PAW - Programowanie Aplikacji Webowych 07
SpringBooks: SpringBoot + Uzupełnienie

Inż. Juliusz Łosiński

```
Hello, World!
```

Model

JPA

Database

CRUD/
Native Query

Client

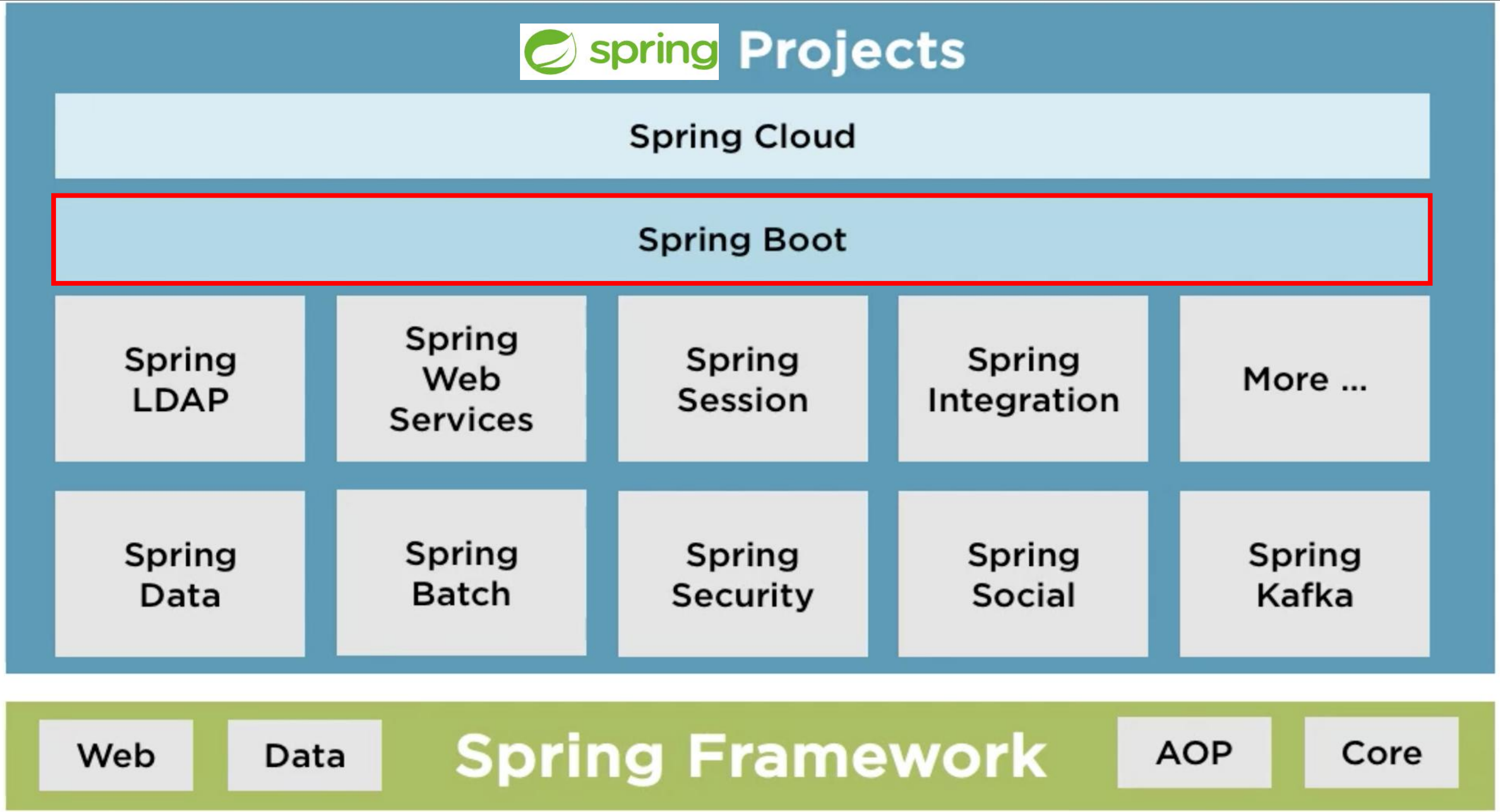
Request

Response

Controller
Layer

Service
Layer

Repository
Layer


spring Projects
Spring Cloud
Spring Boot
Spring LDAP
Spring Web Services
Spring Session
Spring Integration
More ...
Spring Data
Spring Batch
Spring Security
Spring Social
Spring Kafka
Web
Data
Spring Framework
AOP
Core

