

# Bookstore

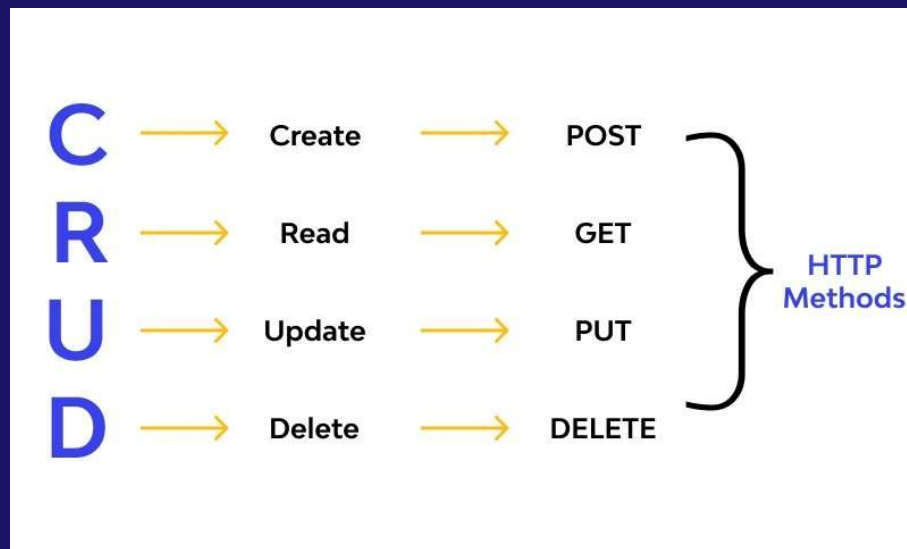
## 3



# Operacje CRUD

**CRUD** to akronim od czterech podstawowych operacji, które można wykonywać na danych w aplikacjach bazodanowych

Uwaga! W formularzu html mamy dostępne tylko metody **GET** i **POST**!!!



# LOMBOK

Lombok to biblioteka Java, która automatycznie generuje kod na podstawie adnotacji.

Pozwoli zaoszczędzić czas podczas pisania klasy Encji Książki: Book.

```
<dependency>  
  <groupId>org.projectlombok</groupId>  
  <artifactId>lombok</artifactId>  
  <scope>provided</scope>  
</dependency>
```



Project  
Lombok

# CRUD

Prosty przykład klasy encji przy użycia LOMBOKa.

Encja zawiera dane na temat książki:  
Id, tytuł, autor, cena oraz ilość książek w sklepie.

Będziemy używali dodatkowo konstruktora z id książki.

```
@NoArgsConstructor
@Getter
@Setter
@ToString
@EqualsAndHashCode
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy =
        GenerationType.IDENTITY)

    private int id;
    private String title;
    private String author;
    private BigDecimal price;
    private int quantity;

    public Book(int id){
        this.id = id;
    }
}
```

# CRUD

## CRUD

### Create

tworzenia nowych danych

### Read

odczytywanie danych

### Update

aktualizacja istniejących  
danych

### Delete

usuwanie danych

## CRUD

**UWAGA!!** Zmiana w konfiguracji aby nasze endpointy mogły działać:

```
//csrf(csrf->csrf.ignoringRequestMatchers("/h2-console/**"))  
.csrf(csrf->csrf.disable())
```

Potrzebujemy Serwisu oraz Repozytorium, za pomocą którego pobierzemy książkę z bazy.

Serwis będzie zawierał metody pokazane w interfejsie.

Połączymy zapisywanie i update w jedną metodę.

```
public interface IBookService {  
    void saveOrUpdate(Book book);  
    Optional<Book> getById(int id);  
    List<Book> getAll();  
    void delete(int id);  
}
```

# CRUD

Potrzebujemy Serwisu oraz Repozytorium, za pomocą którego pobierzemy książkę z bazy.

Serwis będzie zawierał metody pokazane w interfejsie.

Połączymy zapisywanie i update w jedną metodę.

Adnotacja `@Transactional`- automatyczne zarządzanie transakcjami, upraszcza kod i eliminuje konieczność ręcznego zarządzania transakcjami.

