Jakub Ciszak

Praktyczne podejście do testowania w środowisku mikroserwisów

O czym będziemy rozmawiać?

- Testy jednostkowe

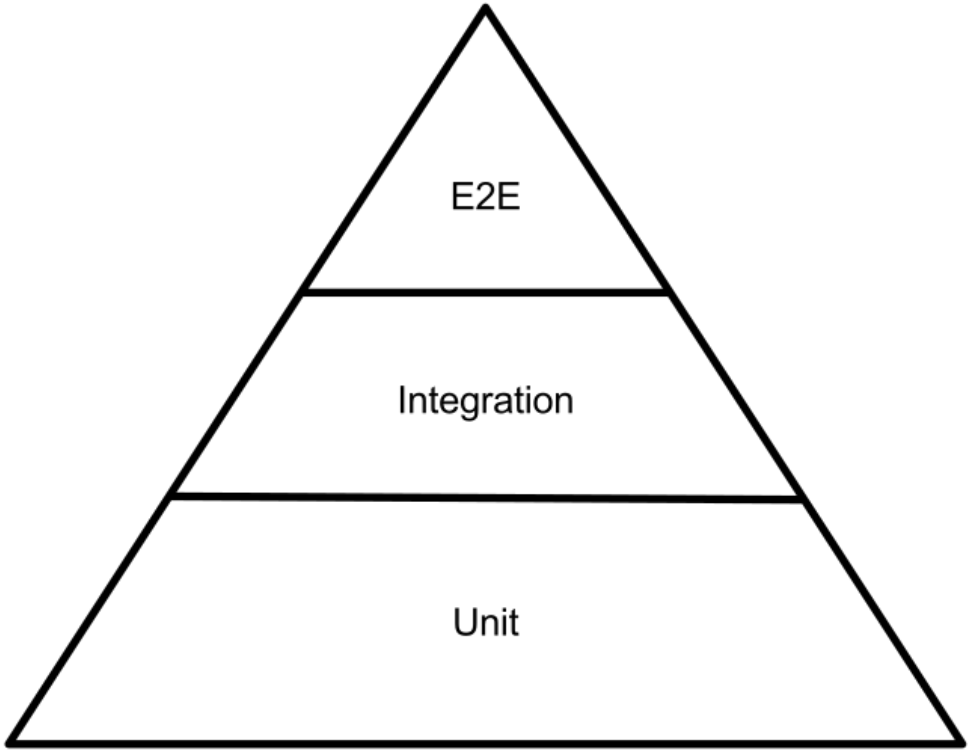
  Do jakich części systemu jaki rodzaj testów?

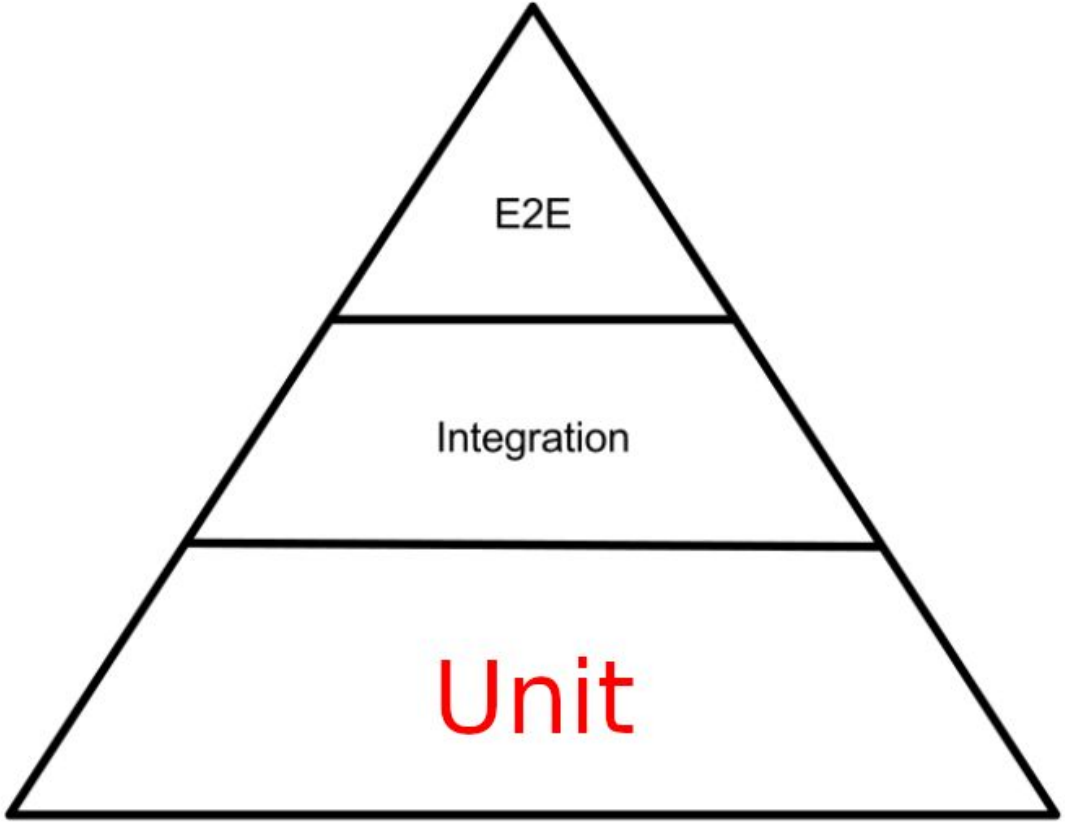
- Testy Integracyjne

  Jak radzić sobie z zewnętrznymi zależnościami

  Testy poprzez kontrakty

- Przykłady testów end-to-end


E2E
Integration
Unit


E2E
Integration
Unit

Czym jest jednostka?

”

Test jednostkowy to weryfikacja poprawności działania pojedynczych elementów ***(jednostek) programu – np. metod lub obiektów w programowaniu obiektowym lub procedur w programowaniu proceduralnym.***

Wikipedia

”

Testy jednostkowe są to małe kawałki kodu,

które służą do testowania innego kodu,

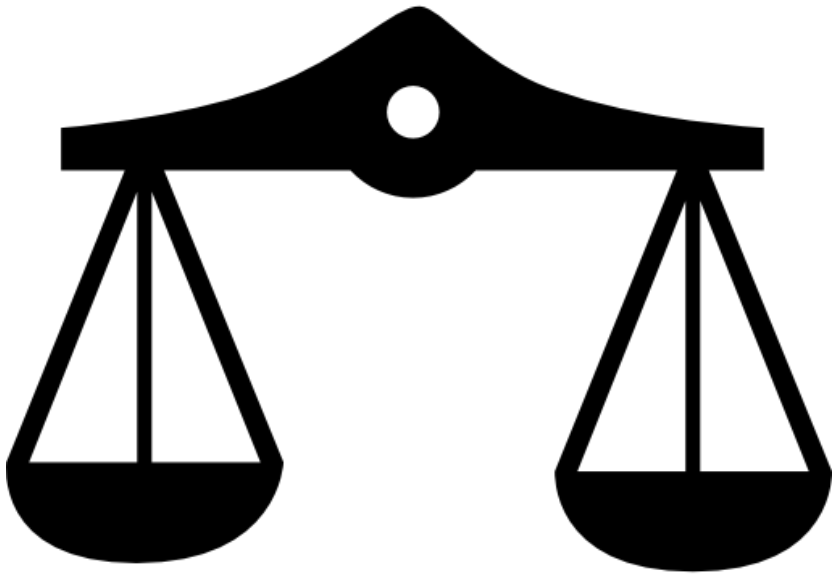
*czyli pojedynczych **jednostek**, to znaczy **klas** i **metod.***

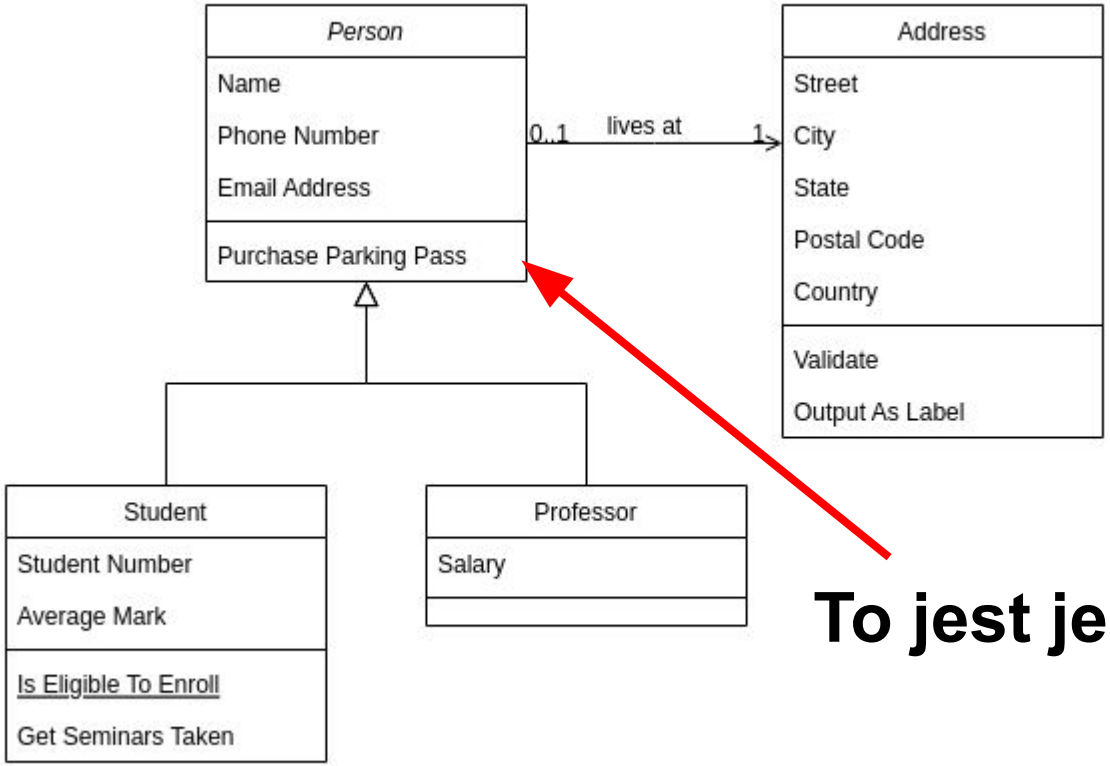
Kazimierz Szpin, www.modestprogrammer.pl