Main Class	@SpringBootApplication	Spring Boot auto configuration
REST Endpoint	@RestController	Class with REST endpoints
	@RequestMapping	REST endpoint method
	@PathVariable	URI path parameter
	@RequestBody	HTTP request body
Periodic Tasks	@Scheduled	Method to run periodically
	@EnableScheduling	Enable Spring's task scheduling
Beans	@Configuration	A class containing Spring beans
	@Bean	Objects to be used by Spring IoC for dependency injection
Spring Managed Components	@Component	A candidate for dependency injection
	@Service	Like @Component
	@Repository	Like @Component, for data base access
Persistence	@Entity	A class which can be stored in the data base via ORM
	@Id	Primary key
	@GeneratedValue	Generation strategy of primary key
	@EnableJpaRepositories	Triggers the search for classes with @Repository annotation
	@EnableTransactionManagement	Enable Spring's DB transaction management through @Beans objects
Miscellaneous	@Autowired	Force dependency injection
	@ConfigurationProperties	Import settings from properties file
Testing	@SpringBootTest	Spring integration test
	@AutoConfigureMockMvc	Configure MockMvc object to test HTTP queries

Spring Boot and Web annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses @Configuration, @EnableAutoConfiguration and @ComponentScan.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

T **@Controller** - marks the class as web controller, capable of handling the requests. T **@RestController** - a convenience annotation of a @Controller and @ResponseBody.

M T **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.

Spring Cloud annotations

Make you application work well in the cloud.

T **@EnableConfigServer** - turns your application into a server other apps can get their configuration from.

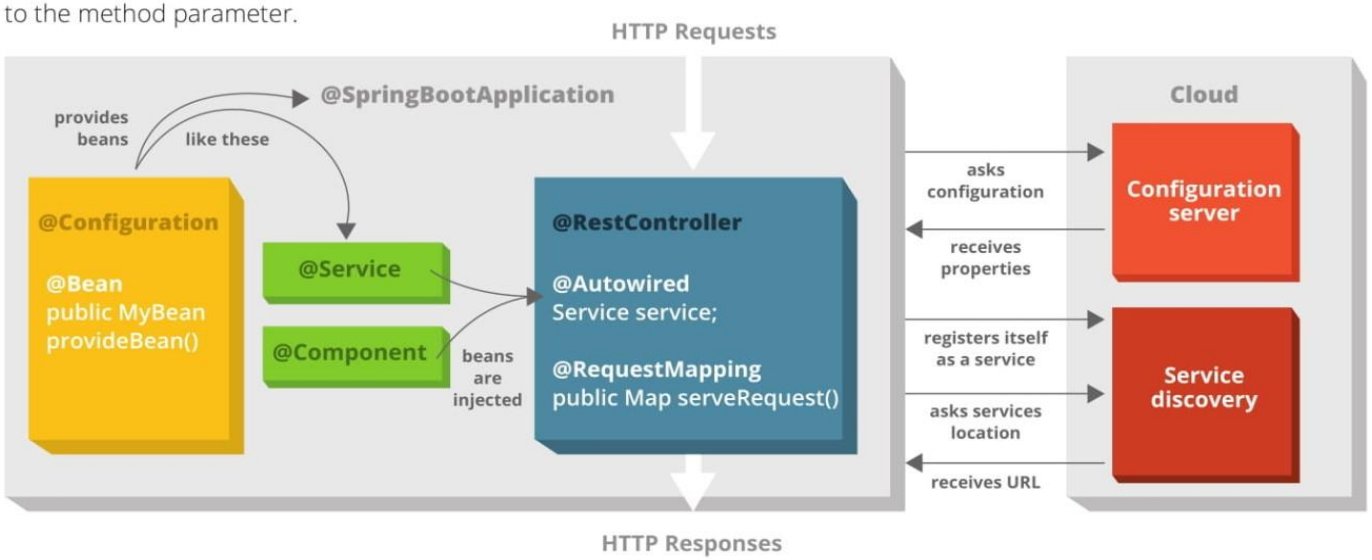
Use `spring.application.cloud.config.uri` in the client @SpringBootApplication to point to the config server.

