Jak być (jeszcze) lepszym programistą

Bartosz Gołek
bartosz.golek@gmail.com
@bartosz_golek

O mnie

Programista - 10 lat doświadczenia

Technologie: PHP, C#, Python

Zainteresowania: OOP, TDD, BDD, Agile

Obowiązki:

Projektowanie i tworzenie rozwiązań dla systemów rozproszonych

Agenda

- Wstęp
- Zarys pojęć
- O architekturze
- Trening czyni mistrza
- Pytania

Co to znaczy być dobrym programistą?

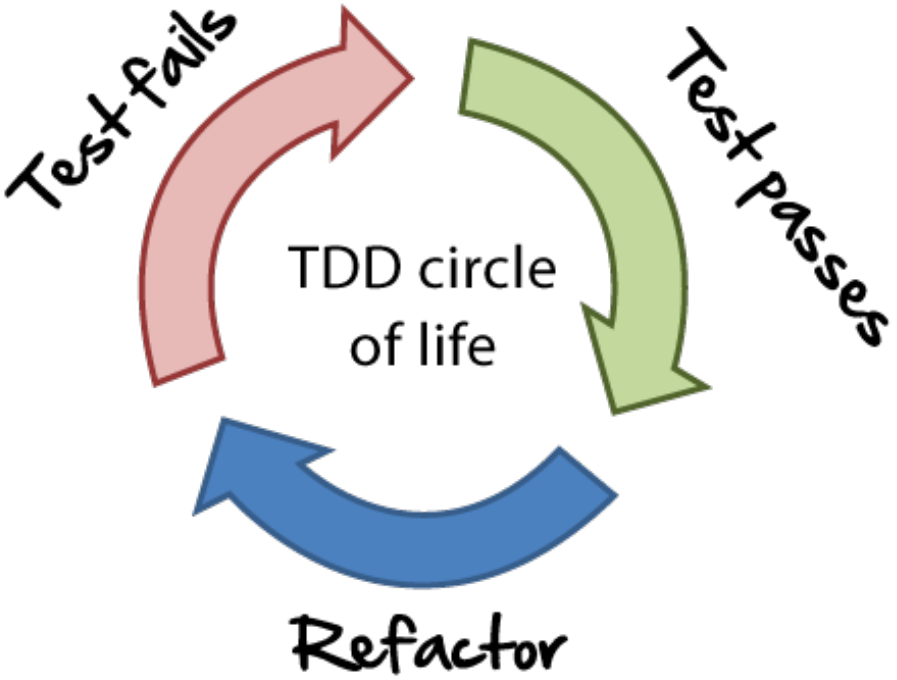
- kompletność kodu?
- czytelność kodu?
- znajomość algorytmów (wydajność)?
- kod podatny na zmiany?
- ilość błędów?
- szybkość pisania kodu?

SOLID

- Single Responsibility Principle
- Open Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Techniki

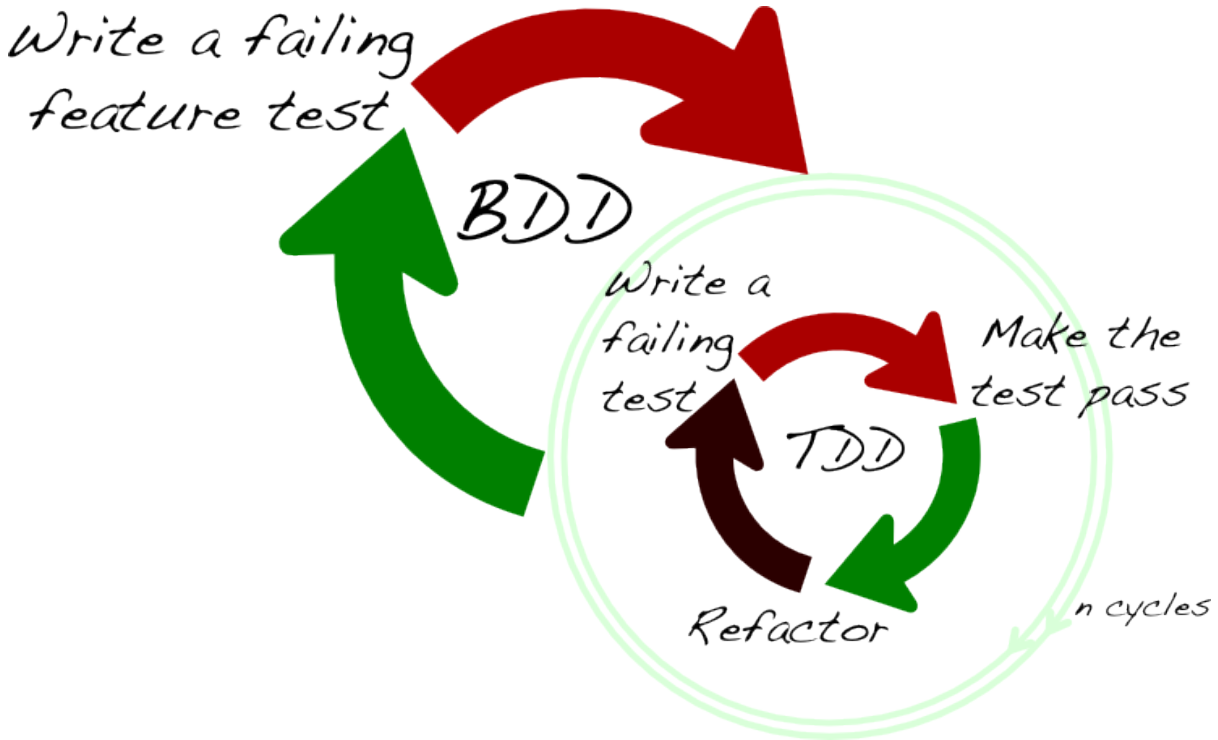
- TDD – test-driven development
- BDD - behavior-driven development




```
Story: Returns go to stock

In order to keep track of stock
As a store owner
I want to add items back to stock when they're returned

Scenario 1: Refunded items should be returned to stock
Given a customer previously bought a black sweater from me
And I currently have three black sweaters left in stock
When he returns the sweater for a refund
Then I should have four black sweaters in stock

Scenario 2: Replaced items should be returned to stock
Given that a customer buys a blue garment
And I have two blue garments in stock
And three black garments in stock.
When he returns the garment for a replacement in black,
Then I should have three blue garments in stock
And two black garments in stock
```