”

*Testy nie powinny weryfikować jednostek kodu. Zamiast tego powinny weryfikować **jednostki zachowania** - element, który ma znaczenie dla domeny problemu. (...) Liczba klas jaka jest potrzebna do zaimplementowania takiej jednostki zachowania, nie ma znaczenia. Taka jednostka może rozciągać się na wiele klas, mieścić w jednej lub zajmować wyłącznie jedną małą metodę.*

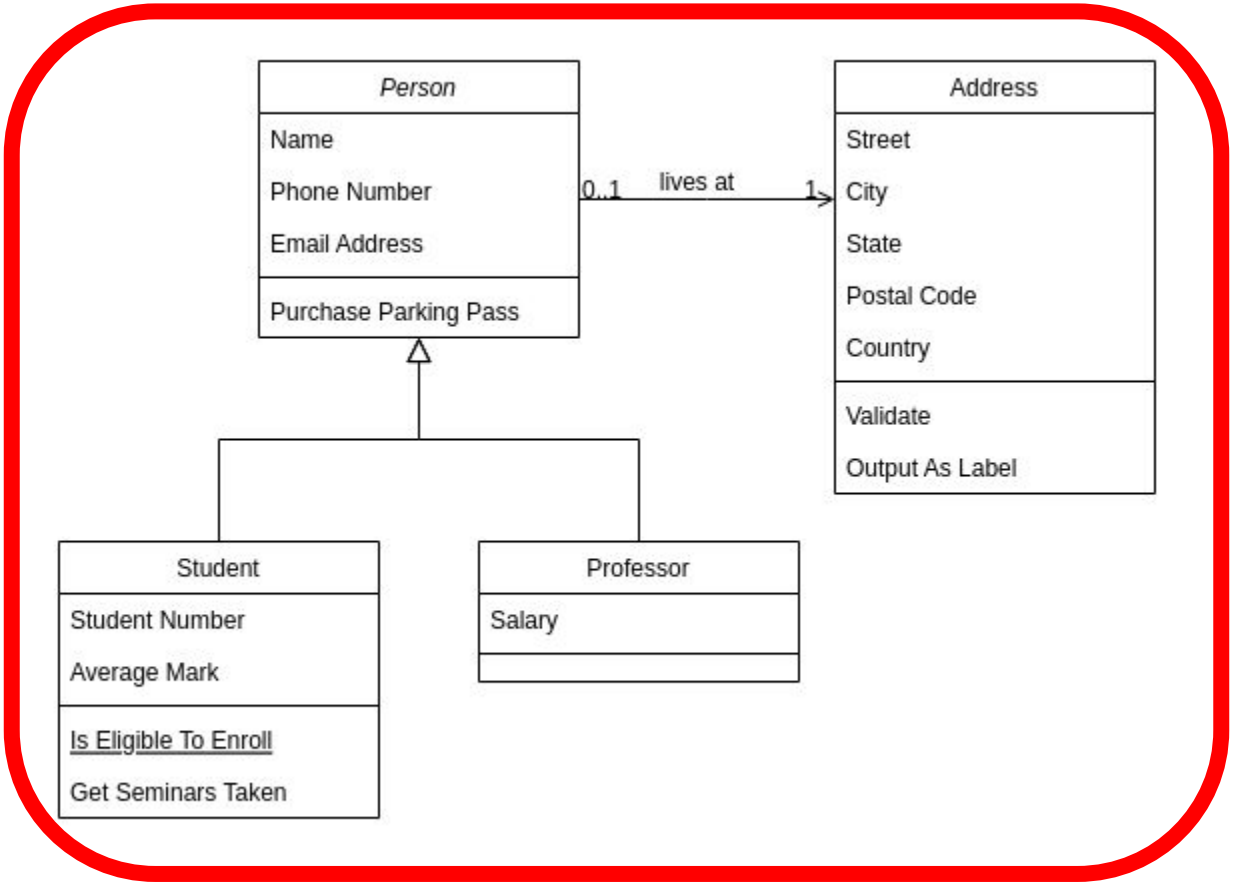
Vladimir Khorikov,
Testy Jednostkowe - zasady, praktyki, wzorce,
Helion 2020





To jest jednostka

Szkoła londyńska



To jest jednostka

Szkoła z Detroit

```php
public function test_Exchange(): void
{
    $bank = $this->createMock( originalClassName: Bank::class);
    $bank->expects($this->once())->method( constraint: 'exchange')->willReturn( value: 2306);
    $money = new Money(new Amount( value: 10000),  currency: Currency::PLN);

    $result = $money->exchangeTo( newCurrency: Currency::USD, $bank);

    self::assertEquals( expected: 2306, $result->amount()->value());
    self::assertEquals( expected: Currency::USD, $result->currency());
}
```

```
Time: 00:00.047, Memory: 6.00 MB

OK (1 test, 3 assertions)

Process finished with exit code 0
```

```php
class Money
{
    public function __construct(
        private Amount $amount,
        private Currency $currency
    ) {
    }

    public function exchangeTo(Currency $newCurrency, Bank $bank): Money
    {
        $value = $bank->exchange($this->amount(), $this->currency(), $newCurrency);
        return new Money(new Amount($value), $newCurrency);
    }

    public function amount(): Amount{...}

    public function currency(): Currency{...}
}
```

```php
class Bank
{
    public function __construct(private readonly array $rates)
    {
    }

    public function exchange(Amount $amount, Currency $fromCurrency, Currency $toCurrency): int
    {
        $rate = $this->getRate($fromCurrency, $toCurrency);
        return (int)round( num: $amount->value() * $rate);
    }

    private function getKeyPair(Currency $fromCurrency, Currency $toCurrency): string{...}

    private function getRate(Currency $fromCurrency, Currency $toCurrency): float{...}
}
```

```php
class Bank
{
    public function __construct(private readonly array $rates){...}

    public function exchange(Amount $amount, Currency $fromCurrency, Currency $toCurrency): Money
    {
        $rate = $this->getRate($fromCurrency, $toCurrency);
        $newAmount = $this->getNewAmount($amount, $rate);
        return new Money($newAmount, $toCurrency);
    }

    private function getRate(Currency $fromCurrency, Currency $toCurrency): float{...}

    private function getKeyPair(Currency $fromCurrency, Currency $toCurrency): string{...}

    private function getNewAmount(Amount $amount, float $rate): Amount
    {
        $value = (int)round( num: $amount->value() * $rate);
        return new Amount($value);
    }
}
```

```php
class Money
{
    public function __construct(
        private Amount $amount,
        private Currency $currency
    ) {
    }

    public function exchangeTo(Currency $newCurrency, Bank $bank): Money
    {
        return $bank->exchange($this->amount(), $this->currency(), $newCurrency);
    }

    public function amount(): Amount{...}

    public function currency(): Currency{...}
}
```

Method exchange may not return value of type integer, its declared return type is "Przelewy24\TestExamples\Domain\MoneyMarket\Money"

[redacted]TestExamples/tests/unit/DomainL/MoneyMarket/MoneyTest.php:16

```php
public function test_Exchange(): void
{
    $bank = new Bank($this->getRates());
    $money = new Money(new Amount( value: 10000),  currency: Currency::PLN);

    $result = $money->exchangeTo( newCurrency: Currency::USD, $bank);

    self::assertEquals( expected: 2306, $result->amount()->value());
    self::assertEquals( expected: Currency::USD, $result->currency());
}

public function getRates(): array
{
    return [
        'USD:PLN' => 4.33585,
        'PLN:USD' => 0.230635,
    ];
}
```

```
Time: 00:00.004, Memory: 4.00 MB

OK (1 test, 2 assertions)

Process finished with exit code 0
```

```php
class Bank
{
    public function __construct(private readonly array $rates){...}

    public function exchange(Amount $amount, Currency $fromCurrency, Currency $toCurrency): Money
    {
        $rate = $this->getRate($fromCurrency, $toCurrency);
        $newAmount = $this->getNewAmount($amount, $rate);
        return new Money($newAmount, $toCurrency);
    }

    private function getRate(Currency $fromCurrency, Currency $toCurrency): float{...}

    private function getKeyPair(Currency $fromCurrency, Currency $toCurrency): string{...}

    private function getNewAmount(Amount $amount, float $rate): Amount
    {
        $value = (int)round( num: $amount->value() * $rate);
        return new Amount($value);
    }
}
```

```
Time: 00:00.004, Memory: 4.00 MB

OK (1 test, 2 assertions)

Process finished with exit code 0
```

Co chcemy osiągnąć?

