# Wzorzec architektoniczny MVC

## Projektowanie i programowanie systemów internetowych I

mgr inż. Krzysztof Rewak

18 marca 2018

Wydział Nauk Technicznych i Ekonomicznych
Państwowa Wyższa Szkoła Zawodowa im. Witelona w Legnicy

# Plan prezentacji

1. Wzorce projektowe i architektoniczne
2. Idea MVC
3. Reprezentacja danych
4. Warstwa prezentująca
5. Obsługa zapytań
6. Podsumowanie

# Wzorce projektowe i architektoniczne

# Jak należy pisać kod?

„Jeżeli coś jest głupie, ale działa, to nie jest głupie.”

Każdy student programowania wyznający taką zasadę powinien poważnie się zastanowić czy to faktycznie jest dziedzinia, w której chce się rozwijać.

# Jak należy pisać kod?

Tworzone oprogramowanie powinno być:

- czytelne,
- udokumentowane,
- zoptymalizowane względem redundancji i innych wybranych wskaźników,
- napisane zgodnie z konwencjami języka,
- napisane zgodnie z dobrymi praktykami programistycznymi,
- możliwe do rozwijania i rozszerzania.

# Wynajdowanie koła na nowo

Prawda jest taka, że wiele problemów jakie często napotykają programiści, to problemy, które ktoś już kiedyś napotkał i najczęściej rozwiązał.

Warto poznać sprawdzone rozwiązania, aby zaoszczędzić czasu przy budowaniu własnego systemu informatycznego.

Warto też jednak pamiętać, że wykorzystanie opracowanych wzorców to jedno, a budowanie aplikacji z gotowych wtyczek i bibliotek to coś całkiem innego!

# Wynajdowanie koła na nowo

Za Wikipedią:

Wzorce projektowe (*design pattern*) to uniwersalne, sprawdzone w praktyce rozwiązania często pojawiających się, powtarzalnych problemów projektowych. Pokazują powiązania i zależności pomiędzy klasami oraz obiektami i ułatwiają tworzenie, modyfikację oraz pielęgnację kodu źródłowego. **Są opisem rozwiązania, a nie jego implementacją.**

# Wzorce projektowe

Warto zainteresować się kilkoma popularnymi wzorcami:

- singleton,
- adapter,
- strategia,
- budowniczy,
- fabryka abstrakcyjna,
- wstrzykiwanie zależności.

# A gdyby objąć wzorcem podwaliny całego systemu?

Gdy wzorce projektowe rozwiązują problemy zachodzące przede wszystkim między obiektami, wzorzec architektoniczny jest rozwiązaniem pomagającym na skalę całej aplikacji.

# Wzorce architektoniczne

Wzorce architektoniczne opisują:

- założenia architektury systemu,
- strukturę aplikacji,
- sposoby komunikacji wewnętrznej i zewnętrznej,
- funkcjonalności systemu.

# Wzorce architektoniczne

Spośród popularnych wzorców można wymienić:

- ETL, *extract-transform-load*,
- P2P, *peer-to-peer*,
- ESB, *enterprise service bus*,
- EDA, *event-driven architecture*,
- szeroko rozumiane mikroserwisy...

