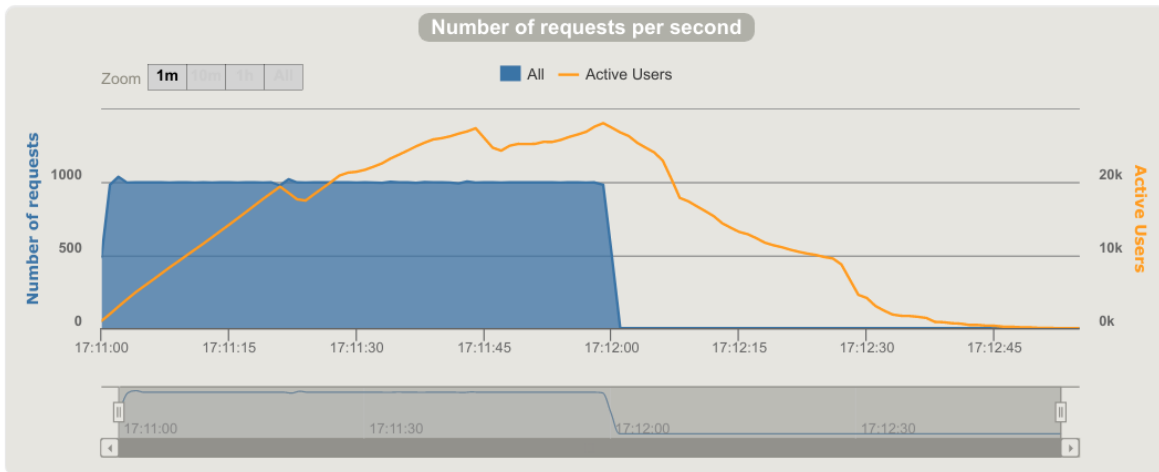
PATH: <http://localhost:8080/v1/transactions> POST

Load test 100 tps 1 node ->



Stress test 1000 tps 1 node ->





Data in PostgreSQL

The image displays two screenshots of PostgreSQL database management tools. The top screenshot shows the PostgreSQL Database Navigator interface, and the bottom screenshot shows the DBeaver 7.3.0 interface.

Top Screenshot: PostgreSQL Database Navigator

The left pane shows the database structure for 'yellowpepper' on 'localhost:5432'. The 'public' schema is selected, showing tables, views, materialized views, indexes, functions, sequences, data types, aggregate functions, topology, roles, administrator, extensions, and storage.

The right pane shows an ER Diagram with the following tables and their attributes:

- transfer**: id (PK), id_origin_account, id_destination_account, amount, tax, id_currency, datetime, status
- account**: id (PK), id_holder, amount, id_currency
- customer**: id (PK), first_name, second_name, surname, second_surname
- currency**: id (PK), symbol, abbreviation
- spatial_ref_sys**: srid (PK), auth_name, auth_srid, srtext
- geography_columns**: f_table_catalog, f_table_schema, f_table_name, f_geometry_column, coord_dimension
- geometry_columns**: f_table_catalog, f_table_schema, f_table_name, f_geometry_column, coord_dimension
- raster_columns**: r_table_catalog, r_table_schema, r_table_name, r_raster_column, srid, scale_x, scale_y, blocksize_x, blocksize_y, same_alignment
- raster_overviews**: o_table_catalog, o_table_schema, o_table_name, o_raster_column, r_table_catalog, r_table_schema, r_table_name

Bottom Screenshot: DBeaver 7.3.0 - account

The left pane shows the database structure for 'yellowpepper' on 'localhost:5432'. The 'public' schema is selected, showing tables, views, materialized views, indexes, functions, sequences, data types, topology, roles, administrator, extensions, and storage.

The right pane shows the 'account' table data in a grid view. The data is as follows:

id	id_holder	amount	id_currency
1	1	1,000	1
2	2	0	2
3	3	2,000	2
4	4	0	1
5	5	1,500.5	2
6	6	12.5	2
7	7	10.99	1
8	8	0.5	1

The bottom pane shows the 'Project - General' tab with the following fields: Name, DataSource, Bookmark, ER Diagram, and Scripts.

