

N704 – Programação Funcional - 2024

Luiz Rebouças Neto - 2317387-5

Murilo Henzo Mendonça Vaz - 2316214

Rhamanda Cathyanna Lima Paiva - 2317121-0

Pedro Vitor Teixeira Ribeiro – 2317122-8

FORTALEZA/CE

MARÇO/2024

Sumário

1. Introdução	3
2. Objetivos	3
3. Divisão de papéis da equipe	3
4. Documentação de requisitos	4
5. Construções Funcionais	6
6. Utilização de Chatbots	8
7. Casos de teste	27
8. Repositório	30

1. Introdução

Este relatório apresenta os resultados da atividade final da disciplina N704 – Programação Funcional, e dispõe a divisão de papéis definida pela equipe, bem como a descrição dos requisitos da aplicação implementada. A aplicação foi desenvolvida em Python.

A aplicação idealizada pela equipe consiste em um sistema de visualização de taxas de câmbio entre moedas, no qual é possível adicionar e listar taxas, e convertê-las entre si.

2. Objetivos

O objetivo da aplicação foi implementar conceitos de programação funcional, a saber:

- Funções lambda;
- *List comprehension*;
- Funções de continuação;
- *Closure*;
- Funções de alta ordem; e
- *Monads*.

3. Divisão de papéis da equipe

A divisão de papéis na equipe ficou definida do seguinte modo:

- **Luiz Rebouças:** desenvolvimento;
- **Murilo Vaz:** desenvolvimento;

- **Rhamanda Lima:** testes unitários da aplicação; e
- **Pedro Ribeiro:** documentação dos requisitos.

4. Documentação de requisitos

4.1 Requisitos Funcionais

- **Obter taxas de câmbio entre diferentes moedas:**
 - A função retorna todas as taxas de câmbio armazenados na base de dados;
 - Implementada pela função *find_all*, no arquivo *src/adapters/rest/CurrencyController.py*;

```
def find_all(self):
    currencies = self.currency_service.find_all()
    # uso de list comprehension
    currency_responses = [CurrencyRepresentationResponse(
        id=currency.id,
        name=currency.name,
        symbol=currency.symbol
    ).to_dict() for currency in currencies]
    return jsonify(currency_responses)
```

- Link: <https://shorturl.at/hmnll>
- **Registrar novas taxas de câmbio:**
 - A função recebe os dados na request e cria uma nova taxa de câmbio na base de dados;
 - Implementada pela função *create*, no arquivo *src/adapters/rest/CurrencyController.py*;

```
def create(self):
    data = request.json
    currency_request = CurrencyRepresentationRequest(**data)
    currency = CurrencyMapper.map_request_to_entity(currency_request)
    created_currency = self.currency_service.create(currency)
    response = CurrencyMapper.map_entity_to_response(created_currency)
    return jsonify(response.to_dict())
```

- Link: <https://shorturl.at/arOT3>

- **Obter taxas pelo nome:**

- A função recebe como parâmetro uma string e retorna uma taxa com este nome, caso encontre;
- Implementada pela função *find_by_name*, no arquivo *src/adapters/rest/CurrencyController.py*;

```
def find_by_name(self, name):
    #Closure para retornar moedas com o nome selecionado
    def get_currencies_by_name(currency):
        return currency.name == name
    currencies = self.currency_service.find_all()
    filtered_currencies = filter(get_currencies_by_name, currencies)
    currency_responses = [CurrencyRepresentationResponse(
        id=currency.id,
        name=currency.name,
        symbol=currency.symbol
    ).to_dict() for currency in filtered_currencies]
    return jsonify(currency_responses)
```

- Link: <https://shorturl.at/eBHST>

- **Obter taxas pelo ID:**

- A função recebe como parâmetro um int e retorna uma taxa com este ID, caso encontre;
- Implementada pela função *find_by_id*, no arquivo *src/adapters/rest/CurrencyController.py*;

```
def find_by_id(self, id):
    currency = self.currency_service.find_by_id(id)
    #uso de Monad
    maybe_currency = Maybe(currency)
    #uso de funções lambda juntamente com o monad
    return maybe_currency.bind(lambda currency: CurrencyMapper.map_entity_to_response(currency)
                              .to_dict()) \
                          .bind(lambda response_dict: jsonify(response_dict)) \
                          .value or jsonify({"error": "Currency not found"}),404
```