T **@EnableEurekaServer** - makes your app an Eureka discovery service, other apps can locate services through it.

T **@EnableDiscoveryClient** - makes your app register in the service discovery server and discover other services through it.

T **@EnableCircuitBreaker** - configures Hystrix circuit breaker protocols.

M **@HystrixCommand(fallbackMethod = "fallbackMethodName")** - marks methods to fall back to another method if they cannot succeed normally.



Spring Framework annotations

Spring uses dependency injection to configure and bind your application together.

T **@ComponentScan** - make Spring scan the package for the @Configuration classes.

T **@Configuration** - mark a class as a source of bean definitions.

M **@Bean** - indicates that a method produces a bean to be managed by the Spring container.

T **@Component** - turns the class into a Spring bean at the auto-scan time. T **@Service** - specialization of the @Component, has no encapsulated state.

C F M **@Autowired** - Spring's dependency injection wires an appropriate bean into the marked class member.

T M **@Lazy** - makes @Bean or @Component be initialized on demand rather than eagerly.

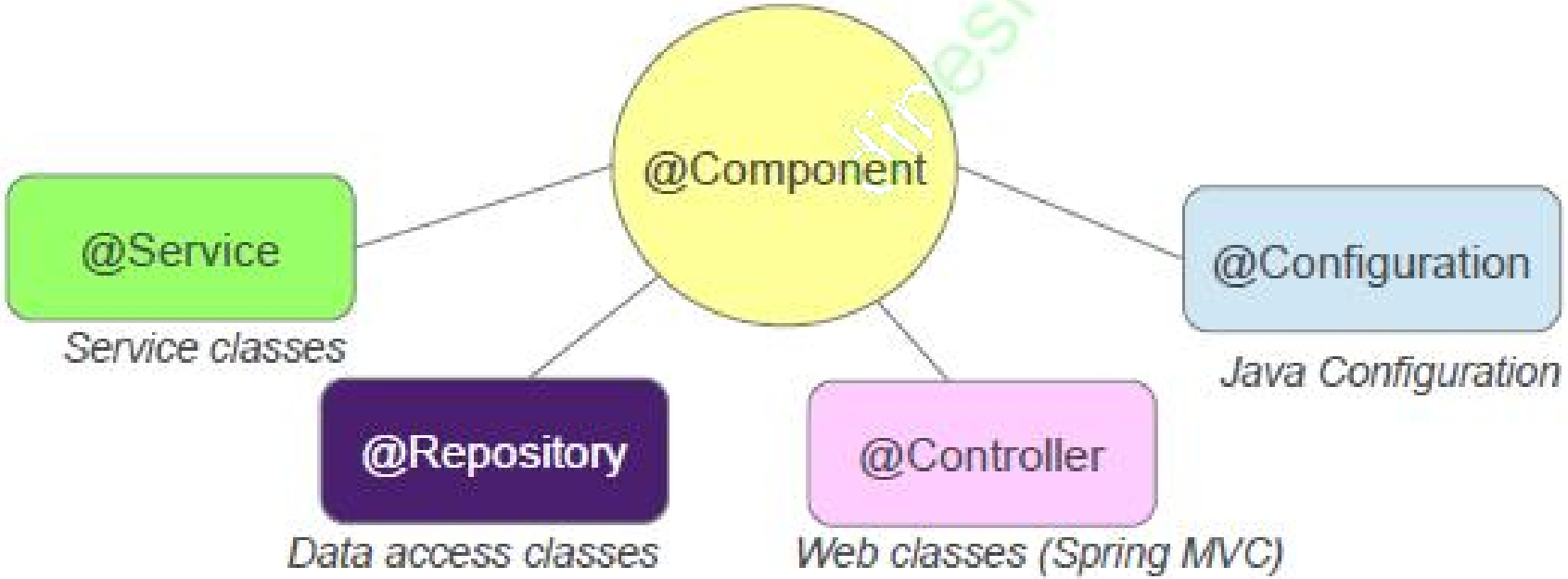
C F M **@Qualifier** - filters what beans should be used to @Autowire a field or parameter.