File Edit Navigate Search SQL Editor Database Window Help

Commit Rollback Auto yellowpapper public@y

Database Navig Projects

Enter a part of table name here

yellowpapper - localhost:5432

- yellowpapper
 - Schemas
 - information_schema
 - pg_catalog
 - public
 - Tables
 - account
 - currency
 - customer
 - spatial_ref_sys
 - transfer
 - Views
 - Materialized Views
 - Indexes
 - Functions
 - Sequences
 - Data types
 - Aggregate functions

<yellowpapper> Script-3 transfer account currency

Properties Data ER Diagram

currency Enter a SQL expression to filter results (use Ctrl+Space)

	id	symbol	abbreviation
1	1	\$	USD
2	2	\$	CAD
3	3	\$	COP
4	4	€	EUR
5	5	R\$	BRL

Save Cancel Script

Project - General #emntv

File Edit Navigate Search SQL Editor Database Window Help

Commit Rollback Auto yellowpapper public@yellowpapper

Database Navig Projects

Enter a part of table name here

yellowpapper - localhost:5432

- yellowpapper
 - Schemas
 - information_schema
 - pg_catalog
 - public
 - Tables
 - account
 - currency
 - customer
 - spatial_ref_sys
 - transfer
 - Views
 - Materialized Views
 - Indexes
 - Functions
 - Sequences
 - Data types
 - Aggregate functions

<yellowpapper> Script-3 transfer account currency customer

Properties Data ER Diagram

customer Enter a SQL expression to filter results (use Ctrl+Space)

	id	first_name	second_name	surname	second_surname
1	1	Jorge	Ulises	Useche	Cuellar
2	2	John		Doe	
3	3	Jane		Doe	
4	4	Simón	José Antonio de la Santísima Trinidad	Bolívar	y Ponte Palacios y Blanco
5	5	Jonny		Deep	

Save Cancel Script

Project - General #empty

Database Navig Projects Commit Rollback Auto yellowpepper public@yellowpepper

Properties Data ER Diagram

Enter a part of table name here

yellowpepper - localhost:5432

- yellowpepper
 - Schemas
 - information_schema
 - pg_catalog
 - public
 - Tables
 - account
 - currency
 - customer
 - spatial_ref_sys
 - transfer
 - Views
 - Materialized Views
 - Indexes
 - Functions
 - Sequences
 - Data types
 - Aggregate functions

transfer

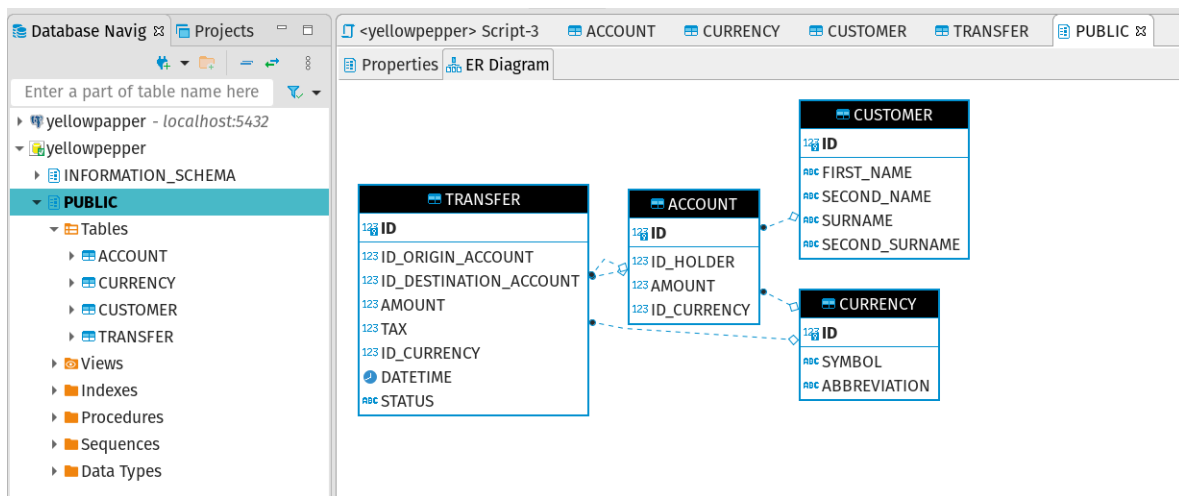
	id	id_origin_account	id_destination_account	amount	tax	id_currency	datetime	status
1	1	1	2	100	0.2	1	1-07-21 03:10:00	DONE
2	2	1	2	10,000	[NULL]	1	1-07-21 03:10:19	INSUFFICIENT_FUNDS
3	3	1	2	80	0.2	1	1-07-21 03:10:28	DONE
4	4	1	2	180	0.5	1	1-07-21 03:10:32	DONE
5	5	1	2	150	[NULL]	1	1-07-21 03:10:38	LIMITS_EXCEED