- Link: <https://shorturl.at/bKLV5>

4.2 Requisitos Não funcionais

- Possuir cobertura de testes igual ou superior a 80%:
 - Os testes unitários devem cobrir pelo menos 80% das funções;
 - Os testes podem ser encontrados no link:
<https://github.com/murilohenzo/currency-exchange-rate/tree/master/tests>
 - Evidência:

```
$ pytest --cov
platform win32 -- python 3.12.2, pytest-8.1.1, pluggy-1.4.0
rootdir: C:\Users\muril\workspace\projects\python\currency-exchange-rate
plugins: cov-4.1.0
collected 22 items

tests\test_currency_controller.py .... [ 100%]
tests\test_currency_mapper.py ..... [ 22%]
tests\test_currency_repository.py ... [ 40%]
tests\test_currency_representation_request.py . [ 45%]
tests\test_currency_representation_response.py .. [ 55%]
tests\test_currency_service.py ... [ 68%]
tests\test_monad.py ... [ 80%]
tests\test_sql_alchemy_currency_repository.py .... [ 100%]

----- coverage: platform win32, python 3.12.2-final-0 -----
Name                                         Stmts   Miss  Cover
-----
src\__init__.py                             0     0   100%
src\adapters\__init__.py                    0     0   100%
src\adapters\representation\currency_representation_request.py  4     0   100%
src\adapters\representation\currency_representation_response.py  7     0   100%
src\adapters\representation\__init__.py      0     0   100%
src\adapters\test\currency_controller.py     30     6    80%
src\adapters\test\__init__.py                0     0   100%
src\domain\__init__.py                       0     0   100%
src\domain\entities\base.py                  3     0   100%
src\domain\entities\currency.py             11     0   100%
src\domain\entities\__init__.py              0     0   100%
src\domain\repositories\currency_repository.py 13     3    77%
src\domain\repositories\__init__.py          0     0   100%
src\domain\service\currency_service.py       14     0   100%
src\domain\service\__init__.py               0     0   100%
src\infrastructure\__init__.py               0     0   100%
src\infrastructure\repositories\sqlalchemy_currency_repository.py 20     0   100%
src\infrastructure\repositories\__init__.py  0     0   100%
src\mappers\currency_mapper.py               10     0   100%
src\mappers\__init__.py                     0     0   100%
src\utils\__init__.py                       0     0   100%
src\utils\monad.py                          7     0   100%
src\utils\__init__.py                       0     0   100%
tests\__init__.py                           5     0   100%
tests\test_currency_controller.py            50     1    98%
tests\test_currency_mapper.py                17     1    94%
tests\test_currency_repository.py            26     1    96%
tests\test_currency_representation_request.py 11     1    91%
tests\test_currency_representation_response.py 21     1    95%
tests\test_currency_service.py               27     1    96%
tests\test_monad.py                          7     1    86%
tests\test_sql_alchemy_currency_repository.py 32     0   100%
TOTAL                                         325    16    95%

----- 22 passed in 1.04s -----
```

5. Construções Funcionais

5.1 Monad:

<https://github.com/murilohenzo/currency-exchange-rate/blob/master/src/adapters/rest/CurrencyController.py#L14>

```
#uso de Monad
maybe_currency = Maybe(currency)
```

5.2 Função Lambda:

<https://github.com/murilohenzo/currency-exchange-rate/blob/master/src/adapters/rest/CurrencyController.py#L16>

```
#uso de funções lambda juntamente com o monad
return maybe_currency.bind(lambda currency: CurrencyMapper.map_entity_to_response(currency)
                           .to_dict()) \
                           .bind(lambda response_dict: jsonify(response_dict)) \
                           .value or jsonify({"error": "Currency not found"}), 404
```

5.3 List Comprehension:

<https://github.com/murilohenzo/currency-exchange-rate/blob/master/src/adapters/rest/CurrencyController.py#L24>

```
# uso de list comprehension
currency_responses = [CurrencyRepresentationResponse(
    id=currency.id,
    name=currency.name,
    symbol=currency.symbol
).to_dict() for currency in currencies]
return jsonify(currency_responses)
```

5.4 Closure:

<https://github.com/murilohenzo/currency-exchange-rate/blob/master/src/adapters/rest/CurrencyController.py#L33>

```
#Closure para retornar moedas com o nome selecionado
def get_currencies_by_name(currency):
    return currency.name == name
currencies = self.currency_service.find_all()
filtered_currencies = filter(get_currencies_by_name, currencies)
currency_responses = [CurrencyRepresentationResponse(
    id=currency.id,
    name=currency.name,
    symbol=currency.symbol
).to_dict() for currency in filtered_currencies]
return jsonify(currency_responses)
```