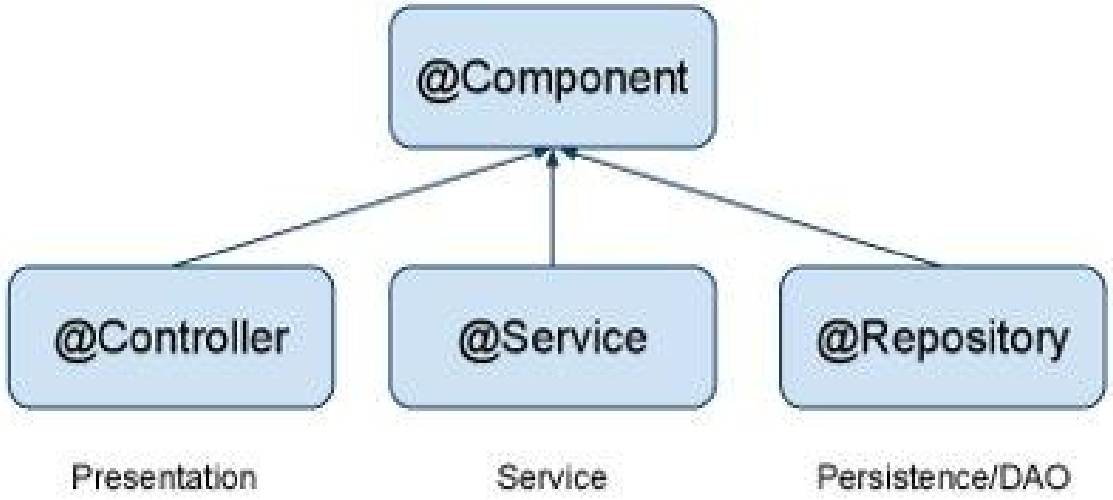
C F M **@Value** - indicates a default value expression for the field or parameter, typically something like `"#{systemProperties.myProp}"`

