Wyższa Szkoła Bankowa
w Gdańsku

MVC

Katarzyna Duchowska

Wzorce projektowe

- Podczas programowania często trafiamy na podobne problemy
- W 99% przypadków nie są to wyjątkowe problemy – ktoś wcześniej już na taki trafił i rozwiązał
- Wzorce projektowe to sztandarowe rozwiązania często spotykanych problemów
- Nie jest to gotowa implementacja – raczej ogólna koncepcja, w jaki sposób osiągnąć założony cel

Wzorce architektoniczne

- za Wikipedią:

  Architektura oprogramowania *– podstawowa organizacja systemu wraz z jego komponentami, wzajemnymi powiązaniami, środowiskiem pracy i regułami ustanawiającymi sposób jej budowy i rozwoju*

- Wzorce architektoniczne, podobnie jak projektowe, to szablony rozwiązań dla często spotykanych problemów, dotyczą jednak dziedziny architektury oprogramowania

Czym się różnią?

- **Wzorce projektowe** zwykle opisują zależności między kilkoma klasami, które mają rozwiązać problem

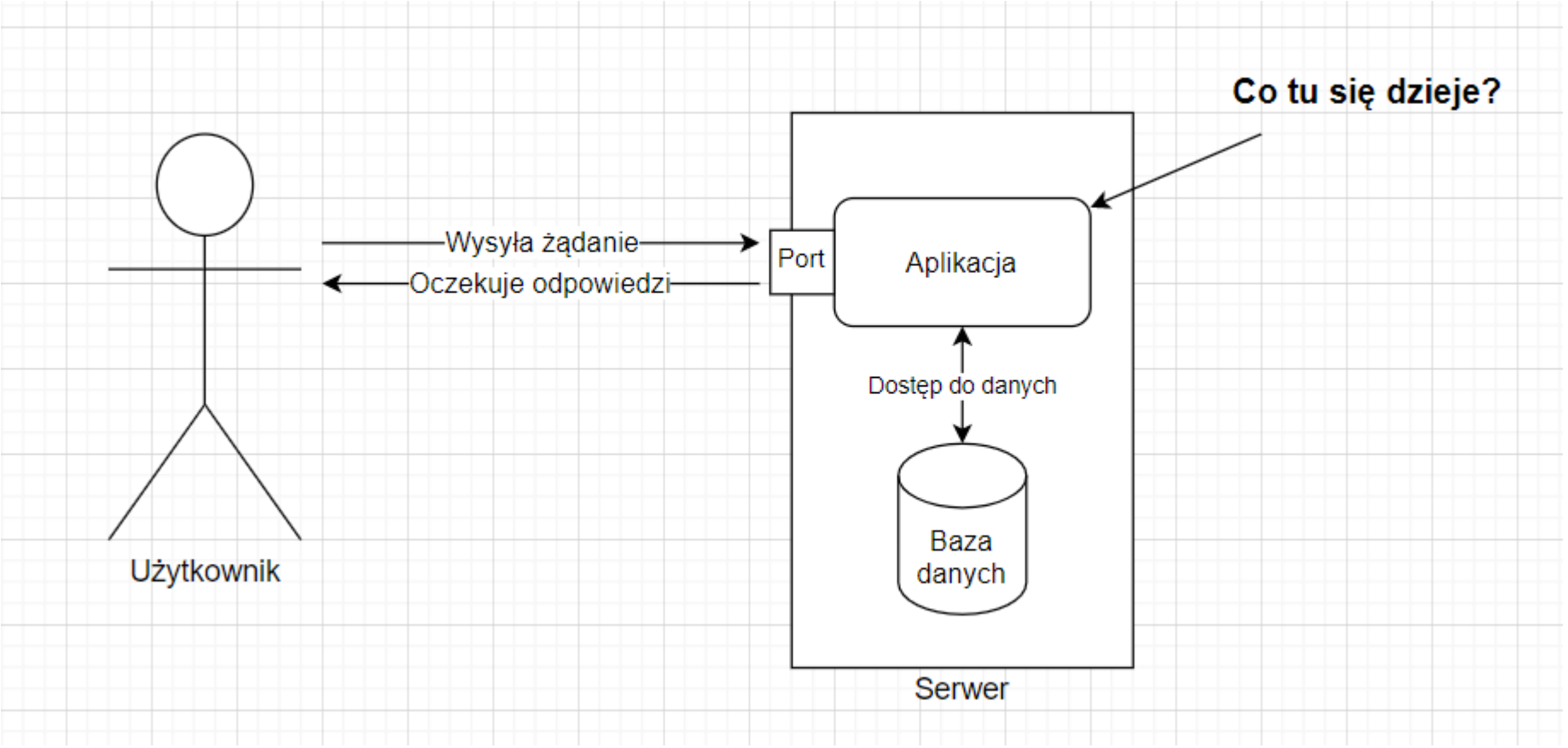
- **Wzorce architektoniczne** rozwiązują problemy na poziomie modułów – mówią o tym, jak zorganizować *cały system*

Jaki mamy dziś problem?

- Na tym kursie zajmujemy się aplikacjami internetowymi i tym, co się dzieje na serwerze takiej aplikacji
  - W jaki sposób powinniśmy zaprojektować architekturę aplikacji webowej?
  - Jak powinniśmy zorganizować klasy i moduły „żeby było dobrze”?

- Omówimy sobie dwa wzorce architektoniczne, na których często są zbudowane frameworki backendowe

