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is obsolete and isn't working without api-key, so I've create one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 check with sonarqube and intellij inspector tools, to prevent missing errors, the ttd testing using the behavior of interface is incredible 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 use some patterns to describe the work

Using entity-collection

Get Customer entity:
GET /v1/customers/:idCustomer
Get Customer collection:
GET /v1/customers
Create a transaction:
POST /v1/transactions

Using master-slave design pattern accounts is slave of customers

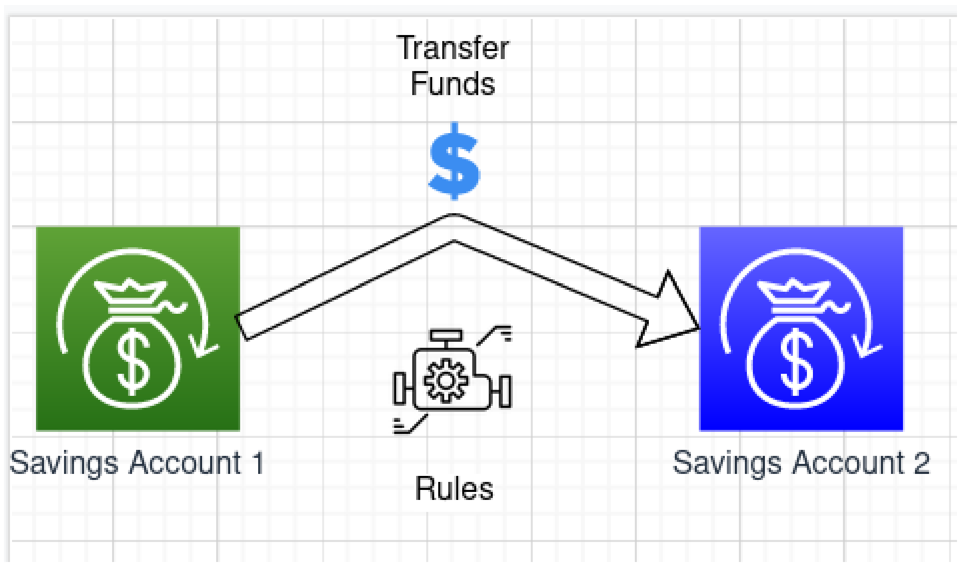
Get single account:
GET /v1/customers/:idCustomer/accounts/:idAccount

Get all accounts of customer:
GET /v1/customers/:idCustomer/accounts

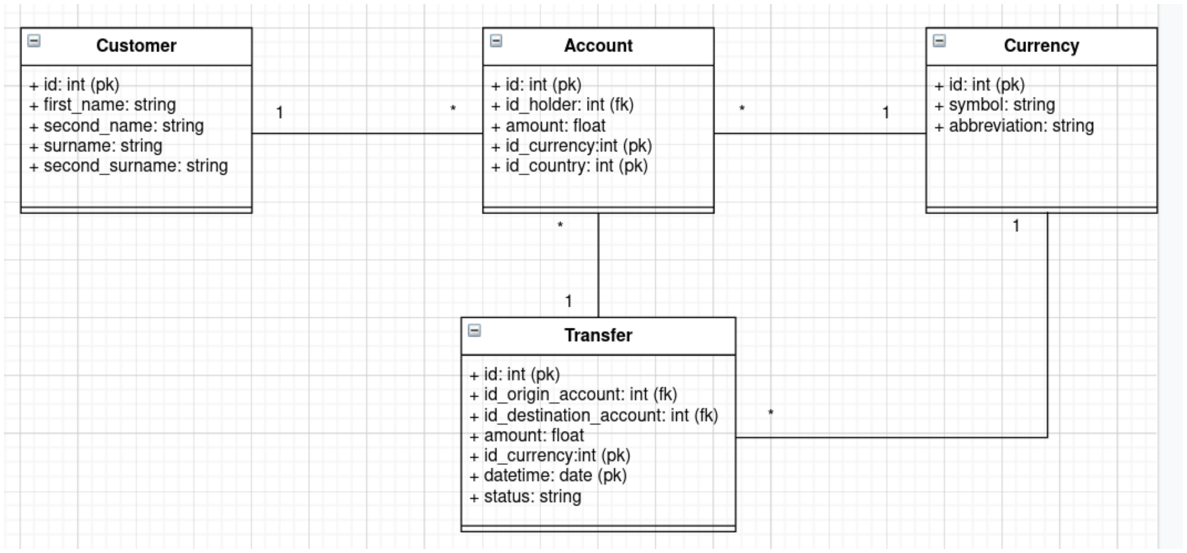
Using command pattern:

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

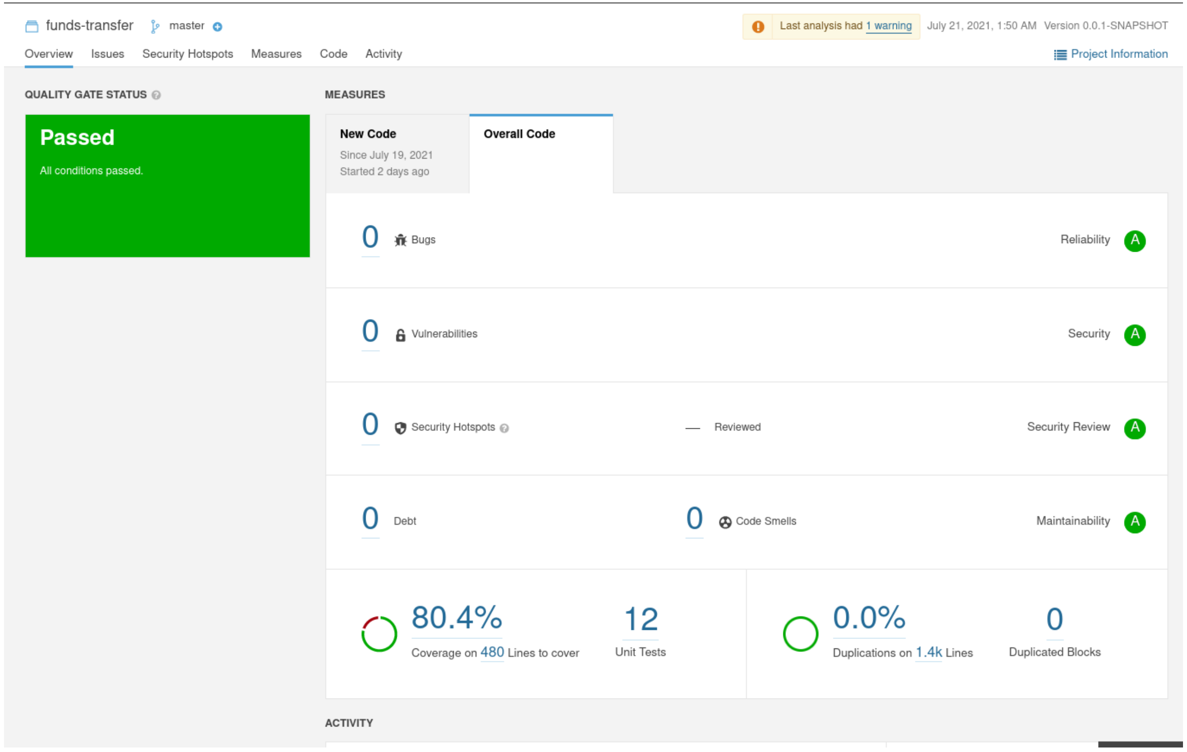
Problem understanding



Entities



Screenshots of SonarQube



funds-transfer master

Last analysis had 1 warning July 21, 2021, 1:50 AM Version 0.0.1-SNAPSHOT

Overview Issues Security Hotspots Measures Code Activity Project Information

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code Since July 19, 2021 Started 2 days ago

Overall Code

0 Bugs — Reliability A

0 Vulnerabilities — Security A

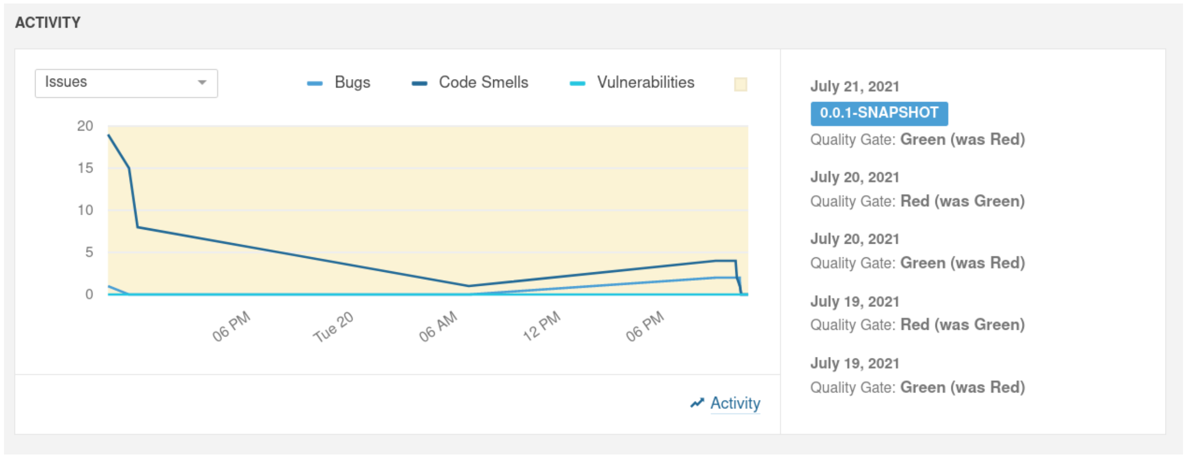
0 Security Hotspots — Reviewed — Security Review A

0 Debt — 0 Code Smells — Maintainability A

80.4% Coverage on 480 Lines to cover — 12 Unit Tests

0.0% Duplications on 1.4k Lines — 0 Duplicated Blocks

ACTIVITY



ACTIVITY

July 21, 2021
0.0.1-SNAPSHOT
Quality Gate: **Green (was Red)**

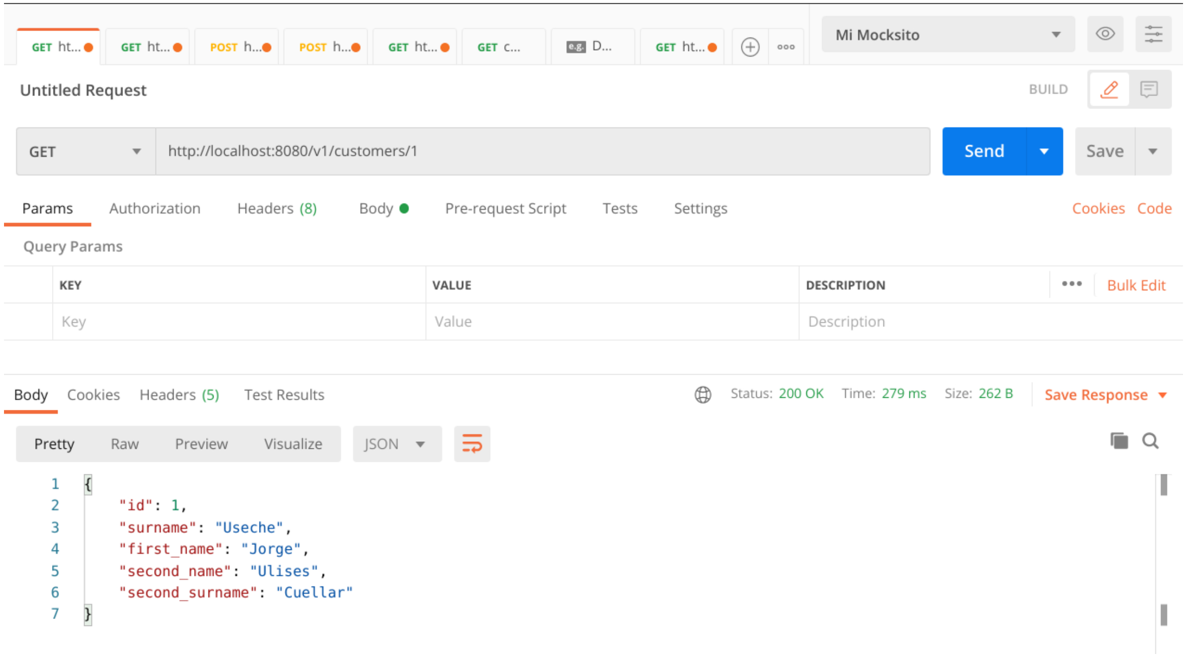
July 20, 2021
Quality Gate: **Red (was Green)**