5.5 Função de Alta Ordem:

<https://github.com/murilohenzo/currency-exchange-rate/blob/master/src/adapters/rest/CurrencyController.py#L36>

```
filtered_currencies = filter(get_currencies_by_name, currencies)
```

6. Utilização de Chatbots

Foi utilizado o auxílio do ChatGPT para desenvolver algumas partes do código, a saber:

6.1 Testes unitários:

6.1.1 Para a classe *currencyService*, a seguinte pesquisa foi realizada junto ao ChatGPT:

- **Pergunta:**

CHAT GPT - CRIACAO DE TESTE UNITARIO COM MOCKS

crie testes unitarios para a seguinte classe

```
from pymysql import IntegrityError
from sqlalchemy.orm import Session
from typing import Optional, List
from domain.entities.Currency import Currency
```



```
from domain.repositories.CurrencyRepository import
CurrencyRepository

class SQLAlchemyCurrencyRepository(CurrencyRepository):
    def __init__(self, session: Session):
        self.session = session

    def find_by_id(self, id: int) -> Optional[Currency]:
        return self.session.query(Currency).filter_by(id=id).first()

    def find_all(self) -> List[Currency]:
        return self.session.query(Currency).all()

    def create(self, currency: Currency) -> Currency:
        try:
            self.session.add(currency)
            self.session.commit()
            return currency
        except IntegrityError as e:
            print(f"[E23] - CurrencyViolationError: {e.orig}")
            self.session.rollback()
```

- **Resposta:**

```
import unittest

from unittest.mock import MagicMock
from sqlalchemy.orm import Session
from sqlalchemy.exc import IntegrityError
from your_module import SQLAlchemyCurrencyRepository, Currency

class TestSQLAlchemyCurrencyRepository(unittest.TestCase):
    def setUp(self):
```

```
self.mock_session = MagicMock(spec=Session)

self.repo = SQLAlchemyCurrencyRepository(self.mock_session)

def test_find_by_id(self):
    # Mocking data
    expected_currency = Currency(id=1, name='USD', code='USD')

    # Mocking session.query().filter_by().first() method
    self.mock_session.query.return_value.filter_by.return_value.first.return_value = expected_currency

    # Test
    result = self.repo.find_by_id(1)

    # Assertion
    self.assertEqual(result, expected_currency)

def test_find_all(self):
    # Mocking data
    expected_currencies = [Currency(id=1, name='USD', code='USD'), Currency(id=2, name='EUR', code='EUR')]

    # Mocking session.query().all() method
    self.mock_session.query.return_value.all.return_value = expected_currencies

    # Test
    result = self.repo.find_all()

    # Assertion
    self.assertEqual(result, expected_currencies)
```

```
def test_create(self):
    # Mocking data
    currency = Currency(id=1, name='USD', code='USD')

    # Mocking session.add() and session.commit() methods
    self.mock_session.add.return_value = None
    self.mock_session.commit.return_value = None

    # Test
    result = self.repo.create(currency)

    # Assertion
    self.assertEqual(result, currency)

def test_create_with_integrity_error(self):
    # Mocking data
    currency = Currency(id=1, name='USD', code='USD')

    # Mocking session.add() and session.commit() methods to
    raise IntegrityError
    self.mock_session.add.side_effect = IntegrityError(' ', ' ',
    '')

    # Test
    result = self.repo.create(currency)

    # Assertion
    self.assertIsNone(result)

if __name__ == '__main__':
    unittest.main()
```

6.1.2 Para a classe `currencyService`, a seguinte pesquisa foi realizada junto ao ChatGPT:

- Pergunta:

Implemente os testes unitários da seguinte classe:

```
from typing import List
from domain.entities.Currency import Currency
from domain.repositories.CurrencyRepository import CurrencyRepository

class CurrencyService:
    def __init__(self, currency_repository: CurrencyRepository):
        self.currency_repository = currency_repository

    def find_by_id(self, id: int) -> Currency:
        currency = self.currency_repository.find_by_id(id)
        return currency

    def find_all(self) -> List[Currency]:
        currencies = self.currency_repository.find_all()
        return currencies

    def create(self, currency: Currency) -> Currency:
        return self.currency_repository.create(currency)
```

- Resposta:

Para testar a classe `CurrencyService`, você precisa criar casos de teste para cada um dos métodos `find_by_id`, `find_all`, e `create`. Aqui está uma implementação possível dos testes usando o módulo `unittest`:

```
import unittest
```

```
from unittest.mock import MagicMock
from domain.entities.Currency import Currency
from domain.repositories.CurrencyRepository import CurrencyRepository
from your_module import CurrencyService

class TestCurrencyService(unittest.TestCase):
    def setUp(self):
        self.mock_repository = MagicMock(spec=CurrencyRepository)
        self.currency_service = CurrencyService(self.mock_repository)

    def test_find_by_id_existing_currency(self):
        # Mocking the return value of the repository method
        expected_currency = Currency(id=1, name="USD", symbol="$")
        self.mock_repository.find_by_id.return_value = expected_currency

        # Testing the method
        result = self.currency_service.find_by_id(1)

        # Asserting that the result is as expected
        self.assertEqual(result, expected_currency)

        # Asserting that the repository method was called with the
        correct argument
        self.mock_repository.find_by_id.assert_called_once_with(1)

    def test_find_by_id_non_existing_currency(self):
        # Mocking the return value of the repository method
        self.mock_repository.find_by_id.return_value = None

        # Testing the method
        result = self.currency_service.find_by_id(1)

        # Asserting that the result is None
```

```
self.assertIsNone(result)

# Asserting that the repository method was called with the
correct argument
self.mock_repository.find_by_id.assert_called_once_with(1)

def test_find_all(self):
    # Mocking the return value of the repository method
    expected_currencies = [
        Currency(id=1, name="USD", symbol="$"),
        Currency(id=2, name="EUR", symbol="€")
    ]

    self.mock_repository.find_all.return_value =
expected_currencies

    # Testing the method
    result = self.currency_service.find_all()

    # Asserting that the result is as expected
    self.assertEqual(result, expected_currencies)
    # Asserting that the repository method was called once
    self.mock_repository.find_all.assert_called_once()

def test_create(self):
    # Creating a new currency object
    new_currency = Currency(id=None, name="GBP", symbol="£")
    # Mocking the return value of the repository method
    self.mock_repository.create.return_value = new_currency

    # Testing the method
    result = self.currency_service.create(new_currency)

    # Asserting that the result is as expected
    self.assertEqual(result, new_currency)
```

```
        # Asserting that the repository method was called with the
        correct argument

self.mock_repository.create.assert_called_once_with(new_currency)

if __name__ == '__main__':
    unittest.main()
```

Esses testes cobrem diferentes cenários para os métodos `find_by_id`, `find_all`, e `create` da classe `CurrencyService`, garantindo que o comportamento esperado seja mantido.

6.1.3 Para a classe *Maybe*, a seguinte pesquisa foi realizada junto ao ChatGPT:

- Pergunta:

Implemente os testes unitários da seguinte classe:

```
#criação de monad para ser implementada
class Maybe:
    def __init__(self, value):
        self.value = value

    def bind(self, function):
        if self.value is None:
            return Maybe(None)
        else:
            return Maybe(function(self.value))
```

- Resposta:

```
import unittest

from your_module import Maybe
```

```
class TestMaybe(unittest.TestCase):
    def test_bind_with_value(self):
        maybe = Maybe(5)
        result = maybe.bind(lambda x: x * 2)

        self.assertEqual(result.value, 10)

    def test_bind_with_none_value(self):
        maybe = Maybe(None)
        result = maybe.bind(lambda x: x * 2)

        self.assertIsNone(result.value)

    def test_bind_chaining(self):
        maybe = Maybe(5)
        result = maybe.bind(lambda x: x * 2).bind(lambda x: x + 3)

        self.assertEqual(result.value, 13)

if __name__ == '__main__':
    unittest.main()
```

6.1.4 Para a classe *CurrencyRepresentationResponse*, a seguinte pesquisa foi realizada junto ao ChatGPT:

- Pergunta:

Implemente os testes unitários da seguinte classe:

```
class CurrencyRepresentationResponse:
    def __init__(self, id: int, name: str, symbol: str):
        self.id = id
        self.name = name
```



```
self.symbol = symbol

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'symbol': self.symbol
        }
```

- Resposta:

```
import unittest
from your_module import CurrencyRepresentationResponse

class TestCurrencyRepresentationResponse(unittest.TestCase):
    def test_init(self):
        id = 1
        name = "USD"
        symbol = "$"

        response = CurrencyRepresentationResponse(id, name,
symbol)

        self.assertEqual(response.id, id)
        self.assertEqual(response.name, name)
        self.assertEqual(response.symbol, symbol)

    def test_to_dict(self):
        id = 1
        name = "USD"
        symbol = "$"
        expected_dict = {'id': id, 'name': name, 'symbol': symbol}
```

```
        response = CurrencyRepresentationResponse(id, name,
symbol)

        self.assertEqual(response.to_dict(), expected_dict)

if __name__ == '__main__':
    unittest.main()
```