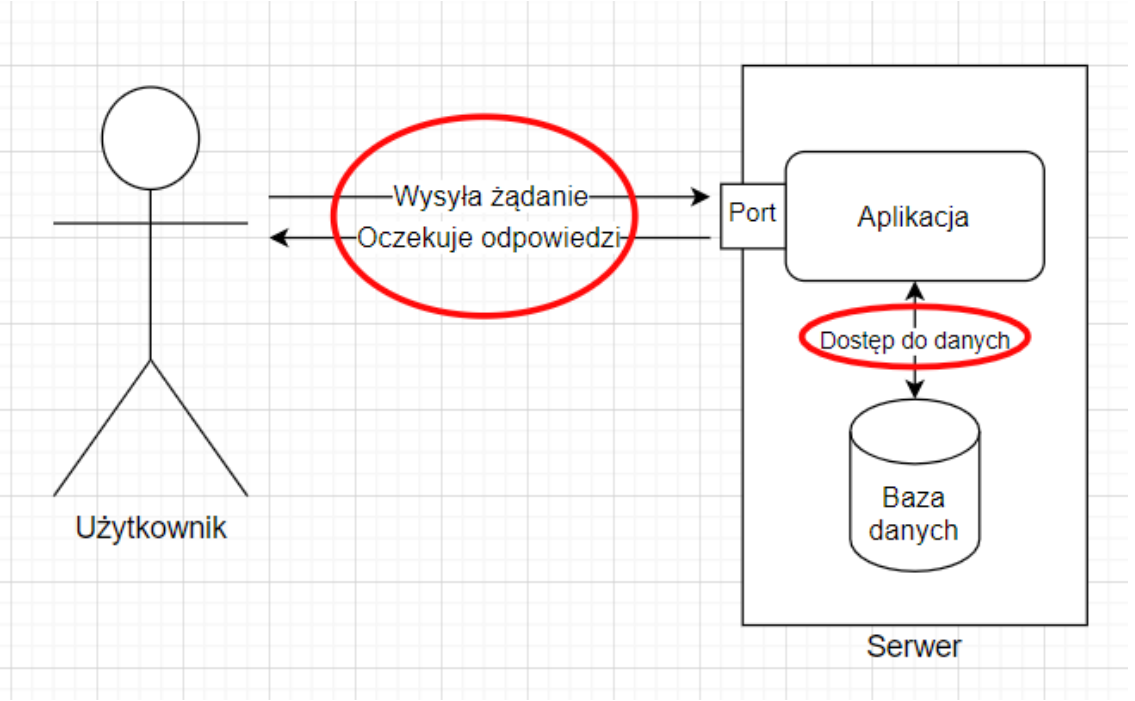
Jak wygląda korzystanie z aplikacji?



Jak wygląda korzystanie z aplikacji?

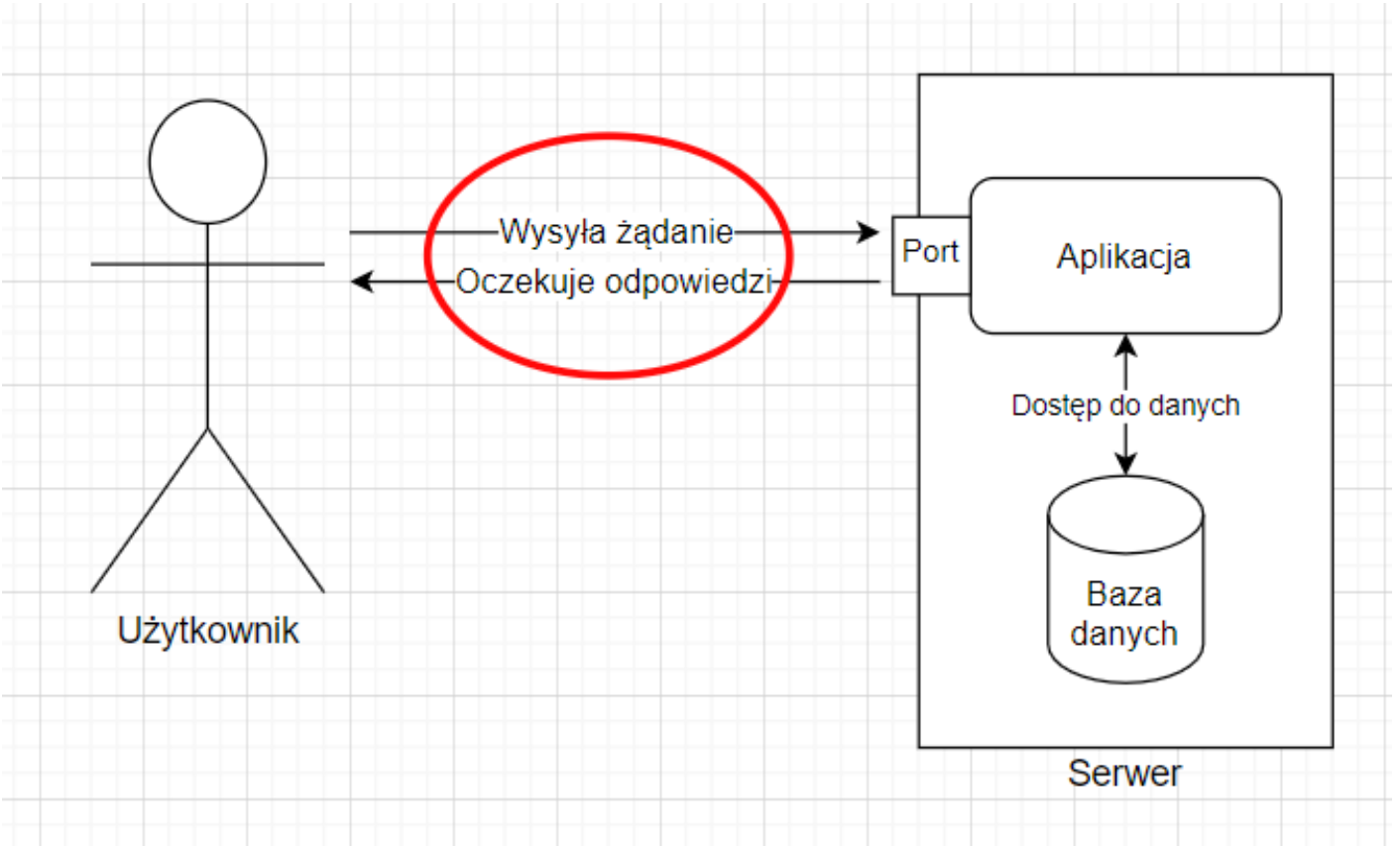
1. Użytkownik wysyła żądanie do serwera
2. Serwer umie przekazać żądanie do aplikacji
3. Aplikacja przyjmuje żądanie
4. Aplikacja obsługuje żądanie; jeśli trzeba:
   - Pobiera potrzebne dane z bazy danych
   - Operuje na tych danych, zgodnie z założoną logiką biznesową
   - Wprowadza zmiany w bazie danych
5. Aplikacja zwraca użytkownikowi odpowiedź:
   - Mogą to być same dane (np. JSON)
   - Może być to widok (dokument HTML, który użytkownik zobaczy w przeglądarce)

Jak wygląda korzystanie z aplikacji?