Serwis wykorzystuje repozytorium `IbookDAO`. Musimy napisać interfejs oraz implementację.

```
@Service
public class BookService implements IBookService {

    @Autowired
    private final IBookDAO bookDAO;

    public BookService(IBookDAO bookDAO) {
        this.bookDAO = bookDAO;
    }

    @Override
    @Transactional
    public Optional<Book> getById(int id) {
        return this.bookDAO.getById(id);
    }

    @Override
    @Transactional
    public List<Book> getAll() {
        return this.bookDAO.getAll();
    }

    @Transactional
    public void saveOrUpdate(Book book) {
        this.bookDAO.saveOrUpdate(book);
    }

    @Override
    @Transactional
    public void delete(int id){
        bookDAO.delete(id);
    }
}
```

# CRUD – wyświetlenie wszystkich książek

W kontrolerze z poprzednich zajęć umieszczamy metodę, która wyświetli stronę z wszystkimi książkami.

Będzie to strona główna naszej księgarni:

```
@RequestMapping(path = {"/main", "/", "/index"}, method = RequestMethod.GET)
public String main(Model model) {
    model.addAttribute("books", this.bookService.getAll());
    return "index";
}
```



# CRUD – wyświetlenie wszystkich książek

Brakuje jeszcze repozytorium książek.  
Na początku napiszmy metodę, która zwróci  
wszystkie książki:

Interfejs:

```
public interface IBookDAO {  
    Optional<Book> getById(int id);  
    List<Book> getAll();  
    void saveOrUpdate(Book book);  
    void delete(int id);  
}
```

```
@Repository  
public class BookDAO implements IBookDAO {  
  
    @PersistenceContext  
    private EntityManager entityManager;  
    private final String GET_ALL_JPQL = "FROM com.umcspro.bookstore.model.Book";  
  
    public BookDAO(EntityManager entityManager) {  
        this.entityManager = entityManager;  
    }  
  
    @Override  
    public List<Book> getAll() {  
        TypedQuery<Book> query = entityManager.createQuery(GET_ALL_JPQL, Book.class);  
        List<Book> result = query.getResultList();  
        return result;  
    }  
}
```

# CRUD – wyświetlenie wszystkich książek

Strona internetowa z wszystkimi książkami:

W modelu przekazujemy listę książek, aby w pętli pobrać z niej wszystkie książki wykorzystujemy znacznik :

```
th:each="book : ${books}"
```

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
</head>
<body>

<div>
  <div th:each="book : ${books}">
    <div th:text="${book.id}"></div>
    <div th:text="${book.title}"></div>
    <div th:text="${book.author}"></div>
    <div th:text="${'Cena: ' + book.price + ' zł'}"></div>
    <div th:text="${'Ilość: ' + book.quantity}"></div>

  </div>
</div>
</body>
</html>
```

# CRUD – wyświetlenie wszystkich książek

Uzupełnienie danych:

Run Run Selected Auto complete Clear SQL statement:

```
INSERT INTO book (author,price,quantity,title) VALUES ('Ray Horstman',100,10,'Java Techniki zaawansowane. Wydanie X');  
INSERT INTO book (author,price,quantity,title) VALUES ('Ray Horstman',100,10,'Java Podstawy. Wydanie XII');
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM BOOK
```

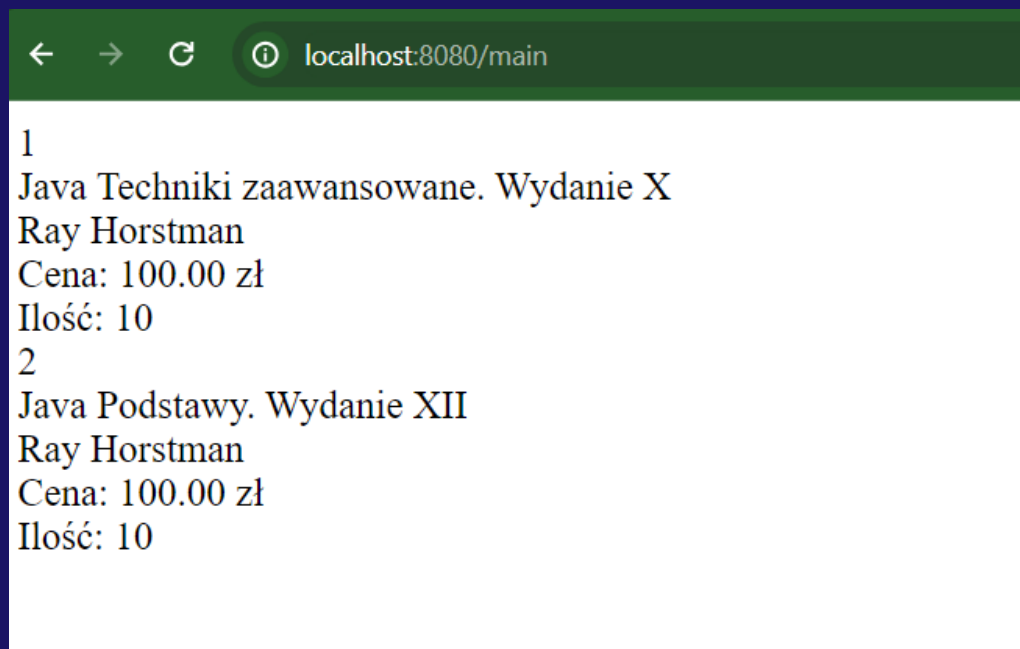
```
SELECT * FROM BOOK;
```

ID	AUTHOR	PRICE	QUANTITY	TITLE
1	Ray Horstman	100.00	10	Java Techniki zaawansowane. Wydanie X
2	Ray Horstman	100.00	10	Java Podstawy. Wydanie XII

(2 rows, 1 ms)

# CRUD – wyświetlenie wszystkich książek

Strona internetowa z wszystkimi książkami:

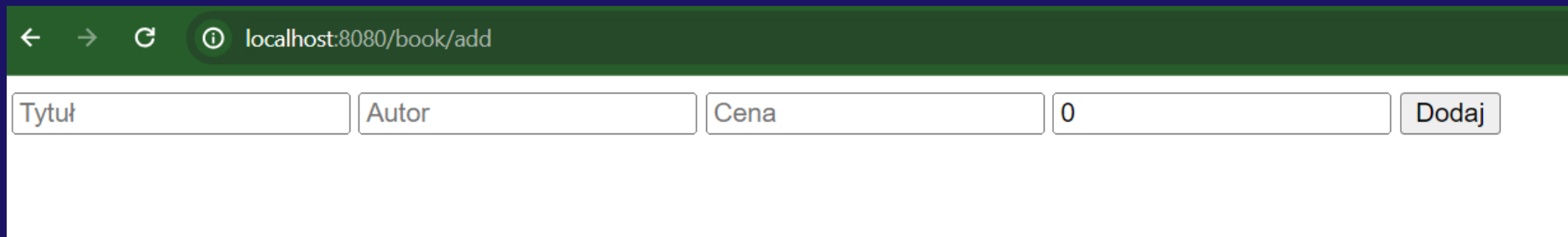


# CRUD – Dodanie nowej książki

Strona book-form.html:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<body>

<div>
  <form method="post">
    <input type="text" placeholder="Tytuł" class="input-field" th:field="*{book.title}" id="title">
    <input type="text" placeholder="Autor" class="input-field" th:field="*{book.author}" id="author">
    <input type="text" placeholder="Cena" class="input-field" th:field="*{book.price}" id="price">
    <input type="text" placeholder="Ilość" class="input-field" th:field="*{book.quantity}" id="quantity">
    <input type="submit" value="Dodaj">
  </form>
</div>
</body>
</html>
```



A screenshot of a web browser window. The address bar shows 'localhost:8080/book/add'. The page contains a form with four input fields: 'Tytuł', 'Autor', 'Cena', and '0'. A 'Dodaj' button is located to the right of the '0' field. The form is styled with a light blue background and rounded corners.

# CRUD – Dodanie nowej książki

Należy zwrócić uwagę na ścieżkę endpointów. Jest "niepełna" w porównaniu do tej w przeglądarce.

Ścieżka powinna być dodana też przed klasą kontrolera.