6.1.5 Para a classe *CurrencyRepresentationRequest*, a seguinte pesquisa foi realizada junto ao ChatGPT:

- Pergunta:

Implemente os testes unitários da seguinte classe:

```
class CurrencyRepresentationRequest:
def __init__(self, name: str, symbol: str):
self.name = name
self.symbol = symbol
```

- Resposta:

Aqui estão os testes unitários para a classe *CurrencyRepresentationRequest*:

```
python
Copy code
import unittest
from your_module import CurrencyRepresentationRequest

class TestCurrencyRepresentationRequest(unittest.TestCase):
def test_init(self):
name = "USD"
symbol = "$"
```

```
request = CurrencyRepresentationRequest(name, symbol)

self.assertEqual(request.name, name)
self.assertEqual(request.symbol, symbol)

if **name** == '**main**':
unittesttest.main()
```

Este teste garante que a inicialização da classe `CurrencyRepresentationRequest` funciona conforme o esperado. Certifique-se de substituir `your_module` pelo nome real do módulo onde está definida a classe `CurrencyRepresentationRequest`.

6.1.6 Para a classe `CurrencyController`, a seguinte pesquisa foi realizada junto ao ChatGPT:

- Pergunta:

Implemente os testes unitários da seguinte classe:

```
from flask import jsonify, request
from adapters.representation.CurrencyRepresentationRequest import
CurrencyRepresentationRequest
from utils.Monad import Maybe
from adapters.representation.CurrencyRepresentationResponse import
CurrencyRepresentationResponse
from mappers.CurrencyMapper import CurrencyMapper

class CurrencyController:
def **init**(self, currency_service):
self.currency_service = currency_service

def find_by_id(self, id):
currency = self.currency_service.find_by_id(id)
#uso de Monad
```

```
maybe_currency = Maybe(currency)

#uso de funções lambda juntamente com o monad
return maybe_currency.bind(lambda currency:
CurrencyMapper.map_entity_to_response(currency)
.to_dict()) \
    .bind(lambda response_dict:
jsonify(response_dict))\
    .value or jsonify({"error":
"Currency not found"}),404

def find_all(self):
    currencies = self.currency_service.find_all()
    # uso de list comprehension
    currency_responses = [CurrencyRepresentationResponse(
        id=currency.id,
        name=currency.name,
        symbol=currency.symbol
    ).to_dict() for currency in currencies]
    return jsonify(currency_responses)

def find_by_name(self, name):
    #Closure para retornar moedas com o nome selecionado
    def get_currencies_by_name(currency):
        return currency.name == name
    currencies = self.currency_service.find_all()
    filtered_currencies = filter(get_currencies_by_name,
currencies)
    currency_responses = [CurrencyRepresentationResponse(
        id=currency.id,
        name=currency.name,
        symbol=currency.symbol
    ).to_dict() for currency in filtered_currencies]
    return jsonify(currency_responses)
```

```
def create(self):
    data = request.json
    currency_request = CurrencyRepresentationRequest(**data)
    currency = CurrencyMapper.map_request_to_entity(currency_request)
    created_currency = self.currency_service.create(currency)
    response = CurrencyMapper.map_entity_to_response(created_currency)
    return jsonify(response.to_dict())
```

- Resposta:

Aqui estão os testes unitários para a classe CurrencyController:

```
python
Copy code
import unittest
from unittest.mock import MagicMock, patch
from flask import jsonify
from your_module import CurrencyController, CurrencyRepresentationRequest, CurrencyRepresentationResponse, CurrencyMapper

class TestCurrencyController(unittest.TestCase):
    def setUp(self):
        self.mock_currency_service = MagicMock()
        self.currency_controller = CurrencyController(self.mock_currency_service)

    def test_find_by_id(self):
        currency_id = 1
        mock_currency = MagicMock()
        self.mock_currency_service.find_by_id.return_value = mock_currency
```

```
        mock_response_dict = {"id": 1, "name": "USD", "symbol":
"$"}

        expected_response = jsonify(mock_response_dict)

        with patch.object(CurrencyMapper,
'map_entity_to_response',
return_value=CurrencyRepresentationResponse(1, "USD", "$")):
            result =
self.currency_controller.find_by_id(currency_id)

        self.assertEqual(result, expected_response)

    def test_find_all(self):
        mock_currencies = [MagicMock(id=1, name="USD",
symbol="$"), MagicMock(id=2, name="EUR", symbol="€")]
        self.mock_currency_service.find_all.return_value =
mock_currencies

        mock_response_dicts = [{"id": 1, "name": "USD", "symbol":
"$"}, {"id": 2, "name": "EUR", "symbol": "€"}]
        expected_response = jsonify(mock_response_dicts)

        result = self.currency_controller.find_all()

        self.assertEqual(result, expected_response)

    def test_find_by_name(self):
        mock_currencies = [MagicMock(id=1, name="USD",
symbol="$"), MagicMock(id=2, name="USD", symbol="€")]
        self.mock_currency_service.find_all.return_value =
mock_currencies

        mock_response_dicts = [{"id": 1, "name": "USD", "symbol":
"$"}, {"id": 2, "name": "USD", "symbol": "€"}]
        expected_response = jsonify(mock_response_dicts)

        result = self.currency_controller.find_by_name("USD")
```

```
self.assertEqual(result, expected_response)

def test_create(self):
    mock_request_data = {"name": "USD", "symbol": "$"}
    mock_currency_request = CurrencyRepresentationRequest("USD", "$")
    mock_created_currency = MagicMock(id=1, name="USD", symbol="$")
    self.mock_currency_service.create.return_value = mock_created_currency
    mock_response_dict = {"id": 1, "name": "USD", "symbol": "$"}
    expected_response = jsonify(mock_response_dict)

    with patch.object(CurrencyMapper, 'map_request_to_entity', \
                      patch.object(CurrencyMapper, 'map_entity_to_response', \
                      return_value=CurrencyRepresentationResponse(1, "USD", "$")):
        with patch('your_module.request') as mock_request:
            mock_request.json.return_value = mock_request_data
            result = self.currency_controller.create()

    self.assertEqual(result, expected_response)

if __name__ == '__main__':
    unittest.main()
```

Estes testes garantem que os métodos da classe `CurrencyController` funcionam conforme esperado, incluindo a manipulação de requisições e respostas do Flask, bem como a interação com o serviço de moeda e o mapeador de moeda. Certifique-se de substituir `your_module` pelo nome real do módulo onde está definida a classe `CurrencyController`.

6.1.7 Para a classe *CurrencyMapper*, a seguinte pesquisa foi realizada junto ao ChatGPT:

- Pergunta:

Implemente os testes unitários da seguinte classe:

```
from adapters.representation.CurrencyRepresentationRequest import CurrencyRepresentationRequest
from adapters.representation.CurrencyRepresentationResponse import CurrencyRepresentationResponse
from domain.entities.Currency import Currency

class CurrencyMapper:
    @staticmethod
    def map_request_to_entity(request_data: CurrencyRepresentationRequest) -> Currency:
        return Currency(
            id=None,
            name=request_data.name,
            symbol=request_data.symbol
        )

    @staticmethod
    def map_entity_to_response(currency: Currency) -> CurrencyRepresentationResponse:
        return CurrencyRepresentationResponse(
            id=currency.id,
            name=currency.name,
            symbol=currency.symbol
        )
```


- Resposta:

Para testar a classe `CurrencyMapper`, podemos criar casos de teste para os métodos `map_request_to_entity` e `map_entity_to_response`. Aqui está uma implementação dos testes unitários usando o módulo `unittest`:

```
python
Copy code
import unittest

from your_module import CurrencyMapper, CurrencyRepresentationRequest, CurrencyRepresentationResponse, Currency

class TestCurrencyMapper(unittest.TestCase):
    def test_map_request_to_entity(self):
        request_data = CurrencyRepresentationRequest(name="USD", symbol="$")

        result = CurrencyMapper.map_request_to_entity(request_data)

        self.assertIsInstance(result, Currency)
        self.assertEqual(result.name, "USD")
        self.assertEqual(result.symbol, "$")
        self.assertIsNone(result.id)

    def test_map_entity_to_response(self):
        currency = Currency(id=1, name="USD", symbol="$")

        result = CurrencyMapper.map_entity_to_response(currency)

        self.assertIsInstance(result, CurrencyRepresentationResponse)
        self.assertEqual(result.id, 1)
        self.assertEqual(result.name, "USD")
        self.assertEqual(result.symbol, "$")
```

```
if **name** == '**main**':  
    unittest.main()
```

Esses testes garantem que os métodos `map_request_to_entity` e `map_entity_to_response` da classe `CurrencyMapper` funcionam conforme esperado, mapeando corretamente os dados entre as representações de requisição e resposta e as entidades de domínio. Certifique-se de substituir `your_module` pelo nome real do módulo onde está definida a classe `CurrencyMapper`.