Save Cancel Script

200 5 Rows: 1

```
jorge@ulises:~/workspace/YellowPepper$ docker-compose up
Creating network "yellowpepper_docker-net" with driver "bridge"
Creating postgres ... done
Creating pgadmin4 ... done
Attaching to pgadmin4, postgres
postgres | Add rule to pg_hba: 0.0.0.0/0
postgres | Add rule to pg_hba: replication replicator
postgres | Setup master database
postgres | 2021-07-21 08:07:02.644 UTC [26] LOG:  starting PostgreSQL 12.2 (Debian 12.2-2.p
postgres | 2021-07-21 08:07:02.644 UTC [26] LOG:  listening on IPv4 address "127.0.0.1", por
postgres | 2021-07-21 08:07:02.646 UTC [26] LOG:  listening on Unix socket "/var/run/postgre
postgres | 2021-07-21 08:07:02.661 UTC [37] LOG:  database system was interrupted; last know
postgres | 2021-07-21 08:07:02.661 UTC [38] postgres@postgres FATAL:  the database system is
postgres | psql: error: could not connect to server: FATAL:  the database system is starting
postgres | 2021-07-21 08:07:02.767 UTC [37] LOG:  database system was not properly shut down
postgres | 2021-07-21 08:07:02.769 UTC [37] LOG:  redo starts at 0/42A107E8
```

Data in h2 db file



The screenshot shows the SQL Developer interface. On the left, the Database Navigator displays the schema structure for 'yellowpepper' on 'localhost:5432'. The 'PUBLIC' schema is expanded, showing tables: ACCOUNT, CURRENCY, CUSTOMER, and TRANSFER. The 'ACCOUNT' table is selected. On the right, the Data Grid shows the contents of the 'ACCOUNT' table. The grid has columns: ID, ID_HOLDER, AMOUNT, and ID_CURRENCY. The data is as follows:

ID	ID_HOLDER	AMOUNT	ID_CURRENCY
1	1	1,000	1
2	2	0	2
3	3	2,000	2
4	4	0	1
5	5	1,500.5	2
6	6	12.5	2
7	7	10.9241917921	1
8	8	3.5	1

Database Navig Projects

Enter a part of table name here

- yellowpapper - localhost:5432
 - yellowpepper
 - INFORMATION_SCHEMA
 - PUBLIC
 - Tables
 - ACCOUNT
 - CURRENCY
 - CUSTOMER
 - TRANSFER
 - Views
 - Indexes
 - Procedures
 - Sequences
 - Data Types

<yellowpepper> Script-3 ACCOUNT CURRENCY

Properties Data ER Diagram

CURRENCY Enter a SQL expression to filter results (use Ctrl+Space)

ID	SYMBOL	ABBREVIATION
1	\$	USD
2	\$	CAD
3	\$	COP
4	€	EUR
5	R\$	BRL

Database Navig Projects

Enter a part of table name here

- yellowpapper - localhost:5432
 - yellowpepper
 - INFORMATION_SCHEMA
 - PUBLIC
 - Tables
 - ACCOUNT
 - CURRENCY
 - CUSTOMER
 - TRANSFER
 - Views
 - Indexes
 - Procedures
 - Sequences
 - Data Types

<yellowpepper> Script-3 ACCOUNT CURRENCY CUSTOMER

Properties Data ER Diagram

CUSTOMER Enter a SQL expression to filter results (use Ctrl+Space)