Warstwa aplikacji

```php
public function testUpdateDataFromExternalGateway(): void
{
    $externalRatesGateway = $this->getExternalRatesGateway();
    $externalRatesGateway->expects($this->once())->method( constraint: 'getLatestRates')->willReturn([
        ['key' => 'PLN:EUR', 'value' => 4.6572],
        ['key' => 'EUR:PLN', 'value' => 0.0063],
    ]);
    $rateRepository = $this->getRateRepository();
    $rateRepository->expects($this->exactly( count: 2))->method( constraint: 'save');
    $service = new UpdateExchangeRatesService($externalRatesGateway, $rateRepository);

    $result = $service->execute();

    self::assertEmpty($result->errors());
}
```

```php
public function testNotUpdateDataFromExternalGatewayWhenDataMalformed(): void
{
    $externalRatesGateway = $this->getExternalRatesGateway();
    $externalRatesGateway->expects($this->once())->method( constraint: 'getLatestRates')->willReturn([
        ['key' => 'PL:EUR', 'value' => 4.6572],
        ['key' => 'EUR:PLN', 'value' => 0.0063],
    ]);
    $rateRepository = $this->getRateRepository();
    $rateRepository->expects($this->exactly( count: 0))->method( constraint: 'save');
    $service = new UpdateExchangeRatesService($externalRatesGateway, $rateRepository);

    $result = $service->execute();

    self::assertEquals(['Invalid rate data.'], $result->errors());
}
```

Repozytoria i bramy

```php
public function testGetById(): void
{
    $requestFactory = new GetUserDataRequestFactory();

    $apiClient = $this->createMock( originalClassName: UserApiClient::class);
    $repository = new UserRepository($apiClient, $requestFactory);

    $apiClient->expects($this->once())->method( constraint: 'call')
        ->willReturnCallback([$this, 'UserApiClient_call']);

    $user = $repository->getById( id: 1);

    $this->assertApiRequestIsValid();
    $this->assertUserDataAreValid($user);
}

public function UserApiClient_call(UserApiRequest $request): ResponseInterface
{
    $this->apiRequest = $request;
    return $this->getResponse();
}
```

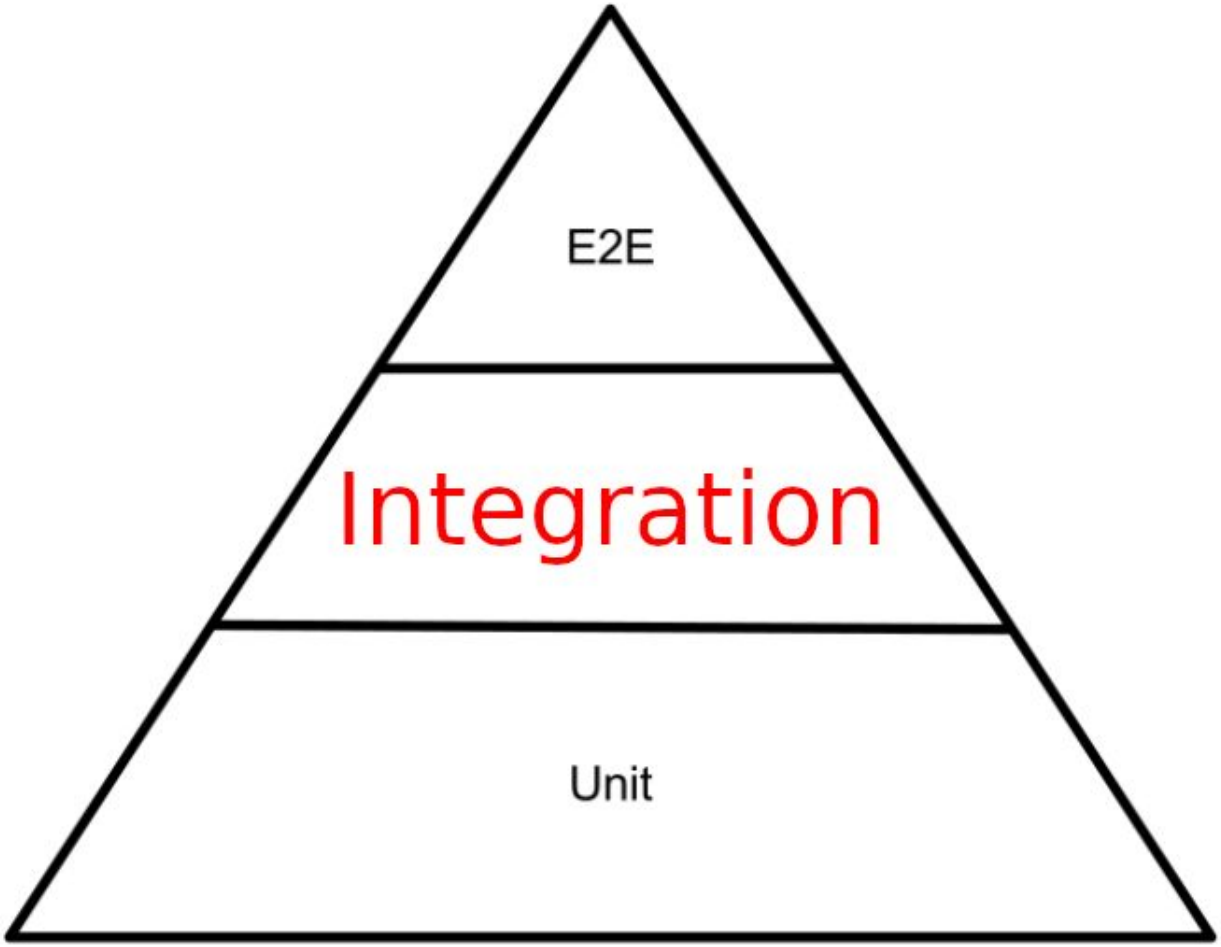
Warstwa logiki

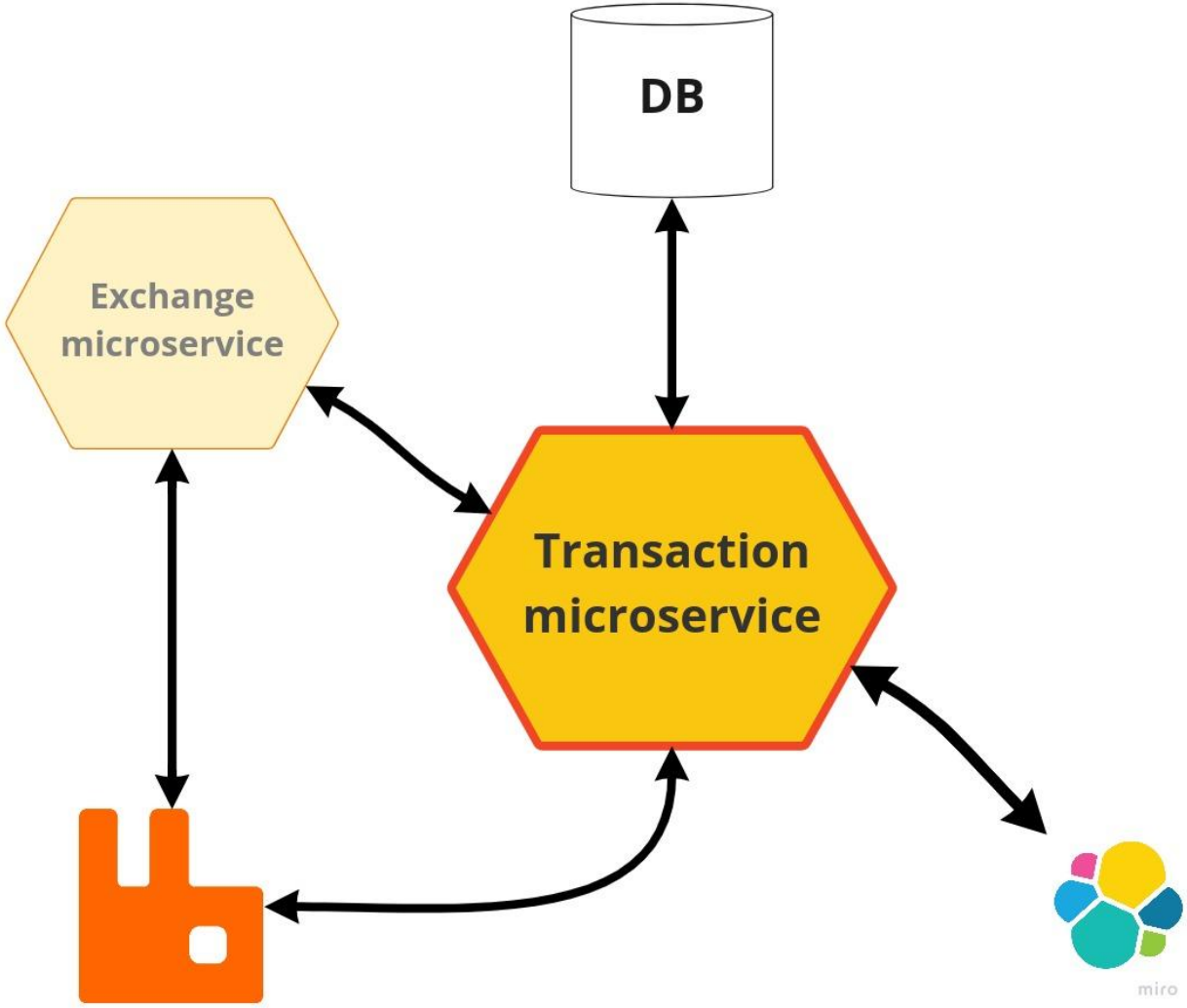
```php
public function test_Exchange(): void
{
    $bank = $this->createMock( originalClassName: Bank::class);
    $bank->expects($this->once())->method( constraint: 'exchange')->willReturn( value: 2306);
    $money = new Money(new Amount( value: 10000),  currency: Currency::PLN);

    $result = $money->exchangeTo( newCurrency: Currency::USD, $bank);

    self::assertEquals( expected: 2306, $result->amount()->value());
    self::assertEquals( expected: Currency::USD, $result->currency());
}
```