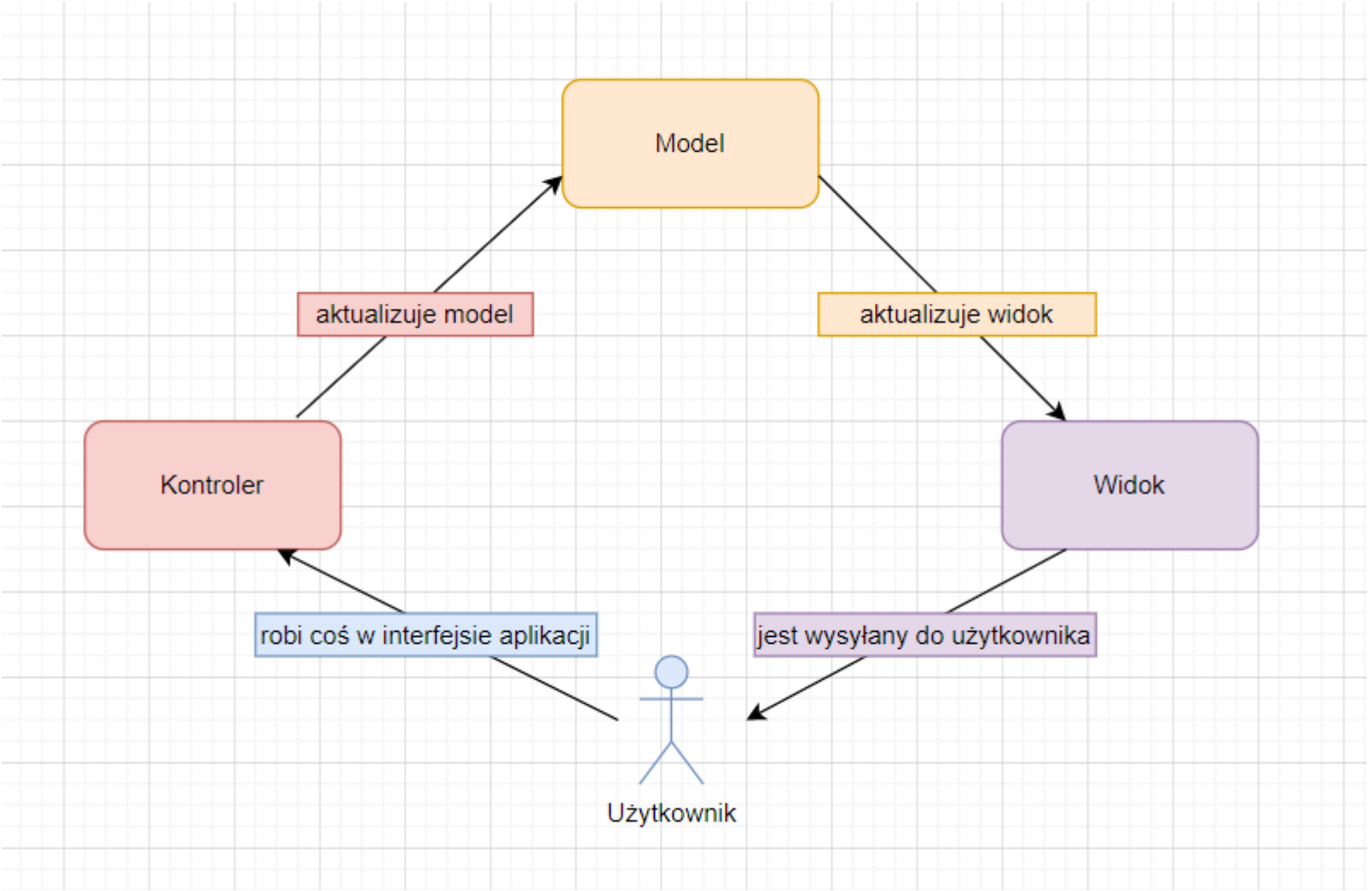


- Te operacje mają miejsce w przypadku niemal każdej aplikacji webowej
- Jak zatem zorganizować strukturę aplikacji?