ID	FIRST_NAME	SECOND_NAME	SURNAME	SECOND_SURNAME
1	Jorge	Ulises	Useche	Cuellar
2	John		Doe	
3	Jane		Doe	
4	Simón	José Antonio de la Santísima Trinidad	Bolívar	y Ponte Palacios y Blanco
5	Jonny		Deep	

Save Cancel Script 200 5 Rows: 1

Database Navig | Projects | <yellowpepper> Script-3 | ACCOUNT | CURRENCY | CUSTOMER | TRANSFER

Enter a part of table name here

yellowpepper - localhost:5432

- yellowpepper
 - INFORMATION_SCHEMA
 - PUBLIC
 - Tables
 - ACCOUNT
 - CURRENCY
 - CUSTOMER
 - TRANSFER
 - Views
 - Indexes
 - Procedures
 - Sequences
 - Data Types

TRANSFER

Enter a SQL expression to filter results (use Ctrl+Space)

ID	ID_ORIGIN_ACCOUNT	ID_DESTINATION_ACCOUNT	AMOUNT	TAX	ID_CURRENCY	DATETIME	STATUS
1	7	8	150	[NULL]	1	2021-07-21 03:15:02	INSUFFICIENT_FUNDS
2	7	8	160	[NULL]	1	2021-07-21 03:15:14	INSUFFICIENT_FUNDS
3	8	7	160	[NULL]	1	2021-07-21 03:15:22	INSUFFICIENT_FUNDS
4	7	8	1	0.2	1	2021-07-21 03:15:35	DONE
5	7	8	1	0.2	1	2021-07-21 03:15:38	DONE
6	7	8	1	0.2	1	2021-07-21 03:15:40	DONE
7	7	8	1	[NULL]	1	2021-07-21 03:15:42	LIMITS_EXCEED

Save Cancel Script

200 7 Rows: 1 7 row(s)

TDD with BDD focus

```

42  @Test
43  @DisplayName(
44      "Given valid accounts and valid parameters "
45      + "When user do a transaction with amount lower or equals to 100 "
46      + "Then the response is OK With 0.2 Tax ")
47  void test1() throws Exception {
48      ObjectNode transaction = getTransaction( amount: 100.0, currency: "USD", originAccount: 3, destinationAccount: 4, description: "Transferring across accounts");
49
50      HttpEntity<Object> entity = new HttpEntity<>(transaction);
51      ResponseEntity<ObjectNode> result =
52          testRestTemplate.exchange(TRANSACTIONS_ENDPOINT, HttpMethod.POST, entity, ObjectNode.class);
53      ObjectNode body = result.getBody();
54
55      assertEquals( expected: 200, result.getStatusCode().value());
56      assertEquals( expected: "OK", body.get("status").asText());
57      assertEquals( expected: 0, body.withArray( propertyName: "errors").size());
58      assertEquals( expected: 0.2, body.get("tax_collected").asDouble());
59      assertTrue(body.get("CAD").isDouble());
60  }