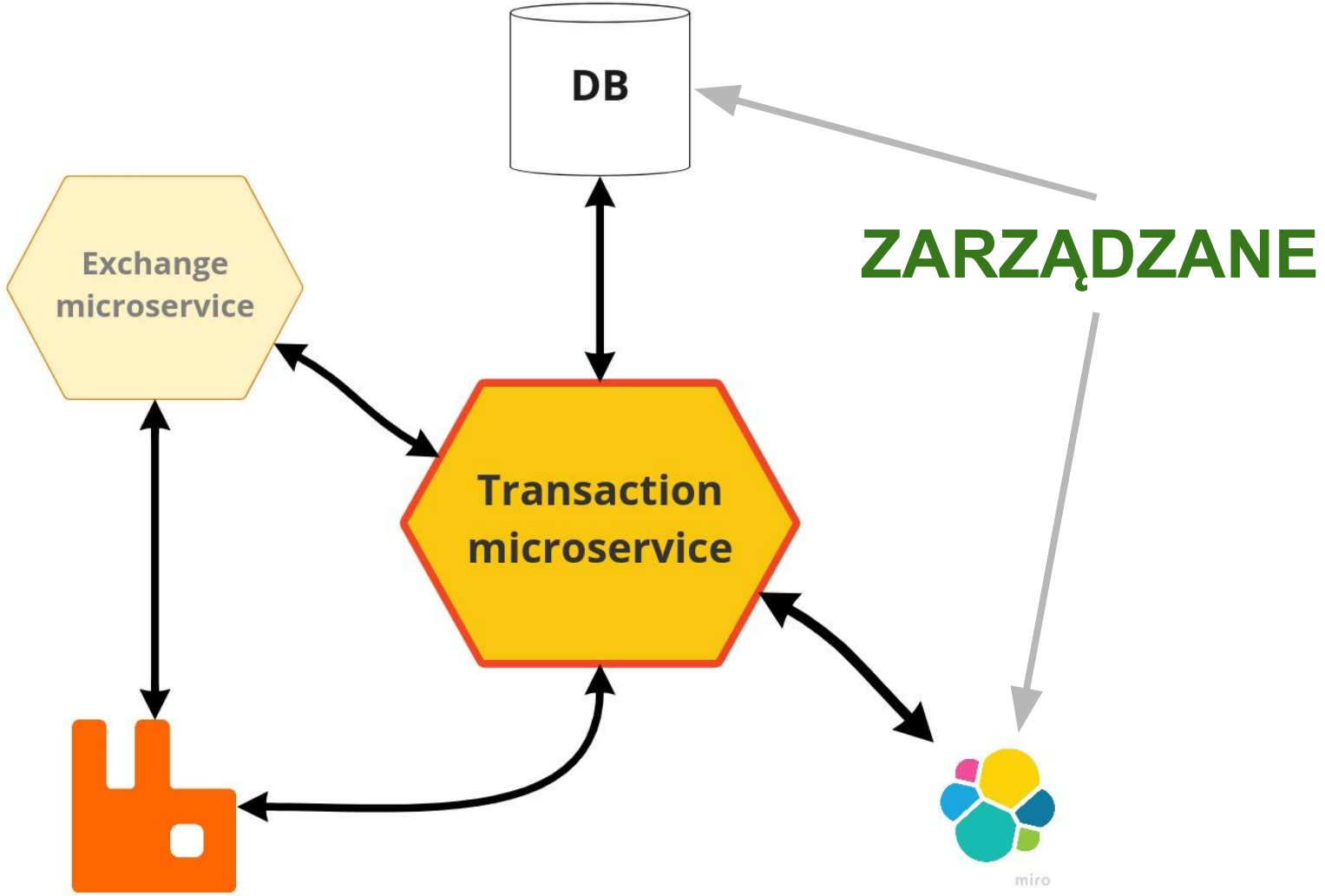
- nie ma jednoznacznej definicji jednostki w testach jednostkowych. Wszystko zależy od konkretnej sytuacji
- w głównej domenie ograniczamy stosowanie atrap
- testując gateway’e skupiamy się na procesie żądanie-odpowiedź
- w warstwie aplikacji testujemy przepływ komunikacji między corem a klientem, warto stosować atrapy dla zerwania łańcucha zależności

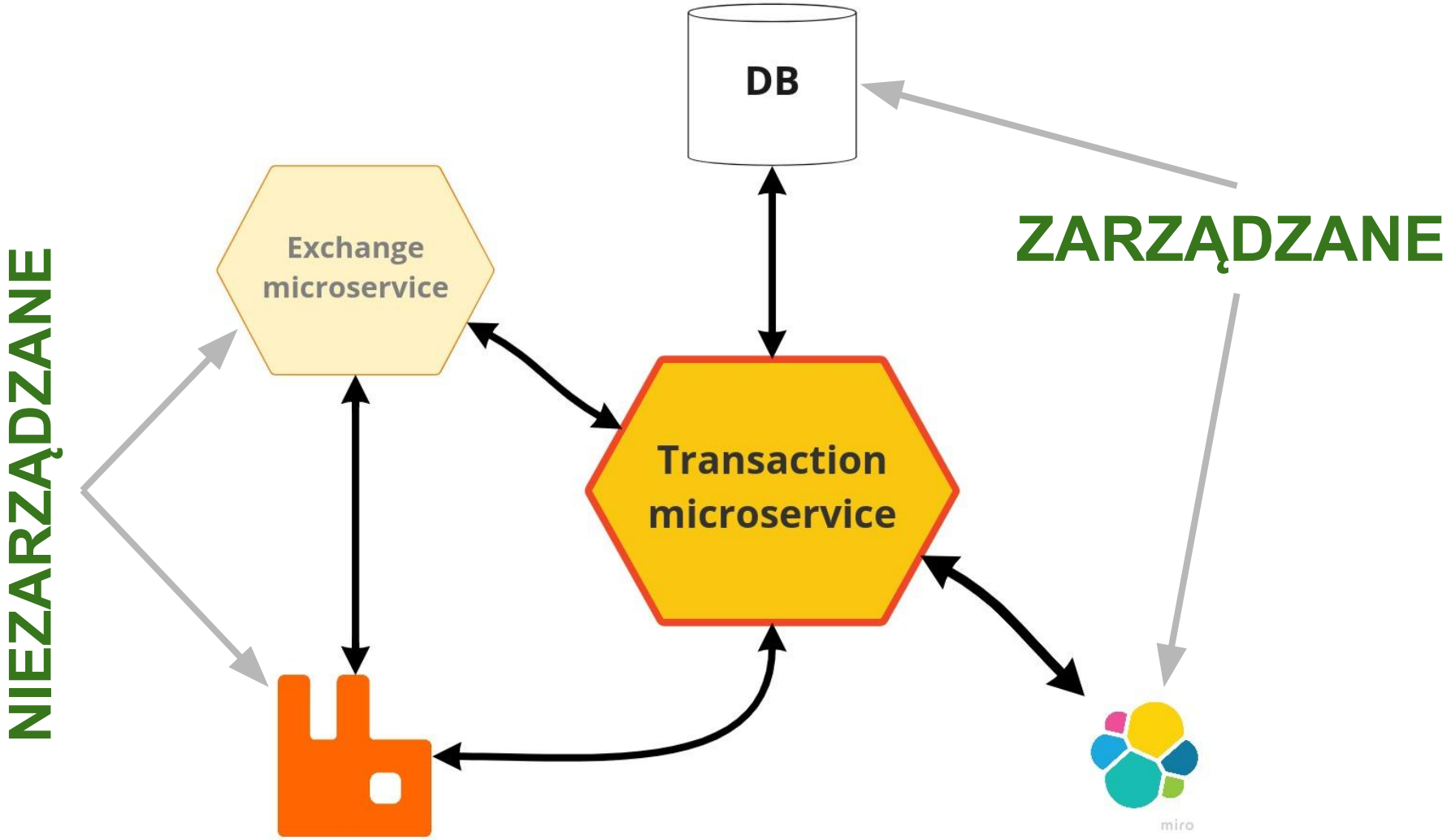
Logika biznesowa = rzeczywiste obiekty

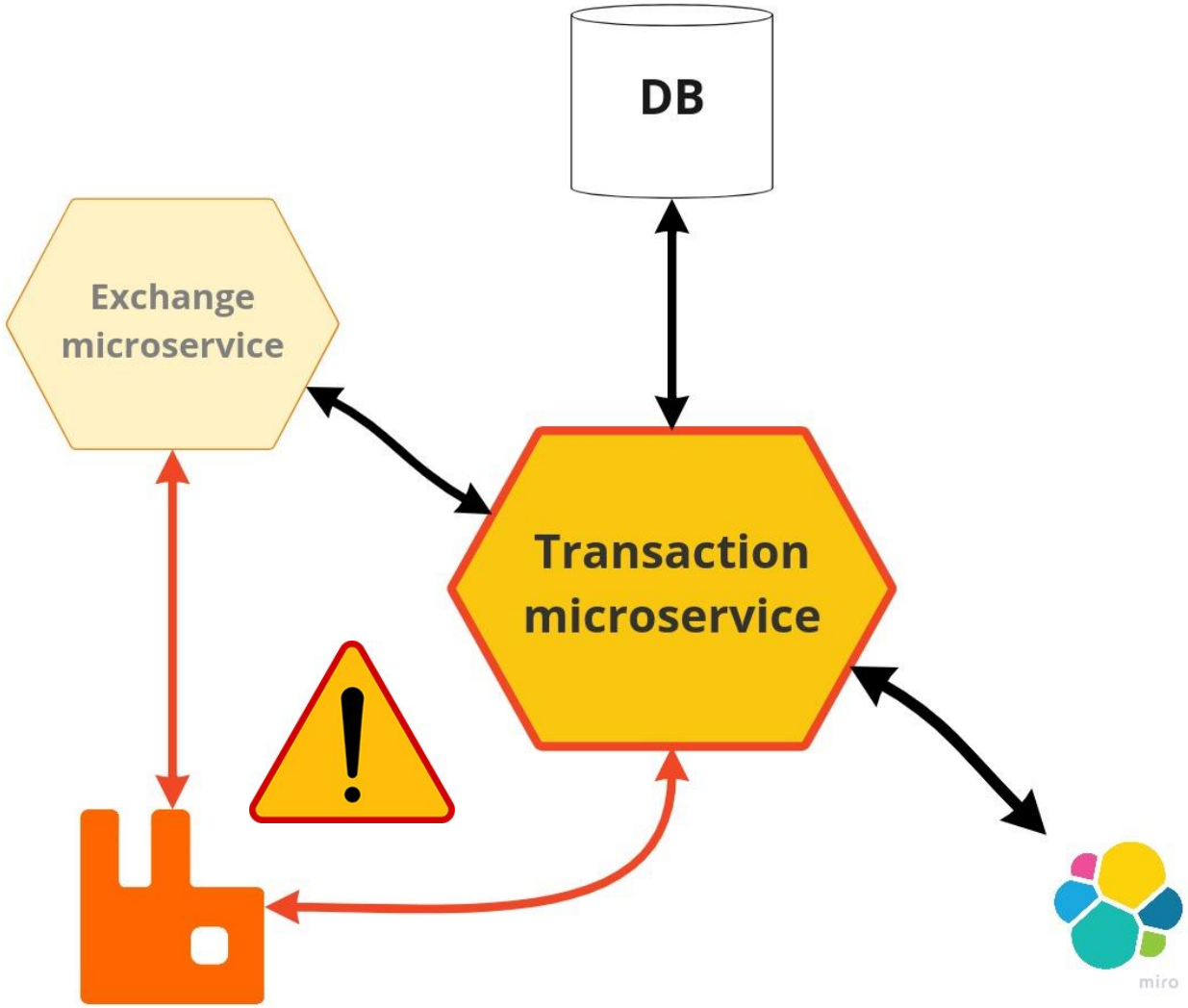
Komunikacja ze światem zewnętrznym = atrapy