July 20, 2021
Quality Gate: **Green (was Red)**

July 19, 2021
Quality Gate: **Red (was Green)**

July 19, 2021
Quality Gate: **Green (was Red)**

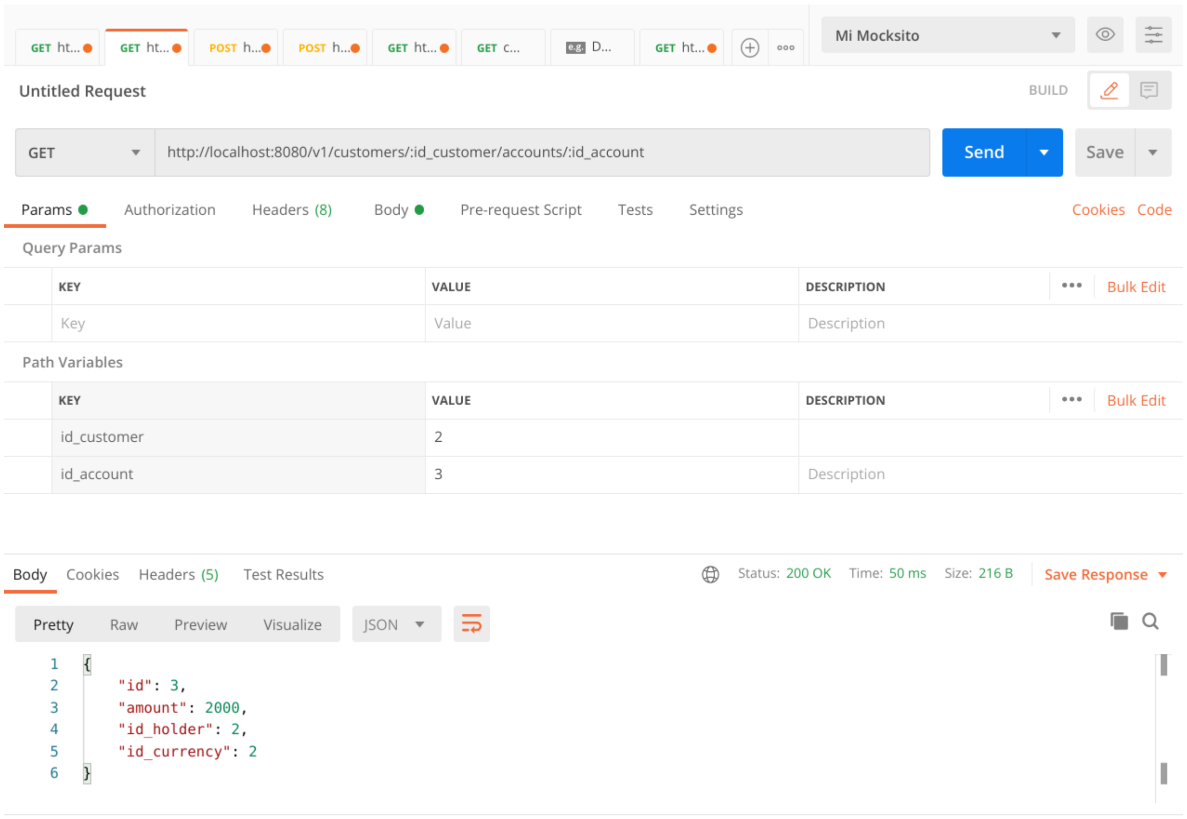
Screenshots of API in postman



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

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

```
{
    "id": 1,
    "surname": "Useche",
    "first_name": "Jorge",
    "second_name": "Ulises",
    "second_surname": "Cuellar"
}
```



Untitled Request

GET http://localhost:8080/v1/customers/:id_customer/accounts/:id_account

Query Params

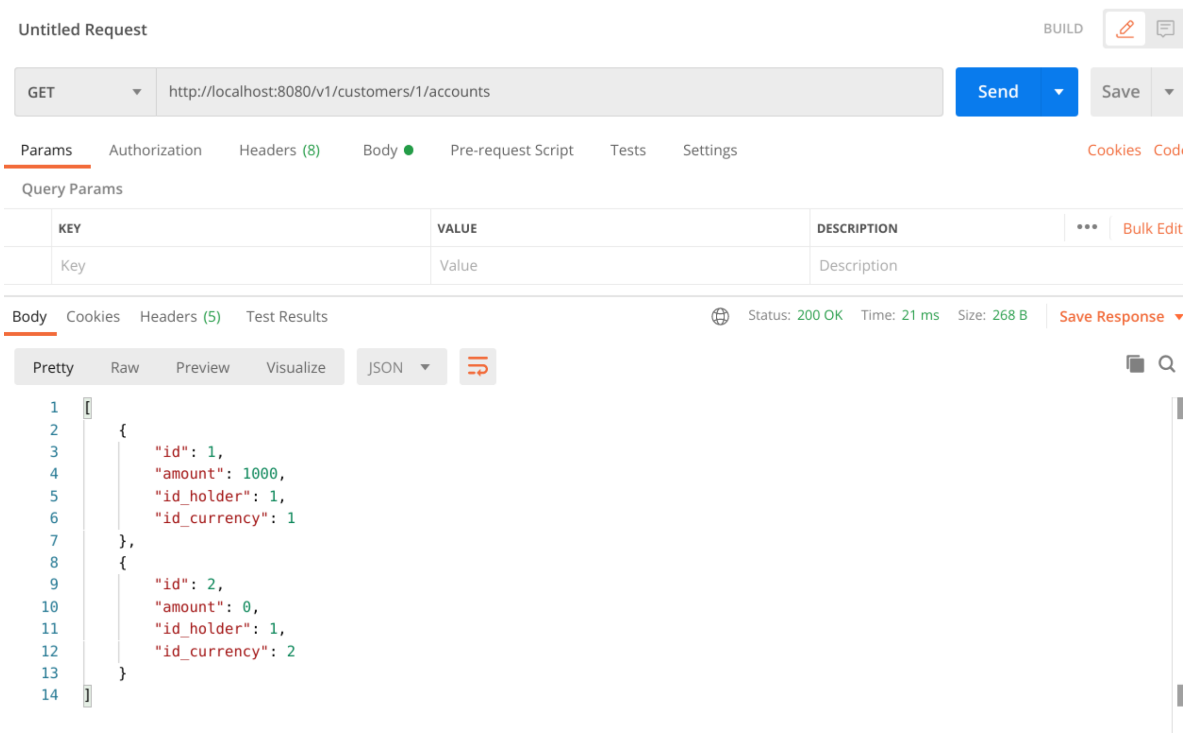
KEY	VALUE	DESCRIPTION
Key	Value	Description

Path Variables

KEY	VALUE	DESCRIPTION
id_customer	2	
id_account	3	Description

Status: 200 OK Time: 50 ms Size: 216 B

```
{
    "id": 3,
    "amount": 2000,
    "id_holder": 2,
    "id_currency": 2
}
```



Untitled Request

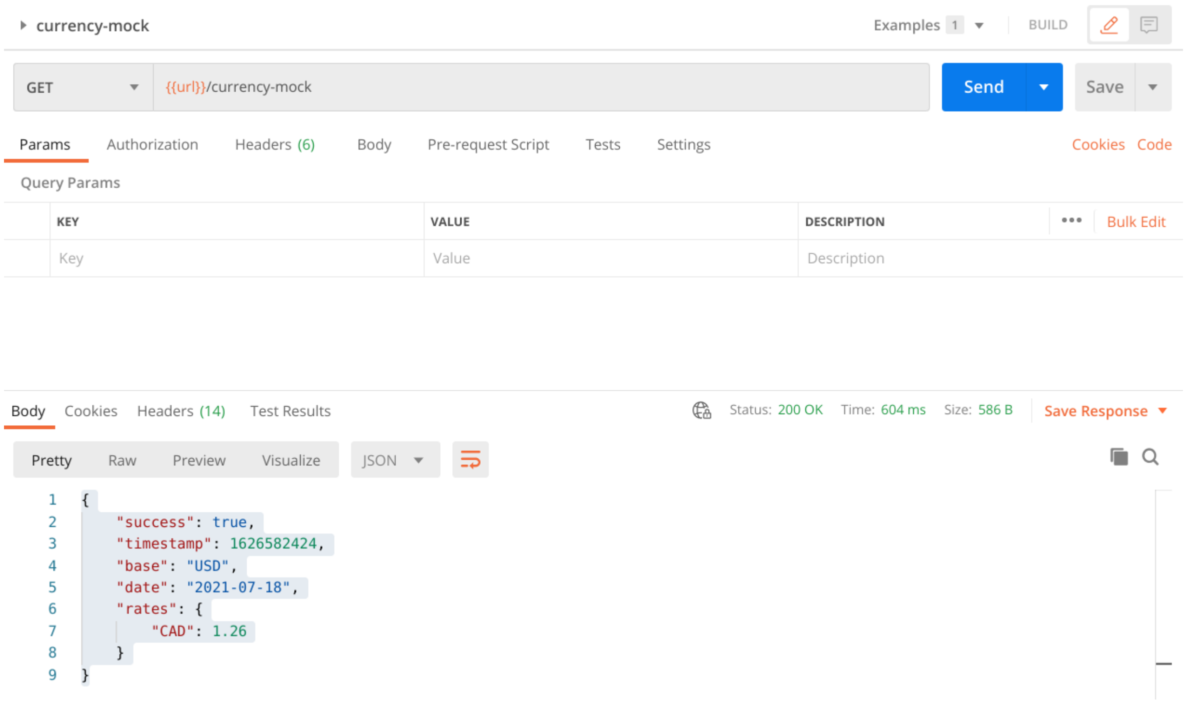
GET http://localhost:8080/v1/customers/1/accounts