```

```

@Test
@DisplayName(
    "Given valid accounts and valid parameters and sufficient amounts "
    + "When the user do a transaction 4 times "
    + "Then 4th time gives a limit error ")
void test4() throws Exception {
    ObjectNode transaction =
        getTransaction(
            amount: 1.0, currency: "USD", originAccount: 5, destinationAccount: 6, description: "Hey dude! I am sendi

    HttpEntity<Object> entity = new HttpEntity<>(transaction);
    ResponseEntity<ObjectNode> tx1 =
        testRestTemplate.exchange(TRANSACTIONS_ENDPOINT, HttpMethod.POST, entity, ObjectNode.class);
    assertEquals( expected: 200, tx1.getStatusCode().value());

    ResponseEntity<ObjectNode> tx2 =
        testRestTemplate.exchange(TRANSACTIONS_ENDPOINT, HttpMethod.POST, entity, ObjectNode.class);
    assertEquals( expected: 200, tx2.getStatusCode().value());

    ResponseEntity<ObjectNode> tx3 =
        testRestTemplate.exchange(TRANSACTIONS_ENDPOINT, HttpMethod.POST, entity, ObjectNode.class);
    assertEquals( expected: 200, tx3.getStatusCode().value());

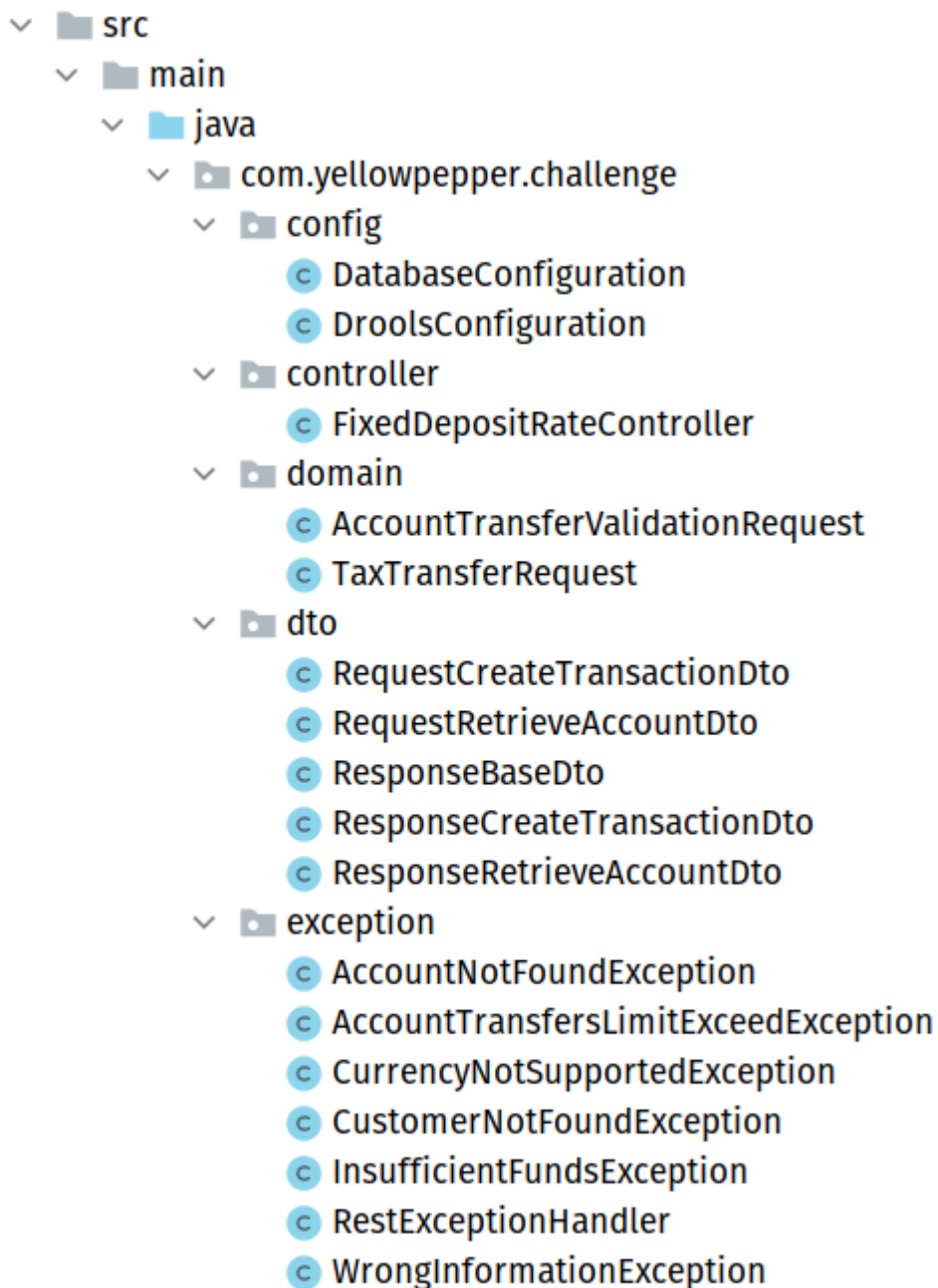
    ResponseEntity<ObjectNode> tx4 =
        testRestTemplate.exchange(TRANSACTIONS_ENDPOINT, HttpMethod.POST, entity, ObjectNode.class);

    ObjectNode body = tx4.getBody();

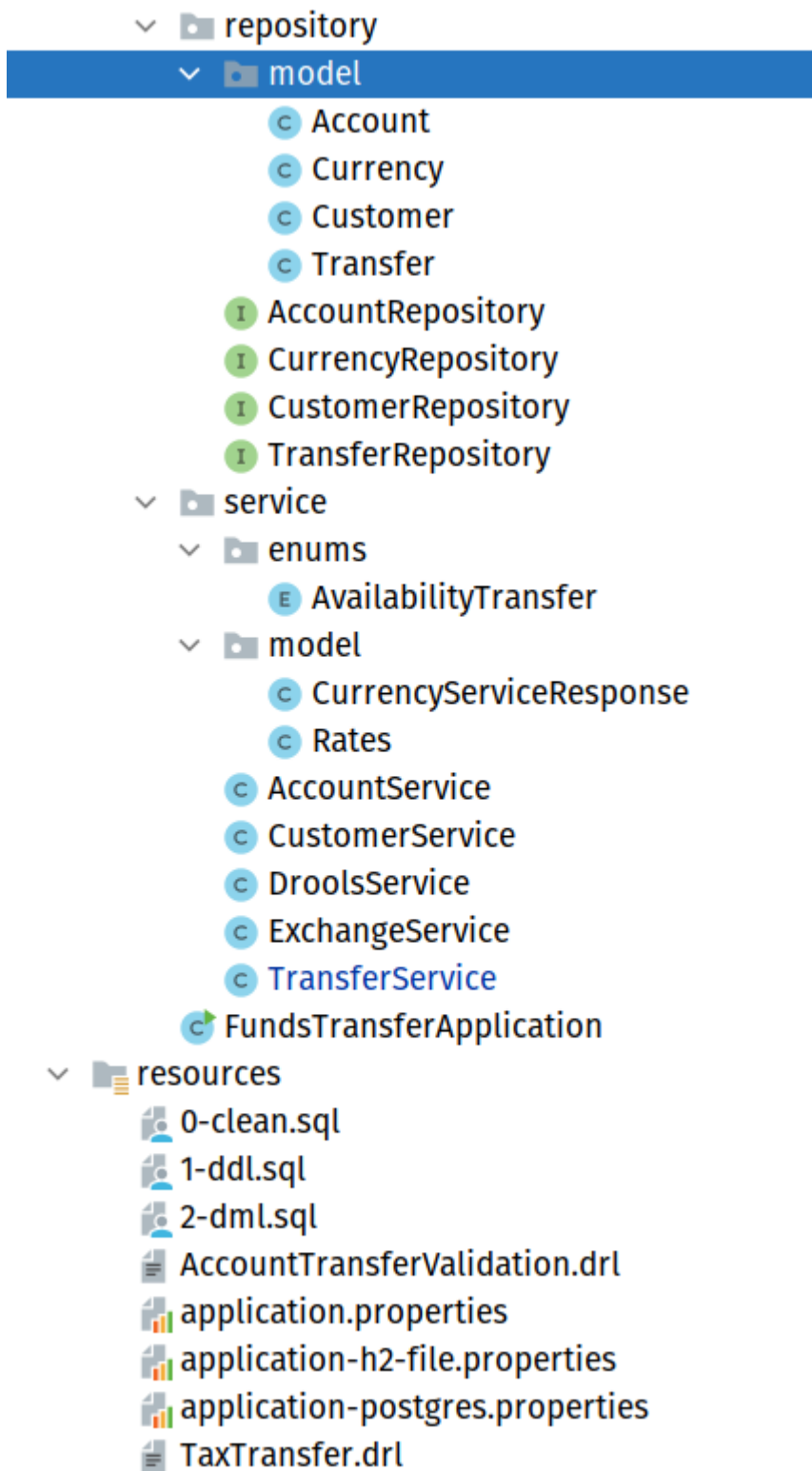
    assertEquals( expected: 412, tx4.getStatusCode().value());
    assertEquals( expected: "K0", body.get("status").asText());
    assertEquals( expected: 1, body.withArray( propertyName: "errors").size());
}

