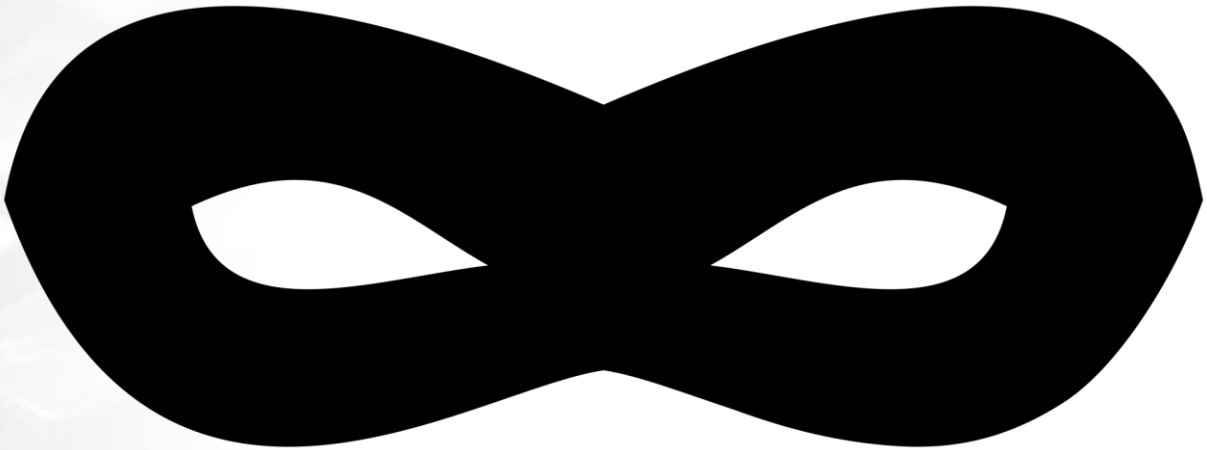
Wprowadzenie do Spring Framework

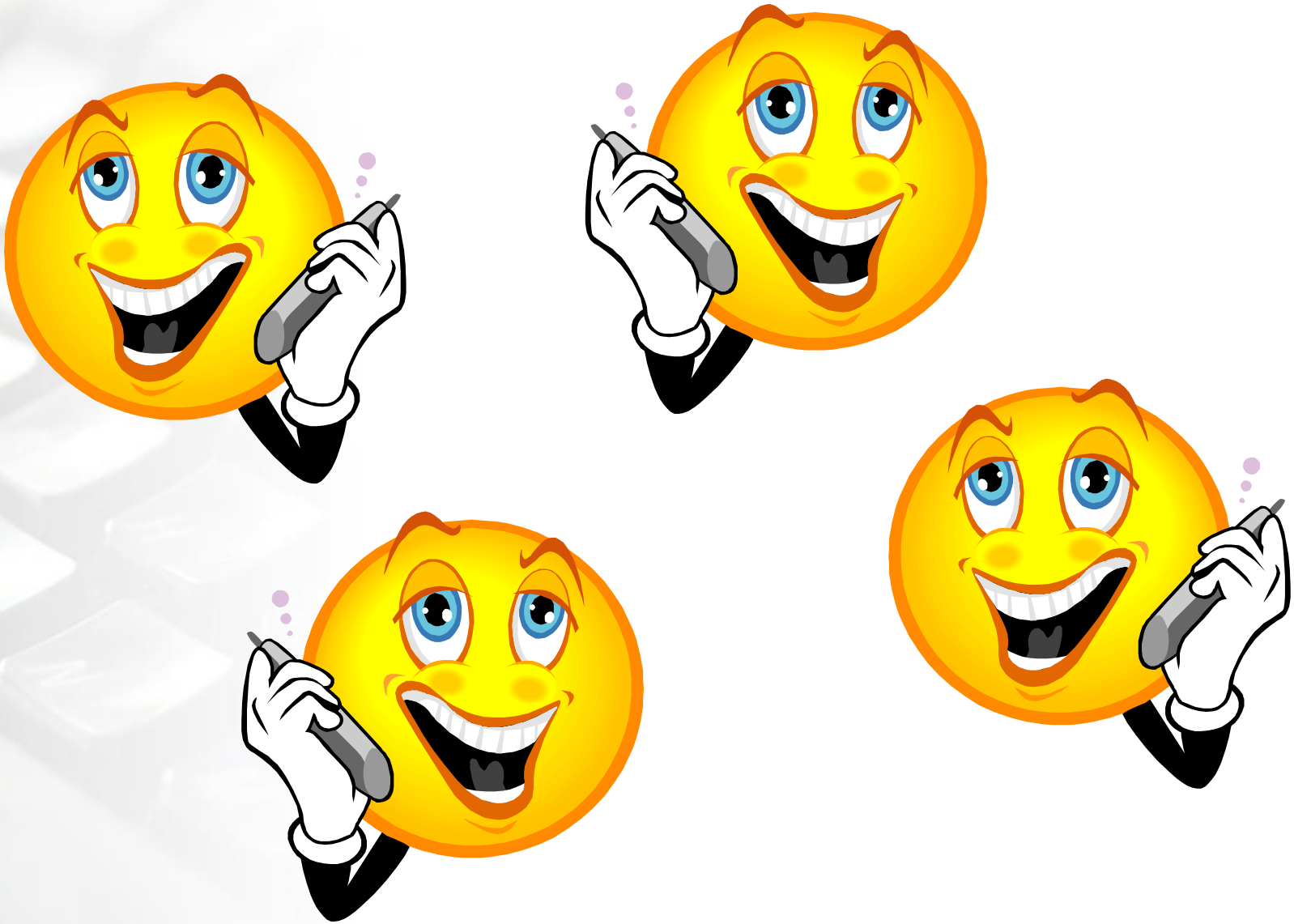
O prowadzącym



Programowanie

- Funkcyjne
- Proceduralne
- Obiektowe

Obiekty



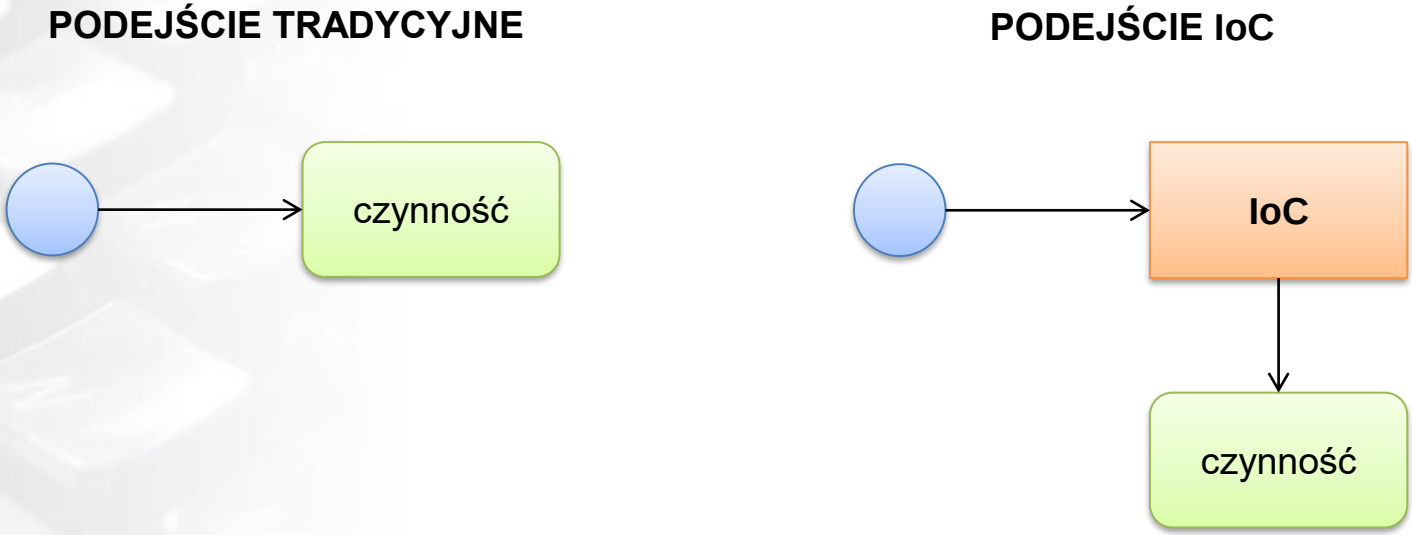
Programowanie obiektowe

Nowoczesna aplikacja

- Data-driven
- Self-contained
- Extensible
- Fast
- Maintainable
- Scalable
- „Reactive”

Inversion of Control

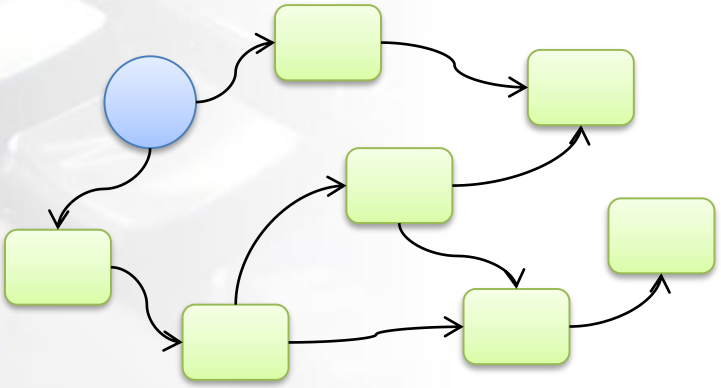
- wzorzec architektoniczny polegający na przeniesieniu na zewnątrz komponentu (np. obiektu) odpowiedzialności za kontrolę wybranych czynności