Techniki

- TDD – test-driven development
- BDD - behavior-driven development



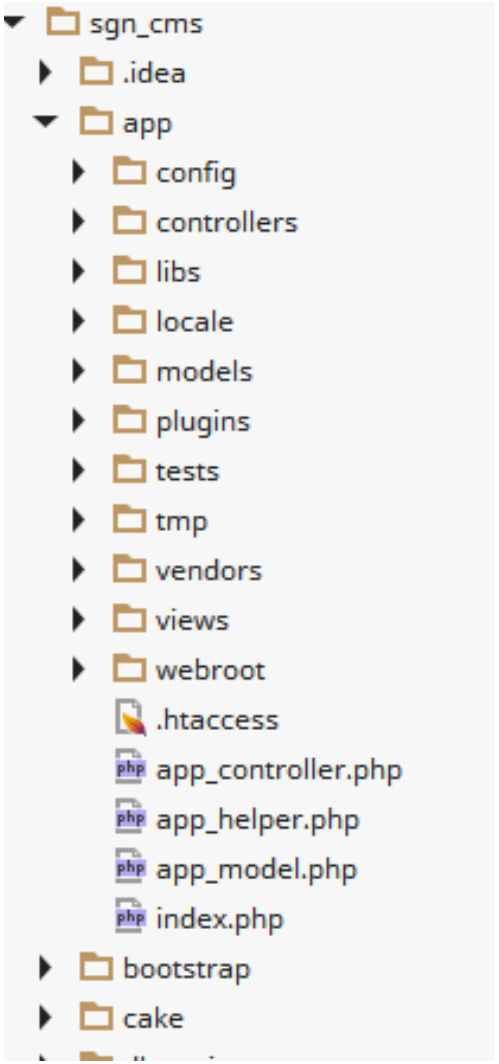
Architektura

- MVC – model view controller
- MVVM – model view view-model
- MVP – model view presenter

Wady MVC, MVVM, MVP

- ODPOWIEDZIALNOŚĆ
- zależności
- baza danych (model) determinuje architekturę aplikacji
- bardziej złożone kontrolery zależą od wszystkich modeli
- gdzie jest miejsce logiki biznesowej?

Logika biznesowa w centrum



- Co wiadomo o tym projekcie?
- Co ta aplikacja potrafi?

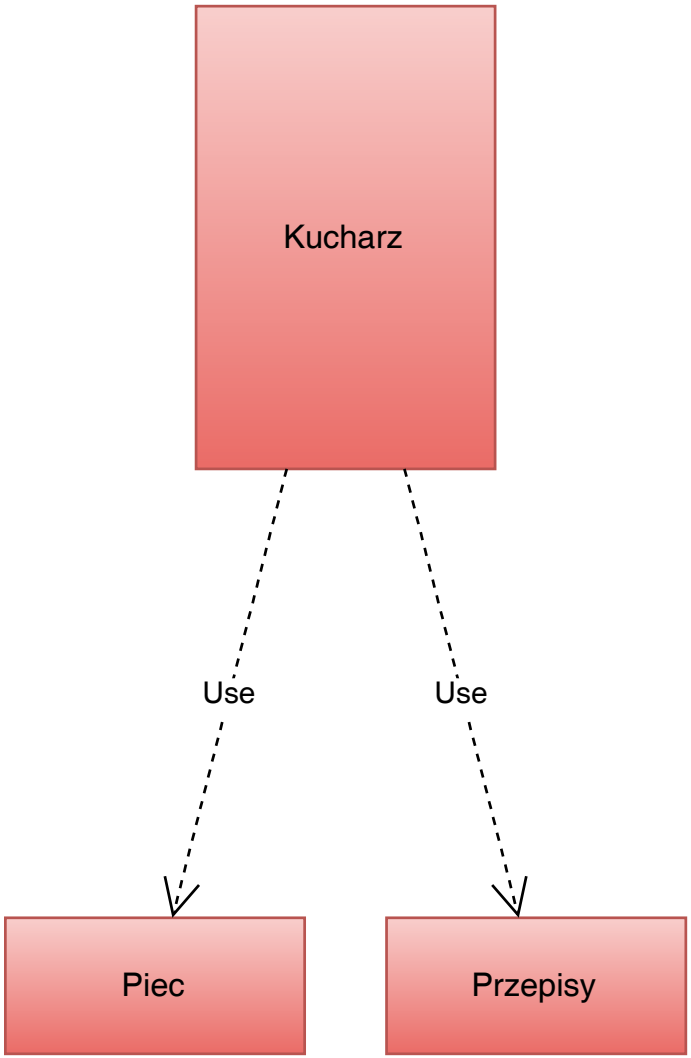
Clean Architecture

- spełnia zasady SOLID
- oferuje czytelny przepływ danych
- ułatwia testowanie
- podział na dane i klasy
- niemutowalne obiekty danych
- struktura aplikacji skupiona wokół logiki biznesowej

