Sprawozdanie 3

Spring – Rafał Bysiek

Przedmiotem zadania jest przerobienie aplikacji RESTowej biblioteki, która pisana była za pomocą czystych servletów na aplikację wykorzystującą framework Spring.

Aplikacja została utworzona w IntelliJ IDEA, oraz uruchamiana jest przy pomocy Tomcat Server w wersji 9.
Obsługa requestów - aplikacji odbywa się przy pomocy programu Postman.

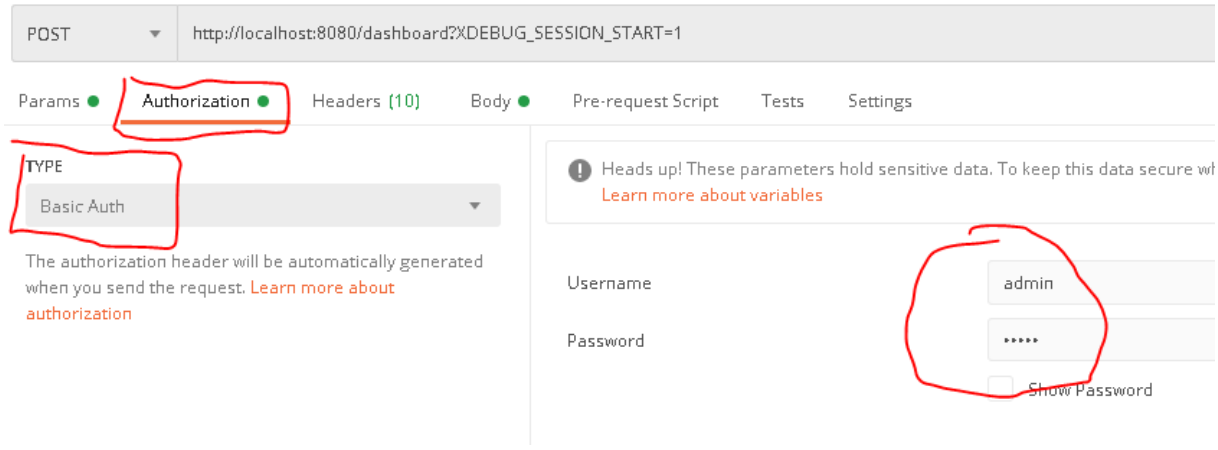
Do testowania działania użyto XDEBUG dodając do request ?XDEBUG_SESSION_START=1

Logowanie

Logowanie w aplikacji zastąpiono metodą autoryzacji Basic Auth.

Polega ona na wprowadzeniu loginu oraz hasła użytkownika jako jednego z elementów requestu.

Tak jak poprzednio ograniczono możliwości użytkowników: user może przeglądać kolekcję, admin może dodatkowo dodawać/usuwać rekordy.




```java
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Bean
    public UserDetailsService userDetailsService() {
        return new UserDetailsServiceModel();
    }

    @SuppressWarnings("deprecation")
    @Bean
    public static NoOpPasswordEncoder passwordEncoder() {
        return (NoOpPasswordEncoder) NoOpPasswordEncoder.getInstance();
    }
```

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth.userDetailsService(userDetailsService());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .httpBasic()
            .and()
            .authorizeRequests()
            .antMatchers(HttpMethod.GET, "/dashboard").hasAnyRole("ADMIN",
"USER")
            .antMatchers(HttpMethod.POST, "/dashboard").hasRole("ADMIN")
            .antMatchers(HttpMethod.DELETE, "/dashboard").hasRole("ADMIN")
            .and()
            .csrf().disable()
            .cors().disable();
    }
}
```

passwordEncoder () – zezwala na użycie hasła w postaci plain text (bez hashowania)

`configure(AuthenticationManagerBuilder auth)- wprowadza dane użytkowników:`

`przy pomocy nadpisanej funkcji`

```
@Override
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

    User user = this.userService.findUserByUsername(username);

    UserBuilder builder = null;
    if (user != null) {
        builder =
org.springframework.security.core.userdetails.User.withUsername(username);
        builder.password(user.getPassword());
        builder.roles(user.getRole().toString());
    } else {
        throw new UsernameNotFoundException("User not found.");
    }

    return builder.build();
}
```

Gdzie dane użytkowników są standardowo przechowywane w Mapach w klasie UserService.

Znaleźć tam też można metodę findUserByUsername.