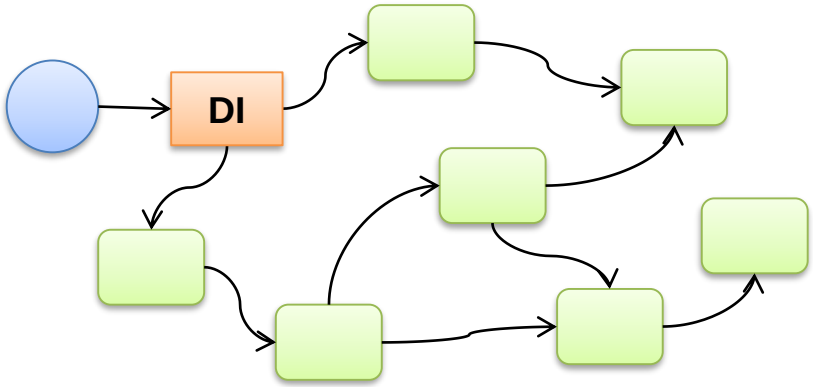
Dependency Injection

- wzorzec architektoniczny polegający na usunięciu bezpośrednich zależności pomiędzy komponentami systemu
- odpowiedzialność za tworzenie obiektów przeniesione zostaje do zewnętrznej fabryki obiektów – kontenera
- kontener na żądanie tworzy obiekt bądź zwraca istniejący z puli ustawiając powiązania z innymi obiektami

PODEJŚCIE TRADYCYJNE

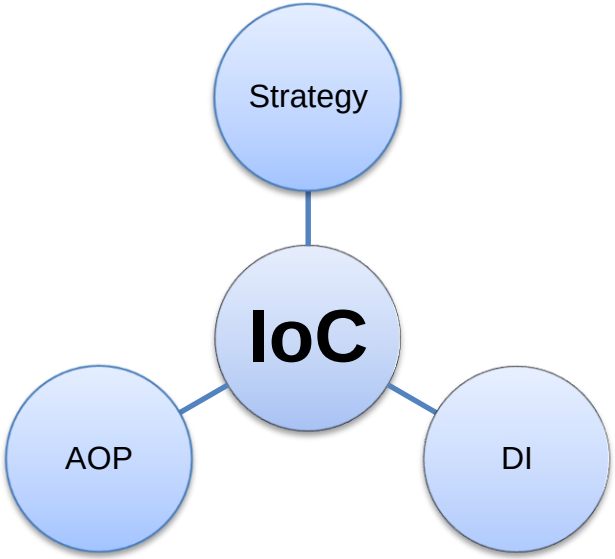


PODEJŚCIE DI

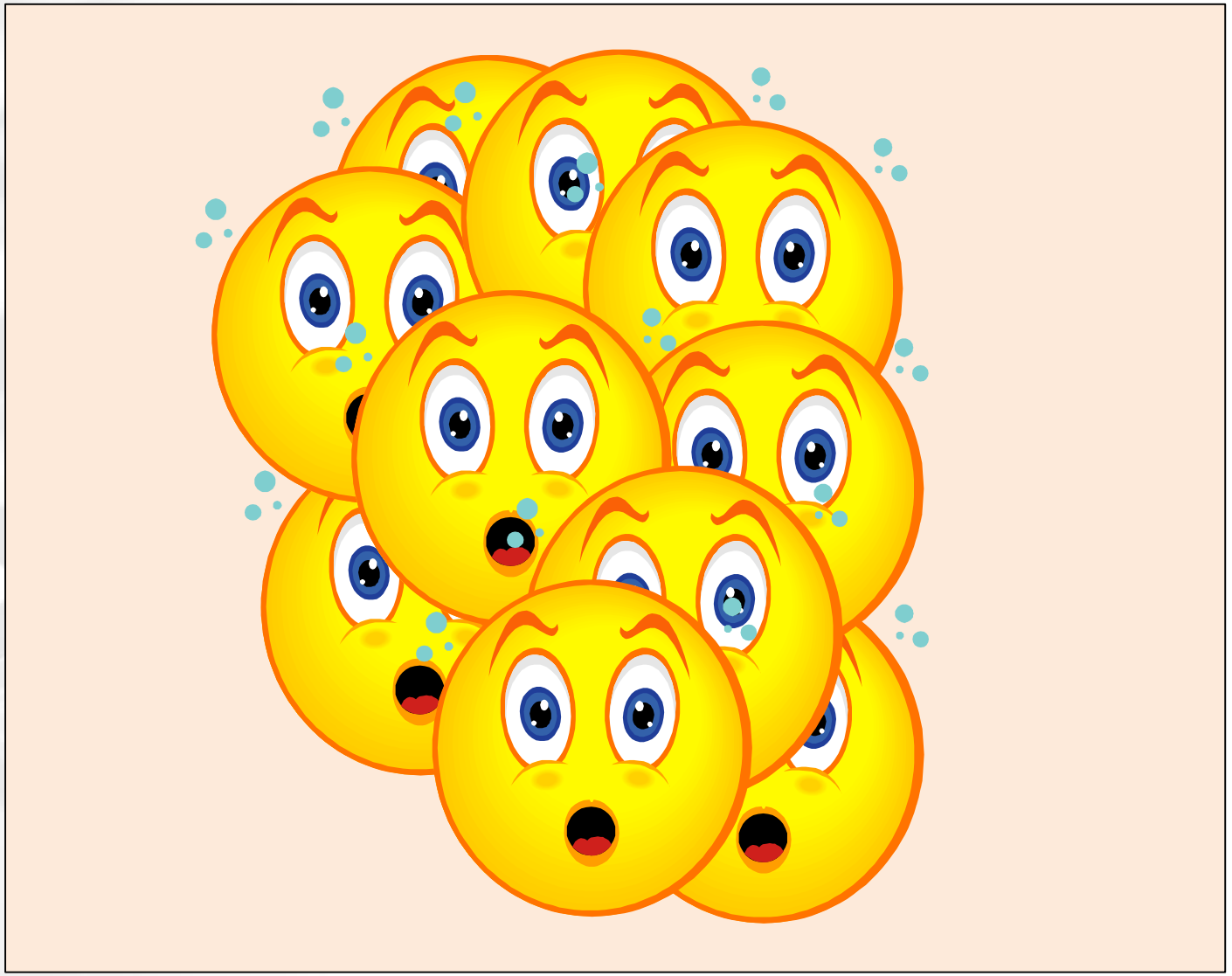


IoC != DI

- Inversion of Control często błędnie utożsamiane jest jednoznacznie z Dependency Injection.
- Dependency Injection to szczególny przypadek IoC, który obejmuje szerszy krąg przypadków.



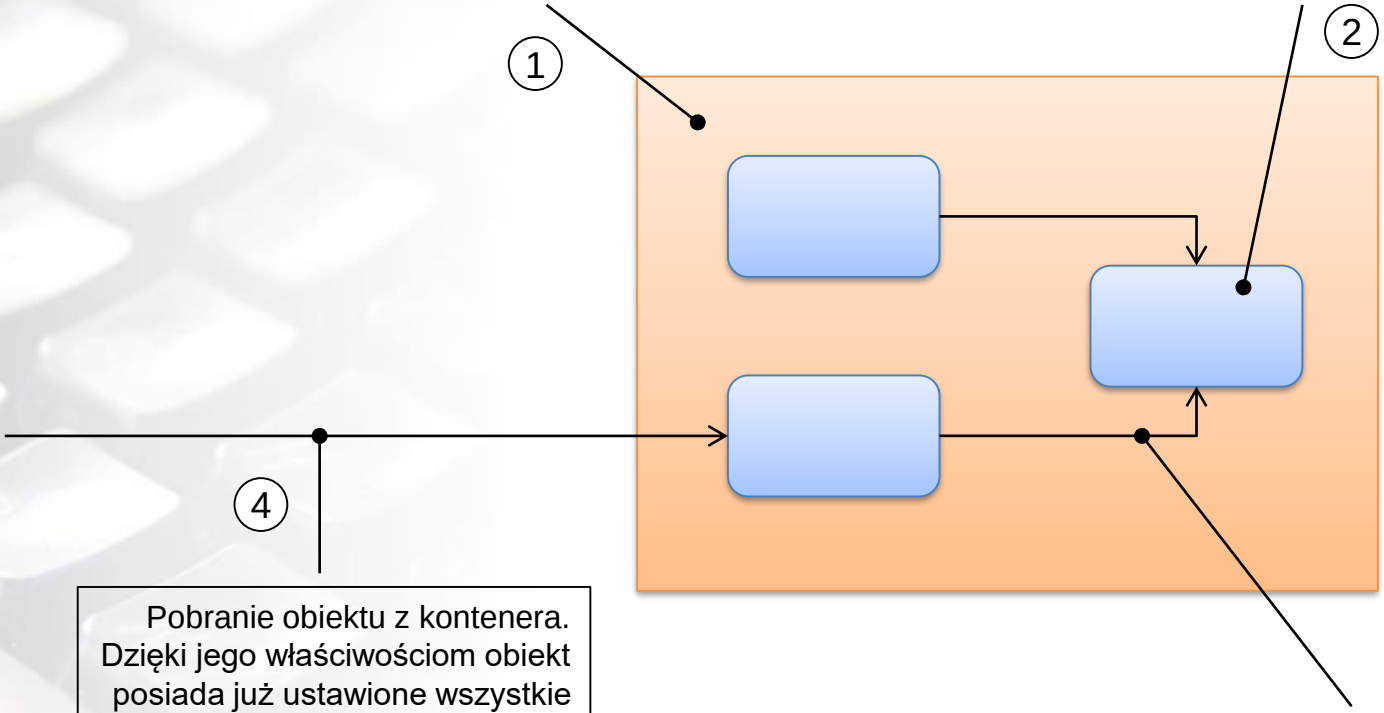
Kontener



Działanie kontenera DI

Pierwszym krokiem jest powołanie do życia samego kontenera zarządzającego obiektami i zależnościami pomiędzy nimi.