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 21 ms Size: 268 B

```
[
    {
        "id": 1,
        "amount": 1000,
        "id_holder": 1,
        "id_currency": 1
    },
    {
        "id": 2,
        "amount": 0,
        "id_holder": 1,
        "id_currency": 2
    }
]
```

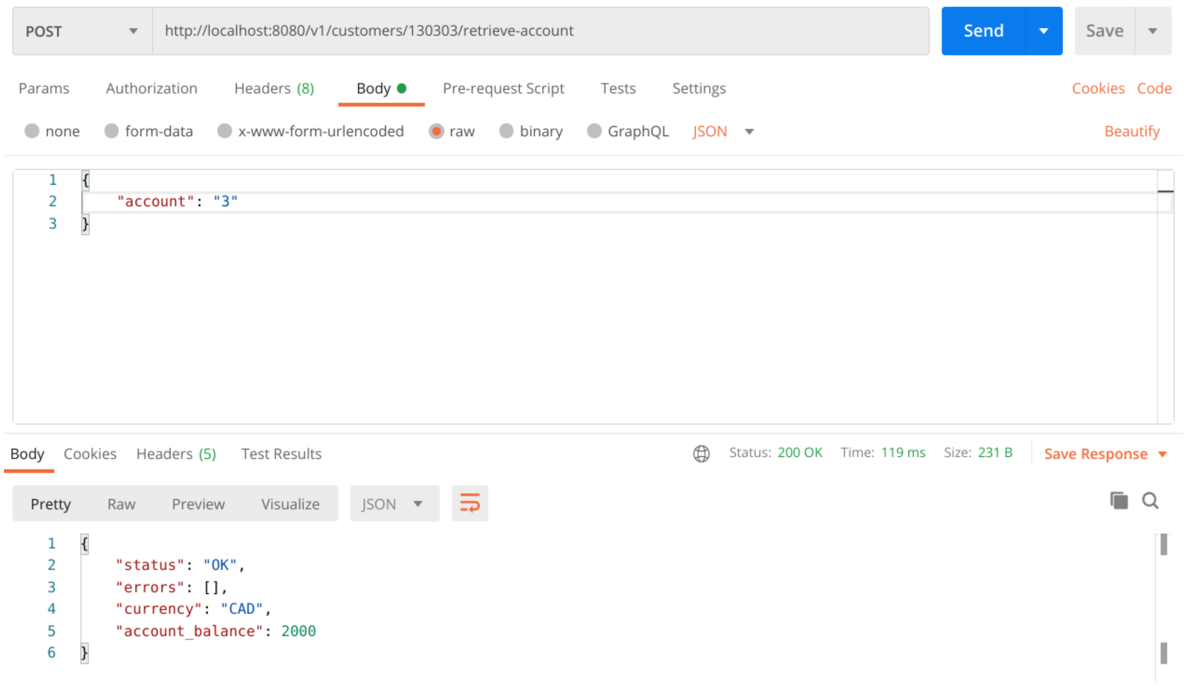


GET {{url}}/currency-mock

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 604 ms Size: 586 B

```
{
    "success": true,
    "timestamp": 1626582424,
    "base": "USD",
    "date": "2021-07-18",
    "rates": {
        "CAD": 1.26
    }
}
```

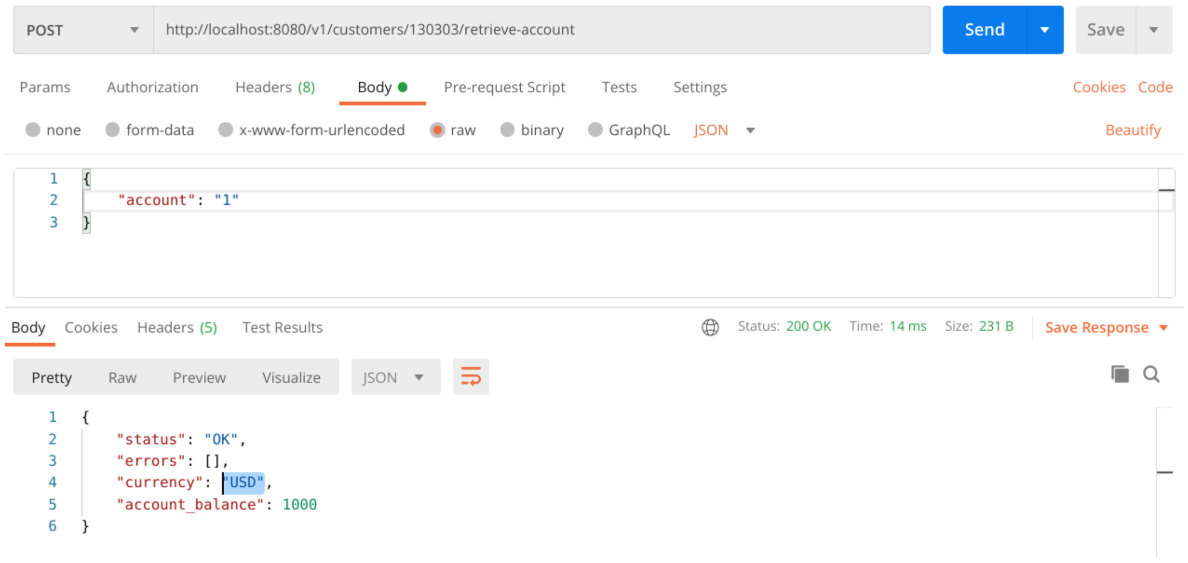


POST http://localhost:8080/v1/customers/130303/retrieve-account

```
{
    "account": "3"
}
```

Status: 200 OK Time: 119 ms Size: 231 B

```
{
    "status": "OK",
    "errors": [],
    "currency": "CAD",
    "account_balance": 2000
}
```



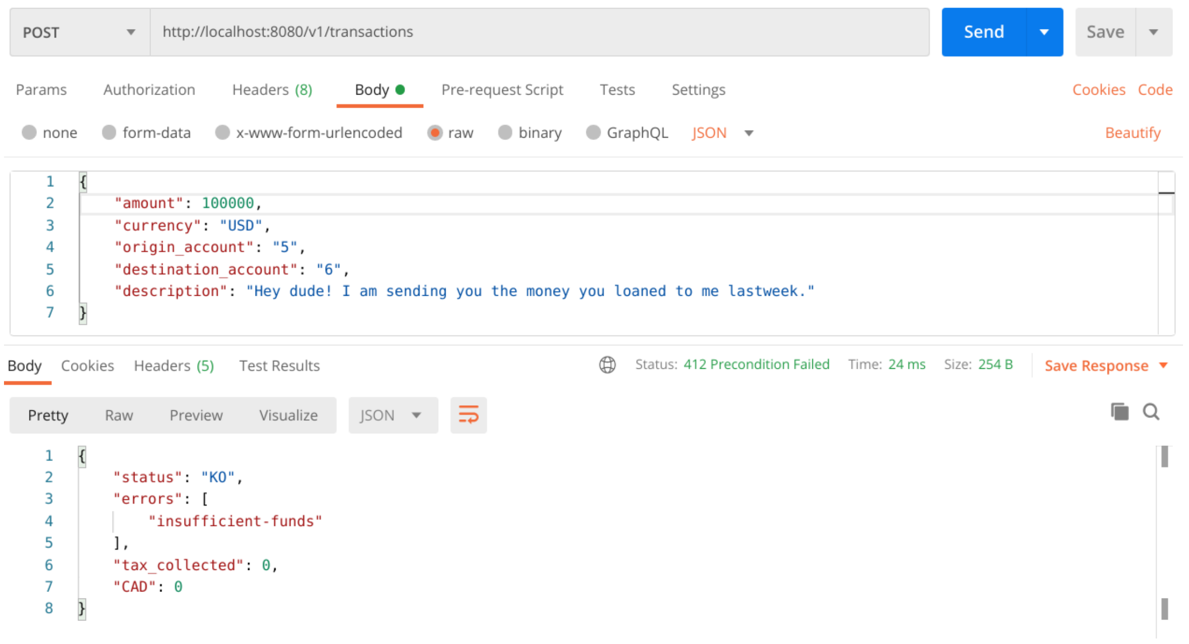
http://localhost:8080/v1/customers/130303/retrieve-account

Body: none ○ form-data ○ x-www-form-urlencoded [✓ raw] ○ binary ○ GraphQL ○ JSON

```
{
    "account": "1"
}
```

Status: 200 OK Time: 14 ms Size: 231 B

```
{
    "status": "OK",
    "errors": [],
    "currency": "USD",
    "account_balance": 1000
}
```

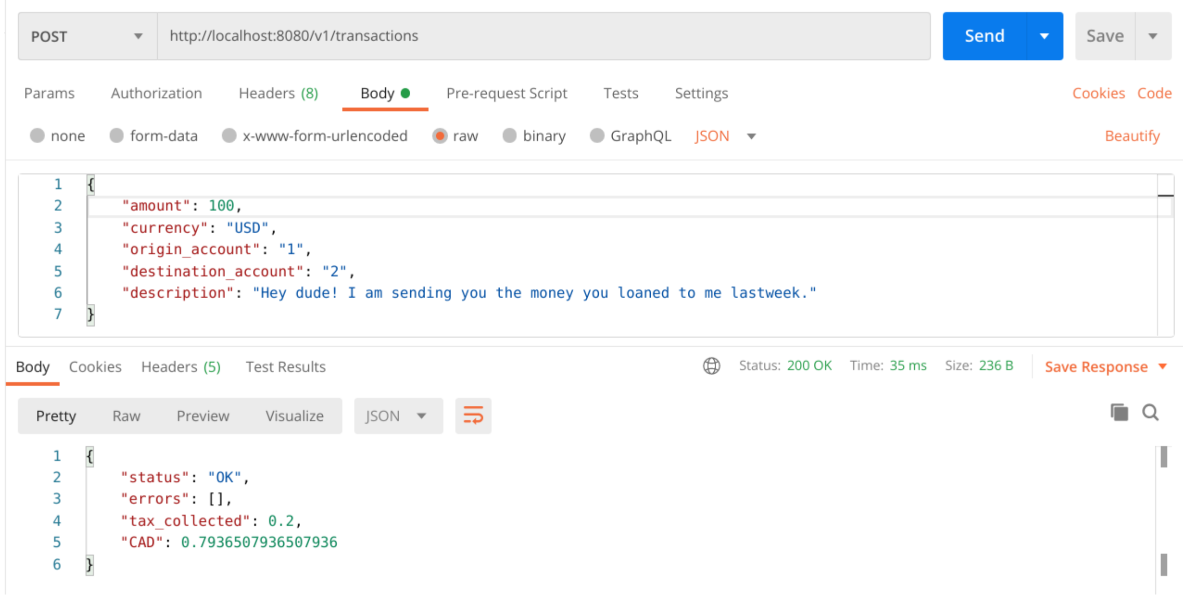


http://localhost:8080/v1/transactions

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

Status: 412 Precondition Failed Time: 24 ms Size: 254 B

```
{
    "status": "KO",
    "errors": [
        "insufficient-funds"
    ],
    "tax_collected": 0,
    "CAD": 0
}
```