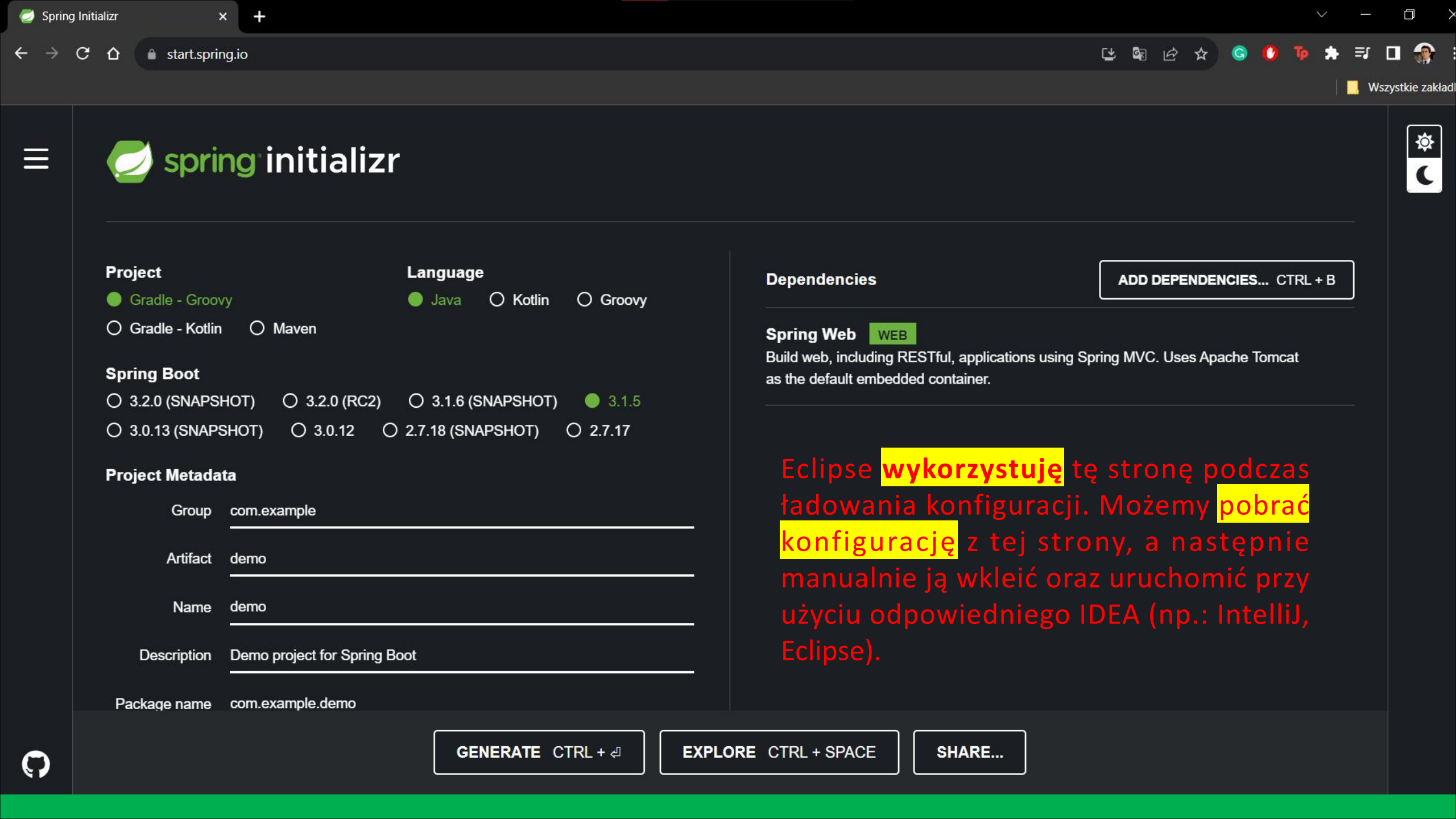
C F M **@Required** - fail the configuration, if the dependency cannot be injected.

Legend

- T - class
- F - field annotation
- C - constructor annotation
- M - method
- P - parameter


@Component
@Service
Service classes
@Configuration
Java Configuration
@Repository
Data access classes
@Controller
Web classes (Spring MVC)