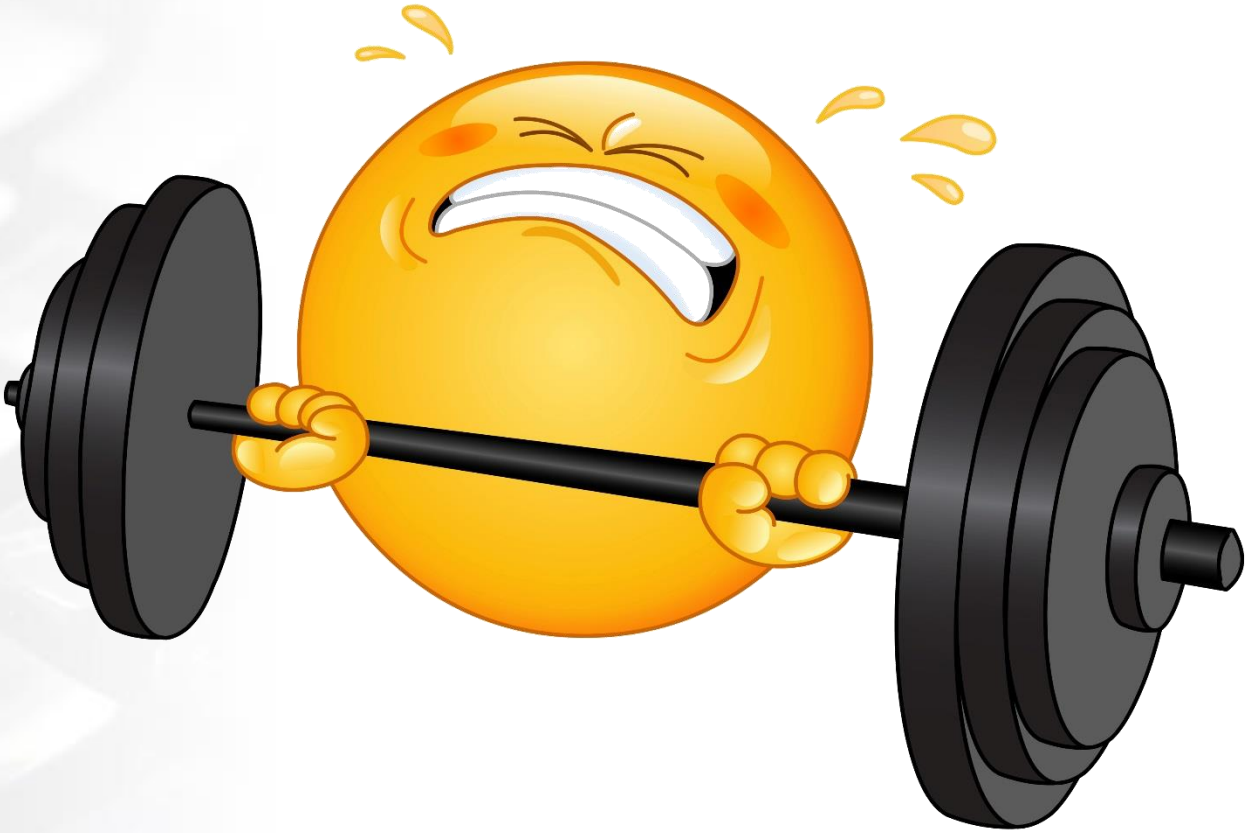
Kontener tworzy instancje obiektów. Klasy oraz sposób powoływania obiektów zdefiniowany jest przez użytkownika.



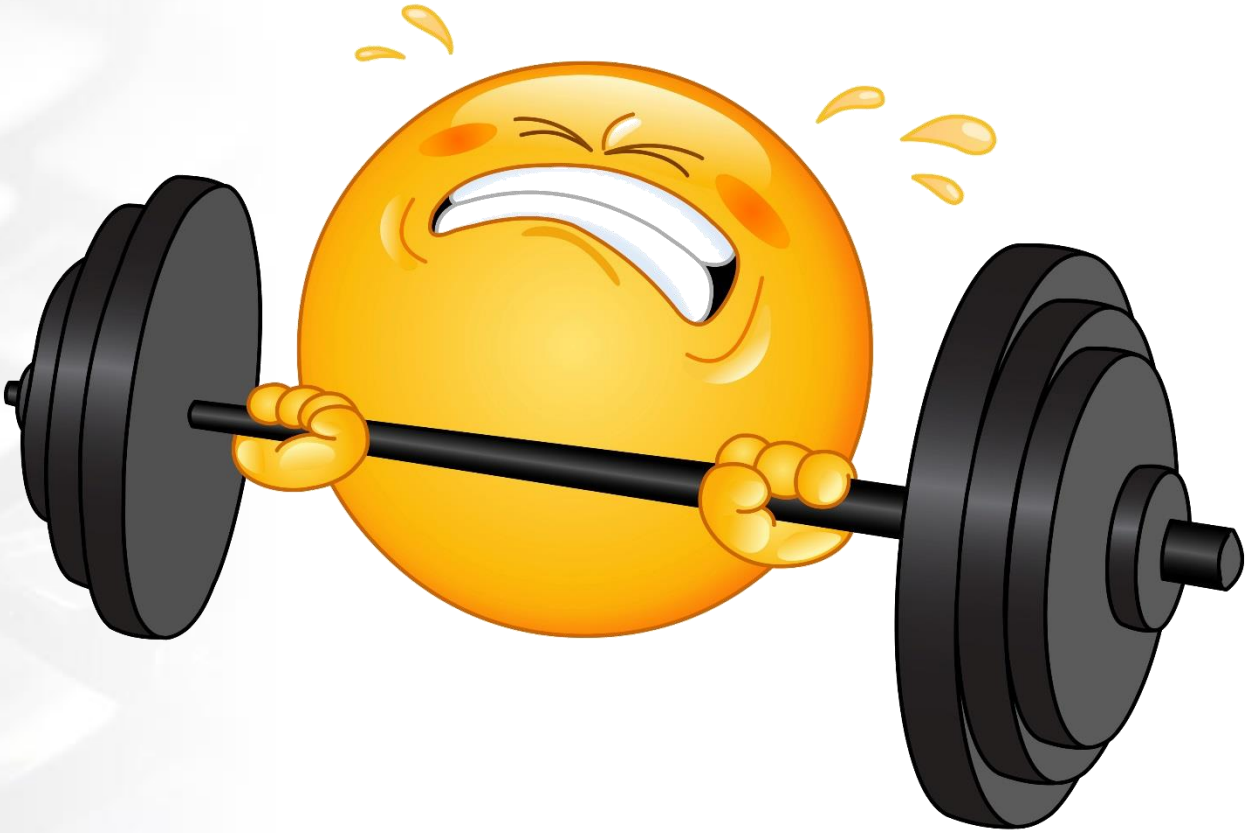
Pobranie obiektu z kontenera. Dzięki jego właściwościom obiekt posiada już ustawione wszystkie zależności do innych obiektów.

Na podstawie warunków zdefiniowanych przez użytkownika kontener ustawia - "wstrzykuje" - zależności pomiędzy komponentami (obiektami).