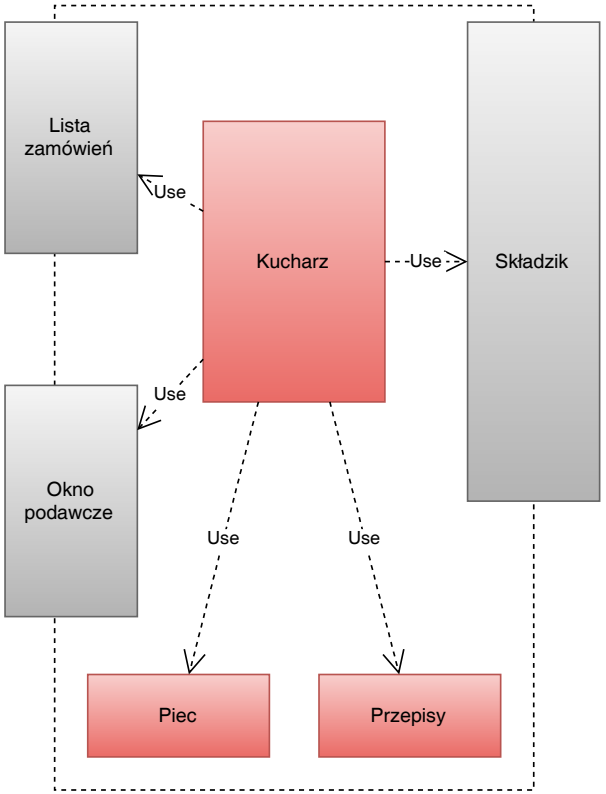
Clean Architecture - autor

- Robert C. Martin
- Architecture the Lost Years:
  https://www.youtube.com/watch?v=WpkDN78P884