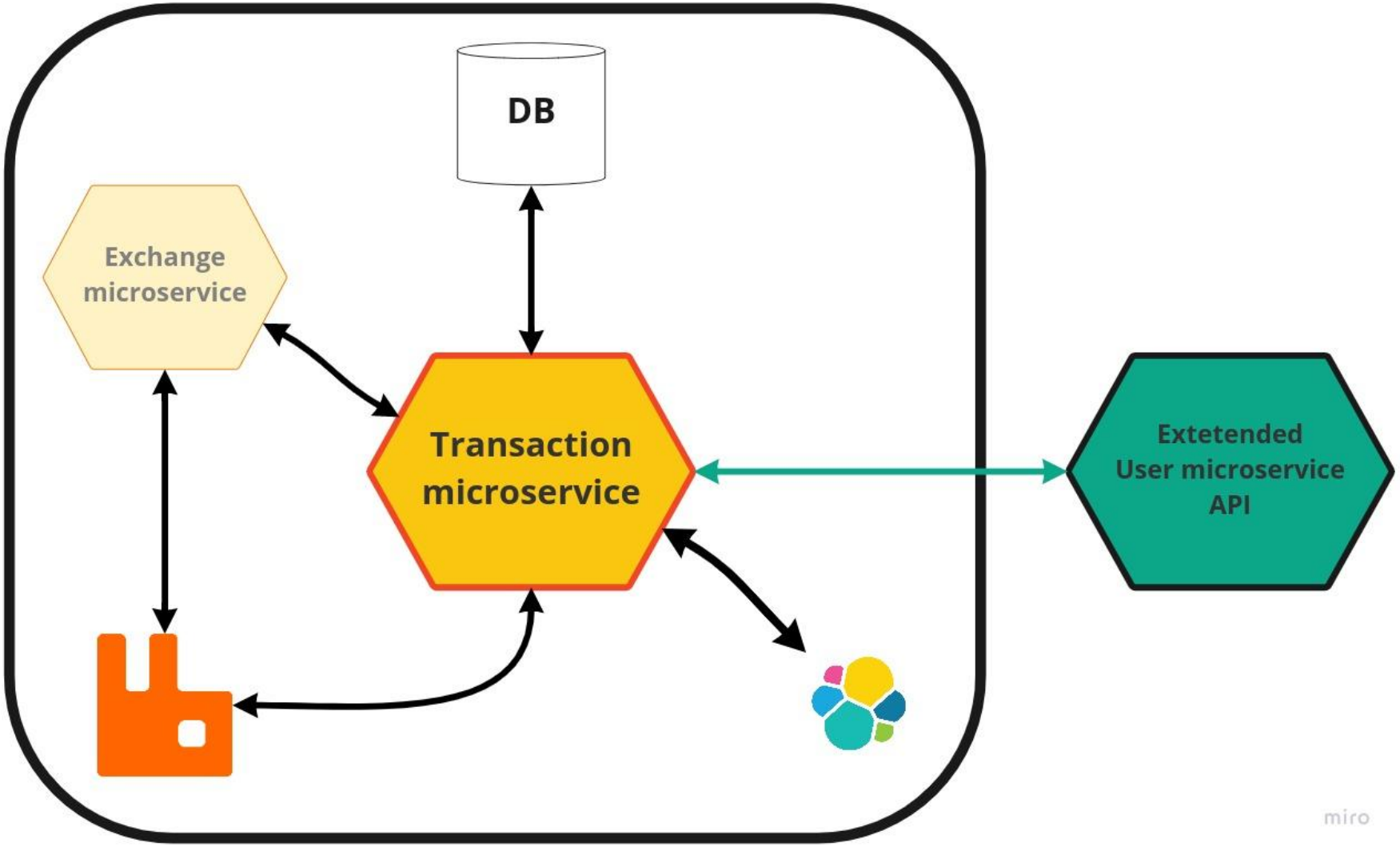

E2E
Integration
Unit

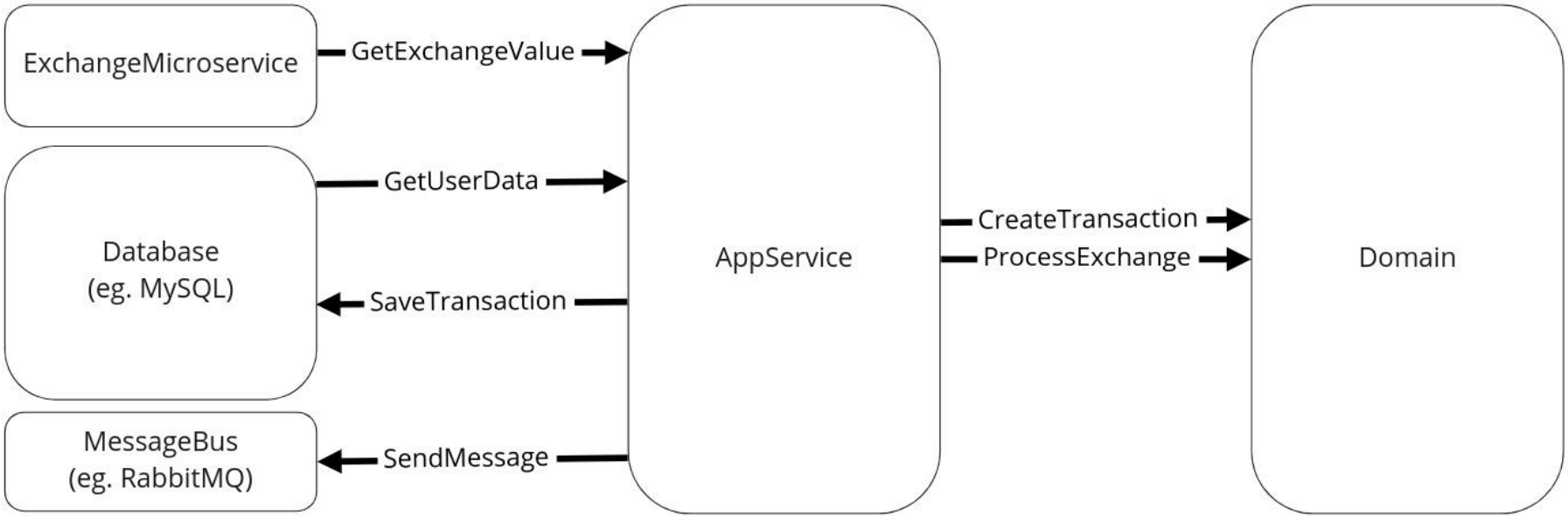

DB
Exchange microservice
Transaction microservice
miro


DB
Exchange microservice
Transaction microservice
ZARZĄDZANE
miro


DB
Exchange microservice
Transaction microservice
NIEZARZĄDZANE
ZARZĄDZANE
miro


DB
Exchange microservice
Transaction microservice
miro


DB
Exchange microservice
Transaction microservice
Extetended User microservice API
miro

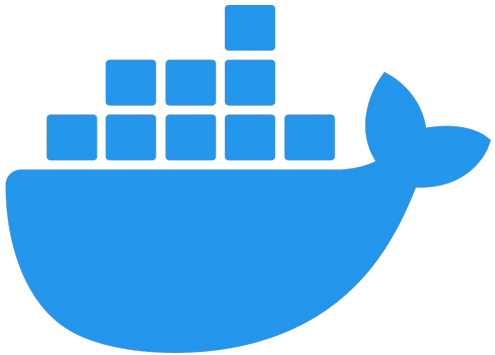

ExchangeMicroservice
GetExchangeValue
Database
(eg. MySQL)
GetUserData
SaveTransaction
MessageBus
(eg. RabbitMQ)
SendMessage
AppService
CreateTransaction
ProcessExchange
Domain

```php
public function testUserExchangeMoney(): void
{
    $this->insertUser();
    $externalRatesGateway = $this->getExternalRatesGatewayMock();
    $externalRatesGateway->expects($this->once())->method( constraint: 'getLatestRates')
        ->willReturn([
            'PLN:EUR' => 4.6572,
            'EUR:PLN' => 0.0063,
        ]);
    $messageBus = $this->getMessageBusMock();
    $messageBus->expects($this->once())->method( constraint: 'dispatch')->with(new TransactionProcessed( transactionId: 1));
    $service = $this->getTransactionService($externalRatesGateway, $messageBus);

    $service->execute($this->getRequest());

    $this->assertTransactionExists();
}
```

PHPUnit - “Przydasie”

```
->withConsecutive(
    [$call1_arg1, $call1_arg2],
    [$call2_arg1, $call2_arg2],
)
->willReturnCallback([$this, 'callbackFunction']);
```