@Component
@Controller
@Service
@Repository
Presentation
Service
Persistence/DAO

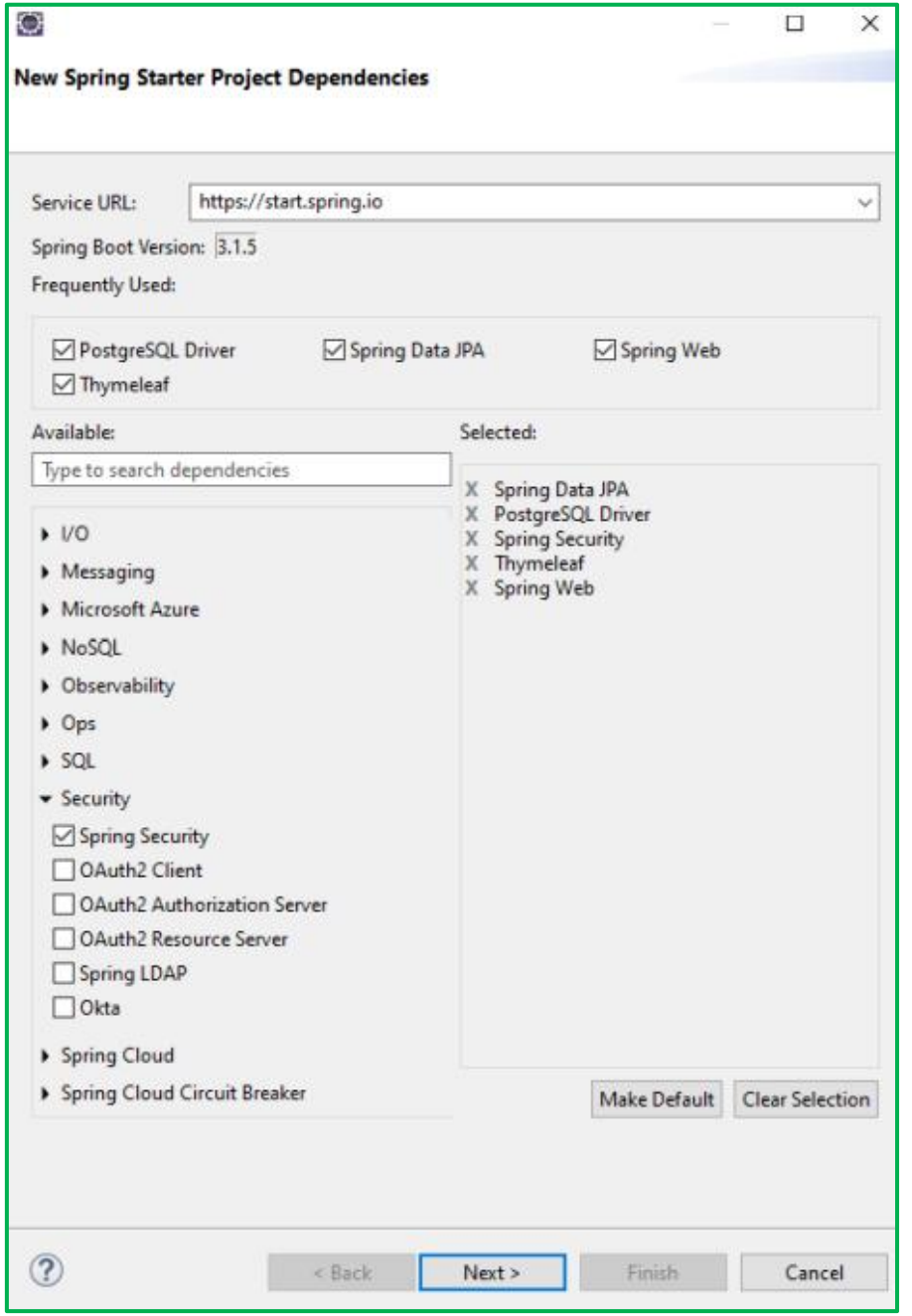

Spring Initializr
start.spring.io
Wszystkie zakładki
spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
3.2.0 (SNAPSHOT)
3.2.0 (RC2)
3.1.6 (SNAPSHOT)
3.1.5
3.0.13 (SNAPSHOT)
3.0.12
2.7.18 (SNAPSHOT)
2.7.17
Project Metadata
Group com.example
Artifact demo
Name demo
Description Demo project for Spring Boot
Package name com.example.demo
Dependencies
ADD DEPENDENCIES... CTRL + B
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Eclipse wykorzystuję tę stronę podczas ładowania konfiguracji. Możemy pobrać konfigurację z tej strony, a następnie manualnie ją wkleić oraz uruchomić przy użyciu odpowiedniego IDEA (np.: IntelliJ, Eclipse).
GENERATE CTRL + ⏎
EXPLORE CTRL + SPACE
SHARE...