MicroDIContext – projekt praktyczny



MicroDIContext – projekt praktyczny



MicroDIContext

- Wstrzykiwanie przez publiczny konstruktor
- Singletons only
- @Component annotation
- Silne sprawdzanie typów
- Brak wsparcia dla interfejsów

MicroDIContext – rozszerzenie

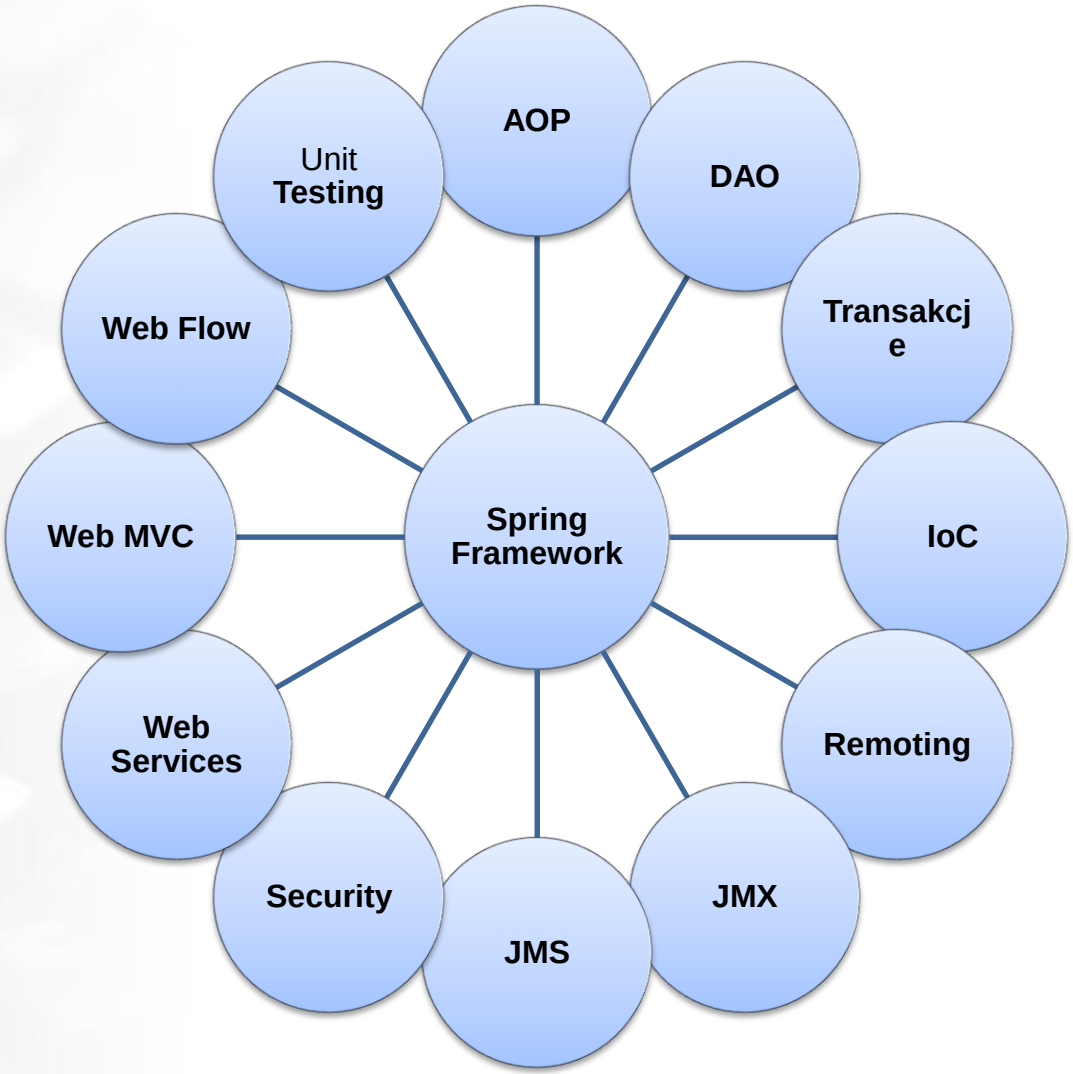
- @PostCreate
- @Prototype
- @Lazy

Spring

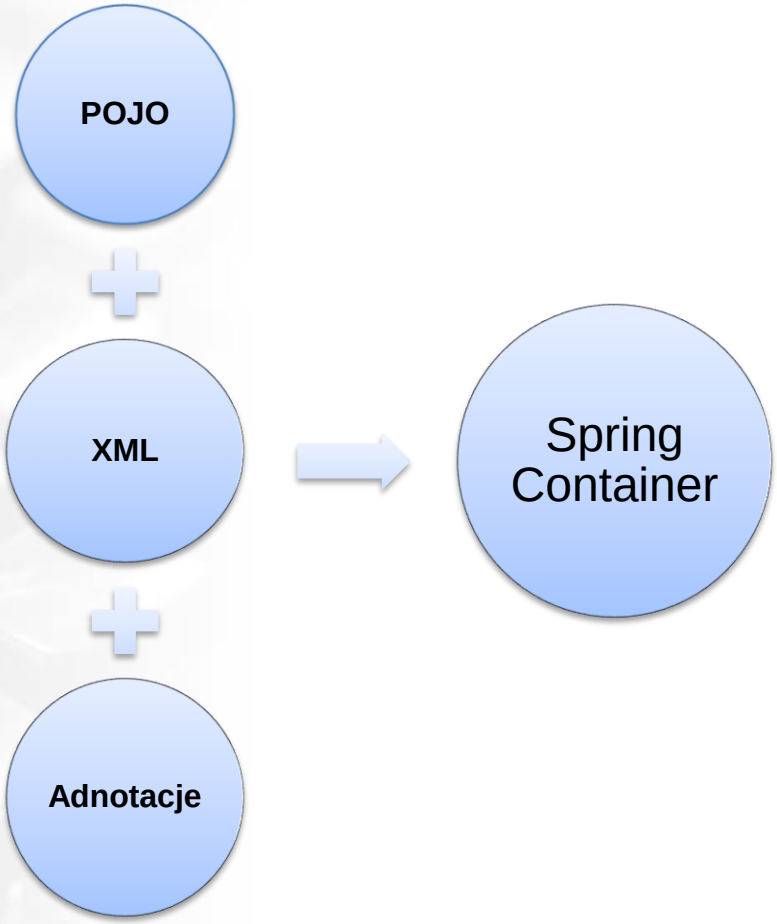
Spring jako alternatywa dla Java EE

- Komponenty EJB (2.1)
  - "ciężkie" i skomplikowane
  - trudne do testowania
- Spring (2003)
  - lekki
  - prosty w konfiguracji
  - łatwy do testowania
  - duża funkcjonalność dodatkowa

Spring Framework



Spring jako kontener



Spring

- Container
  - Beans
  - BeanDefinition class
  - BeanFactory and ApplicationContext classes

Bean scopes

- singleton
- prototype
- request
- session
- globalSession
- application
- websocket
- thread*
- custom

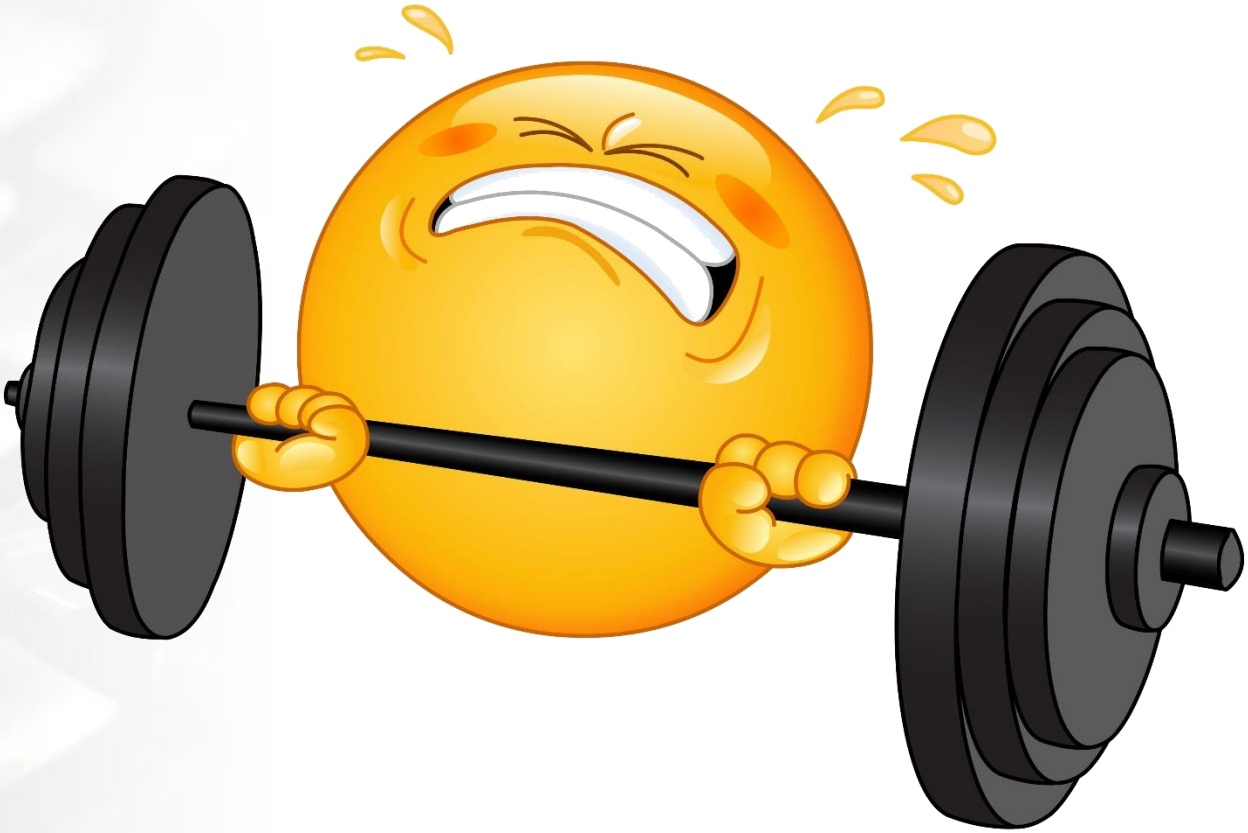
Classpath scanning

- @Configuration and @Bean
- Stereotype annotations:
  - @Component
  - @Repository
  - @Service
  - @Controller