Np:

```
@Controller
@RequestMapping(path = "/book")
public class BookController {
    //...
}
```

W nowej klasie kontrolera np. BookController, przy wyświetleniu formularza, trzeba przekazać nową książkę przez model:

```
@RequestMapping(path = "/add", method = RequestMethod.GET)
public String add(Model model) {
    model.addAttribute("book", new Book());
    return "book-form";
}
```

Przy przesyłaniu trzeba pobrać obiekt z modelu:

```
@RequestMapping(path = "/add", method = RequestMethod.POST)
public String add(@ModelAttribute Book book) {
    this.bookService.saveOrUpdate(book);
    return "redirect:/main";
}
```

# CRUD – Dodanie nowej książki

```
private final String GET_BY_ID_JPQL= "SELECT b FROM com.umcspro.bookstore.model.Book b WHERE b.id = :id";
```

W serwisie od razu dodaliśmy wszystkie potrzebne metody, w repozytorium należy dopisać metodę saveOrUpdate:

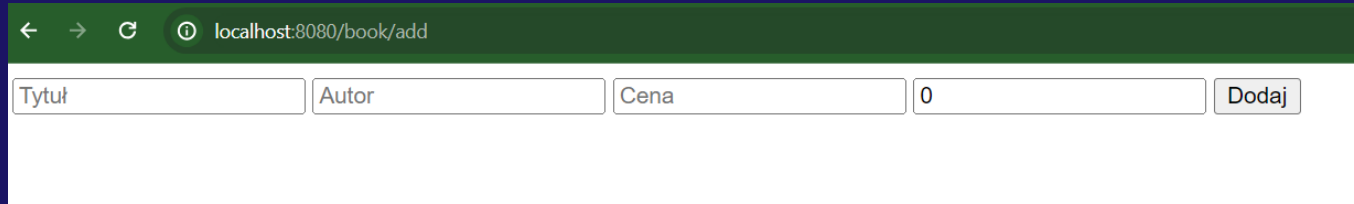
Jeżeli znaleziono książkę, robimy Update-merge.  
Jeżeli nie – tworzymy nową:  
persist.

```
public void saveOrUpdate(Book book) {  
    System.out.println("BOOK "+book);  
    if ( getId(book.getId()).isEmpty() ) {  
        entityManager.persist(book);  
    } else {  
        entityManager.merge(book);  
    }  
}
```

Oraz pobranie Optional<Book> po id:


```
@Override  
public Optional<Book> getId(int id) {  
    TypedQuery<Book> query = entityManager.createQuery(GET_BY_ID_JPQL, Book.class);  
    query.setParameter("id", id);  
  
    try {  
        return Optional.of(query.getSingleResult());  
    } catch (NoResultException e) {  
        return Optional.empty();  
    }  
}
```

# CRUD – Dodanie nowej książki



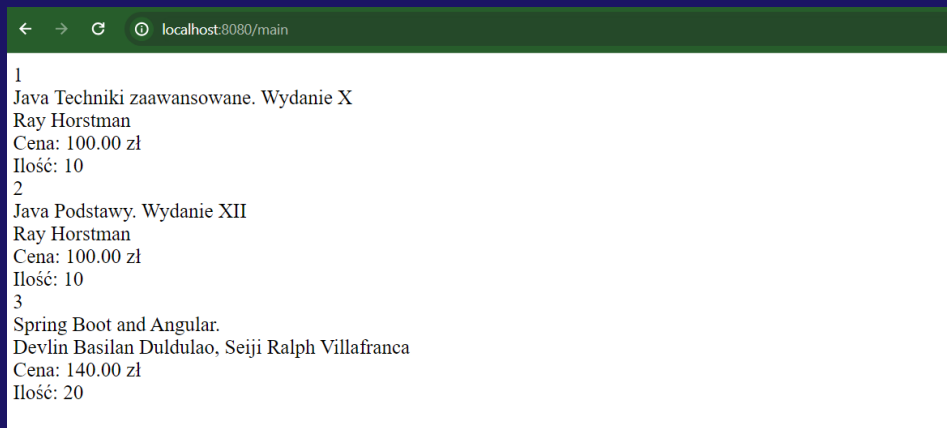
localhost:8080/book/add

Tytuł Autor Cena 0 Dodaj



localhost:8080/book/add

Spring Boot and Angular. Devlin Basilan Duldulao, S 140 20 Dodaj



localhost:8080/main

1	Java Techniki zaawansowane. Wydanie X	Ray Horstman	Cena: 100.00 zł	Ilość: 10
2	Java Podstawy. Wydanie XII	Ray Horstman	Cena: 100.00 zł	Ilość: 10
3	Spring Boot and Angular.	Devlin Basilan Duldulao, Seiji Ralph Villafranca	Cena: 140.00 zł	Ilość: 20