```

Directory structure

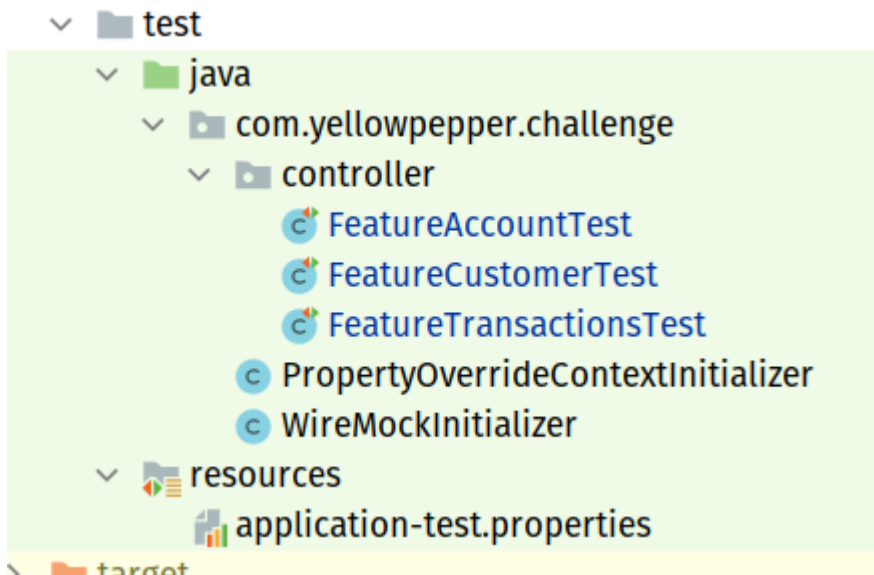


-
- Config: provides from general configuration at start of project like drools rules building and database population
 - Controller: provides all the API resources with their methods
 - Domain: domain specific classes to read drools rules
 - DTO: Objects of information transference
 - Exception: Custom exceptions of the app



- Repository.model: the database models representations
- Repository: the repositories to access to database

- Service.enums: Enums to be used on services
- Service.model: Just some POJOS to transfer info
- Service: All the application services are here
- Resources: resources of the app like SQL scripts for population, configuration files to choose between different database, drools rules (DRL)



- Test: with the testing classes

Reactive programming

```
public Mono<ResponseCreateTransactionDto> applyDiscount(
    Transfer transfer, Account origin, Account destiny, Double tax, BigDecimal amountToTransfer) {
    Mono<Transfer> transferMono = transferRepository.save(transfer);

    return transferMono
        .flatMap(done -> convertCurrenciesToUSD(origin, destiny)) Mono<TransferService.NewAmounts>
        .map(
            oldAmountsInUSD -> {
                Double originInUSD = oldAmountsInUSD.getNewAmountOfOrigin();
                Double destinyInUSD = oldAmountsInUSD.getNewAmountOfDestiny();
                return getNewAmounts(tax, amountToTransfer.doubleValue(), originInUSD, destinyInUSD);
            })
        .flatMap(
            newAmountsInUSD ->
                convertCurrenciesToCAD(transfer, origin, destiny, newAmountsInUSD, tax)) Mono<Boolean>
        .flatMap(done -> exchangeService.fromCADtoUSD(1)) Mono<Double>
        .map(cadInUsd -> createResponse(BigDecimal.valueOf(tax), BigDecimal.valueOf(cadInUsd)));
}
```

```
public Mono<Double> fromUSDtoCAD(double usd) {  
    return getInfoCurrenciesFromService()  
        .map(  
            body -> {  
                if (!body.getSuccess().booleanValue()) {  
                    throw new ResponseStatusException(  
                        HttpStatus.SERVICE_UNAVAILABLE,  
                        "CURRENCIES SERVICE FAILED CONNECTION NOT AVAILABLE TO TRANSFORM FROM USD TO CAD");  
                } else {  
                    saveCache(body);  
                    double cad = usd * body.getRates().getCad();  
                    LOGGER.log(Level.INFO, msg: "From USD to CAD: {0}USD", usd);  
                    LOGGER.log(Level.INFO, msg: "From USD to CAD: {0}CAD", cad);  
                    return cad;  
                }  
            }  
        );  
}
```