Dodatkowe

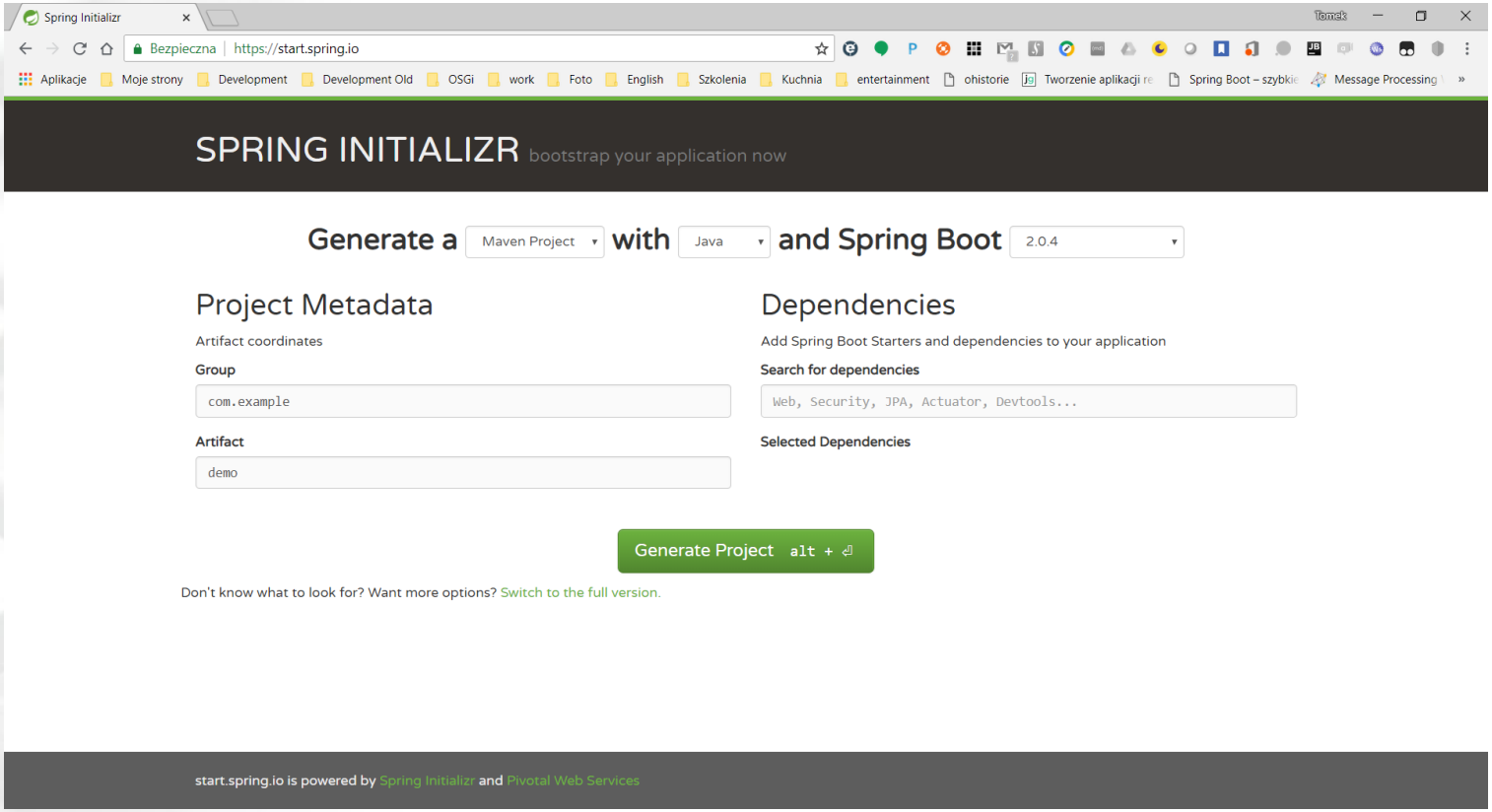
- proxing
- lifecycle callbacks
- @PostConstruct
- @PreDestroy
- AOP

Kontener Spring – projekt praktyczny

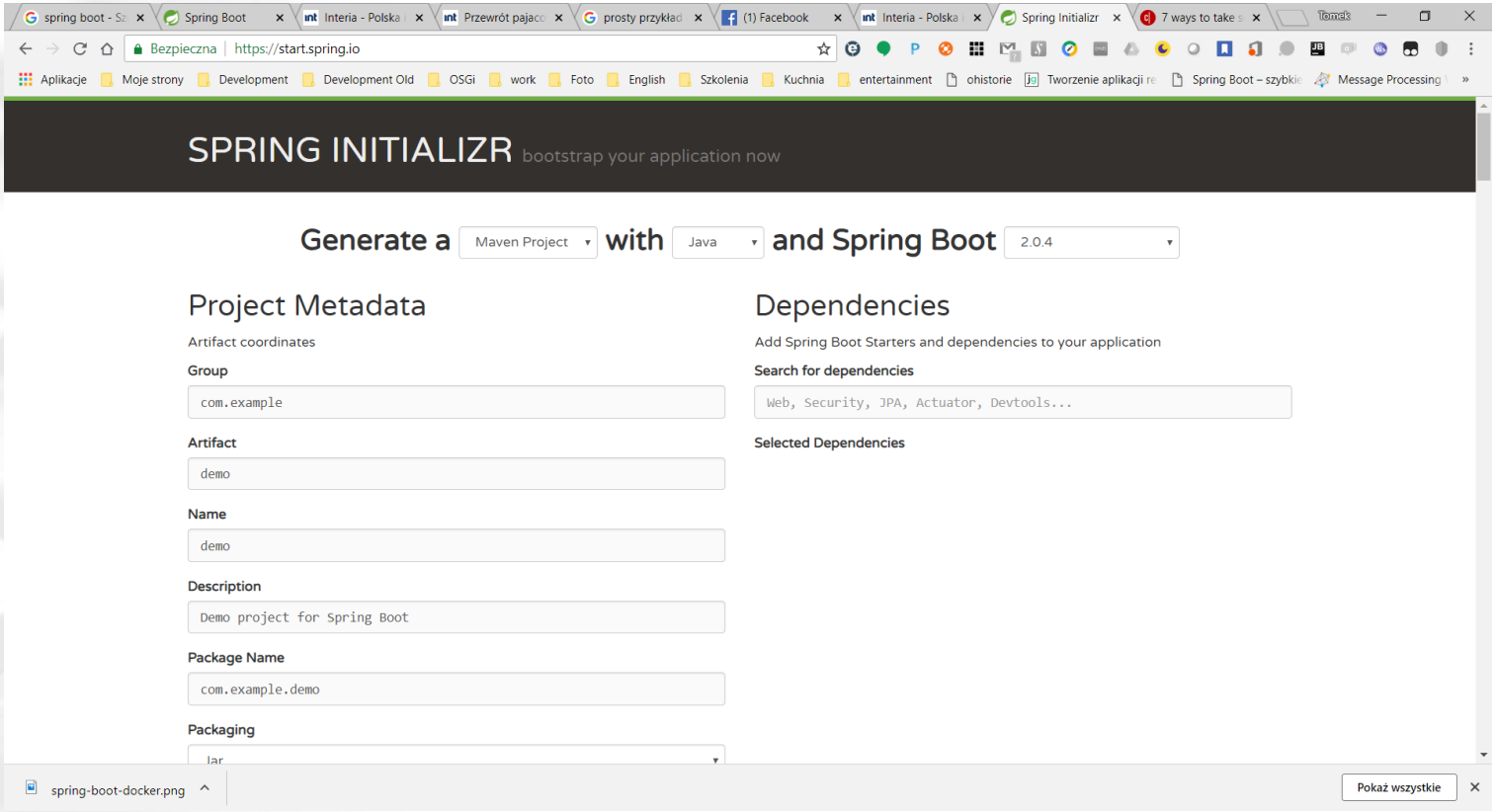



SPRING FRAMEWORK
SPRING BOOT

Spring Initializr



Spring Initializr



Auto configuration

- @ConditionOnBean / @ConditionOnMissingBean
- @ConditionOnClass / @ConditionOnMissingClass
- @ConditionOnExpression
- @ConditionOnJava
- @ConditionOnJndi
- @ConditionOnProperty
- @ConditionOnResource