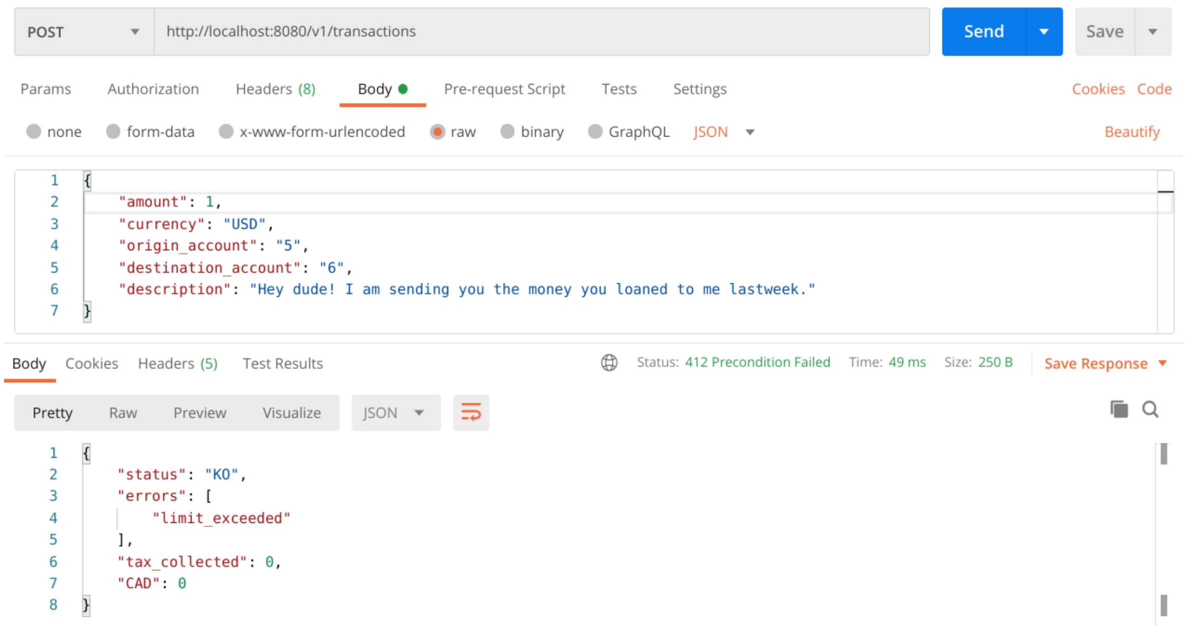


http://localhost:8080/v1/transactions

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

Status: 200 OK Time: 35 ms Size: 236 B

```
{
    "status": "OK",
    "errors": [],
    "tax_collected": 0.2,
    "CAD": 0.7936507936507936
}
```



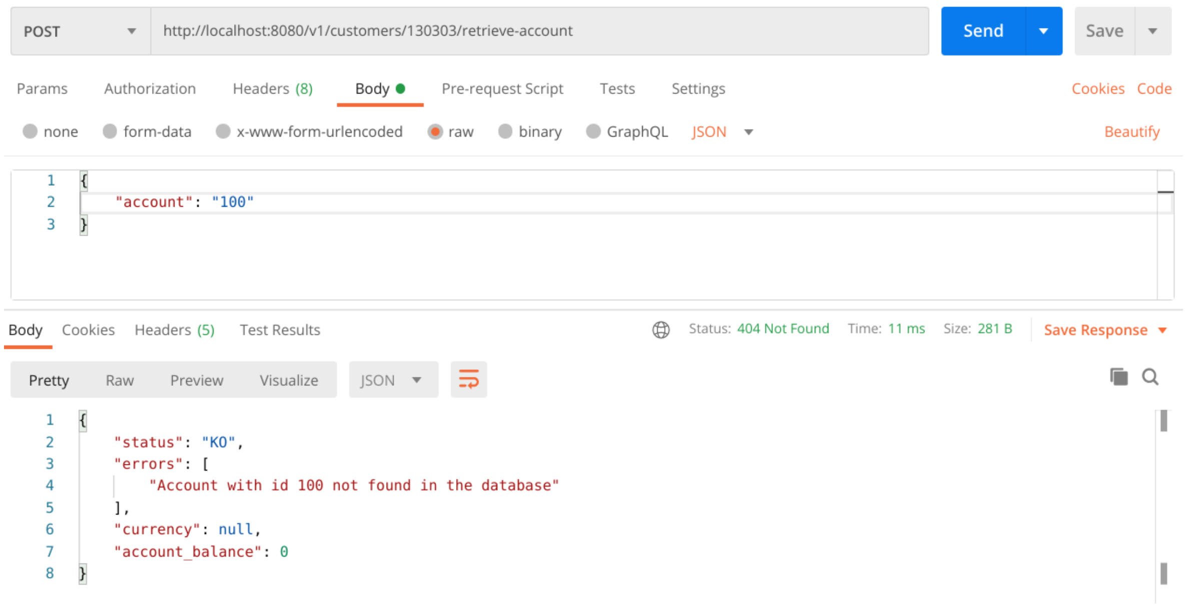
POST http://localhost:8080/v1/transactions

Body: raw, JSON

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

Status: 412 Precondition Failed Time: 49 ms Size: 250 B

```
{
    "status": "KO",
    "errors": [
        "limit_exceeded"
    ],
    "tax_collected": 0,
    "CAD": 0
}
```



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

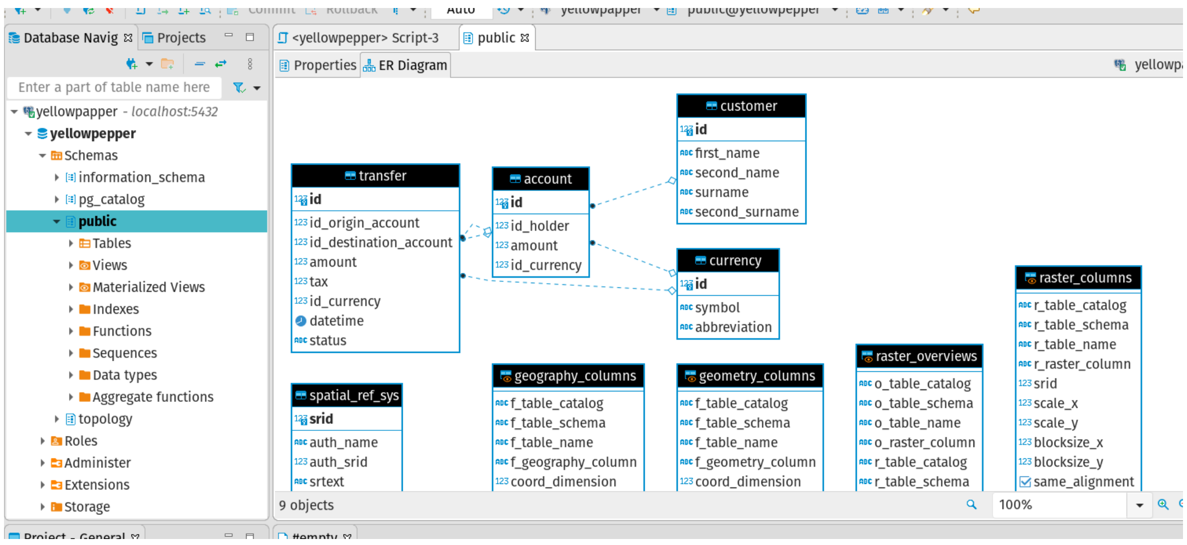
Body: raw, JSON

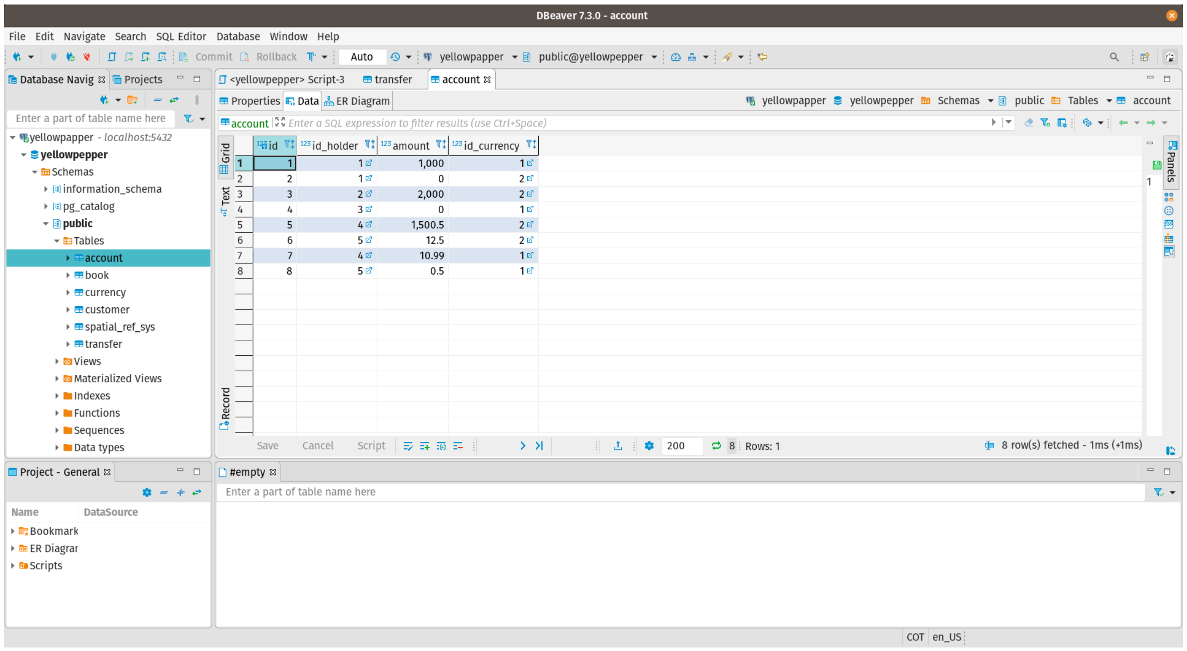
```
{
    "account": "100"
}
```

Status: 404 Not Found Time: 11 ms Size: 281 B

```
{
    "status": "KO",
    "errors": [
        "Account with id 100 not found in the database"
    ],
    "currency": null,
    "account_balance": 0
}
```

