# Użyte mechanizmy

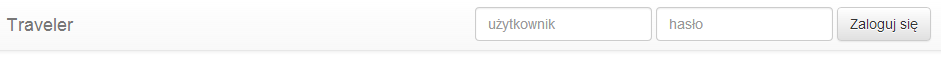
## Autentykacja, Autoryzacja, Zapewnienie poufności danych (hasła)

*Logowanie przy użyciu nazwy użytkownika i hasła. Zastosowanie mechanizmu Spring Security.*

*Dostęp oparty o role i uprawnienia RBAC. Zastosowanie mechanizmu Spring Security.*

*Zastosowanie funkcji skrótu bcrypt z pakietu Spring Security.*

**Widok przed logowaniem:**



**Widok po logowaniu:**



**Konfiguracja logowania, dostępu i szyfrowania hasła:**

<beans …

<security:http auto-config=*"true"* use-expressions=*"true"*>

<security:intercept-url pattern=*"/lista-\*"* access=*"hasRole('ROLE\_USER')"* />

<security:intercept-url pattern=*"/dodaj-\*"* access=*"hasRole('ROLE\_USER')"* />

<security:intercept-url pattern=*"/edytuj-\*"* access=*"hasRole('ROLE\_USER')"* />

<security:intercept-url pattern=*"/usun-\*"* access=*"hasRole('ROLE\_USER')"* />

</security:http>

<bean id=*"bCryptPasswordEncoder"* class=*"org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"*>

<constructor-arg value=*"12"*/>

</bean>

<security:authentication-manager>

<security:authentication-provider>

<security:password-encoder ref=*"bCryptPasswordEncoder"* />

<security:user-service>

<!-- name="user" password="user" -->

<security:user name=*"user"* password=*"$2a$12$iDKtaFDnLPuRNWANU7GdJ.NW/J5zzozFBwhENDJKkxDMweJaNa5Ga"* authorities=*"ROLE\_USER"*/>

</security:user-service>

</security:authentication-provider>

</security:authentication-manager>

</beans>

## Walidacja danych otrzymanych od użytkownika

*Wykorzystanie standardu JSR-303 Bean Validation, zaimplementowanego przez Hibernate Validator.*

**Realizacja przez oznaczenie walidowanych pól odpowiednimi adnotacjami:**

@Data

**public** **class** TouristEventCommand {

**private** Long id;

@NotEmpty

**private** String name;

@NotEmpty

**private** String description;

@NotEmpty

**private** String operator;

@Min(value=1)

@NotNull

**private** Integer peopleLimit;

@NotNull

**private** MultipartFile statue;

**private** String statueUrl;

@NotNull

**private** List<MultipartFile> photos;

**private** List<String> photoUrls;

@Min(value=1)

@NotNull

**private** Long countryId;

@Min(value=1)

@NotNull

**private** Long cityId;

@Min(value=1)

@NotNull

**private** Long hotelId;

@NotNull

@Size(min=3,max=30)

**private** List<Long> touristEventComponentIds;

}

**Oznaczenie obiektu do walidacji adnotacją @Valid w controllerze, w metodzie, która realizuje akcję:**

**public** String addTouristEvent(HttpServletRequest request, Model model, @Valid TouristEventCommand touristEventCommand, BindingResult result) **throws** Exception

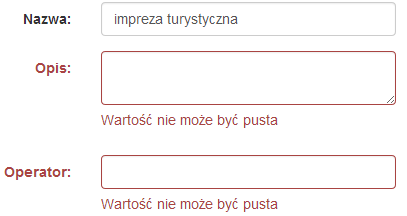
**Obsługa sytuacji wystąpienia błędów w metodzie akcji:**

**if** (result.hasErrors()) {

**return** addTouristEventForm(model, touristEventCommand);

}

**Komunikaty walidacji na stronie:**

****

## Walidacja danych zapisywanych do bazy

*Wykorzystanie mechanizmu restrykcji zapisywanych danych bazy Postgresql. Pola w tabeli zostaną oznaczone poprzez Constraints takie jak Check, Not-Null, Unique. Check wymuszana na danych ewaluację warunku do wartości true. Not-Null wymusza obecność danych a Uniqe unikatowość.*

**Encje zostają oznaczone adnotacjami charakteryzującymi ograniczenia:**

**public** **class** Price {

@Id

@GeneratedValue(generator = "increment")

@GenericGenerator(name = "increment", strategy = "increment")

**private** Long id;

@Max(value = 100000)

@Min(value = 0)

@NotNull

**private** Integer adultValue;

@Max(value = 100000)

@Min(value = 0)

@NotNull

**private** Integer childValue;

@NotNull

@Enumerated(EnumType.*STRING*)

**private** PriceType type;

@ManyToOne(fetch = FetchType.*LAZY*)

@JoinColumn(name = "period\_id")

**private** Period period;

@ManyToOne(fetch = FetchType.*LAZY*)

@JoinColumn(name = "tourist\_event\_component\_id")

**private** TouristEventComponent touristEventComponent;

@ManyToMany(mappedBy = "prices", fetch = FetchType.*LAZY*, cascade = CascadeType.*ALL*)

