Bookstore

4

Koszyk

Koszyk powinien umożliwić użytkownikowi dodawanie produktów z księgarni oraz ich usuwanie.

Każdy User będzie posiadał jeden koszyk, który będzie zapisywany i odświeżany w bazie.

Uwaga – użytkowników powiązanych z koszykami w bazie danych projektu najłatwiej będzie utworzyć na nowo.

Koszyk

Użytkownik będzie posiadał jeden koszyk. Koszyk będzie dodawany do użytkownika podczas jego Rejestracji.

W koszyku dodajemy również Relację OneToMany z CartItem.

W klasie User należy dodać relację oraz obiekt typu Cart:

```
@OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)
@JoinColumn(name = "cart_id")
private Cart cart;
```

W klasie Cart:

```
@Entity
public class Cart {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToMany(mappedBy = "cart", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<CartItem> items = new ArrayList<>();

    @OneToOne(mappedBy = "cart")
    private User user;
```

Należy pamiętać o getterach i setterach.

Koszyk

Koszyk nie przechowuje książek bezpośrednio, tylko obiekty cart_item w liscie items, w obiektach tych oprócz książki jest informacja o jej ilości w koszyku i koszyk w którym się przedmiot znajduje

```java
@Entity
public class CartItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "cart_id")
    private Cart cart;

    @ManyToOne
    private Book book;

    private int quantity;
   //...gettery i settery
```

Koszyk

Operacje wykonywane kaskadowo
jeżeli jest wykonana na jednej encji,
to również na encjach powiązanych:

```
cascade = CascadeType.ALL
```

ALL - Stosuje wszystkie operacje kaskadowo.
PERSIST - zapisania encji.
MERGE - scalania encji.
REMOVE - usunięcia encji.
REFRESH - odświeżenia encji.
DETACH - odłączenia encji.

Koszyk

W UserService przy rejestracji należy dodać tworzenie nowego koszyka.

Uwaga! Koszyk posiada wszystkie kaskadowe operacje więc zapisze się razem z nowym userem.

Rola nie ma ustawionego kaskadowo zapisu – należy zapisać ją ręcznie. Tworzy się tylko wtedy kiedy nie ma w bazie roli „USER”

```
@Transactional
public String registerUser(User user) {
    if
(userRepository.findByUsername(user.getUsername()).isPr
esent()) {
        return "failure";
    }

user.setPassword(passwordEncoder.encode(user.getPasswor
d()));
    user.setCart(new Cart());
    Role userRole =
roleRepository.findByName("USER").orElse(null);
    //System.out.println(userRole.getName());
    if (userRole != null) {
        user.getRoles().add(userRole);
    }
     else {
         Role role = new Role();
         role.setName("USER");
        user.getRoles().add(role);
        roleRepository.save(role);
    }

        userRepository.save(user);
        return "success";
}

@Transactional
public void save(User user) {
    userRepository.save(user);
}
```

Koszyk

CartService powinien umożliwiać dodanie elementu do koszyka, usunięcie elementu, pobranie koszyka zalogowanego użytkownika, oraz zapis/update koszyka w bazie

Koszyk

```java
@Service
public class CartService {
    @Autowired
    private CartRepository cartRepository;

    @Autowired
    private IBookDAO bookRepository;

    @Autowired
    private UserService userService;

    @Transactional
    public Cart getCart() {
        User user = userService.getCurrentUser();
        return user.getCart();
    }

    @Transactional
    public Cart addToCart(int bookId, int quantity) {
        Cart cart = getCart();
        Book book = bookRepository.getById(bookId).orElseThrow(()
            -> new RuntimeException("Book not found"));
        cart.addItem(book, quantity);
        return saveCart(cart);
    }

    @Transactional
    public Cart removeFromCart(int bookId) {
        Cart cart = getCart();
        Book book = bookRepository.getById(bookId).orElseThrow(()
            -> new RuntimeException("Book not found"));
        cart.removeItem(book);
        return saveCart(cart);
    }

    @Transactional
    public Cart saveCart(Cart cart) {
        return cartRepository.save(cart);
    }}
```

Koszyk

Repozytorium Cart wygenerujemy automatycznie:

```
public interface CartRepository extends JpaRepository<Cart, Integer> {
}
```

Pobieranie zalogowanego użytkownika powinno znajdować się w serwisie użytkownika:

```
@Transactional
public User getCurrentUser() {
    Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    String username = ((UserDetails) principal).getUsername();
    return userRepository.findByUsername(username).orElse(null);
}
```

principal w Spring Security reprezentuje uwierzytelnionego użytkownika. Jest to obiekt, który zawiera szczegóły dotyczące aktualnie zalogowanego użytkownika . W naszym przypadku: UserDetails.