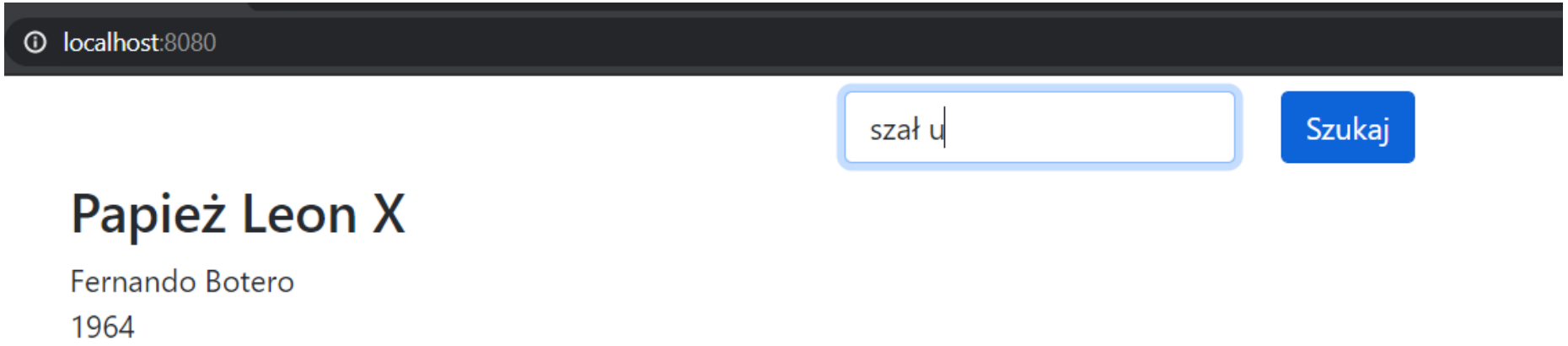
MVC – Model-View-Controller



Podział odpowiedzialności

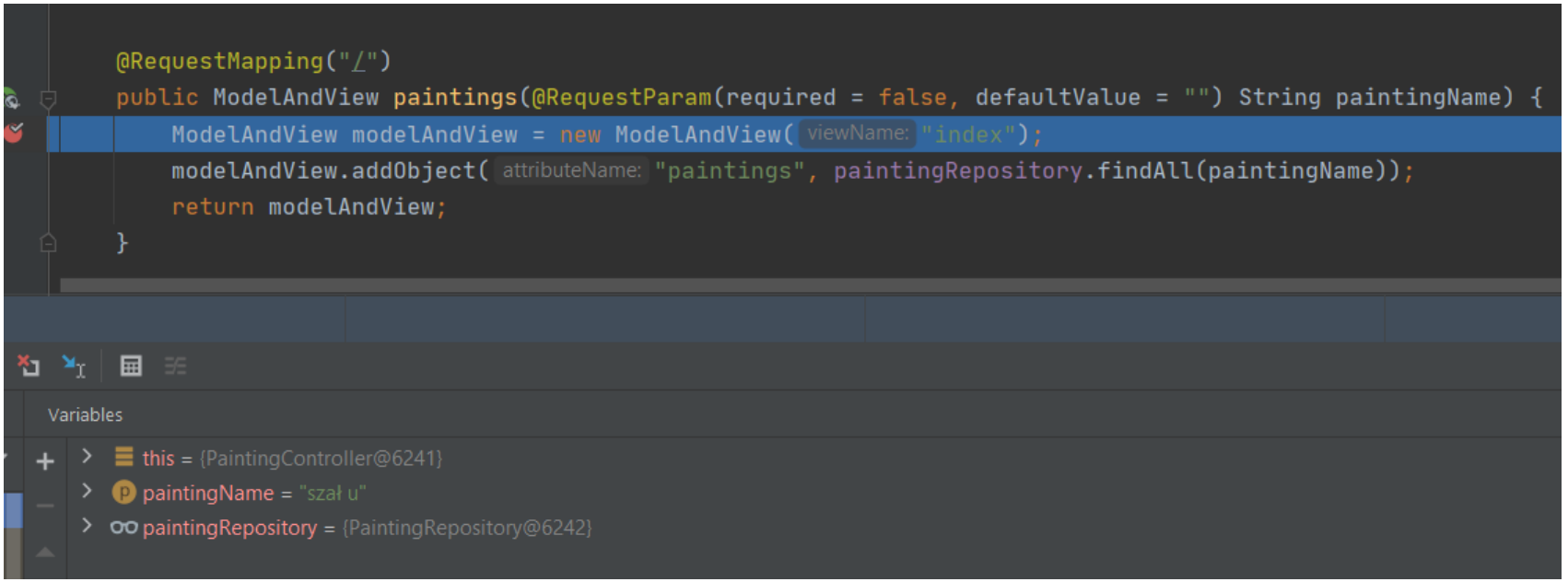


Użytkownik robi coś w interfejsie aplikacji



- np. uzupełnia dane w formularzu, naciska przycisk

Informacja o zdarzeniu trafia do kontrolera



- to znaczy: do kontrolera przychodzi żądanie wraz z danymi, tutaj – z nazwą, którą chce wyszukać użytkownik

Kontroler aktualizuje model

```
@RequestMapping("/")
public ModelAndView paintings(@RequestParam(required = false, defaultValue = "") String paintingName) {
    ModelAndView modelAndView = new ModelAndView( viewName: "index");  modelAndView: "ModelAndView [view="
    modelAndView.addObject( attributeName: "paintings", paintingRepository.findAll(paintingName));  modelAnd
    return modelAndView;
}
```

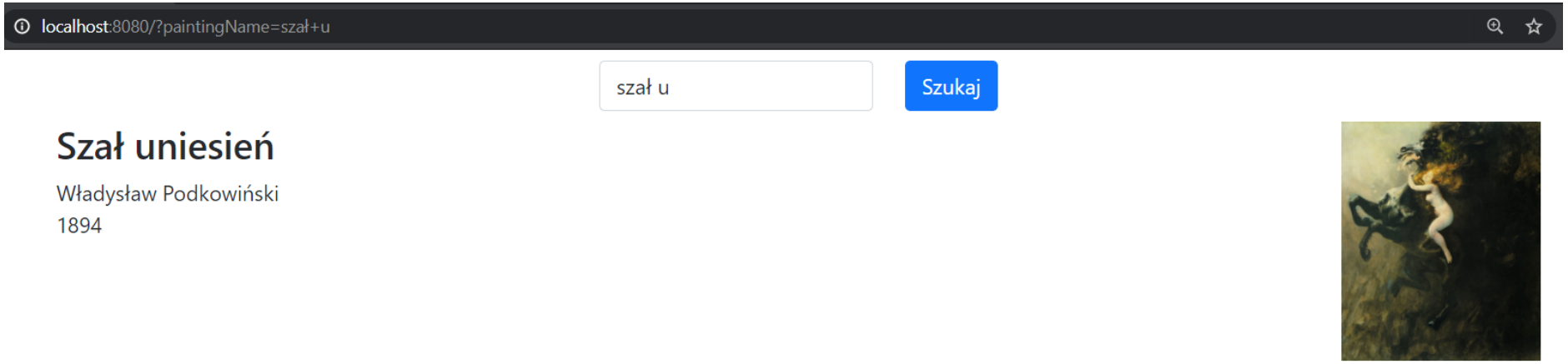
- to znaczy: wyszukuje te obrazy, które pasują do wpisanej przez użytkownika frazy

Model aktualizuje widok

```
<div class="row" th:each="painting : ${paintings}">
    <div class="col-sm-6">
        <h3 th:text="${painting.getName()}"></h3>
        <div th:text="${painting.getAuthor()}"></div>
        <div th:text="${painting.getYear()}"></div>
    </div>
    <div class="col-sm-6">
        <img th:src="${painting.getUrl()}" width="150px" class="float-end"/>
    </div>
</div>
```

- to znaczy: widok wie, że lista obrazów uległa zmianie i należy wyświetlić inny wynikowy HTML

Użytkownik otrzymuje nowy widok



Co to znaczy, że aplikacja „obsługuje” żądanie?

- MVC mówi o tym, że kontroler ma „zaktualizować model”
- Jak było na jednym z poprzednich slajdów:

  Aplikacja obsługuje żądanie; jeśli trzeba:

  - *Pobiera potrzebne dane z bazy danych*
  - *Operuje na tych danych, zgodnie z założoną logiką biznesową*
  - *Wprowadza zmiany w bazie danych*

- Jak ułożyć w kodzie operacje na danych?