Więcej kontroli



```yaml
version: '2.2'
services:
  php: <8 keys>
  mysql:
    image: mysql:latest
    volumes:
      - ./db-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=xxx
      - MYSQL_DATABASE=przelewy24
      - MYSQL_ROOT_HOST=%
  rabbitmq:
    image: rabbitmq:3
    environment:
      - RABBITMQ_DEFAULT_USER=xxx
      - RABBITMQ_DEFAULT_PASS=xxx
  elasticsearch:
    image: elasticsearch:latest
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
```

```yaml
version: '2.2'
services:
  php: <8 keys>
  mysql: <6 keys>
  rabbitmq:
    image: rabbitmq:3
    environment:
      - RABBITMQ_DEFAULT_USER=xxx
      - RABBITMQ_DEFAULT_PASS=xxx
    ..
```

```php
public function testUserExchangeMoney_Rabbit(): void
{
    $this->insertUser();
    $externalRatesGateway = $this->getExternalRatesGatewayMock();
    $externalRatesGateway->method( constraint: 'getLatestRates')
        ->willReturn([
            'PLN:EUR' => 4.6572,
            'EUR:PLN' => 0.0063,
        ]);

    $middleware = $this->getRabbitMiddleware();
    $messageBus = new MessageBus([$middleware]);
    $service = $this->getTransactionService($externalRatesGateway, $messageBus);

    $service->execute($this->getRequest());

    $this->assertTransactionExists();
    $this->assertMessageInRabbit();
}
```

```php
private function assertMessageInRabbit()
{
    $connection = new AMQPStreamConnection( host: 'rabbitmq', port: 5672, user: 'guest', password: 'guest');
    $channel = $connection->channel();
    $channel->queue_declare( queue: 'transaction_create', passive: false, durable: false, exclusive: false, auto_delete: false);
    $callback = function ($msg) {
        self::assertJsonStringEqualsJsonString( expectedJson: '{"transactionId":1}', $msg->body);
    };
    $channel->basic_consume( queue: 'transaction_create', consumer_tag: '', no_local: false, no_ack: true, exclusive: false, nowait: false, $callback);
    while ($channel->is_open()) {
        $channel->wait();
    }
}
```

```yaml
version: '2.2'
services:
  php: <8 keys>
  exchange-microservice:
    image: registry.p24.org.pl/exchange-microservice/latest
  mysql: <3 keys>
  rabbitmq: <2 keys>
  elasticsearch:
    image: elasticsearch:latest
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
```

Wykorzystanie spreparowanego API

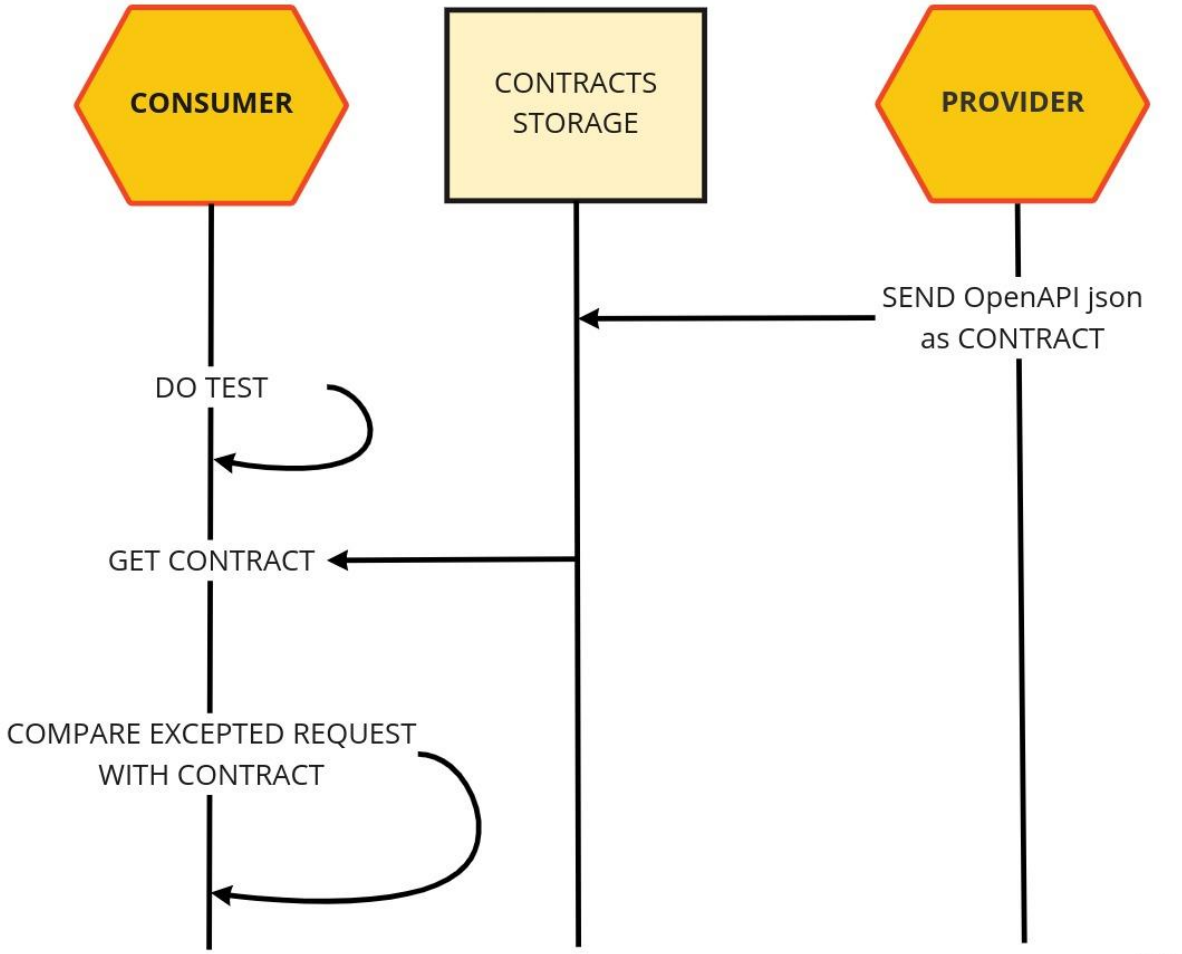
```
server {
    listen 80;
    server_name exchange-microservice.docker;
    root /var/www/html/tests/_data/exchange-microservice;
    index index.php;
    error_page 404 /404.html;

    location / {
        try_files $uri $uri/ /index.php;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

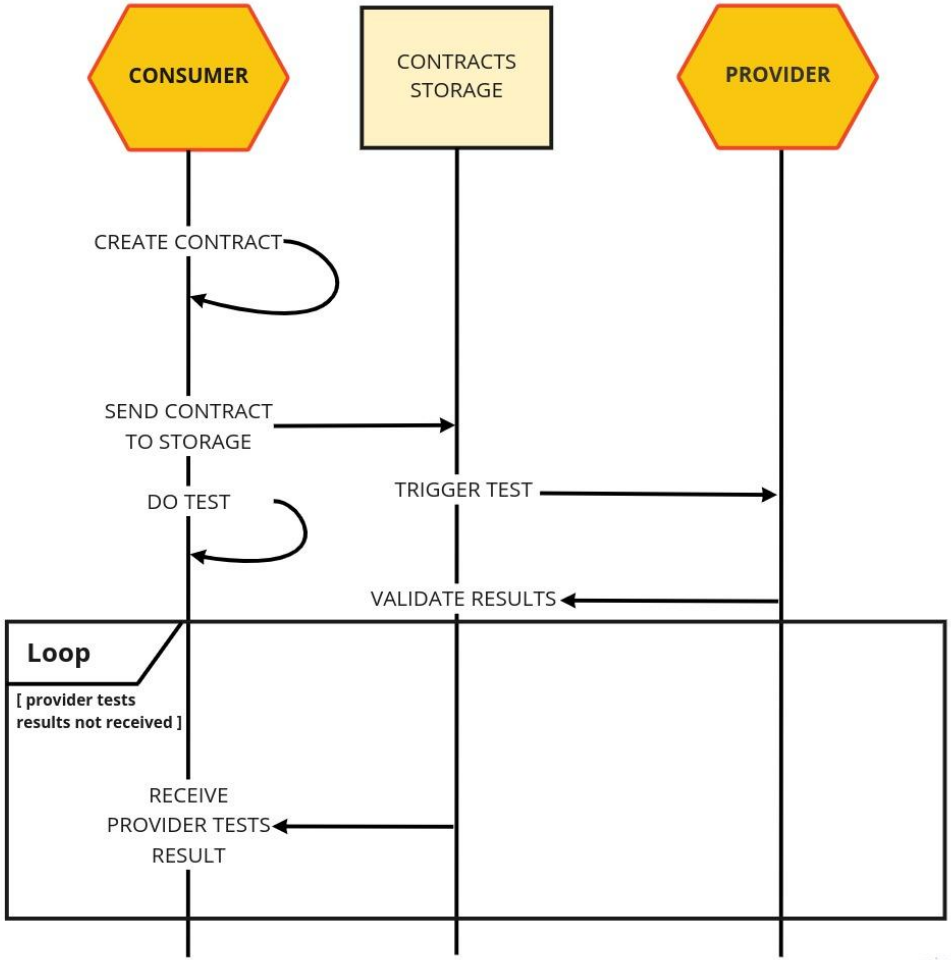
```php
<?php
header( header: 'Content-Type: application/json');

const USER_UNAUTHORIZED = 'unauthorized';
const VALID_DATA = [
    'PLN:EUR' => 4.6572,
    'EUR:PLN' => 0.0063,
];