```
public static final Map<String, String> userCredentials = new HashMap<>();
public static final Map<String, String> adminCredentials = new HashMap<>();

public UserService()
{
    userCredentials.put("user1", "user1");
    userCredentials.put("user2", "user2");
    userCredentials.put("user3", "user3");
    userCredentials.put("user4", "user4");
```

```
    adminCredentials.put("admin", "admin");
}


public User findUserByUsername(String username) {
    if (userCredentials.containsKey(username)) {
        return new User(Role.USER, username,
userCredentials.get(username));
    }
    if (adminCredentials.containsKey(username)) {
        return new User(Role.ADMIN, username,
adminCredentials.get(username));
    }
    return null;
}
```

Dashboard

Więcej zmian można zaobserwować w dashboard.

Powstała klasa **DashboardService** zajmująca się podstawowymi operacjami na liście książek

Takimi jak inicjalizacja listy książek w konstruktorze, pobranie listy książek, dodanie nowej książki czy usunięcie książki z listy.

```
static HashSet<Book> books = new HashSet<>();

public DashboardService() {
    books.add(new Book("Jak zdobyć przyjaciół i zjednać sobie ludzi", "Dale
Carnegie", "2015-01-01"));
    books.add(new Book("Bracia Karamazow", "Fiodor Dostojewski", "1880-11-
01"));
    books.add(new Book("Mój przyjaciel Leonard", "James Frey", "2005-06-
15"));
    books.add(new Book("Sapiens. Od zwierząt do bogów", "Yuval Noah
Harari", "2011-01-01"));
    books.add(new Book("Cień wiatru", "Carlos Ruiz Zafón", "2005-01-01"));
}

public static HashSet<Book> getBooks() {
    return books;
}

public static Book addBook(Book newBook) {
    checkIfBookAlreadyExist(newBook);
    books.add(newBook);
    return newBook;
}

public static boolean checkIfBookAlreadyExist(Book newBook) {
    if (books.contains(newBook)) {
        throw new EntityExistsException("Książka o podanych parametrach już
istnieje w bazie.");
    }
    return false;
}
```