6.2 Implementação de algumas estruturas funcionais:

6.2.1 Usado na construção da classe *Maybe* no caminho *src/domain/service/Monad.py*

- **Pergunta:**

Me explique o que são monads e como eu posso usar, assim com um exemplo, o mais simples possível em python

- **Resposta:**

Um monad é um conceito da teoria da categoria e da programação funcional que oferece uma maneira de encapsular valores com algum contexto computacional e fornecer operações seguras e com efeitos colaterais controlados. Eles são amplamente utilizados em linguagens funcionais, como Haskell, mas também podem ser implementados em linguagens orientadas a objetos como Python.

O conceito-chave do monad é a operação `bind` (também conhecida como `flatMap` em outras linguagens), que permite encadear várias operações de forma segura, passando o resultado de uma operação para a próxima.

Aqui está uma explicação básica de como você pode usar monads em Python, com um exemplo simples:

Suponha que temos uma função `divide_por_dois` que pode dividir um número por 2, mas que retorna `None` se o número for ímpar. Queremos encadear essa função com outra que adiciona 10 ao resultado, mas só queremos adicionar 10 se a divisão for bem-sucedida.

Vamos começar definindo uma classe Maybe para encapsular nossos valores com contexto adicional:

código:

```
class Maybe:
    def __init__(self, valor):
        self.valor = valor

    def bind(self, funcao):
        if self.valor is None:
            return Maybe(None)
        else:
            return Maybe(funcao(self.valor))

    def __repr__(self):
        return f'Maybe({self.valor})'
```

6.2.2 Implementação do Monad no caminho src/adapters/rest/currencyController.py no método find_by_id linha 10

- **Pergunta:**

partindo da classe maybe, como eu poderia usar e implementar numa chamada dentro de um controller flask, por exemplo uma chamada e pesquisa de um id de currency?

- **Resposta:**

Para implementar a classe Maybe em uma chamada dentro de um controller Flask para pesquisar um ID de moeda, você pode encapsular o resultado da operação de pesquisa em um objeto Maybe. Em seguida, você pode encadear operações adicionais usando o método bind, verificando se o valor encapsulado é None ou não. Aqui está um exemplo de como você pode fazer isso:

código:

```
from flask import jsonify
```

```
from domain.service import CurrencyService
from monad import Maybe

class CurrencyController:
    def __init__(self):
        self.currency_service = CurrencyService() # Suponha que
        # você tenha uma classe CurrencyService que fornece os dados

    def find_currency_by_id(self, id):
        # Encapsula o resultado da operação de pesquisa em um
        # objeto Maybe
        maybe_currency =
        Maybe(self.currency_service.find_by_id(id))

        # Encadeia operações adicionais usando bind
        return maybe_currency.bind(lambda currency: jsonify({"id":
        currency.id, "name": currency.name, "type": currency.type})) \
            .value or jsonify({"error": "Currency not found"}),
404
```

continuação resposta:

Neste exemplo:

A função `find_currency_by_id` recebe o ID da moeda como parâmetro.

O resultado da operação `self.currency_service.find_by_id(id)` é encapsulado em um objeto `Maybe`.

Em seguida, é usada uma função `lambda` para mapear o objeto `Currency` para um objeto JSON (`jsonify`) se o objeto `Maybe` não for `None`.

O método `bind` é usado para encadear essas operações, garantindo que elas sejam realizadas apenas se o valor encapsulado não for `None`.

Finalmente, é retornado o valor encapsulado usando `.value`, ou, se for `None`, é retornado um JSON com uma mensagem de erro e o código de status 404.