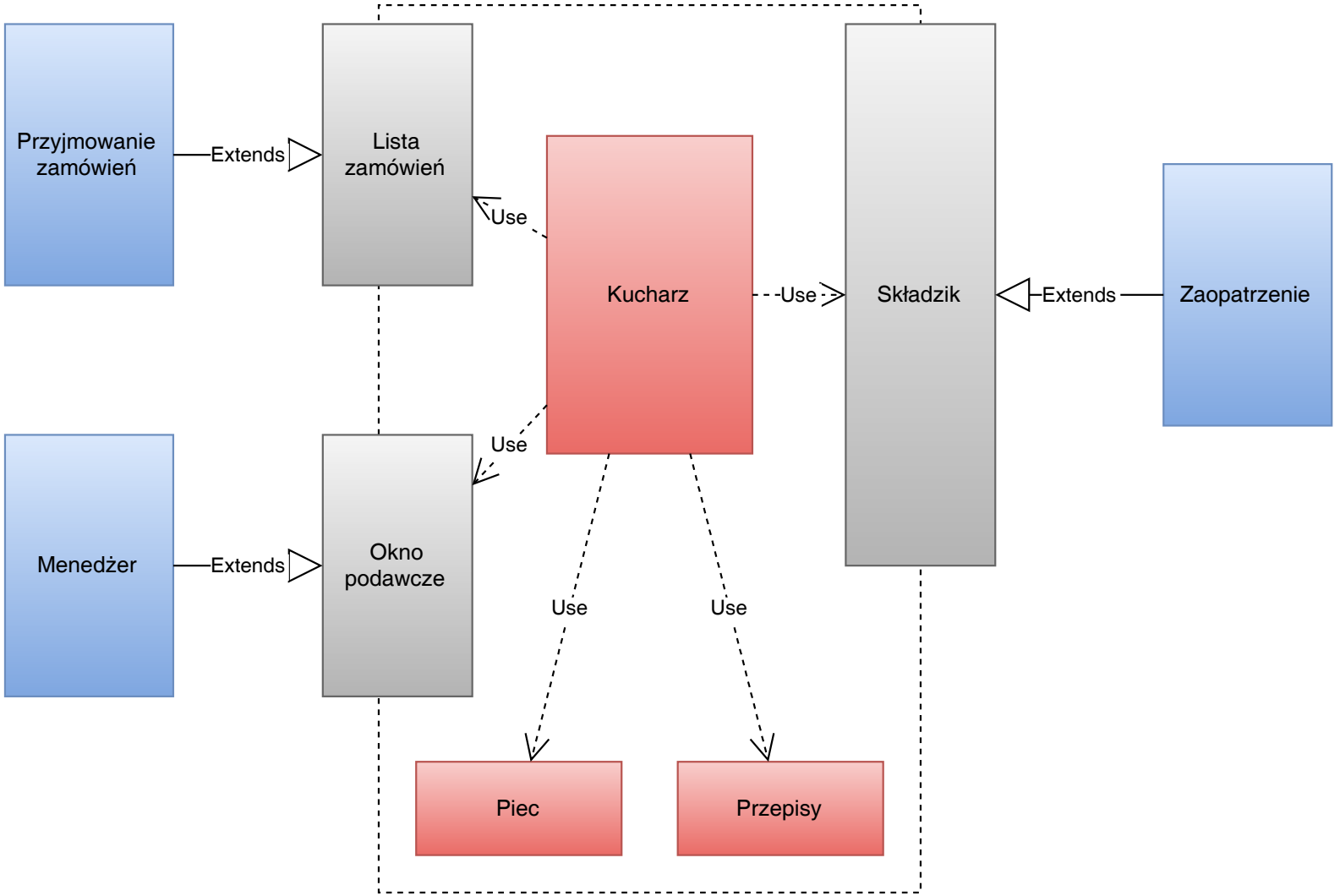
Pizzeria



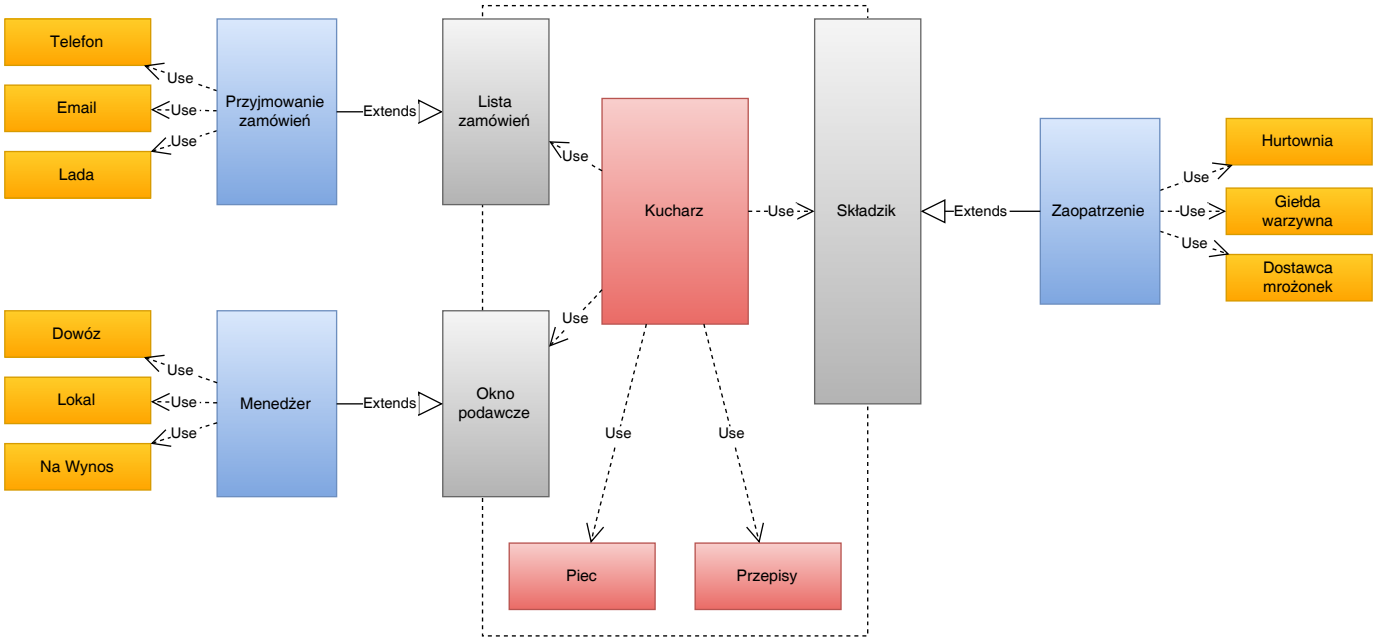
Pizzeria


Lista zamówień
Use
Kucharz
Use
Składzik
Use
Okno podawcze
Use
Use
Piec
Przepisy

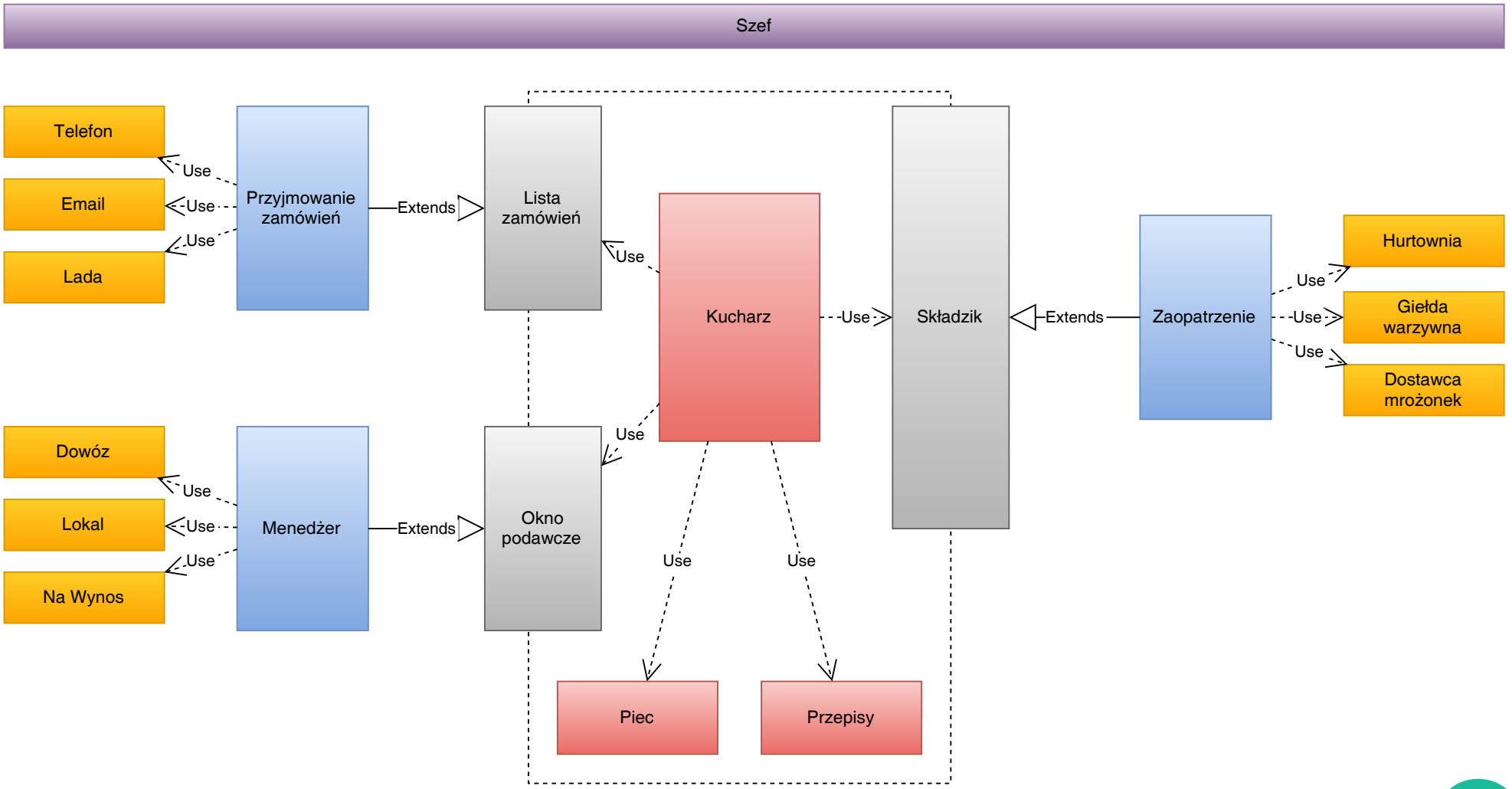
Pizzeria


Przyjmowanie zamówień
Extends
Lista zamówień
Use
Kucharz
Use
Składzik
Extends
Zaopatrzenie
Use
Menedżer
Extends