Architektura wielowarstwowa

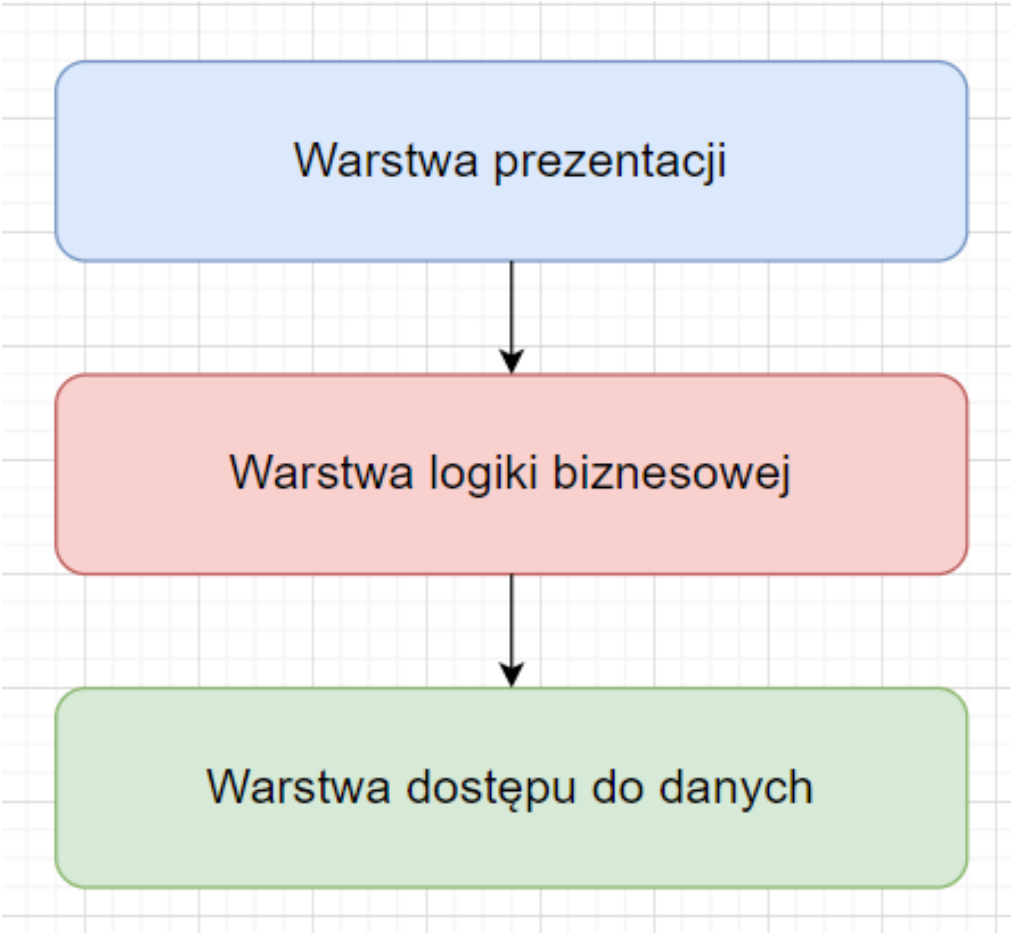
- Dzielimy aplikację na warstwy
- Każda z warstw ma określoną odpowiedzialność – warstwy są oddzielone logicznie, a czasem nawet fizycznie (są uruchamiane na innych maszynach)
- Liczba warstw jest zależna od konkretnej implementacji; innymi słowy – warstw jest tyle, ile potrzeba

Architektura wielowarstwowa

- Wyższe warstwy mogą korzystać z usług niższych warstw, ale nie odwrotnie
  - **Architektura otwarta –** warstwa może korzystać z usług wszystkich niższych warstw
  - **Architektura zamknięta –** warstwa może korzystać z usług warstwy tylko leżącej bezpośrednio pod nią

- W aplikacjach webowych najczęściej pojawia się **architektura trójwarstwowa**

Architektura trójwarstwowa



Warstwa prezentacji

- ***@Controller***
- Odpowiada za prezentację danych
- Wszystko, co związane z frontendem (a więc HTML, CSS, JS – w postaci pojedynczych plików bądź oddzielnej aplikacji), ale też obsługa przychodzącego żądania i przygotowanie stosownej odpowiedzi

```
@PostMapping("/save")
public String save(@ModelAttribute Painting painting) {
    paintingService.save(painting);
    return "redirect:/";
}
```

Warstwa logiki biznesowej

- ***@Service***
- Tutaj mamy implementację wymagań klienta

```
public void save(Painting painting) {
    log.info("Próbujemy zapisać obraz...");
    log.info("Wykonujemy skomplikowane operacje biznesowe - np. wysyłamy powiadomienia mejlowe o dodanym obrazie...");
    log.info("Tu się dzieje cała logika biznesowa!");

    paintingRepository.save(painting);

    log.info("Zapisaliśmy obraz!");
}
```

Warstwa dostępu do danych

- ***@Repository***
- Baza danych + mechanizmy dostępu do niej i wykonywania w niej operacji

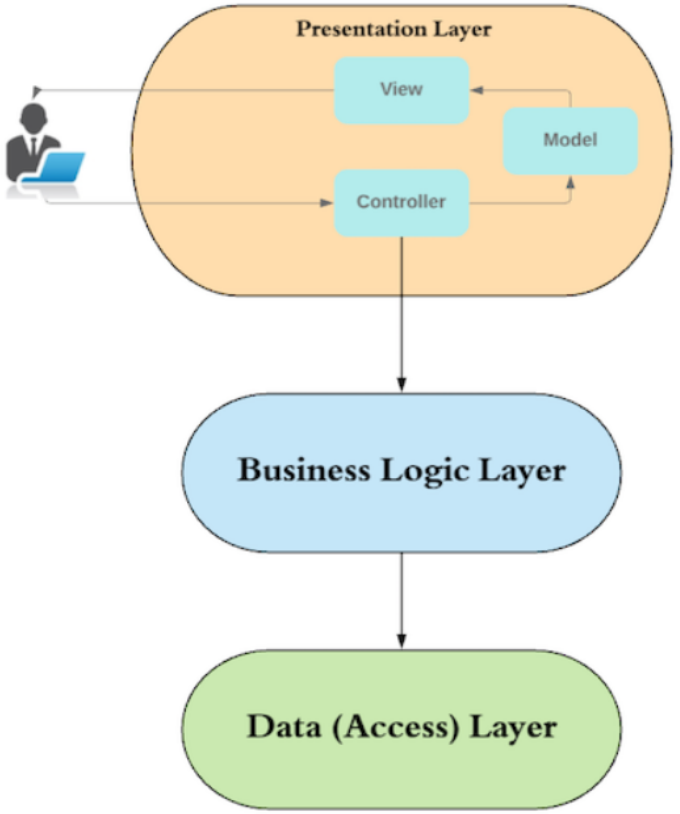
```
@Override
public <S extends Painting> S save(S s) {
    paintings.add(s);
    return s;
}
```

Chwila na przejrzenie kodu!

- https://github.com/Kaatarzyna/WSB_MVC

Jak się ma MVC do warstw?