# Idea MVC

# MVC?

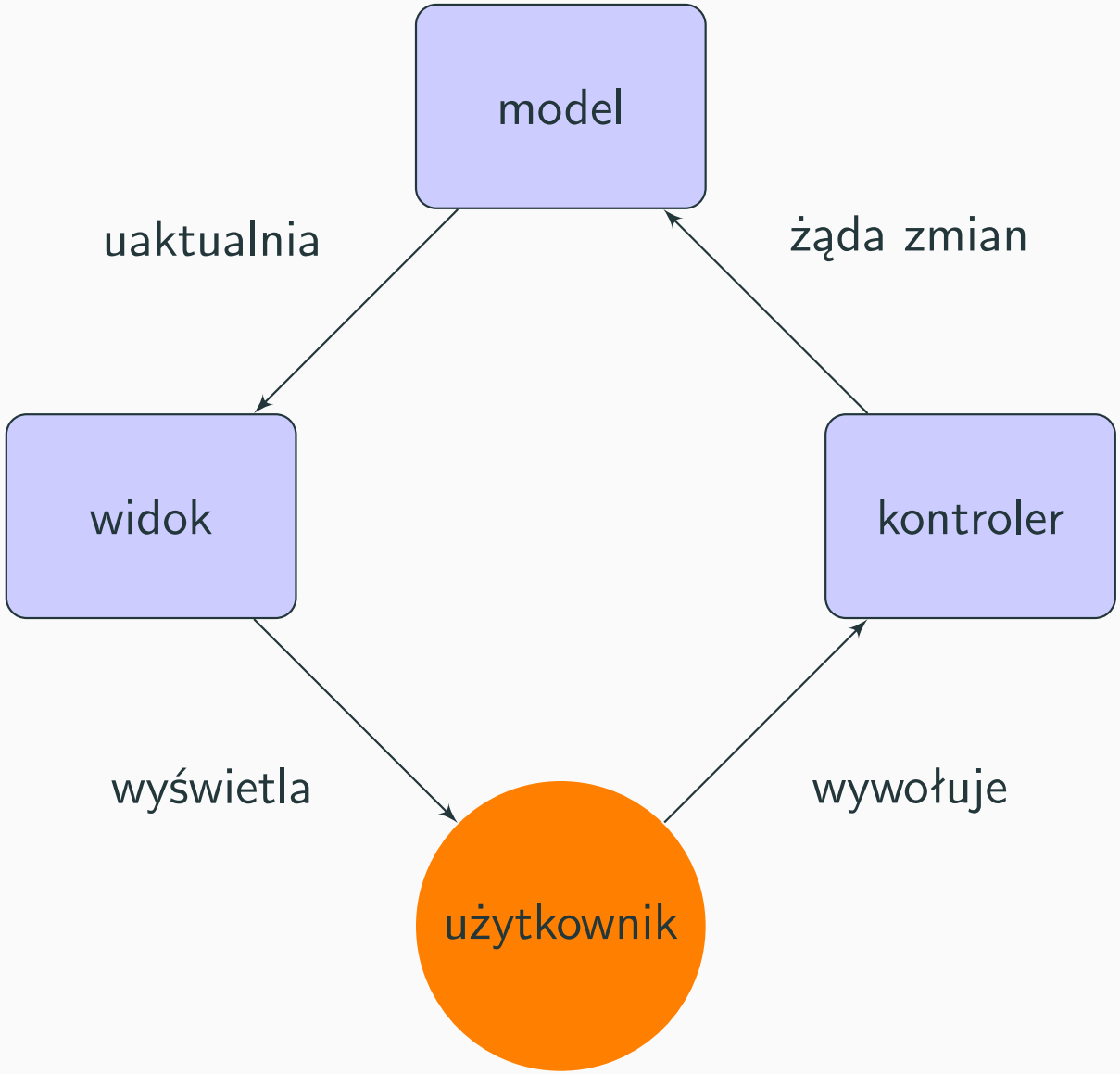
Skrót **MVC** należy rozumień następująco:

**M**odel - reprezentacja danych

**V**iew, widok - warstwa prezentująca

**C**ontroller, kontroler - obsługa zapytań

# Podstawowy schemat interakcji



# Występowanie

MVC to podstawowy wzorzec wykorzystywany w najpopularniejszych frameworkach:

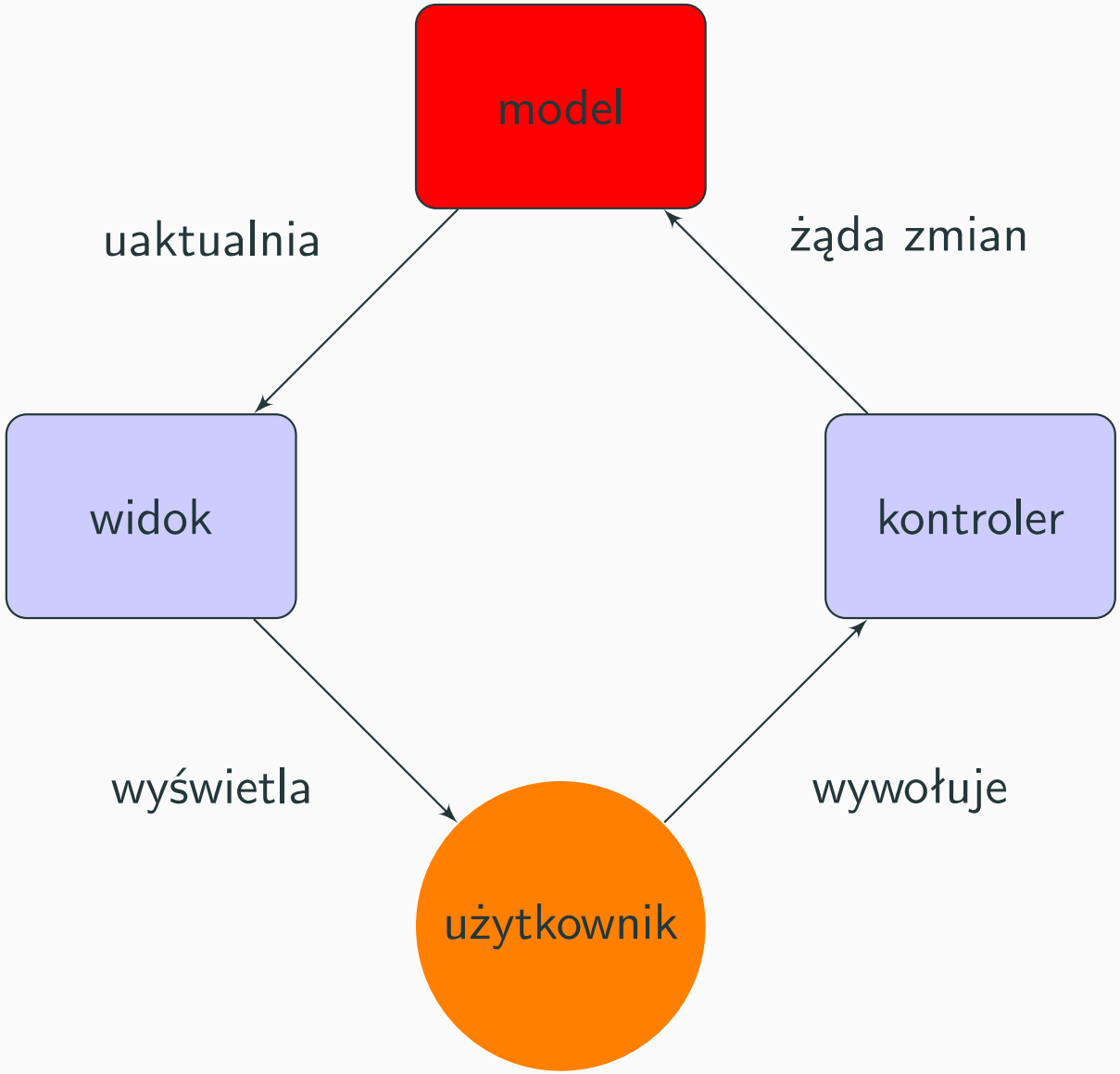
- PHP: Laravel, Symfony, Phalcon, Yii
- ASP.NET MVC
- Java: Swing, Spring
- Ruby on Rails
- Django

# Reprezentacja danych

# Wielkie M

Model należy rozumieć jako reprezentację danych, pewną strukturę danych albo nawet sposób na pobieranie i zarządzanie danymi.

# Wielkie M


model
uaktualnia
żąda zmian
widok
kontroler
wyświetla
wywołuje
użytkownik

# Więc chodź, zamodeluj mój świat, na żółto i na niebiesko...

```php
<?php

use namespace App\Models;

class User {
  private $uid;
  private $login;
  private $password;

  public function __construct(string $login, string $password)
  {
    $this->id = uniqid();
    $this->login = $login;
    $this->password = password_hash($password, PASSWORD_BCRYPT);
  }
}
```

# Więc chodź, zamodeluj mój świat, na żółto i na niebiesko...

```
from bcrypt import hashpw, gensalt
import uuid

class User(object):

  def __init__(self, name, password):
    self.id = uuid.uuid4()
    self.name = name
    self.password = hashpw(password, gensalt())
```

# Skąd brać dane?

Dane do modelu można pobrać z różnych źródeł i wszystko tak naprawdę zależy od tego, czego potrzebuje aplikacja. Dane mogą pochodzić z:

- bazy danych,
- systemu pamięci podręcznej,
- z serializowanych obiektów,
- z plików,
- prosto z kodu.