Okno podawcze
Use
Use
Piec
Przepisy

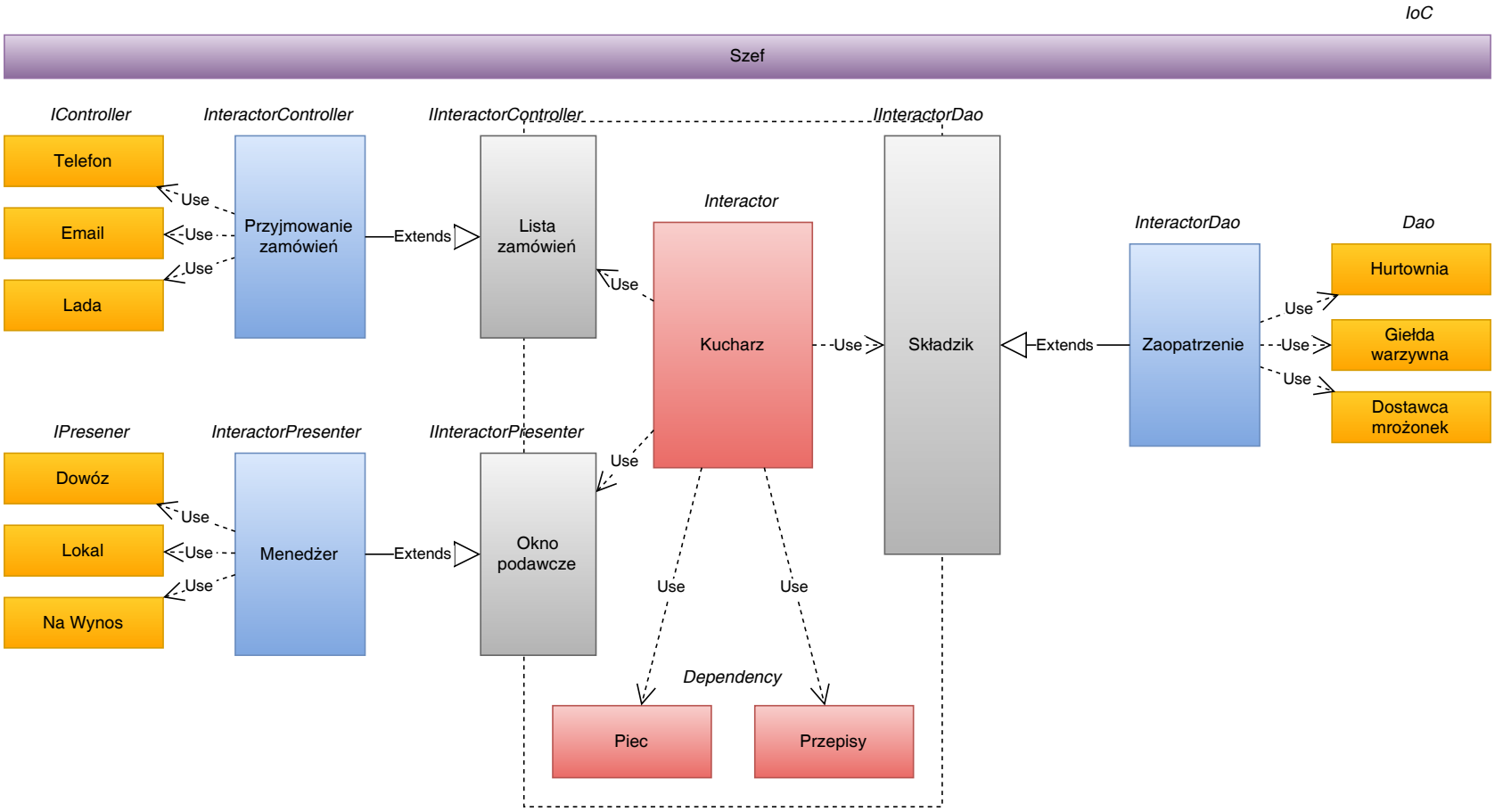
Pizzeria


Telefon
Email
Lada
Use
Use
Use
Przyjmowanie zamówień
Extends
Lista zamówień
Use
Kucharz
Use
Składzik
Extends
Zaopatrzenie
Use
Use
Use
Hurtownia
Giełda warzywna
Dostawca mrożonek
Dowóz
Lokal
Na Wynos
Use
Use
Use
Menedżer
Extends
Okno podawcze
Use
Use
Use
Piec
Przepisy