The controller component of MVC is the connection point between the two layers:



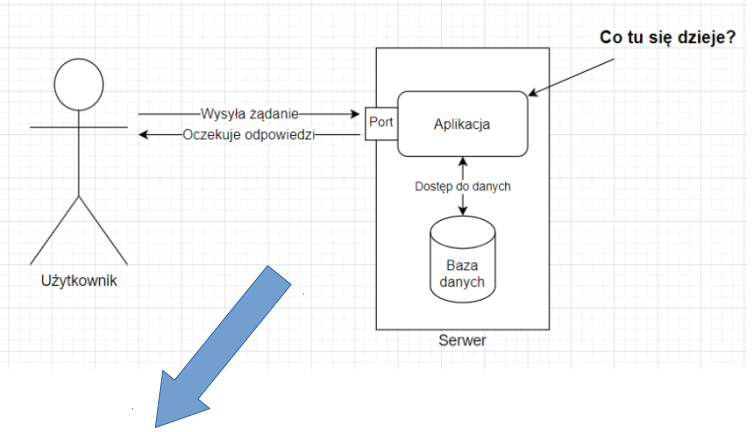
How MVC and 3-tier architecture relate to each other

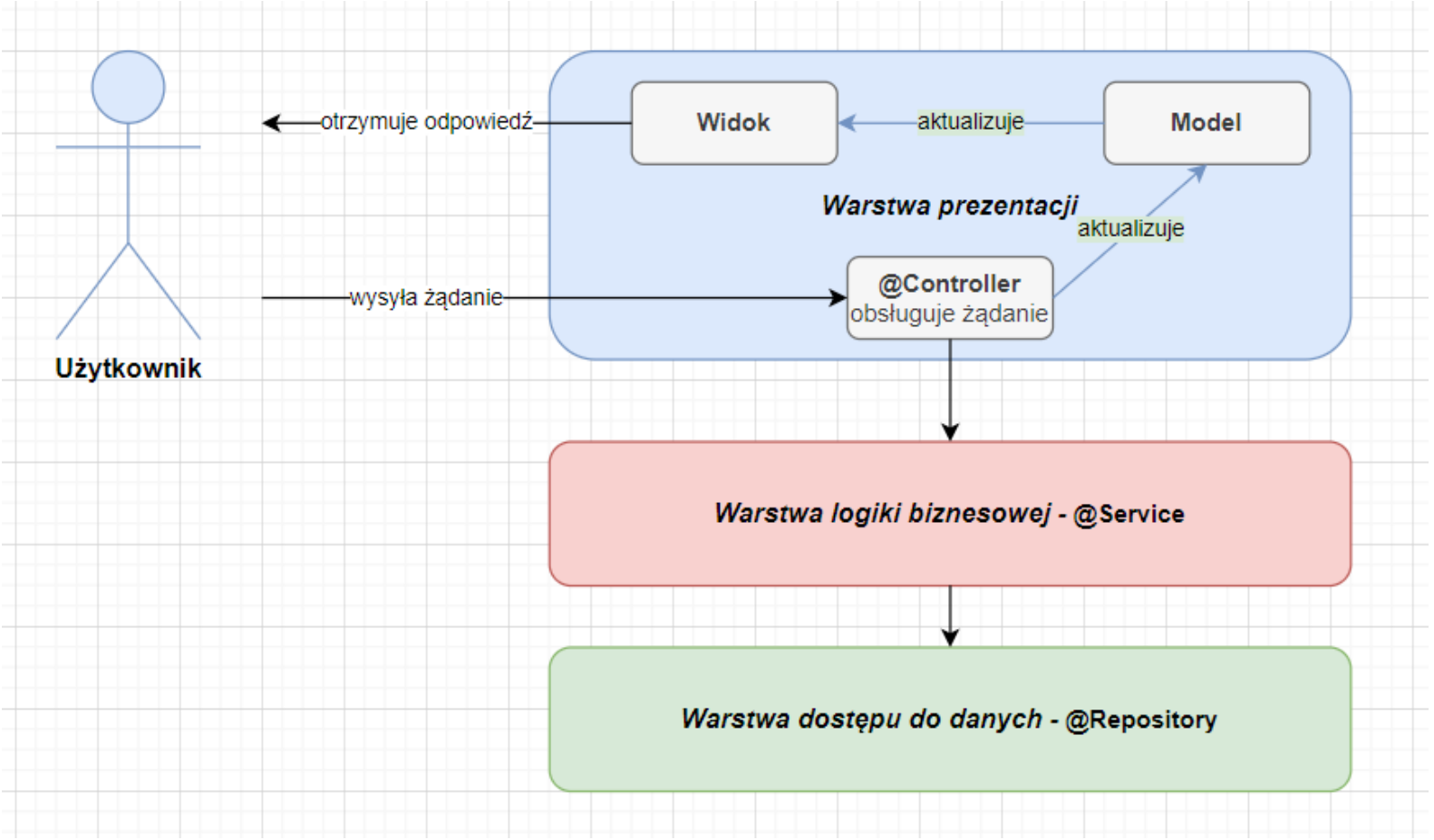
Źródło:
https://openclassrooms.com/en/courses/5684146-create-web-applications-efficiently-with-the-spring-boot-mvc-framework/6156961-organize-your-application-code-in-three-tier-architecture

Po co nam warstwy?

- Zrozumiały podział zadań między warstwami – łatwiej się odnaleźć w kodzie
- Mniejsze zależności między warstwami – mniejsze ryzyko, że drobna zmiana wywoła pożar w zupełnie nieoczekiwanym miejscu kodu
- Warstwy mogą być rozwijane niezależnie
- Przyjemny artykuł Martina Fowlera czemu warstwy są spoko: https://www.martinfowler.com/bliki/PresentationDomainDataLayering.html

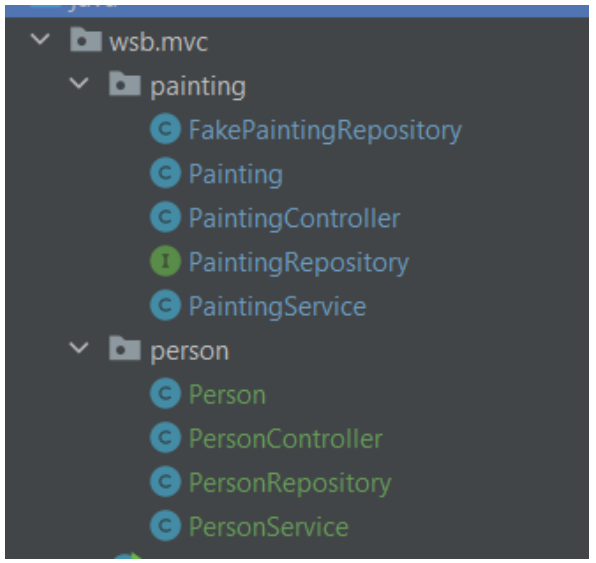
Podsumowanie



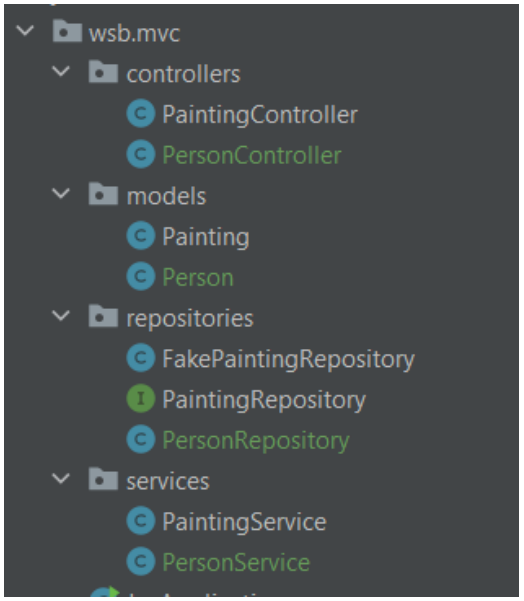


Bonus: jak organizować kod w pakiety?