# Co ten model robi?

Istnieją dwa podstawowe sposoby zarządzania modelami, ale o tym porozmawiamy przy okazji wykładu 9. *Mapowanie relacyjno-obiektowe*.

Klasyczny wzorzec MVC mówi, że model zawiera w sobie podstawową logikę biznesową. Dlatego na każdym obiekcie modelu powinno móc się wywoływać konkretne metody, np.:

```
public class Application
{
  public static void Main()
  {
    User user = new User("krewak", "secretpassword");
    user.SendNotificationEmail();
    user.ToggleStatus();
    user.Save();
  }
}
```

# Zastrzeżenie!

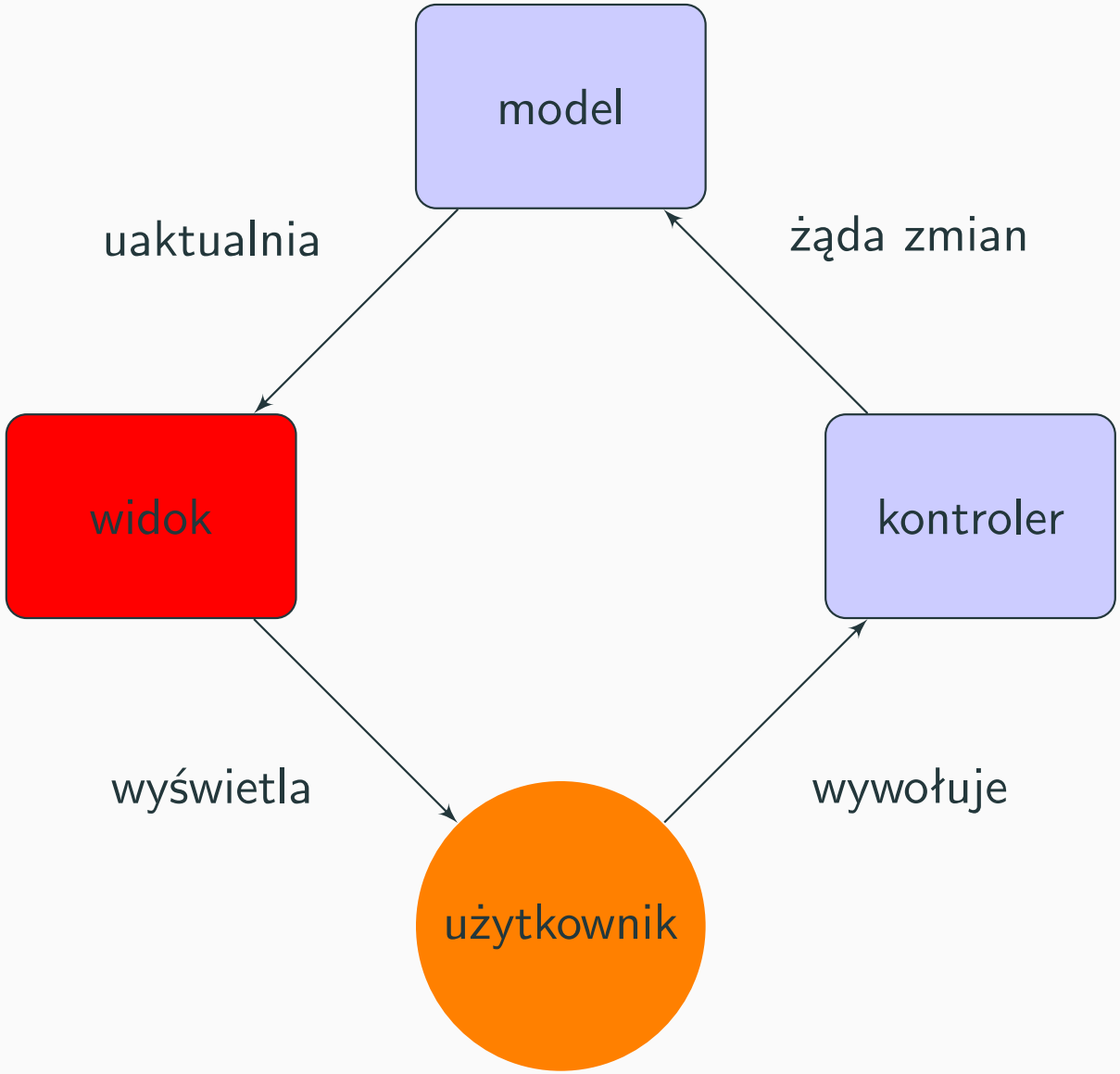
Należy jednak pamiętać, że im większy system, tym bardziej skomplikowana staje się jego architektura. Często MVC rozszerza się o dodatkowe komponenty i wówczas pokazane przed chwilą podejście nie jest już prawidłowe.