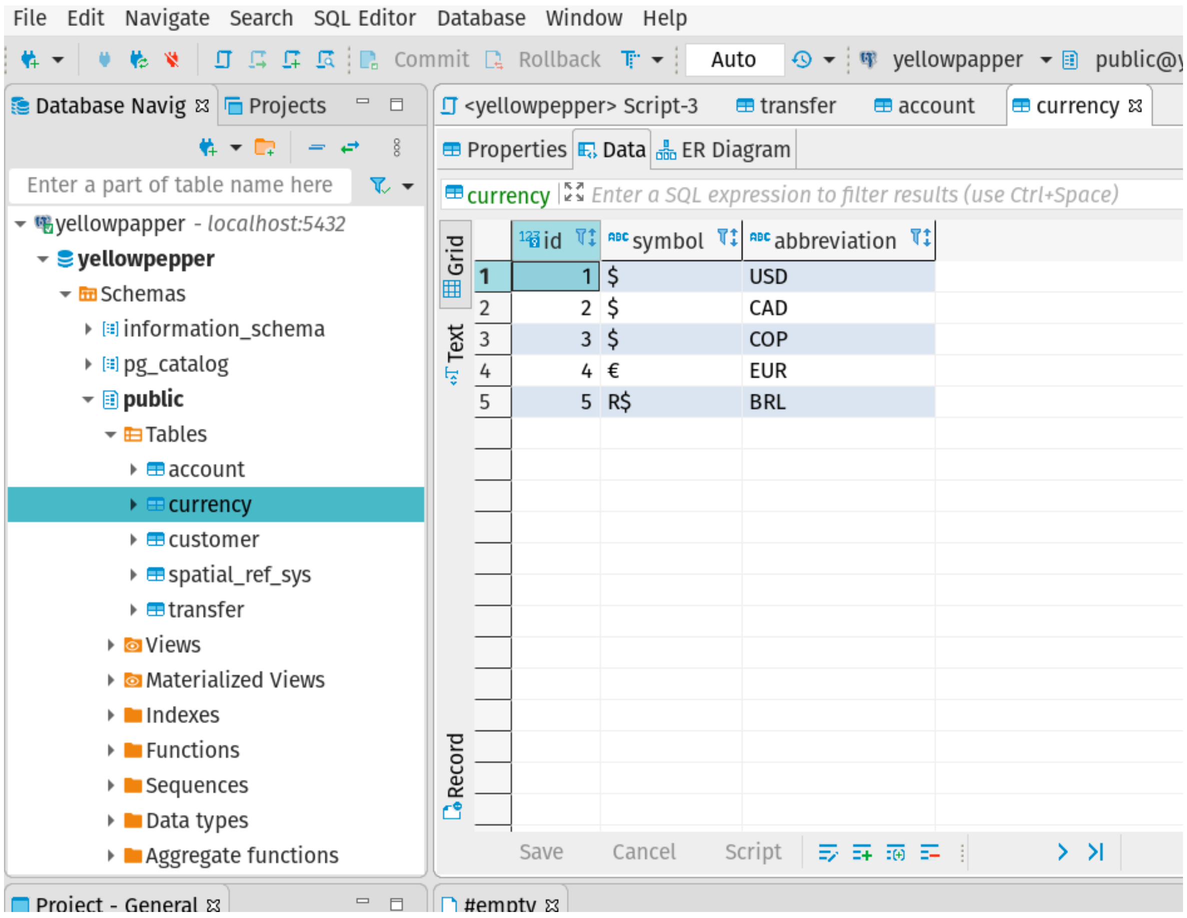
Data in PostgreSQL





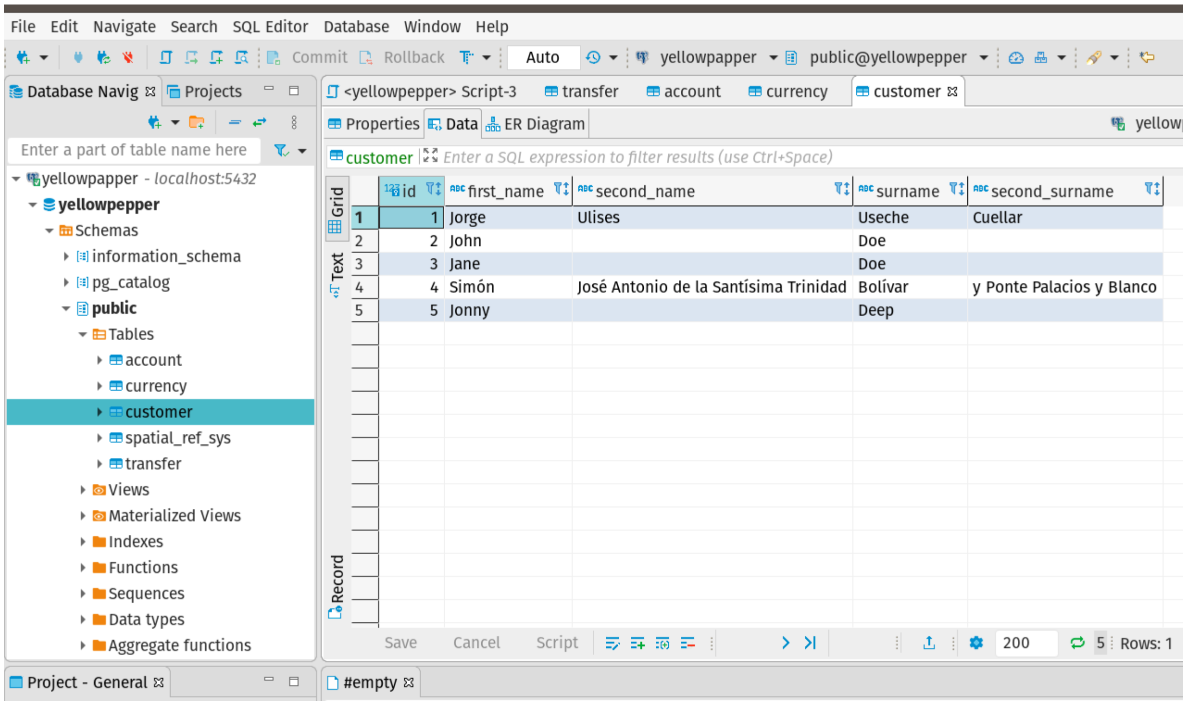
account | Enter a SQL expression to filter results (use Ctrl+Space)