Główne pakiety:

- ***spring-boot-starter-web*** - dostarcza automatyczną deklaracje, mapowanie oraz konfigurację servlet'u dyspozytora,
- ***spring-boot-starter-data-jpa*** - obsługuję relacyjne bazy danych przy użyciu interfejsów Spring Data JPA oraz systemu **ORM** Hibernate,
- ***spring-boot-starter-thymeleaf*** - to implementacja silnika szablonów Thymeleaf, który umożliwia tworzenie elementów interfejsu użytkownika, zawiera takzwane dialekty: **Standard** oraz **SpringStandard**.

Zaktualizowana zawartość pliku **build.gradle**:

```
dependencies {
        implementation 'org.springframework.boot:spring-boot-starter-web'
        implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
        implementation 'org.springframework.boot:spring-boot-starter-security'
        //  Temporary explicit version to fix Thymeleaf bug
        implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity6:3.1.1.
        implementation 'org.springframework.security:spring-security-test'
        testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```



Domyślny panel logowania

Pakiet startowy bezpieczeństwa **springboot-starter-security** uzupełnił automatycznie kod aplikacji i zablokowałdostęp do wszystkich stron wyświetlając podstawową stronę do logowania. Może to być podstawowe zabezpieczenie dla dowolnej aplikacji.

''**Using generated security password:** ***dbdb0988-a47b-43bd-b066-3225b0832954***''

Podstawowa konfiguracja bezpieczeństwa:

1. W katalogu springbooks **dodaj nowy folder** do implementacji bezpieczeństwa o nazwie *security*.
2. W pakiecie security **wygeneruj klasę** do konfiguracji bezpieczeństwa o nazwie *SecurityConfig*.
3. Żeby klasa pełniła rolę konfiguracyjnej trzeba **ustawić** jej **adnotację @Configuration**.
4. Żeby zapewnić bezpieczeństwo dla aplikacji Web, a jednocześnie integrację z MVC trzeba **dodać** do tej klasy jeszcze jedną **adnotację @EnableWebSecurity**.
5. Wewnątrz klasy potrzebna będzie metoda z adnotacją **@Bean** do wstrzyknięcia ziarenka Javy **SecurityFilterChain**. Definiuje ona które URLe aplikacji zastaną zabezpieczone, a które nie. Zwykle ścieżka do katalogu głównego i /home lub /index pozostaje niezabezpieczona

Dodaj do klasy SecurityConfig kod, który nie pozwoli **edytować** ani **dodawać** nowych książek niezalogowanym użytkownikom:

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
      http
      .csrf(csrf -> csrf.disable())
      .authorizeHttpRequests(authorize -> authorize
      .requestMatchers("/new_book,"/edit_book/**")
      .authenticated()
      .anyRequest().permitAll()
      )
      .formLogin(Customizer.withDefaults());
      return http.build();
}
```

Gdy użytkownik poprawnie się zaloguje, zostanie automatycznie przekierowany na URL, którego żądał wcześniej. Metoda formLogin wskazuje withDefaults czyli zachowanie zdefiniowane domyślnie w SpringSecurity.

Dodawanie użytkowników

Na początek wygenerujemy jednego użytkownika do testowania i zapiszemy go w pamięci. Służy do tego klasa **UserDetails** oraz **UserDetailsService** implementowana w klasie **InMemoryUserDetailsManager.** Kod dla użytkownika o loginie *user1* i haśle *passu1* (bez kodowania {noop}) może wyglądać następująco:

```
@Bean
UserDetailsService userDetailsService(){
    UserDetails user = User.builder()
      .username("user1")
      .password("{noop}passu1")
      .roles("USER")
      .build();
    return new InMemoryUserDetailsManager(user);
}
```

Ustawienie kodowania:

```
@Bean
public PasswordEncoder passwordEncoder() {
  return new BCryptPasswordEncoder();
}
```

Spring Security udostępnia również inne rodzaje kodowania, w tym zgodne wstecz, w tym również takie, które nie są już uważane za bezpieczne. Dostępne kodowania to **BCrypt, Argon2, Pbkdf2 i CScrypt** oraz niezalecane SCrypt i Standard.