# Warstwa prezentująca

# Wielkie V

Widok należy rozumieć jako warstwę prezentującą, ale również umożliwiającą interakcję z systemem.

# Wielkie V


model
uaktualnia
żąda zmian
widok
kontroler
wyświetla
wywołuje
użytkownik

# Widok na widok

Poniżej widać dobry przykład odseparowania warstwy wizualnej za pomocą PHP-owego Twiga.

```
<table class="table">
  {% for user in users %}
    <tr data-user-id="{{ user.id }}">
      <td>{{ user.id }}</td>
      <td>{{ user.login }}</td>
      <td>{{ user.email }}</td>
      <td>{{ user.status }}</td>
      <td>
        <button class="btn btn-danger remove-user">
          delete
        </button>
      </td>
    </tr>
  {% endfor %}
</table>
```

# Formy widoku

Widok najczęściej utożsamiany był w systemach internetowych z HTML-em, jednakże warstwą prezentacji może być cokolwiek, co zrozumie użytkownik lub inny system:

- wspomniany HTML,
- XML,
- JSON,
- lub własny format lub standard.

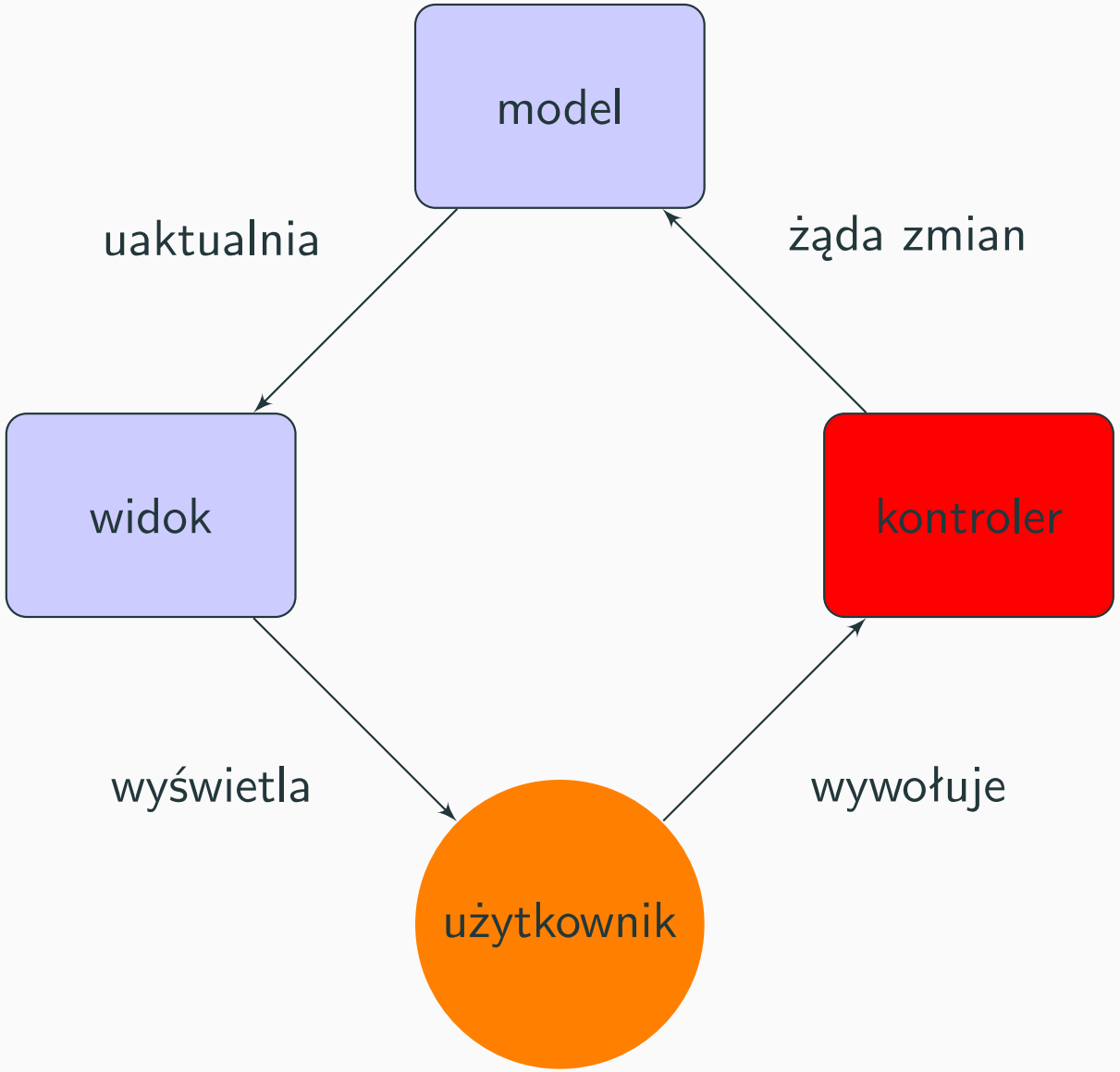
Oczywiście HTML daje największe możliwości dla przeglądarki oraz możliwość łatwego wywołania kontrolera.