Cargo cult programming is a style of computer programming characterized by the ritual inclusion of code or program structures that serve no real purpose.



?



- Lepsza enkapsulacja
- Klasy, które często korzystają ze swoich usług są blisko siebie – lepsza spójność oprogramowania
- Łatwiej budować zamknięte funkcjonalnie moduły

A to co za czary?

```java
@Controller
public class PaintingController {

    private final PaintingService paintingService;

    public PaintingController(PaintingService paintingService) {
        this.paintingService = paintingService;
    }
```

- Nigdzie w kodzie nie tworzymy nowych kontrolerów ani serwisów – próżno szukać w naszym kodzie wywołania ich konstruktorów
- A jednak powstają, co więcej, potrafią skorzystać z podanych w argumencie konstruktora klas
- Jak?

Czym się różni biblioteka od frameworku?

Biblioteka

- Zestaw funkcji, klas, typów, zwykle pogrupowanych w pakiety, z którego programista może skorzystać podczas pisania swojej aplikacji
- Przykład – biblioteka standardowa Javy – żeby wypisać coś na konsolę korzystamy z gotowej funkcji

System.out.println(„Hello world!”);

- My decydujemy, w którym momencie chcemy ją wywołać

Framework

- Gotowy szkielet aplikacji pod dane zastosowanie
- Od razu jest zdefiniowany ogólny mechanizm działania aplikacji i jej architektura
- Zadaniem programisty jest nie tyle napisanie całej aplikacji od zera, co ***uzupełnienie kodu frameworku***, w celu osiągnięcia pożądanego rezultatu
- *Hollywood Principle - „Don't Call Us, We'll Call You”*
- Oddajemy władzę frameworkowi!

IoC – Inversion of Control

- za Wikipedią:

  Odwrócenie sterowania *(ang. Inversion of Control, IoC) – paradygmat (czasami rozważany też jako wzorzec projektowy lub wzorzec architektury) polegający na przeniesieniu funkcji sterowania wykonywaniem programu do używanego frameworku. Framework w odpowiednich momentach wywołuje kod programu stworzony przez programistę w ramach implementacji danej aplikacji.*

DI – Dependency Injection

- Wstrzykiwanie zależności – najpopularniejsza realizacja IoC
- za Wikipedią:

  Wstrzykiwanie zależności *(ang. Dependency Injection, DI) – wzorzec projektowy i wzorzec architektury oprogramowania polegający na usuwaniu bezpośrednich zależności pomiędzy komponentami na rzecz architektury typu plug-in. Polega na przekazywaniu gotowych, utworzonych instancji obiektów udostępniających swoje metody i właściwości obiektom, które z nich korzystają (np. jako parametry konstruktora).*

687

IoC is a generic term meaning that rather than having the application call the implementations provided by a *library* (also know as *toolkit*), a *framework* calls the implementations provided by the application.

DI is a form of IoC, where implementations are passed into an object through constructors/setters/service lookups, which the object will 'depend' on in order to behave correctly.

https://stackoverflow.com/questions/6550700/inversion-of-control-vs-dependency-injection

Czyli o co chodzi?

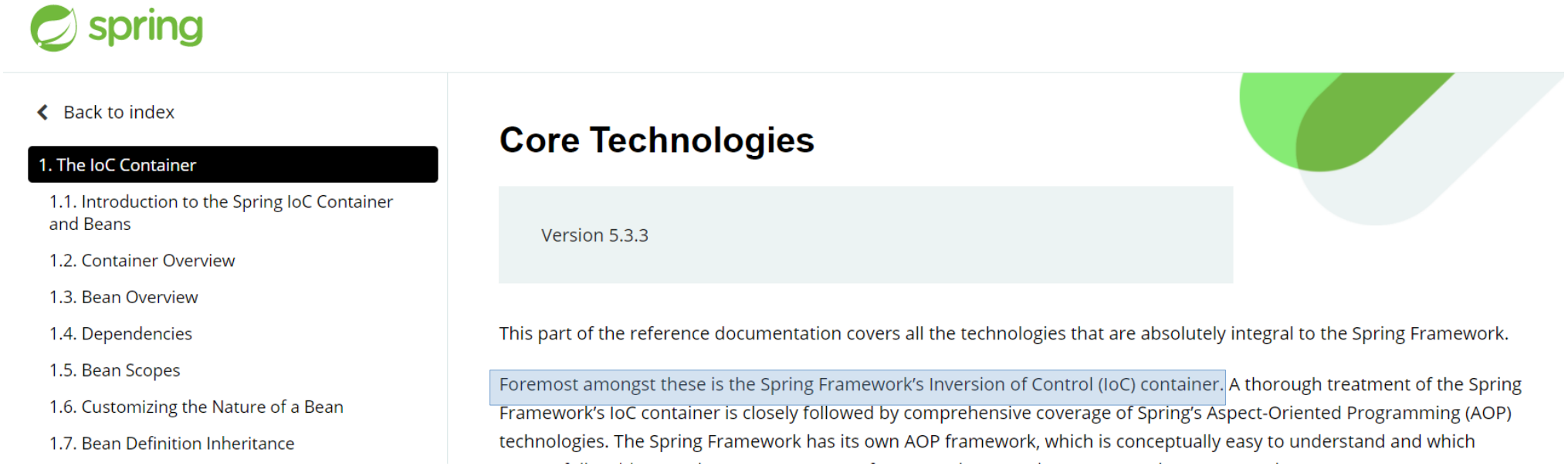
- Obiekt definiuje swoje zależności bez tworzenia ich
- Proces tworzenia tych zależności jest delegowany do kontenera IoC

```java
@Controller
public class PaintingController {

    private final PaintingService paintingService;

    public PaintingController(PaintingService paintingService) {
        this.paintingService = paintingService;
    }
```

IoC, DI a Spring Framework



https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#spring-core

IoC, DI a Spring Framework

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

IoC, DI a Spring

- Trzy zdania z dokumentacji, které niejako załatwiają nam cały temat:
  - *The org.springframework.context.ApplicationContext interface represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the beans.*
  - *The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata.*
  - *The configuration metadata is represented in XML, Java annotations, or Java code.*

Realizacja DI w Springu

- Wstrzykiwanie przez konstruktor:

```java
@Controller
public class PaintingController {

    private final PaintingService paintingService;

    public PaintingController(PaintingService paintingService) {
        this.paintingService = paintingService;
    }
```

- Wstrzykiwanie przez setter:

```java
@Controller
public class PaintingController {

    private PaintingService paintingService;

    @Autowired
    public void setPaintingService(PaintingService paintingService) {
        this.paintingService = paintingService;
    }
```

Skąd wiadomo, które klasy są zarządzane przez kontener IoC?