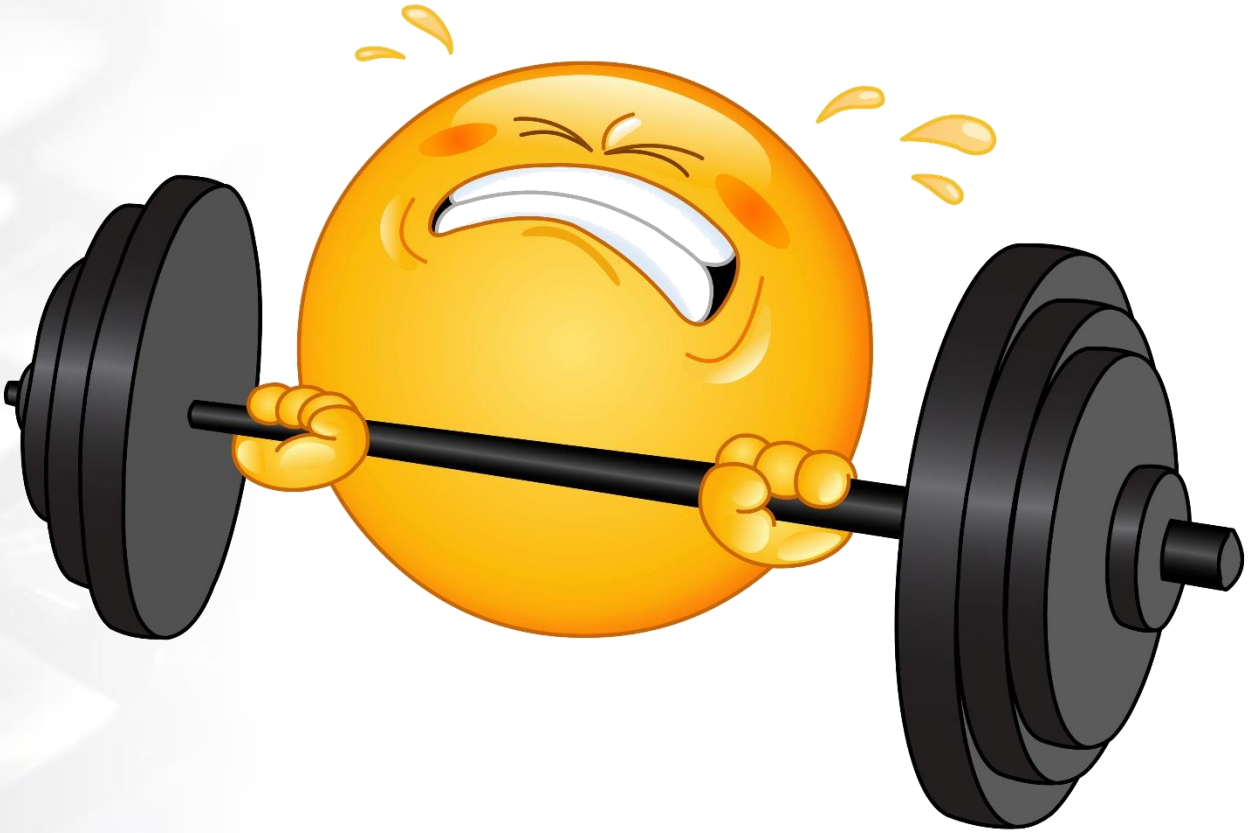
JAR or WAR?

Josh „starbuxman” Long

SPRING ACTUATOR

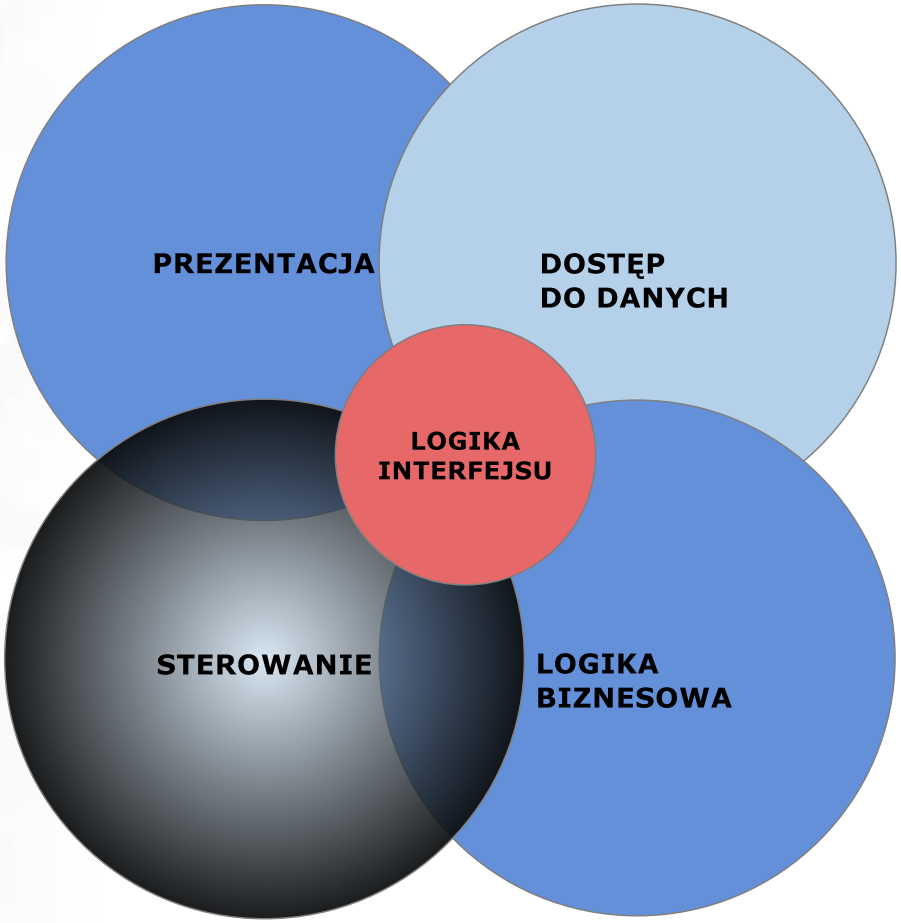
- /actuator
- /auditevents
- /autoconfig
- /beans
- /dumps
- /health
- /info
- /metrics
- /mappings
- /shutdown (!!!)
- /trace