6.2.3 Implementação do closure no método `find_by_name` no caminho `src/adapters/rest/currencyController.py`

- **Pergunta:**

como eu posso facilmente colocar um closure partindo do ponto que estou criando uma api, pode ser um closure de função chamando um método tipo `find_by_name`

- **Resposta:**

Para criar um closure que encapsule um método como `find_by_name` dentro do contexto de uma API Flask, você pode definir uma função dentro do método de seu controller Flask e referenciar o método desejado dentro dessa função interna. Aqui está um exemplo de como você pode fazer isso:

código:

```
python
Copy code
from flask import jsonify
from domain.service import CurrencyService

class CurrencyController:
    def __init__(self):
        self.currency_service = CurrencyService() # Suponha que
        # você tenha uma classe CurrencyService que fornece os dados

    def find_by_name_closure(self, name):
        def find_by_name():
            currencies = self.currency_service.find_by_name(name)
            return jsonify([{"id": currency.id, "name":
currency.name, "type": currency.type} for currency in currencies])
        return find_by_name
```

Neste exemplo:

continuação da resposta:

A função `find_by_name_closure` cria um closure que encapsula o método `find_by_name` dentro do contexto do controller.

Dentro do closure, é definida uma função interna chamada `find_by_name`, que chama o método `find_by_name` e retorna o resultado em um formato JSON.

O closure é então retornado pela função `find_by_name_closure`, permitindo que ele seja usado em outro lugar, como em um roteamento Flask.

7. Casos de teste

7.1. Casos de teste da classe `SQLAlchemyCurrencyRepository`:

- **`test_find_by_id`:**

É realizada a verificação do método `find_by_id` da classe `SQLAlchemyCurrencyRepository`, o qual deve retornar a moeda correta quando recebe um ID válido.

- **`test_find_all`:**

É realizada a verificação do método `find_all` da classe `SQLAlchemyCurrencyRepository`, o qual deve retornar todas as moedas presentes no banco de dados.

- **`test_create`:**

Neste, verifica-se se o método `create` da classe `SQLAlchemyCurrencyRepository` cria uma nova moeda corretamente.

- **`test_create_with_integrity_error`:**

É realizada a verificação do modo como método `create` da classe `SQLAlchemyCurrencyRepository` lida um erro de integridade.

7.2. Casos de teste da classe Maybe:

- *test_bind_with_value:*

Verifica-se o método bind da classe Maybe retorna o valor correto.

- *test_bind_with_none_value:*

Verifica-se o método bind da classe Maybe, quando o valor a ser retornado é None.

- *test_bind_chaining:*

Verifica-se o método bind da classe Maybe ao lidar com um encadeamento.

7.3. Casos de teste da classe CurrencyService:

- *test_find_by_id:*

Verifica-se o método find_by_id da classe CurrencyService, o qual deve retornar a moeda correta ao receber um ID válido.

- *test_find_all:*

Verifica-se o método find_all da classe CurrencyService, o qual deve retornar todas as moedas presentes no repositório.

- *test_create:*

Verifica-se o método create da classe CurrencyService, o qual deve criar uma moeda corretamente.

7.4. Casos de teste da classe CurrencyRepresentationResponse:

- *test_init:*

Verifica-se se o método **init** da classe CurrencyRepresentationResponse inicializa os atributos corretamente.

- *test_to_dict:*

Verifica-se se o método **to_dict** da classe CurrencyRepresentationResponse retorna um dicionário corretamente formatado.

7.5. Casos de teste da classe CurrencyRepresentationRequest:

- *test_init:*

Verifica-se se os atributos estão sendo inicializados conforme o esperado.

7.6. Casos de teste da classe CurrencyMapper:

- *test_map_request_to_entity:*

Verifica-se o método **map_request_to_entity** da classe CurrencyMapper, o qual deve mapear corretamente uma instância de CurrencyRepresentationRequest para uma instância de Currency.

- *test_map_entity_to_response:*

Verifica-se o método **map_entity_to_response** da classe CurrencyMapper, o qual deve mapear corretamente uma instância de Currency para uma instância de CurrencyRepresentationResponse.

7.7. Casos de teste da classe CurrencyController:

- test_find_by_id:

Verifica-se o método find_by_id do controlador de moeda (CurrencyController), o qual retorna uma resposta HTTP adequada e os dados corretos da moeda quando é fornecido um ID válido.

- test_find_all:

Verifica-se o método find_all do controlador de moeda, o qual deve retornar uma resposta HTTP adequada com uma lista de moedas quando é chamado.

- test_find_by_name:

Verifica-se o método find_by_name do controlador de moeda, o qual retorna uma resposta HTTP adequada com as moedas correspondentes ao nome fornecido.

- test_create_with_error:

Verifica-se o método create do controlador de moeda, o qual retorna um código de status 404 (Not Found) quando ocorre um erro durante a criação da moeda.

8. Repositório

Repositório da aplicação:

<https://github.com/murilohenzo/currency-exchange-rate/tree/master>