Własny formularz logowania

1. Dodanie do aplikacji **szablonu** o nazwie **login.html** i umieszczenie w nim formularza. Domyślny mechanizm Spring Security **wymaga** żeby w formularzu znalazły się **standardowe pola** o nazwach username i password.

```
<form action="#" th:action="@{/login}"  method="post">
  <table>
    <tr>
      <td>Login</td>
      <td><input type="text" id="username" name="username"/></td>
    </tr>
    <tr>
      <td>Password</td>
      <td><input type="password" id="password" name="password"/></td>
    </tr>
  </table>
</form>
```

Własny formularz logowania

2. Zdefiniowanie klasy **LoginController** z metodą do wyświetlania formularza o nazwie **showFormLogin** z mapowaniem **@GetMapping(”/login”)** oddającą w wyniku ”login”.

3. Przekierowanie obsługi logowania z domyślnego formularza do naszego. W tym celu w pliku **SecurityConfig**, w metodzie **securityFilterChain** trzeba zmienić parametr metody formLogin na następującą postać:

```
.formLogin(form -> form
    .loginPage("/login")
    .loginProcessingUrl("/login")
    .defaultSuccessUrl("/")
    .permitAll());
```

Nawigacja – opcje w menu

W menu aplikacji **Spring Books** powinny się znaleźć **pola** Login i Logout wyświetlane na zmianę w zależności od tego czy użytkownik został **uwierzytelniony czy nie**. Służy do tego parametr **sec:authorize**, który daje dostęp do obiektu **Authentication**. Odnośik **Login/Logout** w kodzie HTML Thymeleaf może mieć postać:

```
<a sec:authorize="!isAuthenticated()" th:href="@{/login}">Login</a>
<a sec:authorize="isAuthenticated()" th:href="@{/logout}">Logout</a>
```

Login powinien już działać, natomiast Logout możemy uruchomić dołączając do łańcucha wywołań w **metodzie securityFilterChain** funkcję logout ustawiającą adres strony do której nastąpi przekserowanie,np. strony głównej naszej aplikacji:

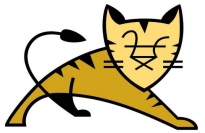
```
.logout(logout -> logout
        .logoutSuccessUrl("/")
);
```

Zadania dodatkowe

1. Zdefiniuj drugiego użytkownika i pozwól mu na wyświetlanie i edycję tylko listy książek.
2. Przydziel wybrane prawa dostępu na podstawie roli użytkownika.
3. Zmień miejsce przechowywania informacji o użytkownikach z pamięci na tabelę user w bazie danych.
4. Włącz zabezpieczenia, np. CSRF, CORS i dostosuj do nich logowanie.

Opublikowanie aplikacji na serwerze foka

Wyeksportuj aplikację do pliku .war wybierając projekt i opcję Export |WAR file. Opublikuj aplikację na swoim egzemplarzu Tomcat na serwerze foka.



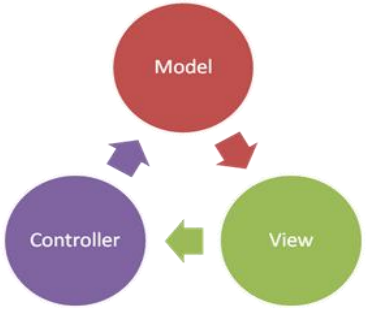
Apache Tomcat

Sprawozdanie

DO: 29.11.2023

W sprawozdaniu w systemie Sprawer wyślij link do działającej aplikacji wystawionej na serwerze foka oraz folder `main` i plik build.gradle z projektu ze źródłami spakowany do pliku .zip. W przypadku, gdy projekt nie pochodzi z Eclipse IDE, proszę dodatkowo w komentarzu napisać w jakim IDE był realizowany.


spring


Model
Controller
View


Apache Tomcat

Powodzenia!


Java


PostgreSQL

Źródła

- https://www.infoworld.com/article/3336161/what-is-jsp-introduction-to-javaserver-pages.html,
- https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server,
- https://spring.io/projects/spring-boot,
- https://www.baeldung.com/spring-core-annotations,