# CRUD – Edycja książki



Aby edytować książkę dodajmy na stronie z listą książek w pętli link do odpowiedniego endpoint'a:

```
<a th:href="@{/book/update/{id}(id=${book.id})}">Edytuj</a>
```

Id przekazywane w urlu pobierane jest dla danej książki znajdującej się w liście książek z modelu.

# CRUD – Edycja książki

W kontrolerze update podobnie jak  
Przy dodawaniu zwraca formularz do  
Dodawania książki – z tym, że  
Najpierw wyszukuje książki po id  
A następnie dodaje ją do modelu.

W wypadku dodawania – w modelu  
znajdowała się pusta książka.

Formularz na stronie  
Dodawania/updateu przekazuje dane do  
Endpointa z takim samym adresem jaki jest  
podczas jego wyświetlania.

Dla update przy metodzie POST ustawienie id  
można pominąć – odpowiednie id znajduje się  
już w książce.

```
@RequestMapping(path = "/update/{id}", method = RequestMethod.GET)
public String update(@PathVariable int id, Model model) {
    Optional<Book> bookOpt = this.bookService.getById(id);
    if(bookOpt.isEmpty()) {
        return "redirect:/main";
    }
    model.addAttribute("book", bookOpt.get());
    return "book-form";
}

@RequestMapping(path = "/update/{id}", method = RequestMethod.POST)
public String update(@PathVariable int id, @ModelAttribute Book book) {
    //book.setId(id);
    this.bookService.saveOrUpdate(book);
    return "redirect:/main";
}
```

# CRUD – Edycja książki

← → ↻ ⓘ localhost:8080/main

1  
Java Techniki zaawansowane. Wydanie X  
Ray Horstman  
Cena: 100.00 zł  
Ilość: 10  
[Edytuj](#)

2  
Java Podstawy. Wydanie XII  
Ray Horstman  
Cena: 100.00 zł  
Ilość: 10  
[Edytuj](#)

3  
Spring Boot and Angular.  
Devlin Basilan Duldulao, Seiji Ralph Villafranca  
Cena: 140.00 zł  
Ilość: 20  
[Edytuj](#)



← → ↻ ⓘ localhost:8080/book/update/3



1  
Java Techniki zaawansowane. Wydanie X  
Ray Horstman  
Cena: 100.00 zł  
Ilość: 10  
[Edytuj](#)

2  
Java Podstawy. Wydanie XII  
Ray Horstman  
Cena: 100.00 zł  
Ilość: 10  
[Edytuj](#)

3  
Spring Boot and Angular.  
Devlin Basilan Duldulao, Seiji Ralph Villafranca  
Cena: 200.00 zł  
Ilość: 200  
[Edytuj](#)

# CRUD – Usuwanie książki

Tym razem w pętli na stronie dodajemy przycisk do usuwania z id książki z listy.

W repozytorium – jeżeli znajdziemy książkę o danym id – usuwamy ją.

W kontrolerze używamy serwisu do usunięcia książki ( metodę w serwisie już zaimplementowaliśmy)

```
<form method="post" th:action="@{/book/delete}">
  <input type="hidden" th:name="id" th:value="${book.id}" />
  <input type="submit" value="Usuń" />
</form>
```

```
@Override
public void delete(int id) {
    Book book = getById(id).orElse(null);
    if (book != null) {
        entityManager.remove(book);
    }
}
```

```
@PostMapping("/delete")
public String deleteBook(@RequestParam int id) {
    bookService.delete(id);
    return "redirect:/main";
}
```

# CRUD – Usuwanie książki