$user = $_SERVER['PHP_AUTH_USER'];
$body = json_encode(['data' => VALID_DATA, 'error' => false], flags: JSON_THROW_ON_ERROR);

if ($user === USER_UNAUTHORIZED) {
    header( header: 'HTTP/1.1 401 Unauthorized');
    $body = json_encode(['error' => 'Unauthorized'], flags: JSON_THROW_ON_ERROR);
} else {
    header( header: 'HTTP/1.1 200 OK');
}

print $body;
```

Kontrakty

provider-driven

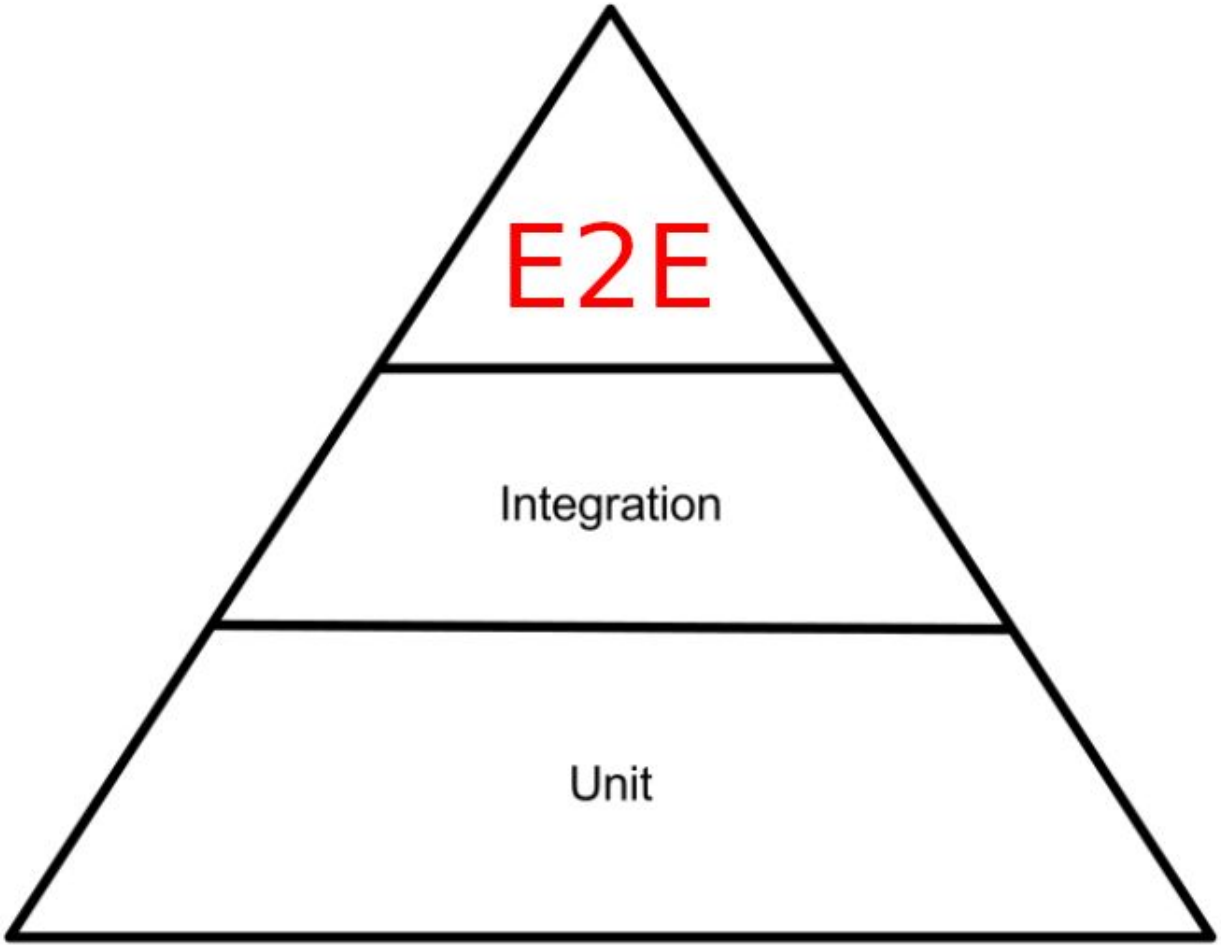
consumer-driven

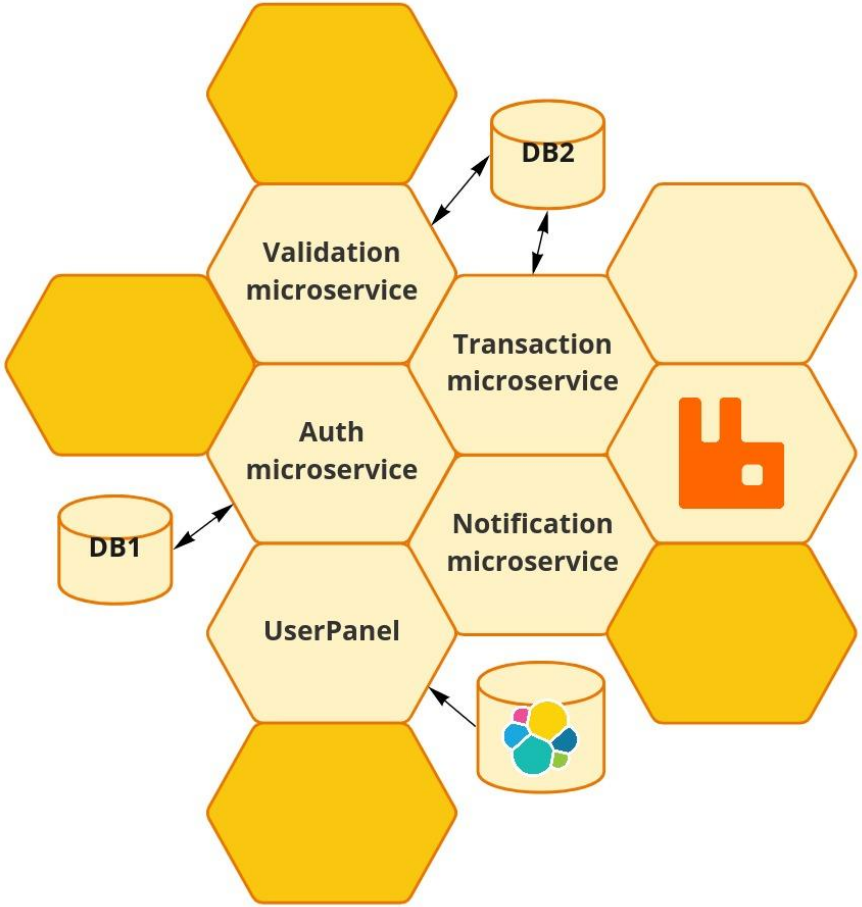

CONSUMER
CONTRACTS STORAGE
PROVIDER
SEND OpenAPI json as CONTRACT
DO TEST
GET CONTRACT
COMPARE EXCEPTED REQUEST WITH CONTRACT
miro


CONSUMER
CONTRACTS STORAGE
PROVIDER
CREATE CONTRACT
SEND CONTRACT TO STORAGE
DO TEST
TRIGGER TEST
VALIDATE RESULTS
Loop
[provider tests results not received]
RECEIVE PROVIDER TESTS RESULT

www.pact.io

- testy integracyjne = weryfikacja współpracy między zależnościami
- zależności zarządzane = rzeczywiste instancje
- zależności niezarządzane = atrapy
- testy w zdokeryzowanym środowisku
- warto wykorzystać rzeczywisty obraz zewnętrznego mikroserwisu
- atrapy przez proste skrypty i vhost-y
- testy kontraktowe


E2E
Integration
Unit

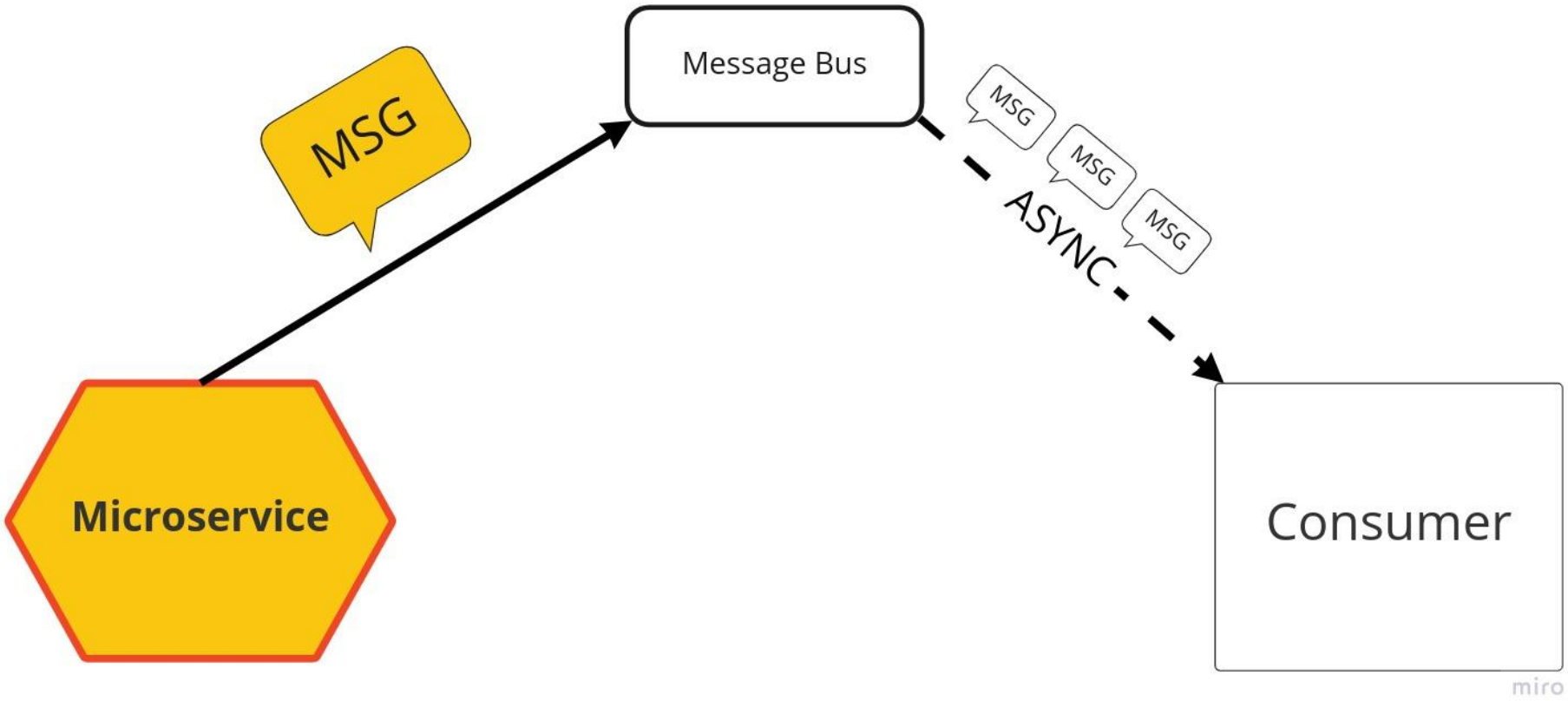

DB2
Validation microservice
Transaction microservice
Auth microservice
Notification microservice
DB1
UserPanel

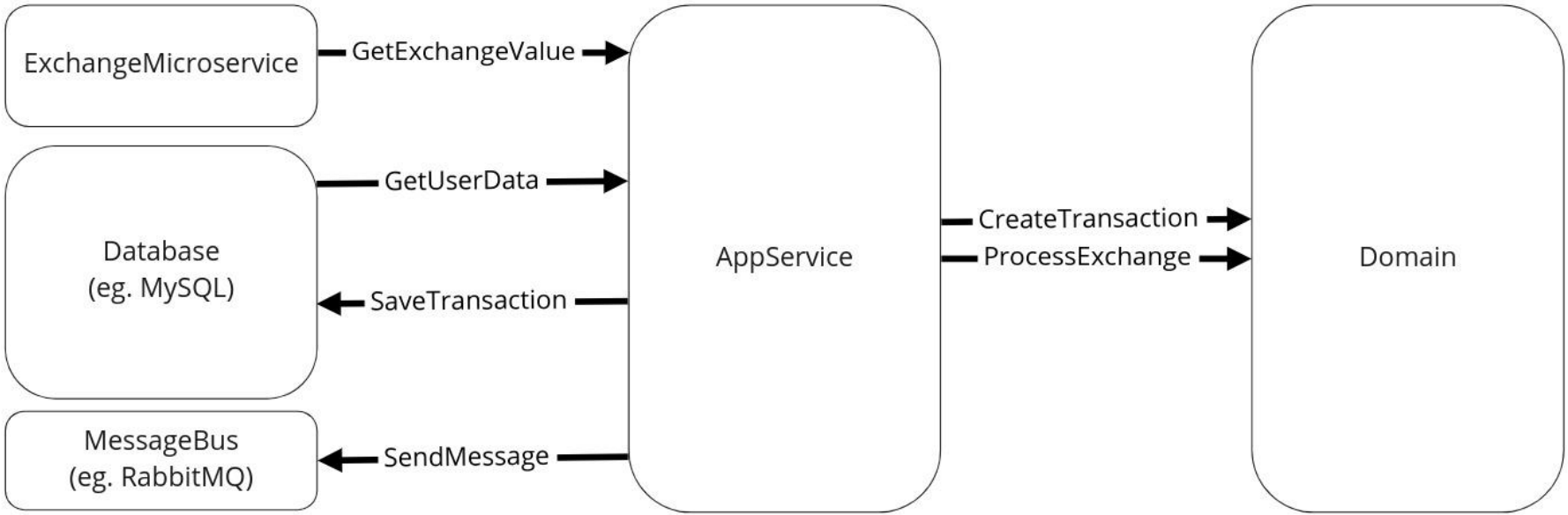
Narzędzia

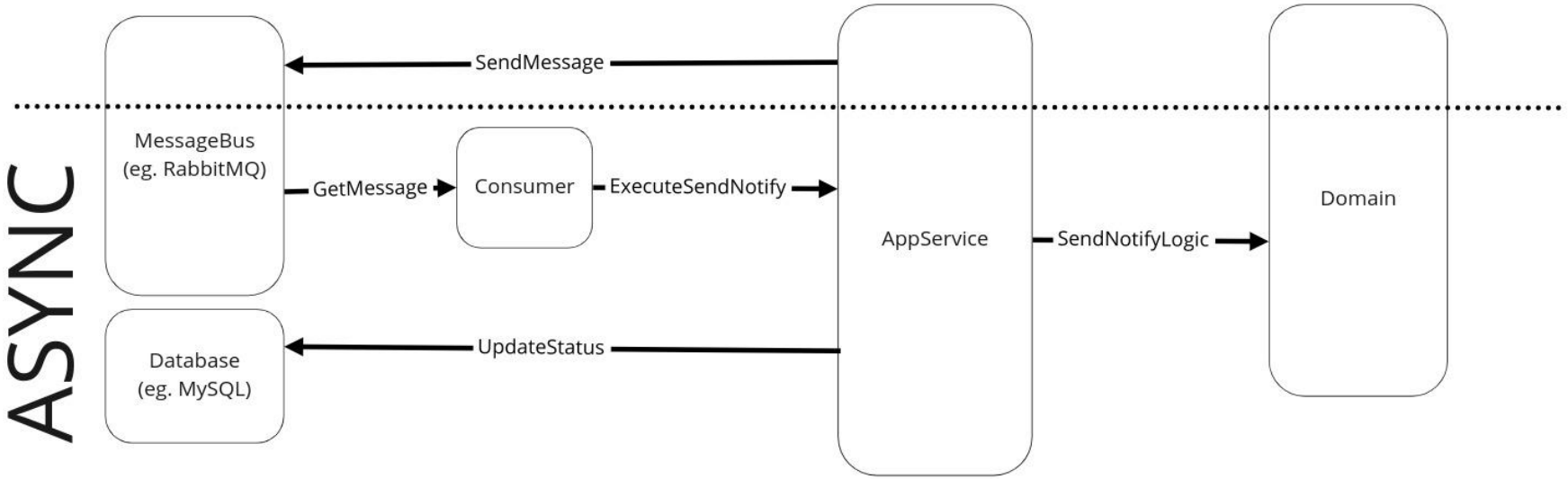
```php
public function _before(ApiTester $I): void
{
    $this->prepareUserData($I);
}