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



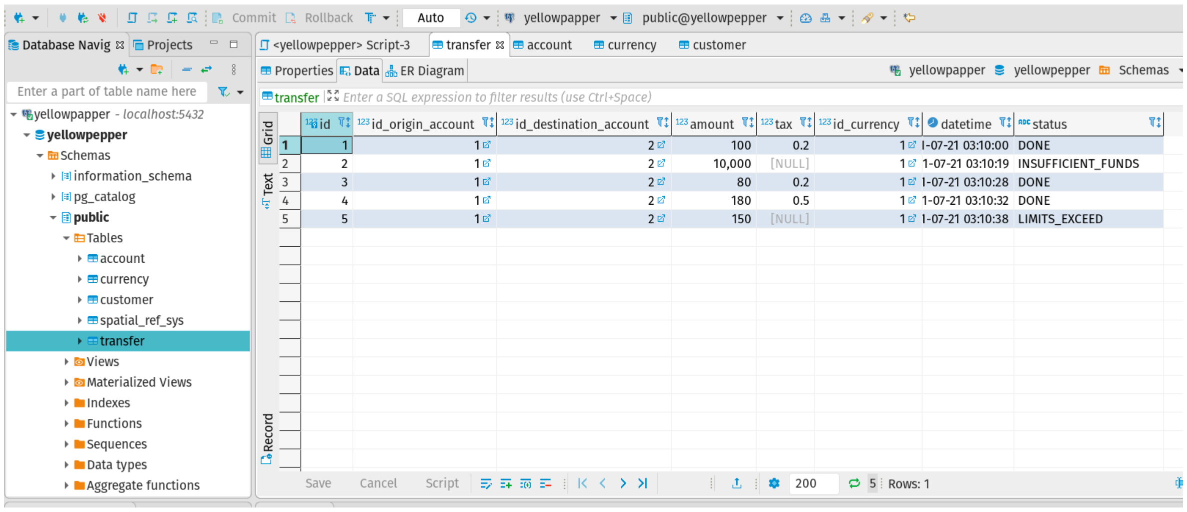
currency

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

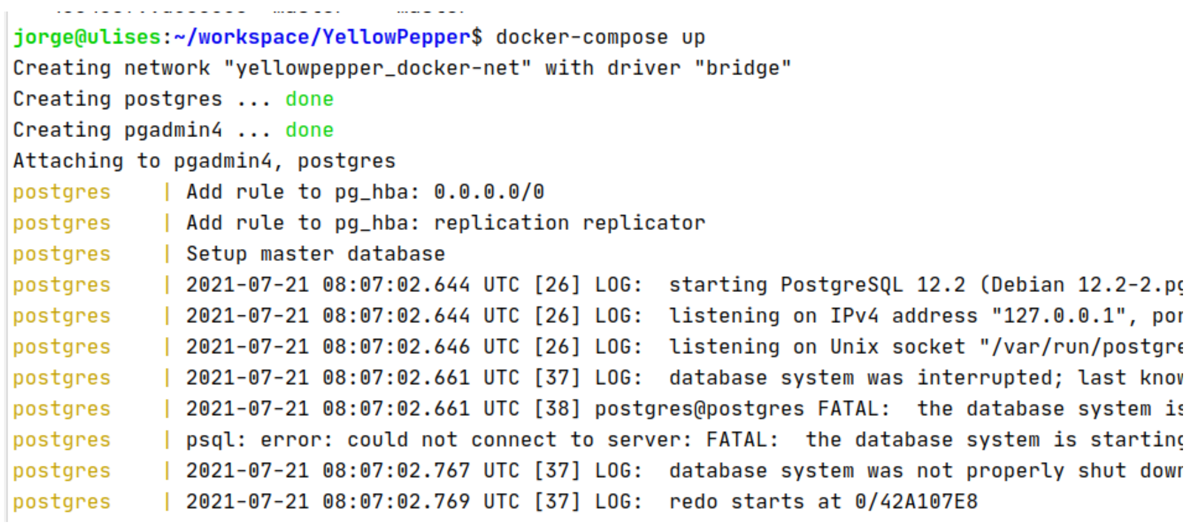


customer

	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	

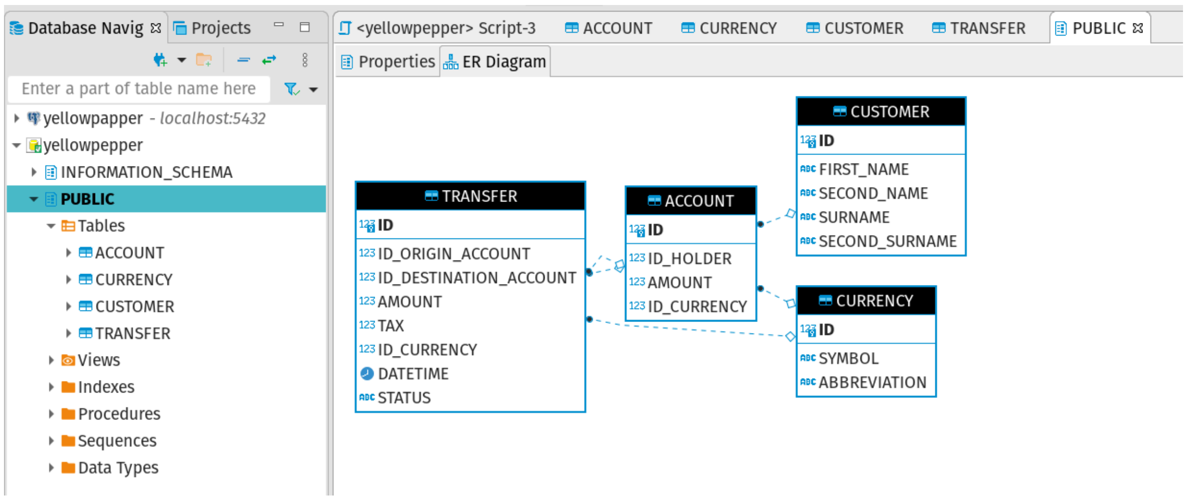


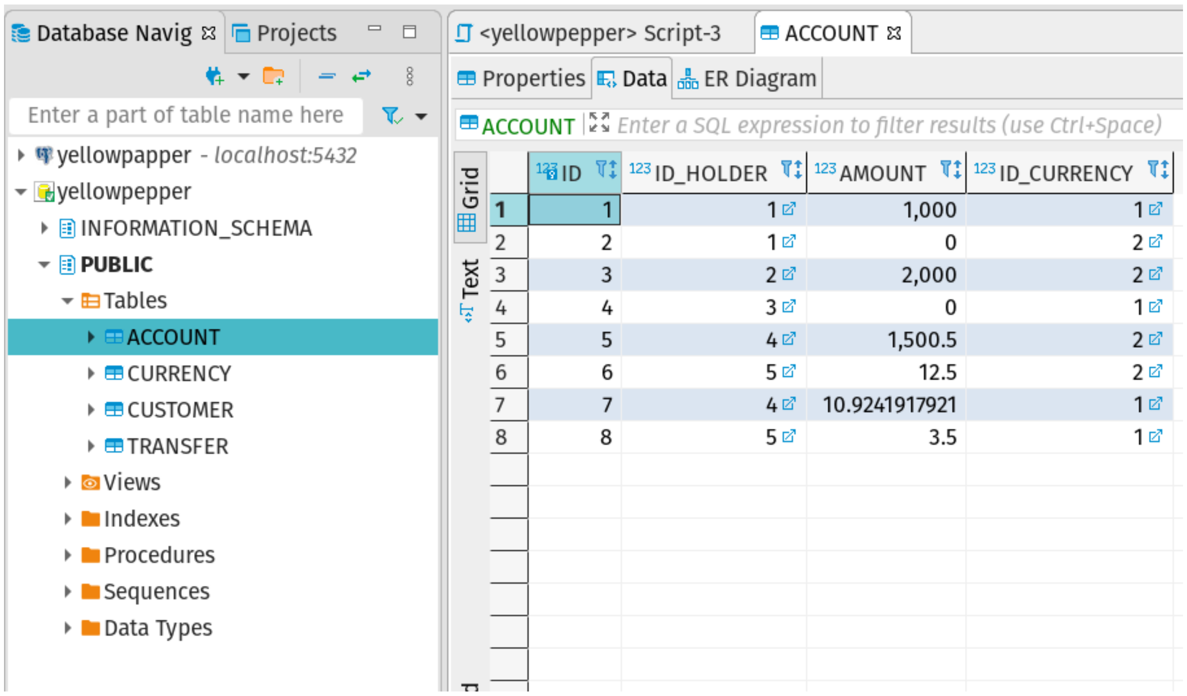
	id	id_origin_account	id_destination_account	amount	tax	id_currency	datetime	status
1	1	1	2	100	0.2	1	1-07-21 03:10:00	DONE
2	2	1	2	10,000	[NULL]	1	1-07-21 03:10:19	INSUFFICIENT_FUNDS
3	3	1	2	80	0.2	1	1-07-21 03:10:28	DONE
4	4	1	2	180	0.5	1	1-07-21 03:10:32	DONE
5	5	1	2	150	[NULL]	1	1-07-21 03:10:38	LIMITS_EXCEED



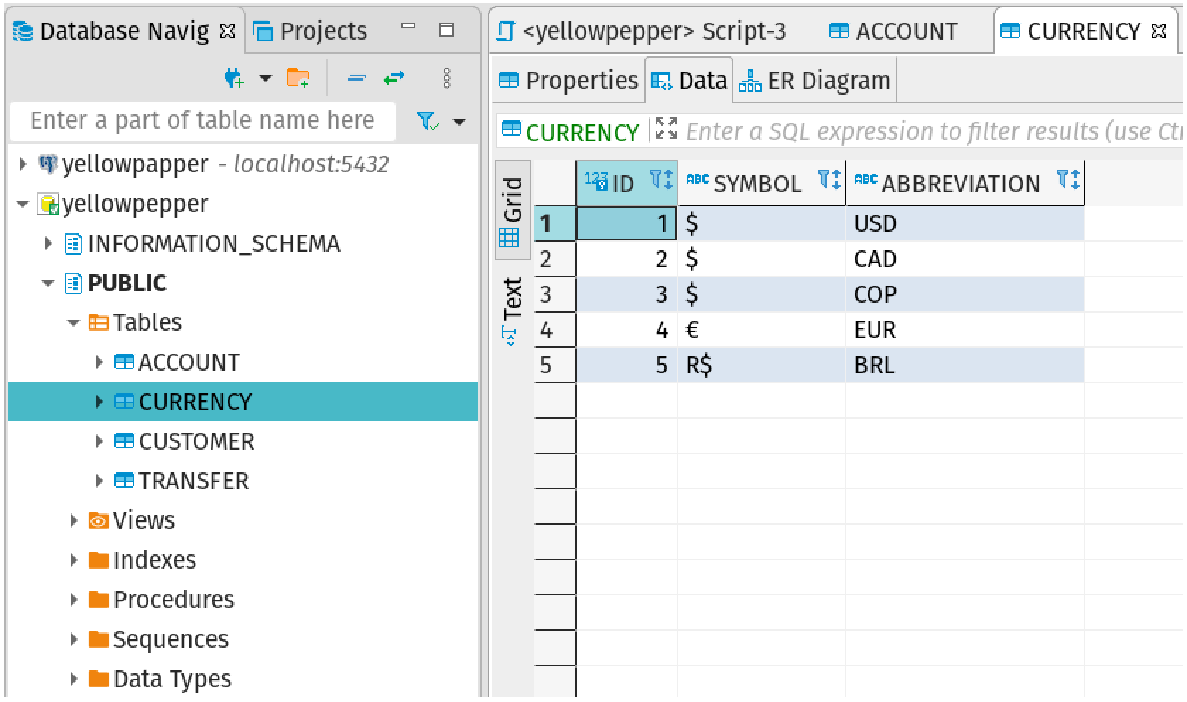
```
jorge@ulises:~/workspace/YellowPepper$ docker-compose up
Creating network "yellowpepper_docker-net" with driver "bridge"
Creating postgres ... done
Creating pgadmin4 ... done
Attaching to pgadmin4, postgres
postgres     | Add rule to pg_hba: 0.0.0.0/0
postgres     | Add rule to pg_hba: replication replicator
postgres     | Setup master database
postgres     | 2021-07-21 08:07:02.644 UTC [26] LOG:  starting PostgreSQL 12.2 (Debian 12.2-2.pg
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 dowr
postgres     | 2021-07-21 08:07:02.769 UTC [37] LOG:  redo starts at 0/42A107E8
```

Data in h2 db file

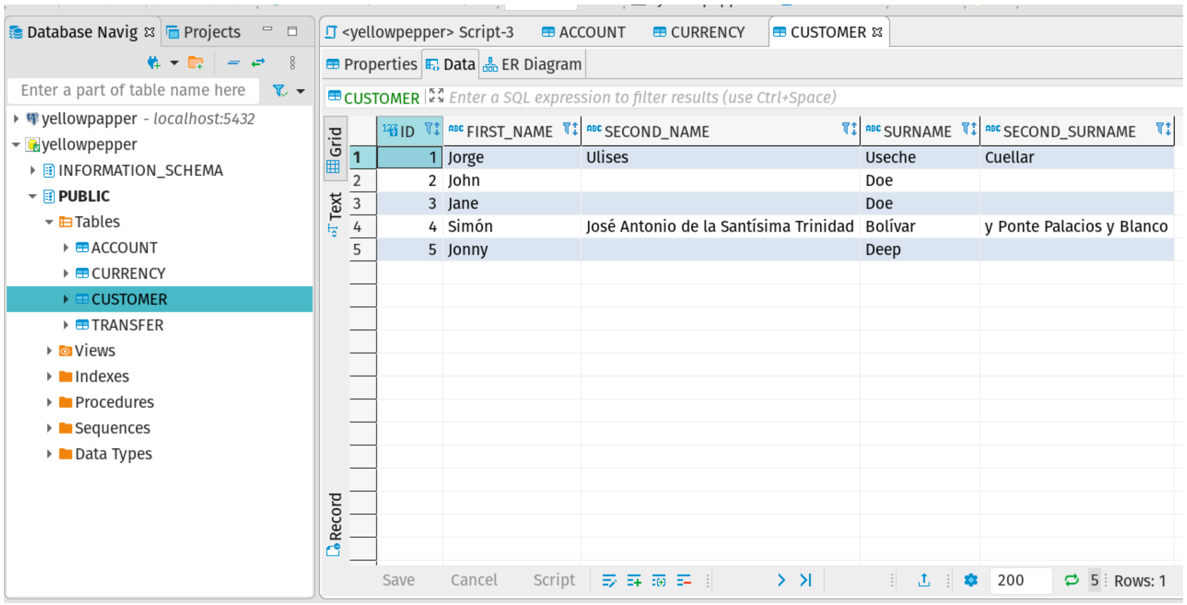




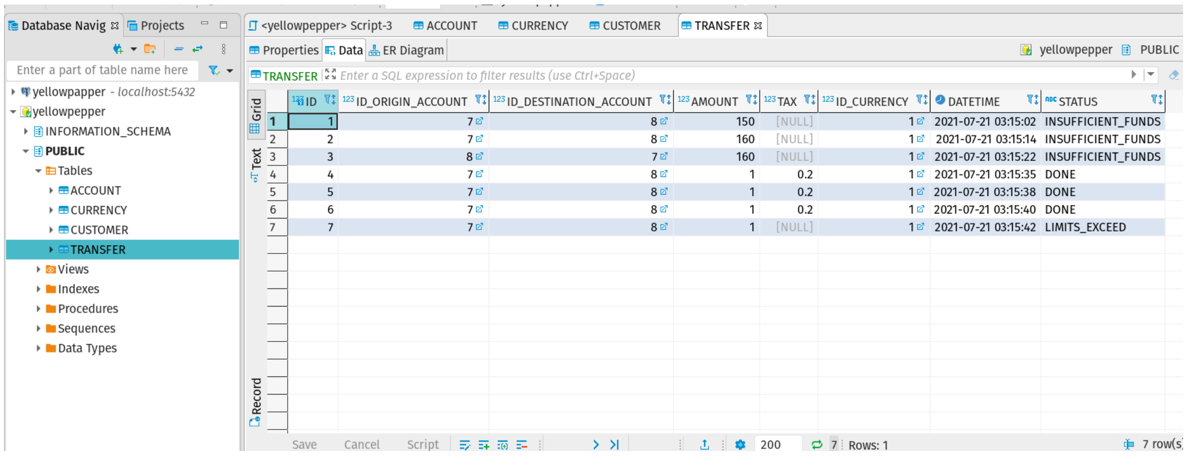
	ID	ID_HOLDER	AMOUNT	ID_CURRENCY
1	1	1	1,000	1
2	2	1	0	2
3	3	2	2,000	2
4	4	3	0	1
5	5	4	1,500.5	2
6	6	5	12.5	2
7	7	4	10.9241917921	1
8	8	5	3.5	1