- W naszych przykładach – odpowiadają za to adnotacje:

```
@Controller
public class PaintingController {
```

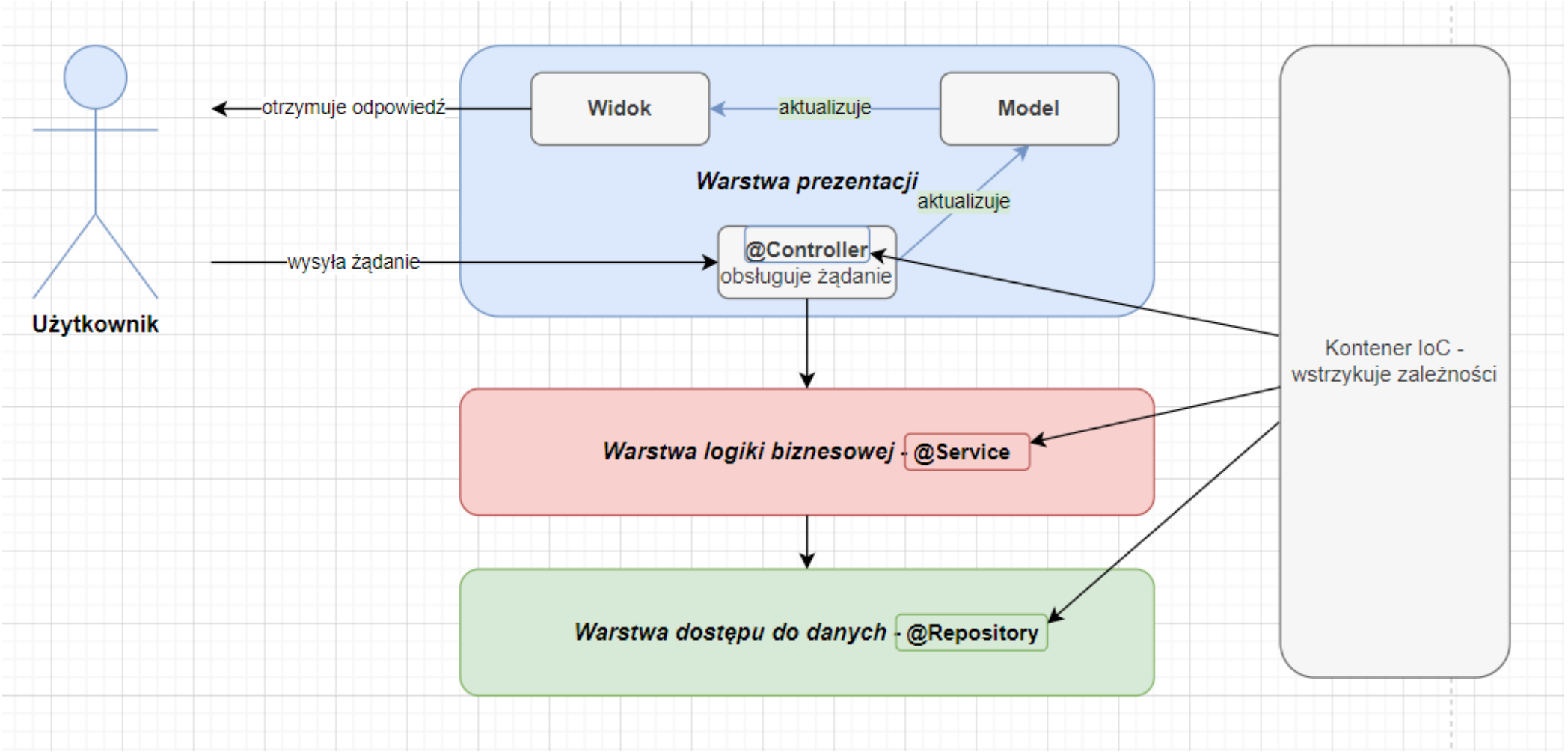
```
@Service
public class PaintingService {
```

```
@Repository("fake")
public class FakePaintingRepository
```

Podsumowanie IoC i DI

- Pisząc aplikację w Springu oddajemy część kontroli nad kodem frameworkowi
- Zależności między warstwami aplikacji są zarządzane przez framework – wykorzystując adnotacje i wstrzykiwanie zależności przez konstruktor lub setter możemy zrzucić odpowiedzialność za utworzenie instancji obiektów na framework

Uff!


Użytkownik
otrzymuje odpowiedź
wysyła żądanie
Widok
aktualizuje
Model
Warstwa prezentacji
aktualizuje
@Controller
obsługuje żądanie
Kontener IoC -
wstrzykuje zależności
Warstwa logiki biznesowej - @Service
Warstwa dostępu do danych - @Repository

Źródła

- https://docs.microsoft.com/en-US/azure/architecture/guide/architecture-styles/n-tier [03.02.2021]
- https://openclassrooms.com/en/courses/5684146-create-web-applications-efficiently-with-the-spring-boot-mvc-framework/6156961-organize-your-application-code-in-three-tier-architecture [03.02.2021]
- https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/ [03.02.2021]
- https://blog.indrek.io/articles/structuring-packages-in-java-web-applications/ [03.03.2021]
- https://bulldogjob.pl/news/1448-najwazniejsze-wzorce-architektoniczne [5.02.2021]
- https://refactoring.guru/pl/design-patterns/catalog [5.02.2021]
- https://www.martinfowler.com/bliki/InversionOfControl.html [5.02.2021]
- https://www.martinfowler.com/articles/injection.html [5.02.2021]
- https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#spring-core [5.02.2021]
- https://www.baeldung.com/spring-bean [5.02.2021]
- https://www.baeldung.com/spring-bean-annotations [5.02.2021]