**private** List<Reservation> reservations;

}

**Z których generowany jest schemat z nałożonymi ograniczeniami w bazie danych:**

CREATE TABLE prices

(

id bigint NOT NULL,

adultvalue integer NOT NULL,

childvalue integer NOT NULL,

type character varying(255) NOT NULL,

period\_id bigint,

tourist\_event\_component\_id bigint,

CONSTRAINT prices\_pkey PRIMARY KEY (id ),

CONSTRAINT fk\_8a029o3967hjbtlk520rir8k3 FOREIGN KEY (period\_id)

REFERENCES periods (id) MATCH SIMPLE

ON UPDATE NO ACTION ON DELETE NO ACTION,

CONSTRAINT fk\_b01804rs70rgkw9y98y5pgncp FOREIGN KEY (tourist\_event\_component\_id)

REFERENCES tourist\_event\_components (id) MATCH SIMPLE

ON UPDATE NO ACTION ON DELETE NO ACTION,

CONSTRAINT prices\_adultvalue\_check CHECK (adultvalue >= 0 AND adultvalue <= 100000),

CONSTRAINT prices\_childvalue\_check CHECK (childvalue >= 0 AND childvalue <= 100000)

)

## Wprowadzenie cache bazy danych

*Wykorzystanie mechanizmu keszowania frameworku ORM Hibernate.*

## Wprowadzenie cache przetworzonych danych

*Wykorzystanie mechanizmu biblioteki Ehcache. Keszowane będą dane pobrane i przetworzone przez warstwę serwisową aplikacji. Zapisane w pamięci podręcznej informacje będą wykorzystywane przez kontrolery.*

**Metodę serwisu, której wyniki chcemy umieścić w pamięci podręcznej oznaczamy adnotacją:**

@Cacheable(cacheName = "cities")

**public** List<City> listCities(String condition)

Elementy są przechowywane pojedynczo – nie jako jeden obiekt kolekcji.

**Metodę serwisu, która unieważnia pamięć podręczną również oznaczamy adnotacją:**

@TriggersRemove(cacheName = {"cities","hotels"}, removeAll = **true**)

**public** **void** removeCity(Long cityId)

**Konfiguracja pamięci podręcznej w pliku ehcache.xml:**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<ehcache xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:noNamespaceSchemaLocation=*"http://ehcache.org/ehcache.xsd"*>

<defaultCache maxElementsInMemory=*"1000"* eternal=*"false"*

timeToIdleSeconds=*"3600"* overflowToDisk=*"false"* />

<cache name=*"countries"* maxElementsInMemory=*"300"* eternal=*"true"*

overflowToDisk=*"false"* />

<cache name=*"cities"* maxElementsInMemory=*"500"* maxElementsOnDisk=*"1000"*

eternal=*"false"* timeToIdleSeconds=*"3600"* overflowToDisk=*"true"* />

<cache name=*"hotels"* maxElementsInMemory=*"2000"*

maxElementsOnDisk=*"2000"* eternal=*"false"* timeToIdleSeconds=*"3600"*

overflowToDisk=*"true"* />

</ehcache>

**Przykładowe dostępne parametry pamięci podręcznej:**

* name – nazwa pamięci podręcznej
* maxElementsInMemory – maksymalna liczba elementów w pamięci
* eternal – jeżeli ustawione, to elementy ignorują timeouty i nigdy nie wygasają.
* overflowToDisk – jeżeli zostanie przekroczony limit pamięci elementy zapisywane są na dysk
* maxEntriesOnlDisk – maksymalna liczba elementów zapisanych na dysku
* timeToIdleSeconds – czas nieużywania elementu po jakim element wygasa i jest usuwany
* timeToLiveSeconds– czas po jakim element wygasa i jest usuwany
* copyOnRead – jeżeli ustawione to pamięć podręczna zwraca kopię elementu
* copyOnWrite – jeżeli ustawione to pamięć podręczna zapisuje kopię elementu

Czyli wpis:

<cache name=*"hotels"* maxElementsInMemory=*"2000"*

maxElementsOnDisk=*"2000"* eternal=*"false"* timeToIdleSeconds=*"3600"*

overflowToDisk=*"true"* />

Tworzy pamięć podręczną o nazwie “hotels”, w której przechowuje do 2000 elementów w pamięci i do 2000 elementów na dysku. Elementy przenosi na dysk po wyczerpaniu miejsca w pamięci. Elementy wygasają i są usuwane z pamięci po godzinie braku trafień.

## Wprowadzanie pośredników - interfejsów

*Wykorzystanie paradygmatu odwrócenia kontroli i mechanizmu wstrzykiwania zależności do inicjalizacji obiektów, zaimplementowanego przez Spring. Przykładowe wykorzystanie:*

**Definicja interfejsu:**

public interface TouristEventService **{**

**}**

**Definicja klasy serwisu:**

@Service

public class TouristEventServiceImpl **implements** TouristEventService **{**

**}**

**Definicja serwisu w kontroler:**

@Controller

public class TouristEventController **{**

@Inject

private TouristEventService touristEventService**;**

**}**