# Obsługa zapytań

# Wielkie C

Kontroler należy przede wszystkim rozumieć jako obsługę zapytań. Jego podstawowym zadaniem jest przyjęcie zapytania od serwera HTTP oraz zwrócenie mu odpowiedzi.

# Wielkie C


model
uaktualnia
żąda zmian
widok
kontroler
wyświetla
wywołuje
użytkownik

# Co może kontroler w MVC?

- przyjąć dane wejściowe,
- znaleźć zbiór lub konkretny model oraz dokonać na nim zmian,
- przekierować na inny kontroler,
- zwrócić odpowiedź w formie widoku.

# Kontrolowanie kontrolerów

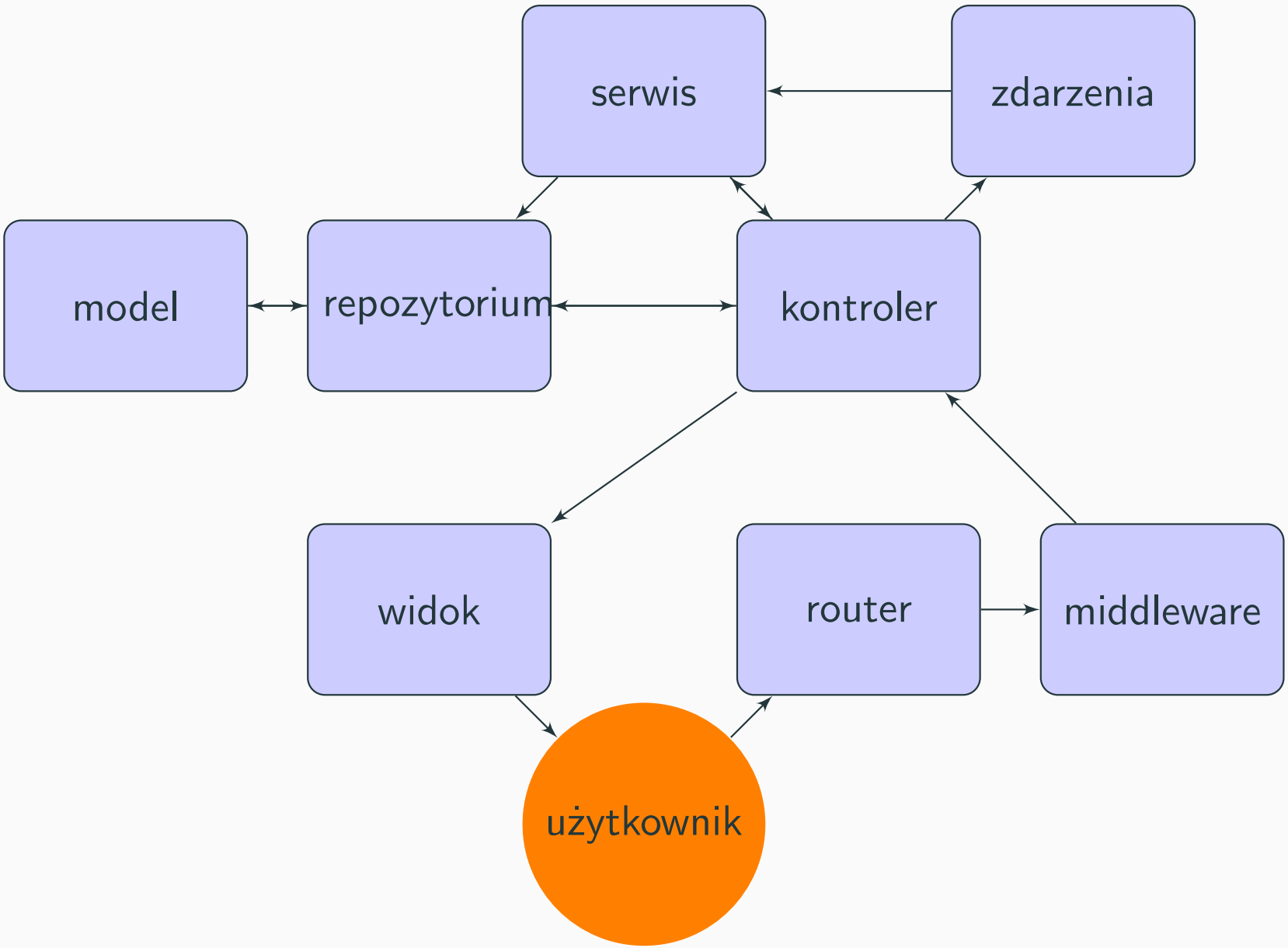
```php
<?php

namespace PWSZ\Controllers;

use Phalcon\Http\Response;

class NewsController extends Controller {

    public function getNewsAction(): Response {
        $this->dataset->setData(News::find())
        return $this->renderResponse();
    }

    public function getEntryAction(int $id): Response {
        $this->dataset->setData(News::findFirst($id))
        return $this->renderResponse();
    }

}
```

# Co może kontroler poza MVC?

Wzorzec MVC jest bardzo popularną koncepcją, jednakże rzadko bywa używany w swojej klasycznej formie. Najczęstszymi rozszerzeniami są:

- serwisy, które wykonują logikę biznesową zamiast modeli,
- repozytoria, które zarządzają modelami,
- zdarzenia, które łamią jednokierunkowy przepływ zapytania,
- i inne - zależne od progrmiasty i wymagań systemu.

# Możliwe rozszerzenie systemu



# Podsumowanie

# Wady i zalety?

- +/- popularność
- +/- separacja odpowiedzialności
- +/- złożoność systemu
- +/- testowanie
- +/- umożliwienie pracy *full-stack developerom*

# Bibliografia i ciekawe źródła


- https://pl.wikipedia.org/wiki/Wzorzec_projektowy_(informatyka)
- https://pl.wikipedia.org/wiki/Wzorzec_architektoniczny
- https://msdn.microsoft.com/en-us/library/ff649643.aspx


Wzorce w praktyce:


- https://designpatternsphp.readthedocs.io/en/latest/
- http://www.inforsoft.nazwa.pl/inforsoft-nowy/kody-c/

# Pytania?

Kod prezentacji dostępny jest w repozytorium git pod adresem
`https://bitbucket.org/krewak/pwsz-ppsi`

Wszystkie informacje dot. kursu dostępne są pod adresem
`http://pwsz.rewak.pl/kursy/4`