Koszyk

Do strony usera z listą książek dodajemy opcję dodania elementu do koszyka:

```
<a th:href="@{/cart/add/{bookId}/1(bookId=${book.id})}">Add to Cart</a>
```

W kontrolerze Koszyka:

```
@GetMapping("/add/{bookId}/{quantity}")
public String addToCart(@PathVariable int bookId, @PathVariable int quantity) {
    cartService.addToCart(bookId, quantity);
    return "redirect:/cart";
}
```

Dodatkowo usuwanie z koszyka i pobranie koszyka:

```
@GetMapping
public String getCart(Model model) {
    Cart cart = cartService.getCart();
    model.addAttribute("cart", cart);
    return "cart";
}
```

```
@GetMapping("/remove/{bookId}")
public String removeFromCart(@PathVariable int bookId) {
    cartService.removeFromCart(bookId);
    return "redirect:/cart";
}
```

Koszyk

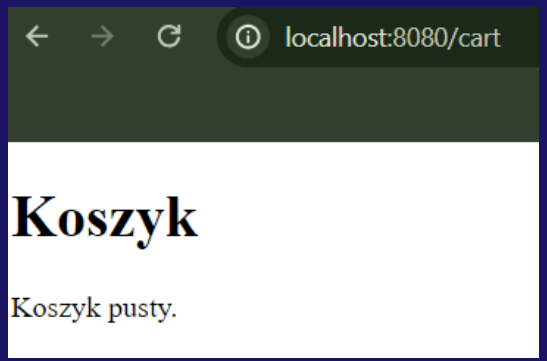
Strona z koszykiem, Gdzie można usunąć element z koszyka.

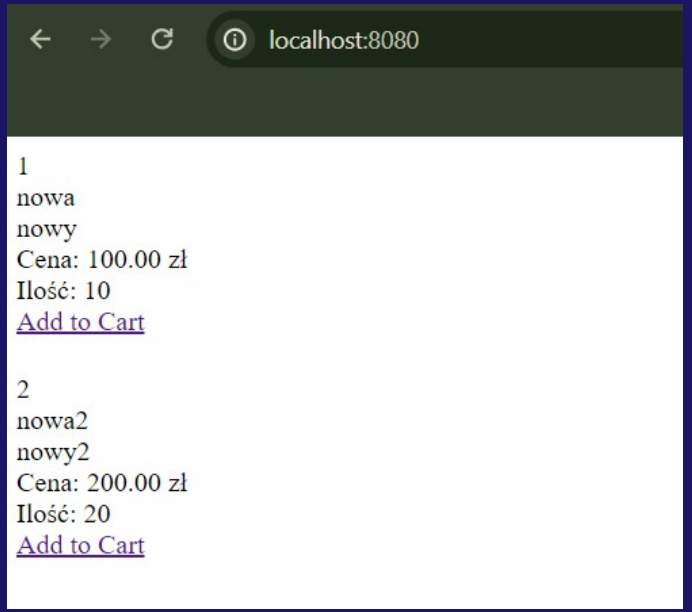
W przykładzie dodajemy do koszyka za pomocą metody get ze zmienną ze ścieżki, podobnie robimy przy usuwaniu.

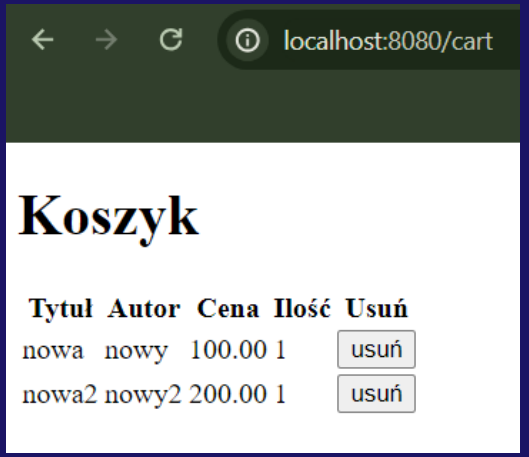
W naszym przykładzie Dodajemy jeden egzemplarz oraz usuwamy wszystkie egzemplarze danej książki

```
<h1>Koszyk</h1>
<div th:if="${cart.items.isEmpty()}">
    <p>Koszyk pusty.</p>
</div>
<div th:unless="${cart.items.isEmpty()}">
    <table>
        <thead>
        <tr>
            <th>Tytuł</th>
            <th>Autor</th>
            <th>Cena</th>
            <th>Ilość</th>
            <th>Usuń</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="item : ${cart.items}">
            <td th:text="${item.book.title}"></td>
            <td th:text="${item.book.author}"></td>
            <td th:text="${item.book.price}"></td>
            <td th:text="${item.quantity}"></td>
            <td>
                <form th:action="@{/cart/remove/{bookId}(bookId=${item.book.id})}" method="get">
                    <button type="submit">usuń</button>
                </form>
            </td>
        </tr>
        </tbody>
    </table>
</div>
```

Koszyk


localhost:8080/cart
Koszyk
Koszyk pusty.


localhost:8080
1
nowa
nowy
Cena: 100.00 zł
Ilość: 10
Add to Cart
2
nowa2
nowy2
Cena: 200.00 zł
Ilość: 20
Add to Cart


localhost:8080/cart
Koszyk
Tytuł Autor Cena Ilość Usuń
nowa nowy 100.00 1 usuń
nowa2 nowy2 200.00 1 usuń