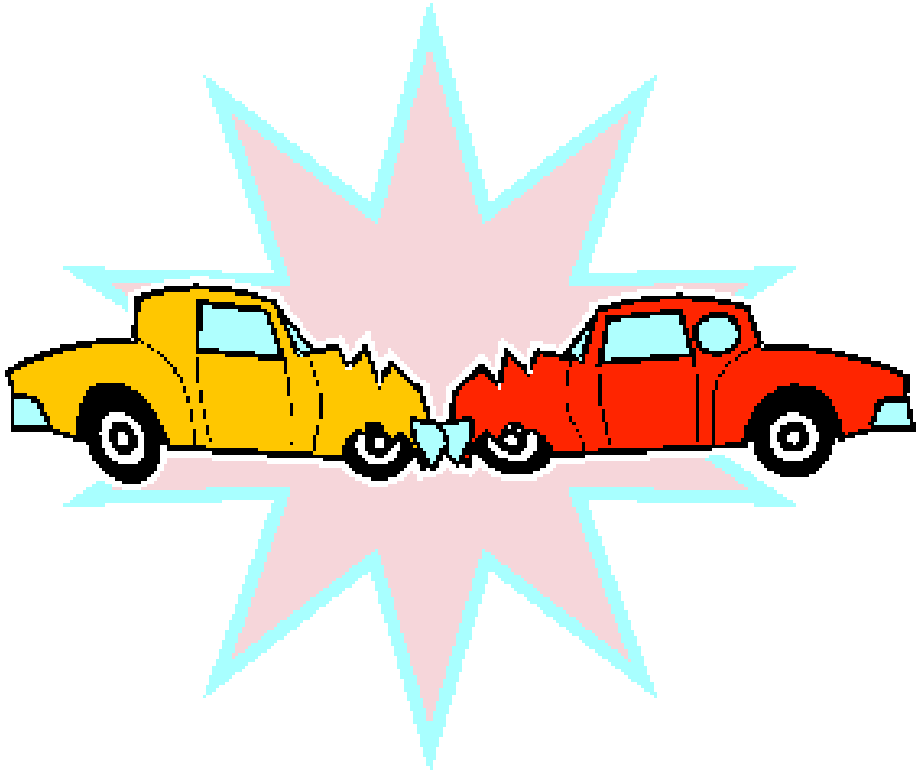
Pizzeria


Szef
Telefon
Email
Lada
Use
Use
Use
Przyjmowanie zamówień
Extends
Lista zamówień
Use
Kucharz
Use
Składzik
Extends
Zaopatrzenie
Use
Use
Use
Hurtownia
Giełda warzywna
Dostawca mrożonek
Dowóz
Lokal
Na Wynos
Use
Use
Use
Menedżer
Extends
Okno podawcze
Use
Use
Use
Piec
Przepisy

Pizzeria


IoC
Szef
IController
InteractorController
IInteractorController
IInteractorDao
Telefon
Email
Lada
Use
Use
Use
Przyjmowanie zamówień
Extends
Lista zamówień
Interactor
Use
Kucharz
Use
Składzik
Extends
InteractorDao
Zaopatrzenie
Dao
Use
Use
Use
Hurtownia
Giełda warzywna
Dostawca mrożonek
IPresener
InteractorPresenter
IInteractorPresenter
Use
Dowóz
Lokal
Na Wynos
Use
Use
Use
Menedżer
Extends
Okno podawcze
Use
Use
Dependency
Piec
Przepisy

Jak to połączyć?



- koszt wejścia
- brak doświadczenia
- błędy wynikające ze złego zrozumienia idei
- goniące terminy
- dług technologiczny*

* - http://wojciszko.com/2015/10/12/kiedy-warto-zaciagnac-dlug-technologiczny/

Własny framework

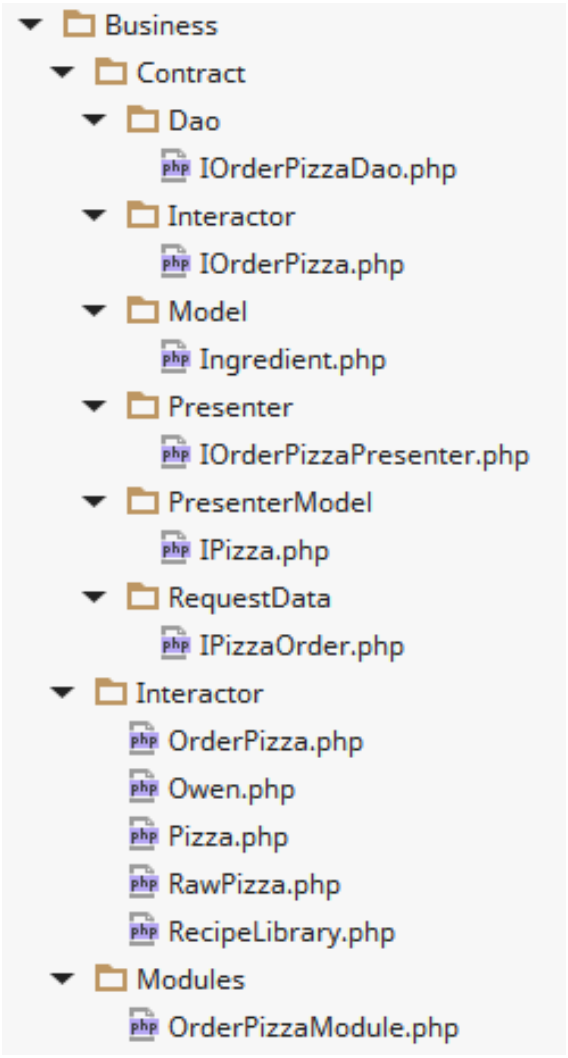
- A po co?
- Tego jest milion!
- Kolejny?!
- Czy będzie lepszy od istniejących?