Kontener Spring – projekt praktyczny

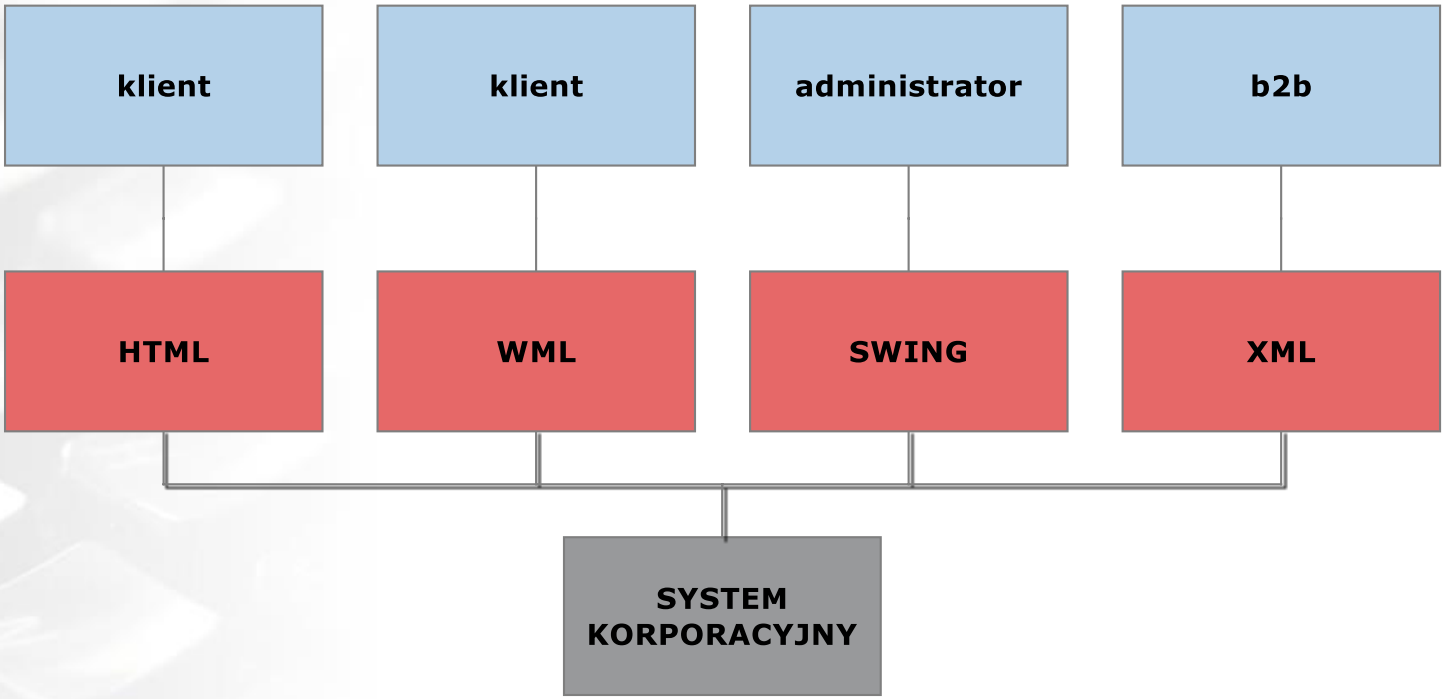


Spring Web

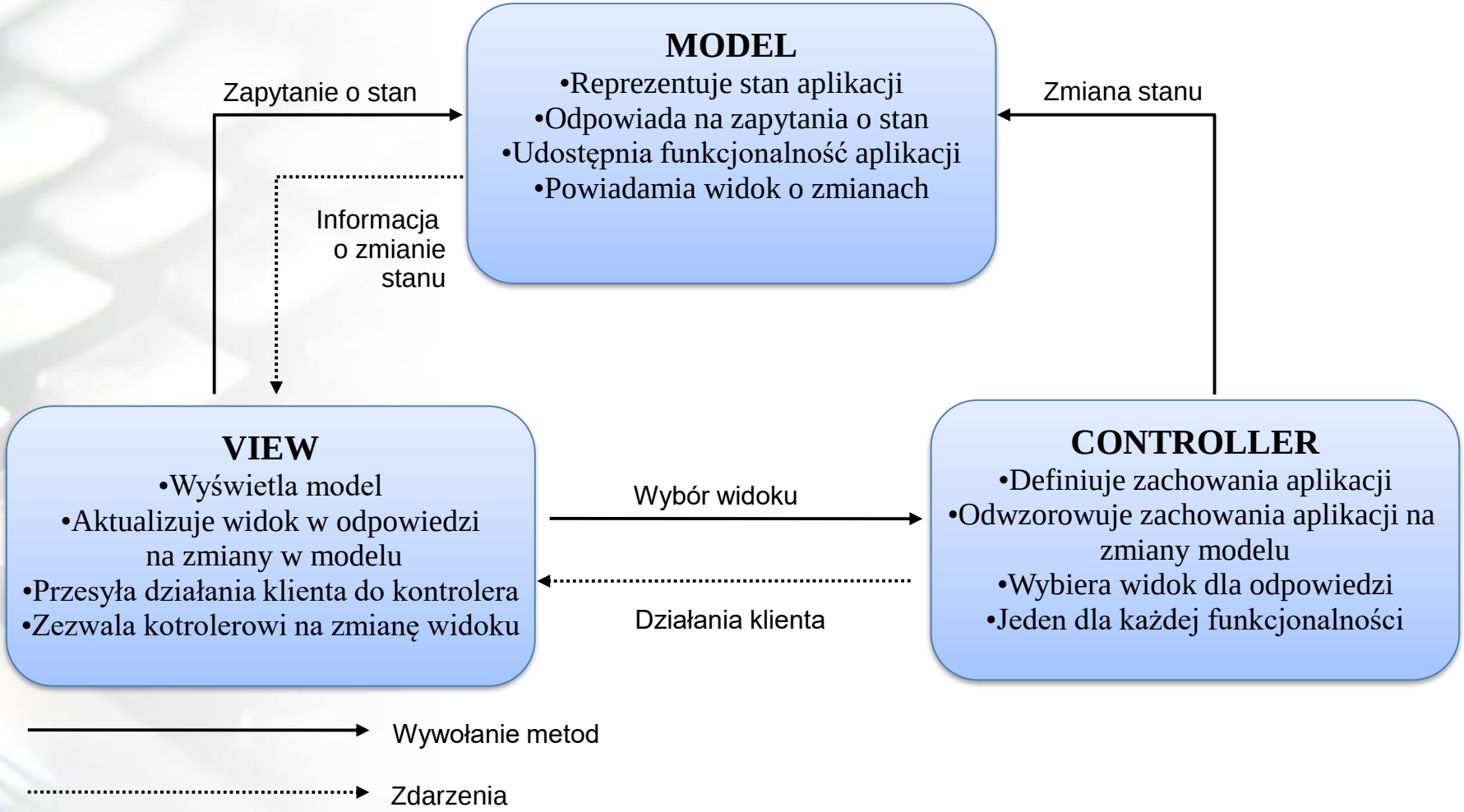
Podejście "tradycyjne"