public function testExchange(ApiTester $I): void
{
    $this->userAuthenticated();
    $I->haveHttpHeader( name: 'Content-Type',  value: 'application/json');

    $I->sendPOST( url: self::PATH, $this->getExchangeTransactionData());

    $I->seeResponseCodeIs( code: HttpCode::OK);
    $I->seeResponseIsJson();
    $I->seeResponseMatchesJsonType(
        [
            'transactionId' => 'integer:!empty'
        ]
    );
}

public function _after(ApiTester $I): void
{
    $this->clean($I);
}
```

```
modules:
    config:
        \App\Test…DbExtended: <8 keys>
        RedisDb: <3 keys>
        REST:
            url: http://127.0.0.1/
            depends: PhpBrowser
            part:
                - Json
                - Xml
        ElasticsearchModule: <9 keys>
```

```
modules:
    config:
        DB:
            dsn: 'mysql:host=mysql;dbname=przelewy24_codeception'
            user: 'xxx'
            password: 'xxx'
            dump: dump.sql
            populate: true
            cleanup: true
            databases: <3 keys>
        RedisDb: <3 keys>
        REST: <3 keys>
        ElasticsearchModule: <9 keys>
```


Message Bus
MSG
MSG
MSG
MSG
ASYNC
Microservice
Consumer
miro


ExchangeMicroservice
GetExchangeValue
Database
(eg. MySQL)
GetUserData
SaveTransaction
MessageBus
(eg. RabbitMQ)
SendMessage
AppService
CreateTransaction
ProcessExchange
Domain


ASYNC
MessageBus
(eg. RabbitMQ)
SendMessage
GetMessage
Consumer
ExecuteSendNotify
AppService
SendNotifyLogic
Domain
Database
(eg. MySQL)
UpdateStatus

```php
public function testUserExchangeMoneyProcess(ApiTester $I): void
{
    $this->prepareData();
    $I->haveHttpHeader( name: 'Content-Type',  value: 'application/json');
    $I->sendPost( url: self::PATH, $this->getTransactionData());
1   $id = $I->grabFromDatabase(
        table: 'transaction',
        column: 'id',
        [
            'customer_id' => 1,
            'amount' => 20000,
            'status' => self::STATUS_NEW
        ]
    );

2   $this->startConsumer();
3   $I->waitUntilSeeInDatabase( tableName: 'transaction', [
        'id' => $id,
        'status' => self::STATUS_SENT
    ]);
    $this->killConsumer();
}
```

```php
public function startConsumer(): int
{
    $consumerPath = '/var/www/html/bin/console consumer:transaction:notification';
    $outputFilename = 'notification-consumer.log';
    $command = "nohup ${consumerPath} >> tests/_output/${outputFilename} 2>&1 & echo $!";
    exec($command, &$output);
    $pid = (int)$output[0];
    $this->cronProcess = $pid;
    return $pid;
}

private function killConsumer(): void
{
    exec( command: "kill {$this->cronProcess}", &$output);
    unset($this->cronProcess);
}
```

```php
public function testUserExchangeMoneyProcess(ApiTester $I): void
{
    $this->prepareData();
    $I->haveHttpHeader( name: 'Content-Type',  value: 'application/json');
    $I->sendPost( url: self::PATH, $this->getTransactionData());
1   $id = $I->grabFromDatabase(
        table: 'transaction',
        column: 'id',
        [
            'customer_id' => 1,
            'amount' => 20000,
            'status' => self::STATUS_NEW
        ]
    );

2   $this->startConsumer();
3   $I->waitUntilSeeInDatabase( tableName: 'transaction', [
        'id' => $id,
        'status' => self::STATUS_SENT
    ]);
    $this->killConsumer();
}
```

- czas na przygotowanie
- czas wykonywania
- podstawowe ścieżki przypadków użycia

- testy jednostkowe
  - logika biznesowa = rzeczywiste obiekty
  - świat zewnętrzny = mocki
- testy integracyjne
  - zależności zarządzane = prawdziwe instancje
  - zależności niezarządzane = atrapy
- testy end-to-end
  - możliwie jak najmniej
  - podstawowe ścieżki

Co mogę przetestować
*w **TAKIM** systemie*
*na **TYM** etapie?*

Do zobaczenia!