NIE!!!

Po co?

- dla nauki
- w projektach produkcyjnych nie ma czasu na naukę nowych rzeczy
- w projektach produkcyjnych nie można sobie pozwolić na niekończące się poprawianie kodu
- testowanie nowych rozwiązań

Przykład implementacji



- Katalog Business zawiera wszystkie rzeczy, które są mu potrzebne wykonania pracy (logiki biznesowej)
- Interfejsy i klasy publiczne, znajdują się w podkatalogu Contract
- Znajduje się tu również moduł kontenera zależności.

Przykład implementacji

1

```php
interface IOrderPizza
{
    function run(IPizzaOrder $order);
}
```

2

```php
interface IOrderPizzaDao {
    /**
     * @param string[] $ingredient_names
     *
     * @return Ingredient[]
     */
    function getIngredients(array $ingredient_names);
}
```

3

```php
interface IOrderPizzaPresenter
{
    public function deliver(IPizza $pizza);
}
```

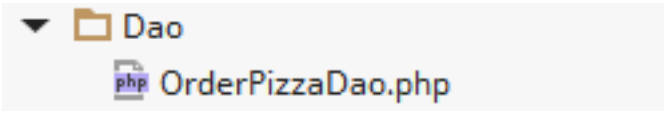
4

```php
interface IPizzaOrder {
    function getName();
    function getDoubleDough();
    function getSauces();
}
```

Przykład implementacji

```php
class OrderPizza implements IOrderPizza
{
    /**
     * OrderPizza constructor.
     *
     * @param IOrderPizzaPresenter $presenter
     * @param IOrderPizzaDao $orderPizzaDao
     */
    public function __construct(
        IOrderPizzaPresenter $presenter,
        IOrderPizzaDao $orderPizzaDao
    )
    {
        $this->recipe_library = new RecipeLibrary();
        $this->owen = new Owen();
        $this->presenter      = $presenter;
        $this->orderPizzaDao = $orderPizzaDao;
    }

    public function run(IPizzaOrder $order)
    {
        $recipe = $this->recipe_library->getRecipe($order->getName());

        $ingredients = $this->orderPizzaDao->getIngredients($recipe);

        $pizza = $this->makePizza($ingredients, $order->getDoubleDough());
        $baked_pizza = $this->owen->bake($pizza);
        $baked_pizza->addSauces($order->getSauces());

        $this->presenter->deliver($baked_pizza);
    }
    //...//
```

Przykład implementacji

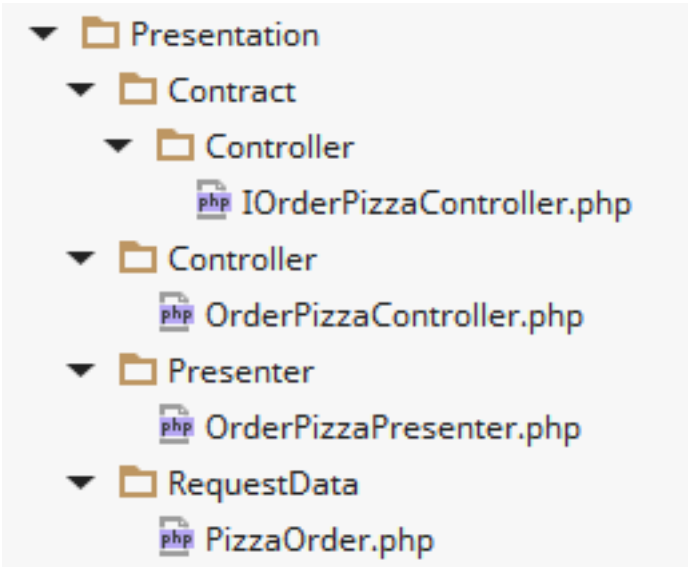


- Katalog Dao zawiera adapter interfejsu IOrderPizzaDao.
- W przypadku implementacji rzeczywistej będzie też zawierał klasy dostępu do bazy danych, web service i/lub innych źródeł danych.
- To warstwa Dao zależy od Business, a nie Business od Dao.

Przykład implementacji

```php
use Conpago\Pizza\Business\Contract\Dao\IOrderPizzaDao;
use Conpago\Pizza\Business\Contract\Model\Ingredient;

class OrderPizzaDao implements IOrderPizzaDao {

    /**
     * @param string[] $ingredient_names
     *
     * @return Ingredient[]
     */
    function getIngredients(array $ingredient_names) {
        $func = function($ingredient){
            return new Ingredient($ingredient);
        };

        return array_map($func, $ingredient_names);
    }
}
```

Przykład implementacji



- Katalog Presentation zawiera adapter interfejsu IOrderPizzaController oraz IOrderPizzaPresenter.
- Zawiera także implementację RequestData definiowanego przez Business.
- To warstwa Dao zależy od Business, a nie Business od Dao.

Przykład implementacji

```php
use Conpago\Pizza\Business\Contract\Interactor\IOrderPizza;
use Conpago\Pizza\Presentation\Contract\Controller\IOrderPizzaController;
use Conpago\Pizza\Presentation\RequestData\PizzaOrder;

class OrderPizzaController implements IOrderPizzaController
{
    /**
     * @var IOrderPizza
     */
    private $orderPizza;

    function __construct(IOrderPizza $orderPizza)
    {
        $this->orderPizza = $orderPizza;
    }

    /**
     * @param IRequestData $data
     *
     * @SuppressWarnings(PHPMD.UnusedFormalParameter)
     */
    function execute(IRequestData $data)
    {
        $parameters = $data->getParameters();
        $order = $parameters["order"];
        $this->orderPizza->run(new PizzaOrder(
            $order["name"],
            $order["double_dough"],
            $order["sauces"]
        ));
    }
}
```