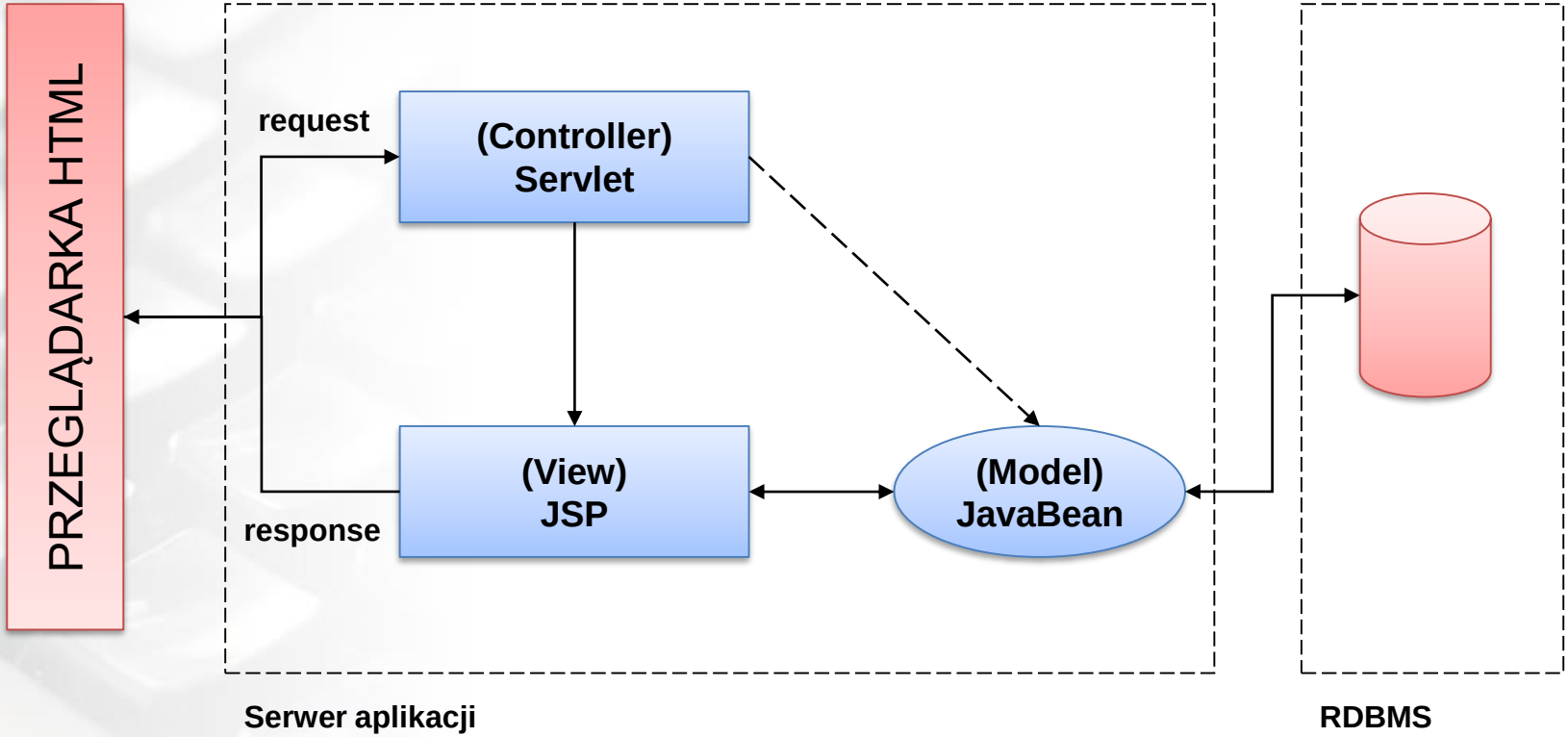
Problem do rozwiązania



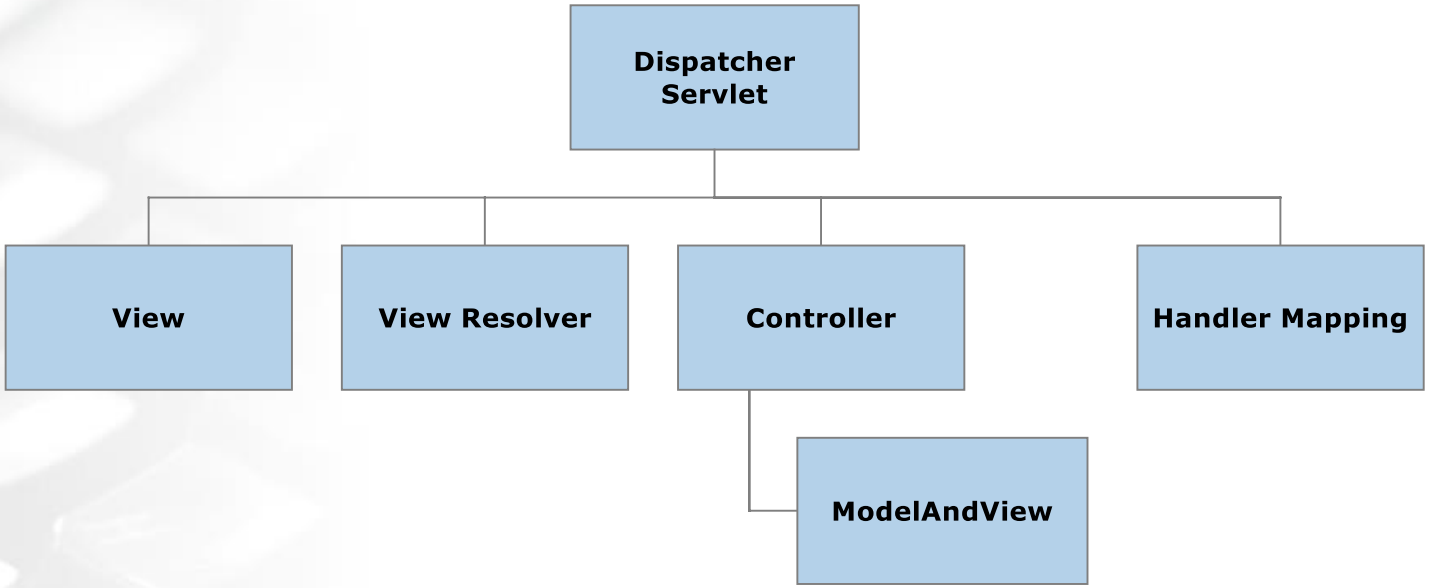
Model View Controller



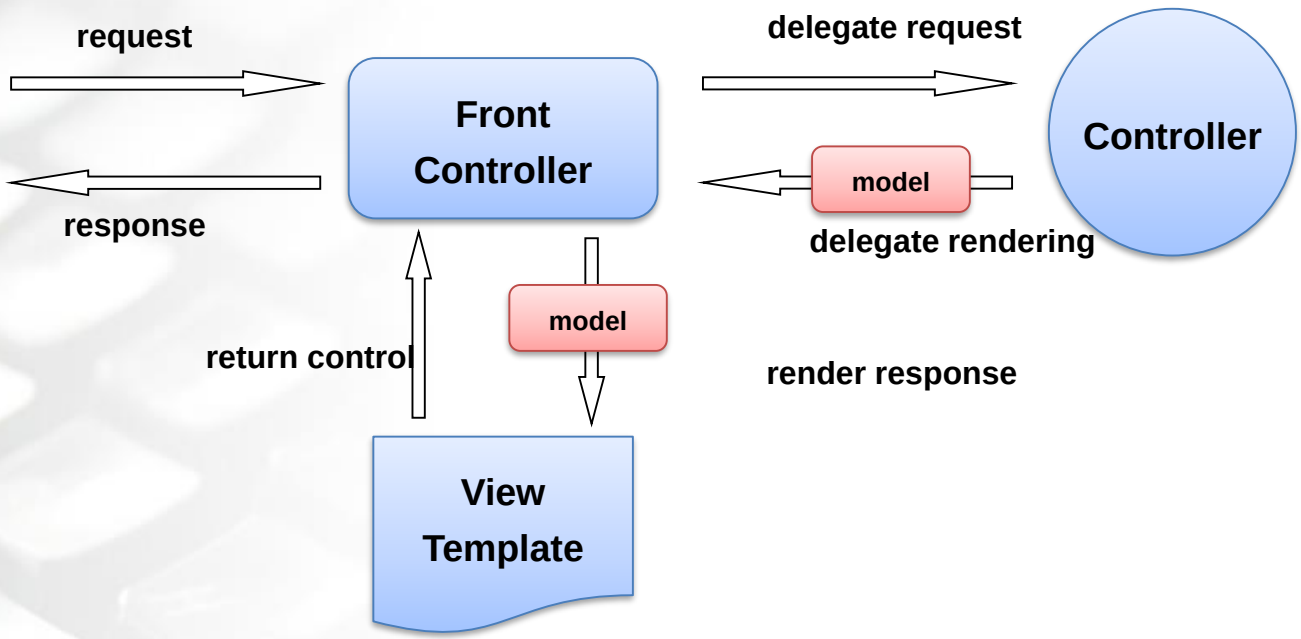
MVC w środowisku Java EE



Podstawowe komponenty Spring MVC



Przepływ danych w Spring MVC



DispatcherServlet odnajduje kontroler przy pomocy obiektu HandlerMapping
DispatcherServlet przekazuje żądanie do kontrolera
Kontroler zwraca ModelAndView
DispatcherServlet odnajduje warstwę prezentacji (View) przy pomocy obiektu ViewResolver
Obiekt View prezentuje odpowiedź serwera

Templates

- Velocity
- FreeMarker
- Groovy Markup Templates
- Thymeleaf
- JSP

- @Controller
- @RequestMapping
- @GetMapping
- @PostMapping
- …
- @ResponseBody
- @RestController
- ResponseEntity

Spring Data

Rodzaje baz

- Relational
- Graph
- Document
- Key-Value
- Column

Wybrane wspierane technologie

- JPA
- LDAP
- MongoDB
- Redis
- Cassandra
- Solr
- Couchbase
- DynamoDB
- Elasticsearch
- Hazelcast
- Neo4j
- Gemfire

Koncepcja

- Mapping
  - JPA
- Repositories
- CrudRepository
  - JpaRepository
  - MongoRepository

Mapping

```
@Entity
public class Data {
        @Id
        @GeneratedValue(strategy =
                GenerationType.IDENTITY)
        private Long id;
        private String content;
        // getters, setters
}
```

Repository

```
public interface DataRepository extends
        JpaRepository<Data, Long> {
        …
        }
```

Repository

- List<Data> findBy…(String prop)
- List<Data> findBy…And…(String prop1, String prop2)
- List<Data> findTop3By…(String prop)
- List<Data> findDistinctBy…(String prop)
- List<Data> findBy…IgnoreCase(String prop)
- List<Data> findBy…IsNull(String prop)

Repository

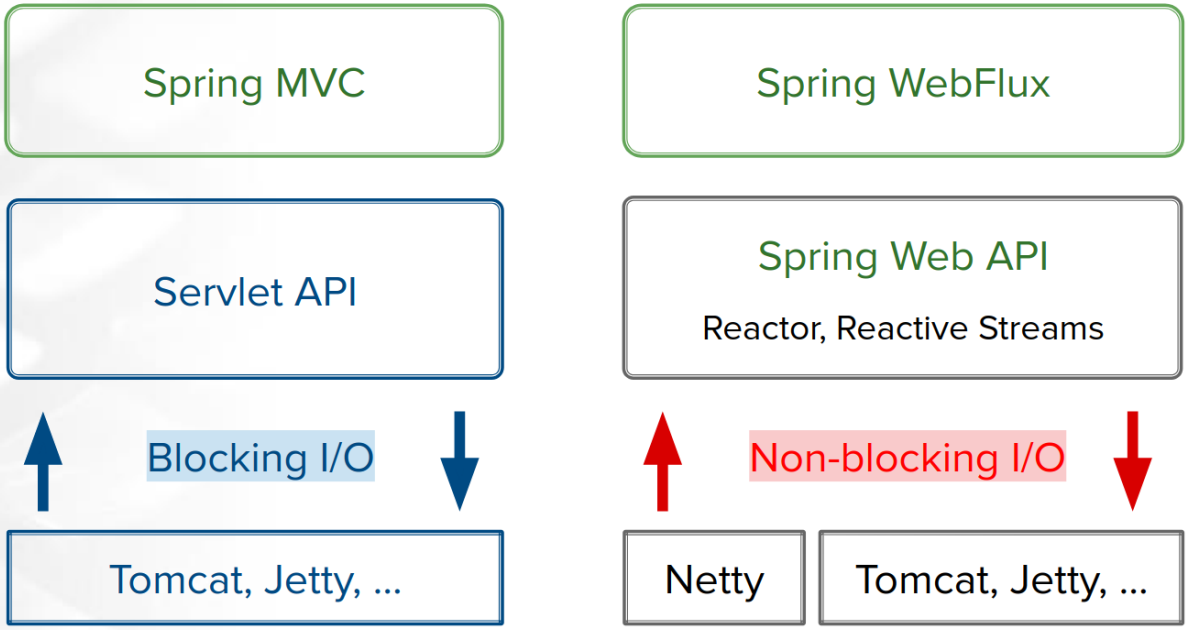
- Stream<Data> findBy…(String prop)
- Future<Data> findBy…(String prop)
- Page<Data> findAll(Pageable p)
- Page<Data> findBy…(String prop, Pageable p)
- List<Data> findAll(Sort s)
- Page<Data> findBy…(String prop, Pageable p, Sort s)

Świetlana przyszłość

Program reaktywny

- Responsive
  - System responds in a timely manner
- Resilent
  - System stays responsive in the face of failure (replication, isolation, delegation)
- Scalable
  - System stays responsive under varying workload (automatic resource allocation, sharding)
- Message Driven
  - Asynchronous message-passing, location transparency, back-pressure, nonblocking

Spring REACTOR



KOTLIN

W prezentacji wykorzystano grafiki z oficjalnych materiałów Spring