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

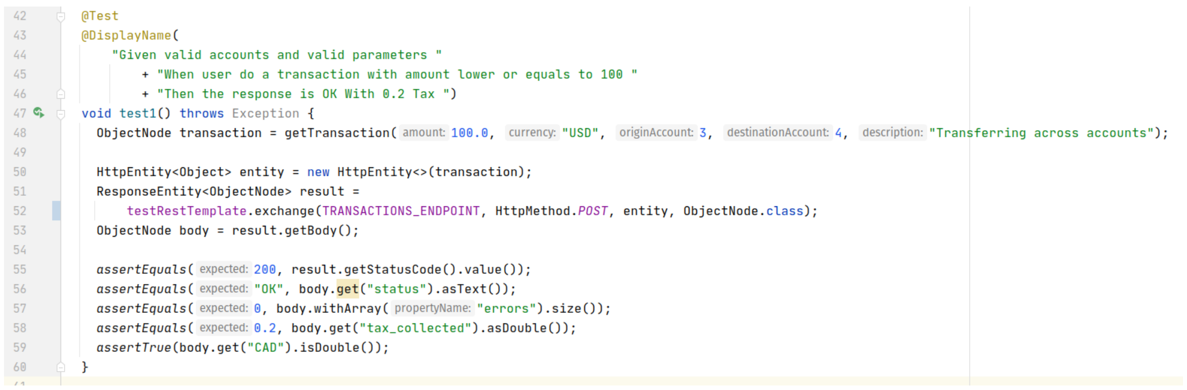


	ID	FIRST_NAME	SECOND_NAME	SURNAME	SECOND_SURNAME
1	1	Jorge	Ulises	Useche	Cuellar
2	2	John		Doe	
3	3	Jane		Doe	
4	4	Simón	José Antonio de la Santísima Trinidad	Bolívar	y Ponte Palacios y Blanco
5	5	Jonny		Deep	

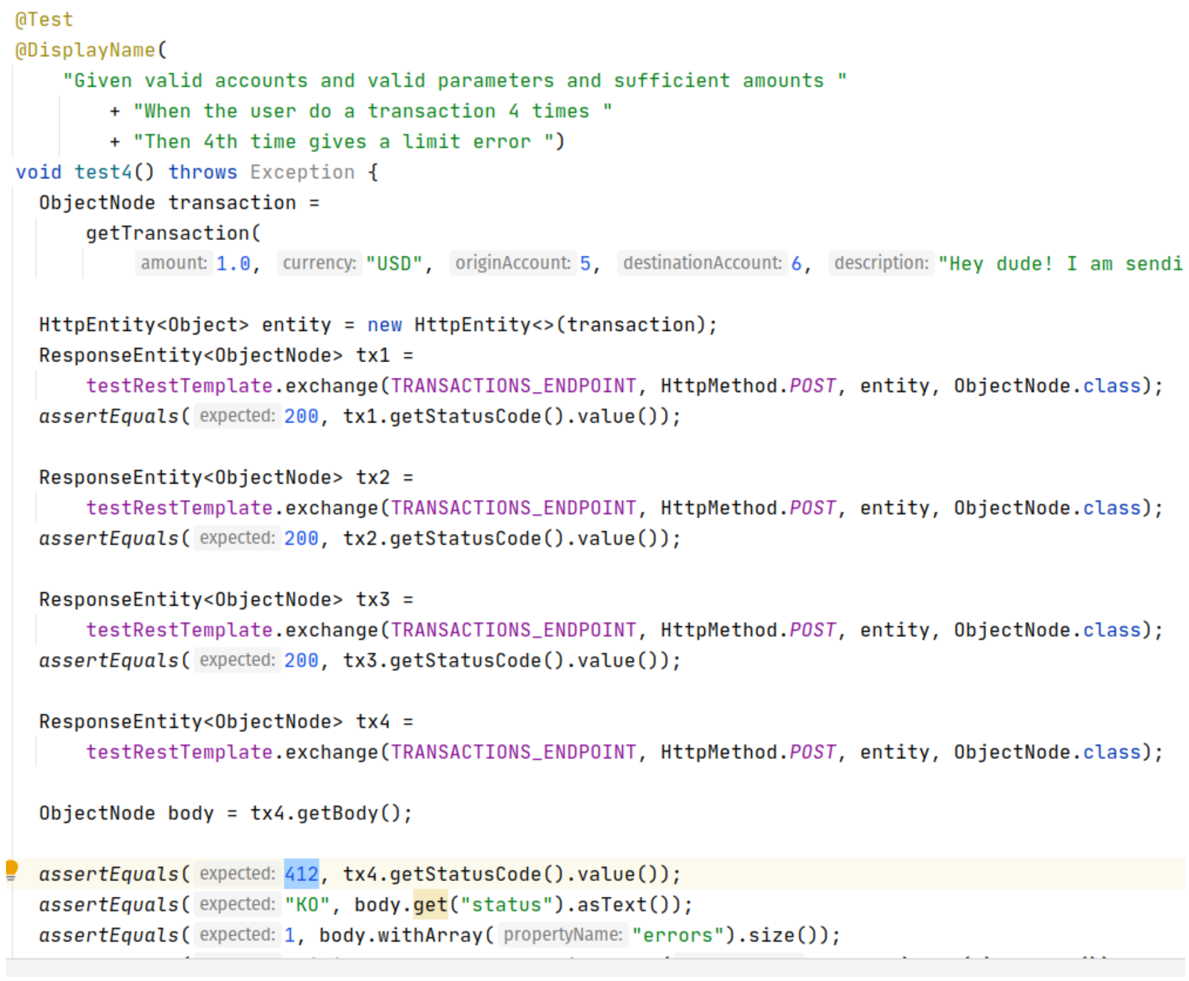


	ID	ID_ORIGIN_ACCOUNT	ID_DESTINATION_ACCOUNT	AMOUNT	TAX	ID_CURRENCY	DATETIME	STATUS
1	1	7	8	150	[NULL]	1	2021-07-21 03:15:02	INSUFFICIENT_FUNDS
2	2	7	8	160	[NULL]	1	2021-07-21 03:15:14	INSUFFICIENT_FUNDS
3	3	8	7	160	[NULL]	1	2021-07-21 03:15:22	INSUFFICIENT_FUNDS
4	4	7	8	1	0.2	1	2021-07-21 03:15:35	DONE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

TDD with BDD focus

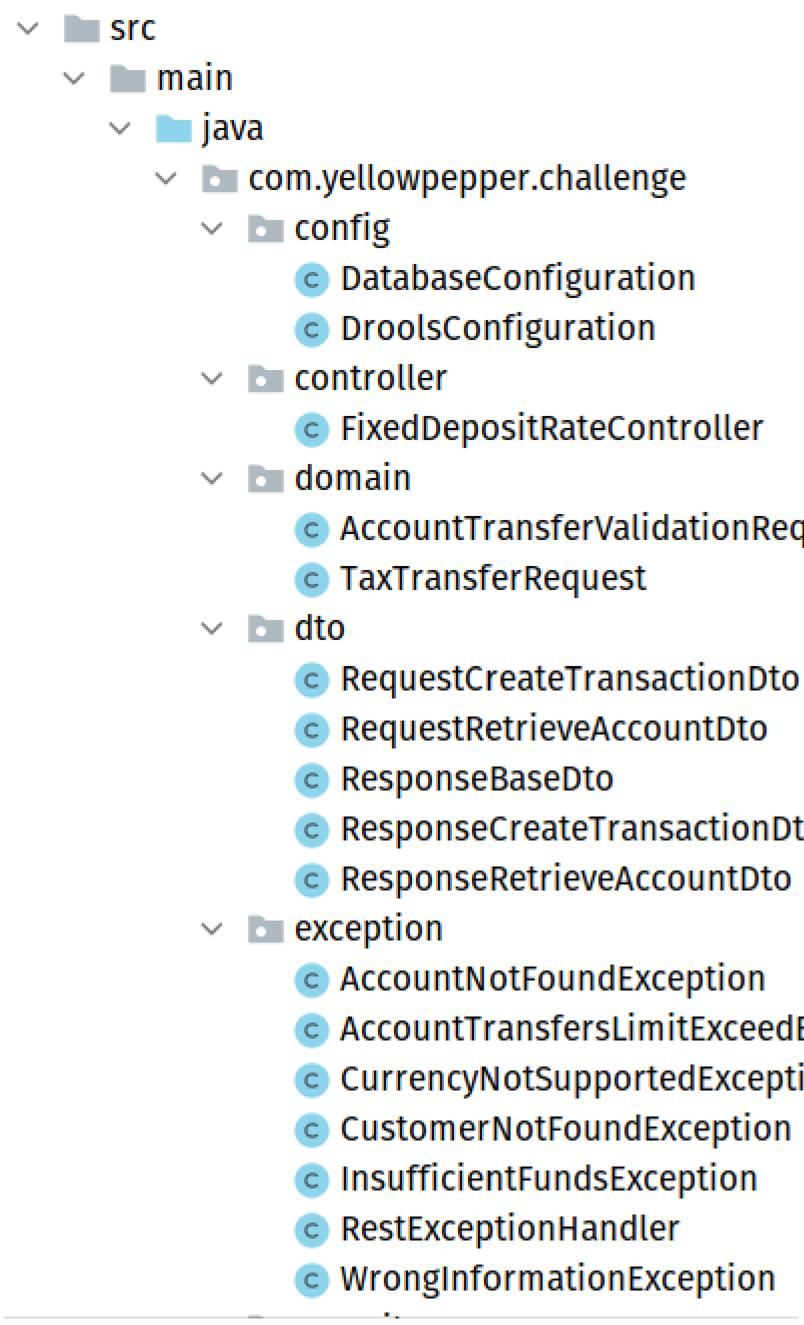


```java
@Test
@DisplayName(
    "Given valid accounts and valid parameters "
        + "When user do a transaction with amount lower or equals to 100 "
        + "Then the response is OK With 0.2 Tax ")
void test1() throws Exception {
  ObjectNode transaction = getTransaction( amount: 100.0,  currency: "USD",  originAccount: 3,  destinationAccount: 4,  description: "Transferring across accounts");

  HttpEntity<Object> entity = new HttpEntity<>(transaction);
  ResponseEntity<ObjectNode> result =
      testRestTemplate.exchange(TRANSACTIONS_ENDPOINT, HttpMethod.POST, entity, ObjectNode.class);
  ObjectNode body = result.getBody();

  assertEquals( expected: 200, result.getStatusCode().value());
  assertEquals( expected: "OK", body.get("status").asText());
  assertEquals( expected: 0, body.withArray( propertyName: "errors").size());
  assertEquals( expected: 0.2, body.get("tax_collected").asDouble());
  assertTrue(body.get("CAD").isDouble());
}
```