Przykład implementacji

```php
use Conpago\Pizza\Business\Contract\Model\Ingredient;
use Conpago\Pizza\Business\Contract\PresenterModel\IPizza;
use Conpago\Pizza\Business\Contract\Presenter\IOrderPizzaPresenter;
use Conpago\Presentation\Contract\IJsonPresenter;
```

```php
class JsonOrderPizzaPresenter implements IOrderPizzaPresenter
{
    /**
     * @var
     */
    private $jsonPresenter;

    function __construct(IJsonPresenter $jsonPresenter)
    {
        $this->jsonPresenter = $jsonPresenter;
    }

    function deliver(IPizza $pizza)
    {
        $func = function(Ingredient $i){
            return $i->getName();
        };

        $data = new StdClass;
        $data->ingredients = array_map($func, $pizza->getIngredients());
        $data->double_dough = $pizza->getDoubleDough();
        $data->sauces = $pizza->getSauces();

        $this->jsonPresenter->showJson($data);
    }
}
```

Przykład implementacji

```php
class OrderPizzaModule implements IModule
{

    /**
     * @param IContainerBuilder $builder
     *
     * @SuppressWarnings(PHPMD.StaticAccess)
     */
    public function build(IContainerBuilder $builder)
    {
        $builder->registerType('Conpago\Pizza\Presentation\Controller\OrderPizzaController')
            ->asA('Conpago\IController')
            ->asA('Conpago\Pizza\Presentation\Contract\Controller\IOrderPizzaController')
            ->keyed('OrderPizzaController')
            ->singleInstance();

        $builder->registerType('Conpago\Pizza\Business\Interactor\OrderPizza')
            ->asA('Conpago\Pizza\Business\Contract\Interactor\IOrderPizza')
            ->named('OrderPizza')
            ->singleInstance();

        $builder->registerType('Conpago\Pizza\Presentation\Presenter\JsonOrderPizzaPresenter')
            ->asA('Conpago\Pizza\Business\Contract\Presenter\IOrderPizzaPresenter')
            ->singleInstance();

        $builder->registerType('Conpago\Pizza\Dao\OrderPizzaDao')
            ->asA('Conpago\Pizza\Business\Contract\Dao\IOrderPizzaDao')
            ->singleInstance();
    }
}
```

BDD i TDD w praktyce

- błędy w BDD
- testowanie fabryk
- adaptacja bibliotek
- testowalna konfiguracja
- kolekcje w encjach biznesowych

Błędy w BDD

```php
/**
 * @param string $interactorName
 * @param IRequestData $data
 */
public function runInteractor($interactorName, IRequestData $data)
{
    /** @var \Conpago\DI\Lazy[] $controllers */
    $controllers = $this->container->resolveAll('Lazy<Conpago\\IController>');
    $controllerLazyInitializer = $controllers[$interactorName."Controller"];

    /** @var IController $controller */
    $controller = $controllerLazyInitializer->getInstance();
    $controller->execute($data);
}
```

Błędy w BDD

```php
private function servicePost($interactor, $data)
{
    $headers = ['Content-Type' => 'application/json'];
    $body = json_encode($data);

    $request = new Request(
            'POST',
            $this->getServiceUrl() . '?interactor=' . $interactor,
            $headers,
            $body);
    try {
        $res = $this->client->send( $request, [ 'cookies' => $this->jar ] );

        $body         = (string) $res->getBody();
        echo "Result body: ".$body;
        $this->result = json_decode( $body, true );
    }
    catch (ClientException $caught)
    {
        $res = $caught->getResponse();
        $body         = (string) $res->getBody();
        echo "Error body: ".$body;
        $this->result = json_decode( $body, true );
    }
}
```

Testowanie fabryk

- Wyprodukowanie klasy i jej uruchomienie to 2 różne odpowiedzialności
- Włożenie tych 2 odpowiedzialności do jednej klasy sprawia, że nie można napisać testów jednostkowych.

Adaptacja bibliotek

- Dla każdej zewnętrznej biblioteki należy napisać adapter w taki sposób aby nasza aplikacja miała z nią jak najmniej punktów stykowych (np. tylko Dao), tak aby w razie potrzeby można ją było stosunkowo łatwo zastąpić nie wykonując przy tym zmian w kodzie biznesowym.

Testowalna konfiguracja

- *array $config* – nie dostarcza informacji ani o strukturze konfiguracji ani o danych w niej zawartych.
- Lepszym rozwiązaniem jest dostarczenie drzewa interfejsów.

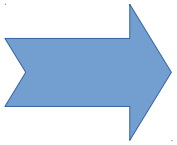
```php
interface ILoggerConfigProvider {
    /**
     * @return ILoggerConfig[]
     */
    function getConfigs();
}
```

```php
interface ILoggerConfig
{
    /**
     * @return string
     */
    function getLogLevel();

    /**
     * @return string
     */
    function getLogFilePath();
}
```

Kolekcje w encjach biznesowych

```php
class Document
{
  /**
   *  @return DocumentPosition[]
   */
  function getPositions(){
    //...//
  }
}
```

```php
class DocumentPosition
{
  /**
   *  @return Document
   */
  function getDocument(){
    //...//
  }
}
```



```php
class Document {
    /**
     * @return DocumentPosition
     */
    function getPosition($index) {
        //...//
    }
    /**
     * @return int
     */
    function countPositions() {
        //...//
    }
    function addPosition(DocumentPosition $position) {
        //...//
    }
    function removePosition(DocumentPosition $position) {
        //...//
    }
}
```

```php
class DocumentPosition {
    /**
     * @return Document
     */
    function getDocument() {
        //...//
    }
}
```

Pytania?

Dziękuję za uwagę