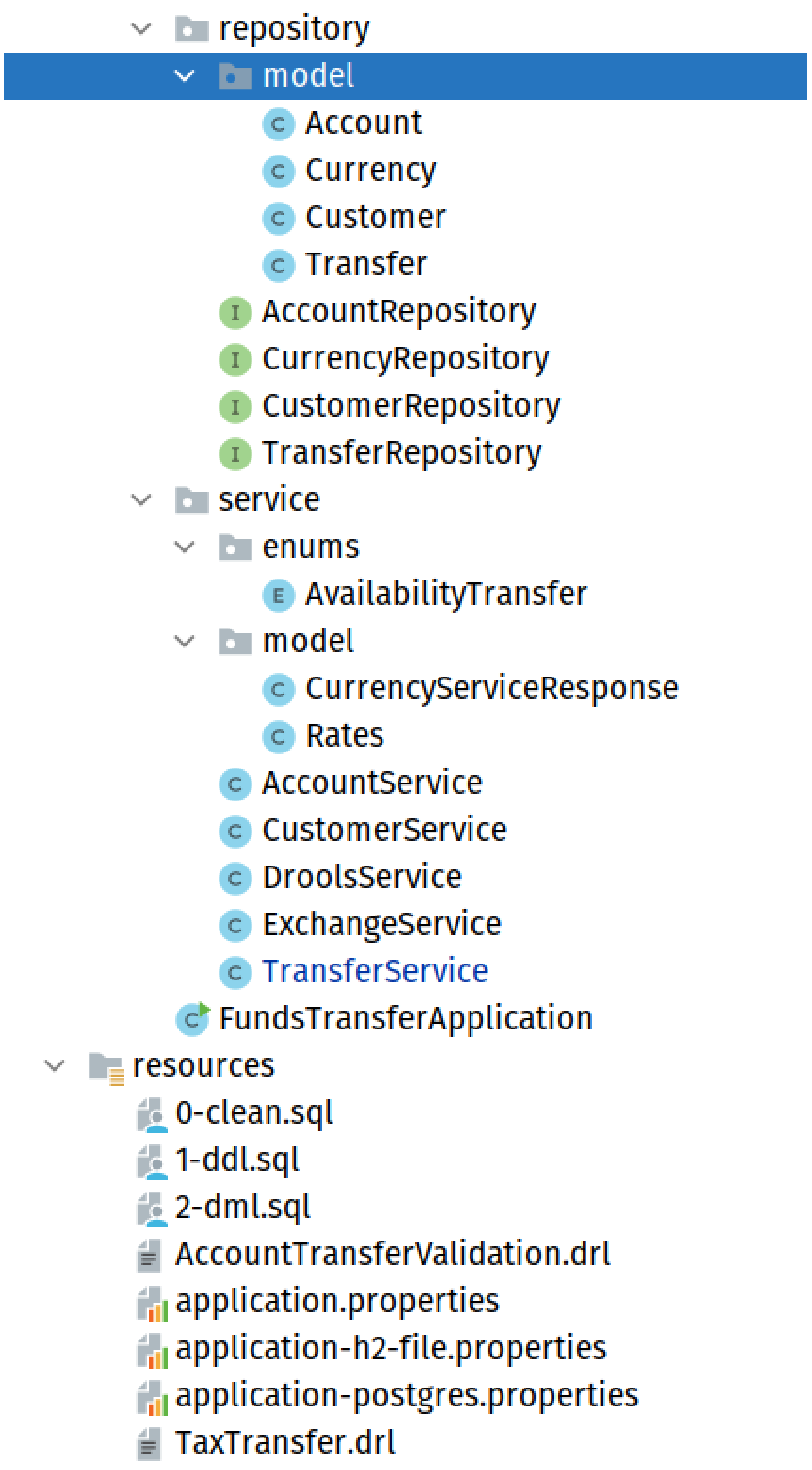
```
@Test
@DisplayName(
    "Given valid accounts and valid parameters and sufficient amounts "
        + "When the user do a transaction 4 times "
        + "Then 4th time gives a limit error ")
void test4() throws Exception {
  ObjectNode transaction =
      getTransaction(
          amount: 1.0,  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: "KO", body.get("status").asText());
  assertEquals( expected: 1, body.withArray( propertyName: "errors").size());
```

Directory structure



```
src
  main
    java
      com.yellowpepper.challenge
        config
          DatabaseConfiguration
          DroolsConfiguration
        controller
          FixedDepositRateController
        domain
          AccountTransferValidationRequest
          TaxTransferRequest
        dto
          RequestCreateTransactionDto
          RequestRetrieveAccountDto
          ResponseBaseDto
          ResponseCreateTransactionDto
          ResponseRetrieveAccountDto
        exception
          AccountNotFoundException
          AccountTransfersLimitExceedException
          CurrencyNotSupportedException
          CustomerNotFoundException
          InsufficientFundsException
          RestExceptionHandler
          WrongInformationException
```

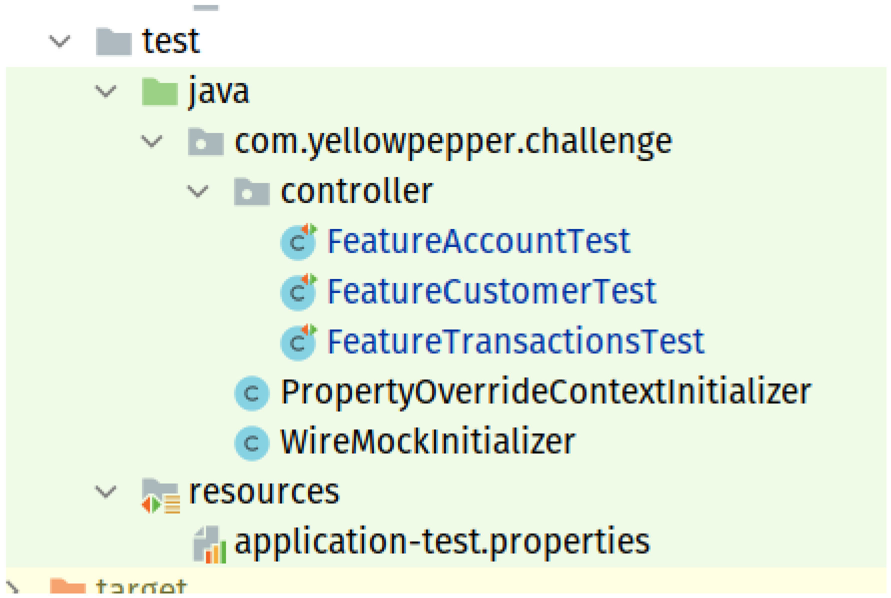
- Config: provides from general configuration at start of project like drools rules building and database population
- Controller: provides al the API resources with ther methods
- Domain: domain specific classes to read drools rules
- DTO: Objects of information transference
- Exception: Custom exceptions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



```
repository
  model
    Account
    Currency
    Customer
    Transfer
  AccountRepository
  CurrencyRepository
  CustomerRepository
  TransferRepository
service
  enums
    AvailabilityTransfer
  model
    CurrencyServiceResponse
    Rates
  AccountService
  CustomerService
  DroolsService
  ExchangeService
  TransferService
FundsTransferApplication
resources
  0-clean.sql
  1-ddl.sql
  2-dml.sql
  AccountTransferValidation.drl
  application.properties
  application-h2-file.properties
  application-postgres.properties
  TaxTransfer.drl
```

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

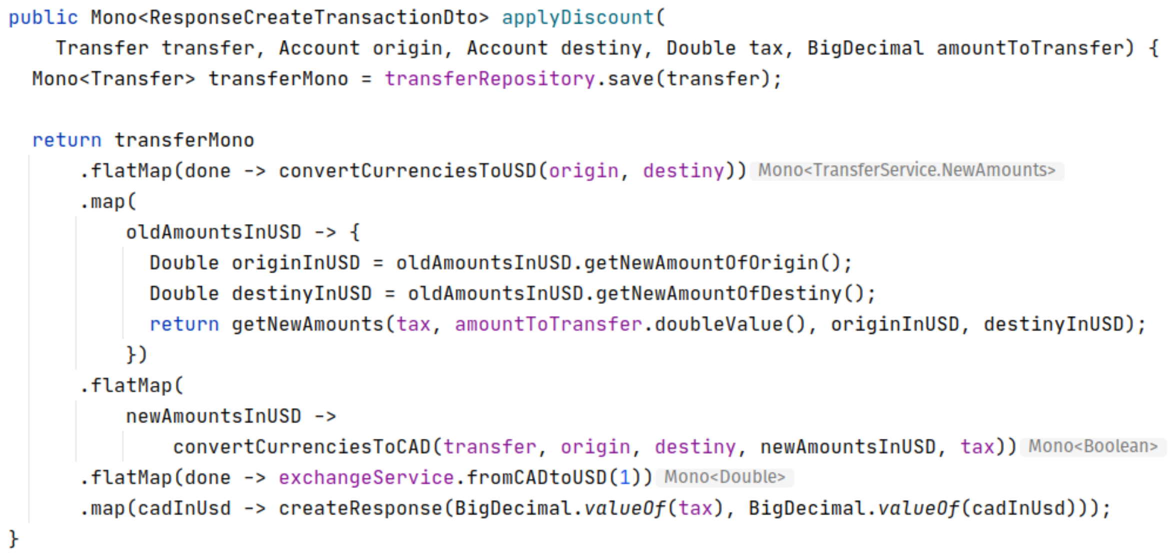
- Service.enums: Enums to be used on services
- Service.model: Just some POJOS to transfer info
- Service: All the application services are here
- Resources: resources of the app like SQL scripts for population, configuration files to choose between different database, drools rules (DRL)
-



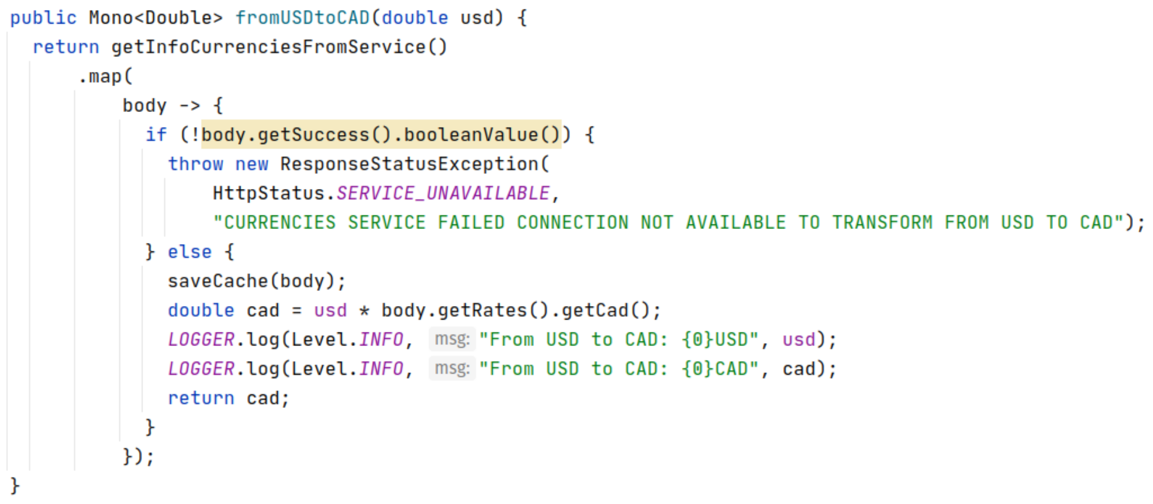
```
test
  java
    com.yellowpepper.challenge
      controller
        FeatureAccountTest
        FeatureCustomerTest
        FeatureTransactionsTest
      PropertyOverrideContextInitializer
      WireMockInitializer
  resources
    application-test.properties
target
```

- 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)));
}
```



```java
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;
            }
          });
}
```