```
public static Book removeBook(Integer removeBookId) {
    Book removedBook = Book.getBookById(books, removeBookId);
    books.removeIf(book -> book.getId().equals(removeBookId));
    return removedBook;
}
```

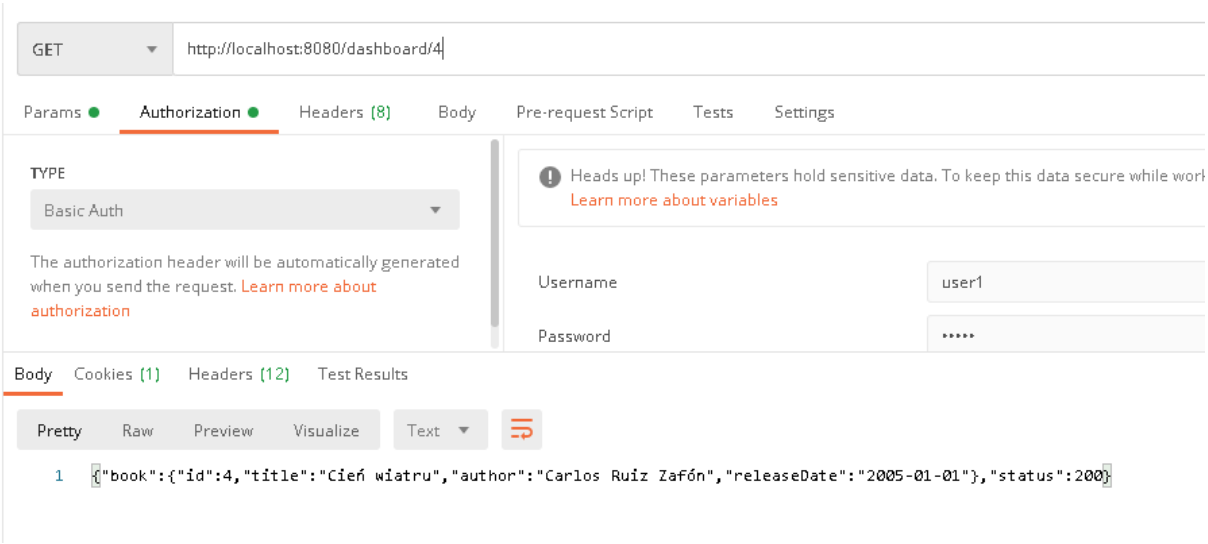
Obsługą requestów zajmuje się głównie klasa **DashboardController**, gdzie wydzielono osobne metody dla każdego obsługiwanego requestu.

Dane zwracane są w formacie JSON.

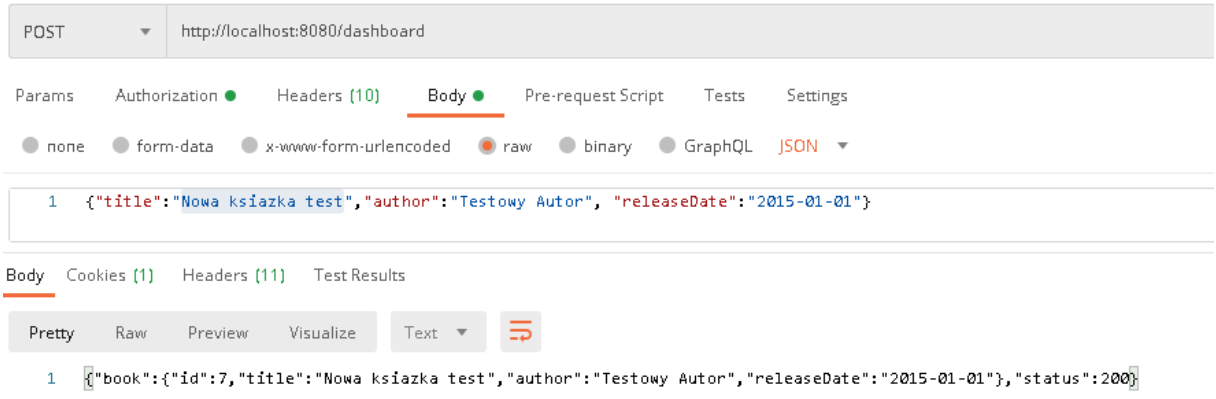
```
@GetMapping("/dashboard")
public String doGet() throws Exception {
    HashSet<Book> books = dashboardService.getBooks();
    if (books != null) {
        GetDashboardResponse response = new GetDashboardResponse(books,
200);
        return gson.toJson(response);
    }
    throw new Exception();
}

@GetMapping("/dashboard/{id}")
public String doGetWithParam(@PathVariable String id) {
    HashSet<Book> books = dashboardService.getBooks();
    Book book = Book.getBookById(books, Integer.parseInt(id));
    if (book != null) {
        GetDashboardResponse response = new GetDashboardResponse(book,
200);
        return gson.toJson(response);
    }
    throw new NoSuchEntityException("Rekord o podanym id nie istnieje.");
}

@PostMapping("/dashboard")
protected String doPost(@RequestBody Book newBook) {
    //add new book, return created obj
    dashboardService.addBook(newBook);
    GetDashboardResponse response = new GetDashboardResponse(newBook, 200);
    return gson.toJson(response);
}

@DeleteMapping("/dashboard")
protected String doDelete(@RequestParam String bookId) {
    //delete book

    Book removedBook =
dashboardService.removeBook(Integer.parseInt(bookId));
    if (removedBook == null) {
        throw new NoSuchEntityException("Rekord o podanym id nie
istnieje.");
    }
    GetDashboardResponse res = new GetDashboardResponse(removedBook, 200);

    return gson.toJson(res);
}
```

Swoistą nowością w tym projekcie jest wyświetlenie konkretnej pozycji-książki za pomocą wprowadzonego id.

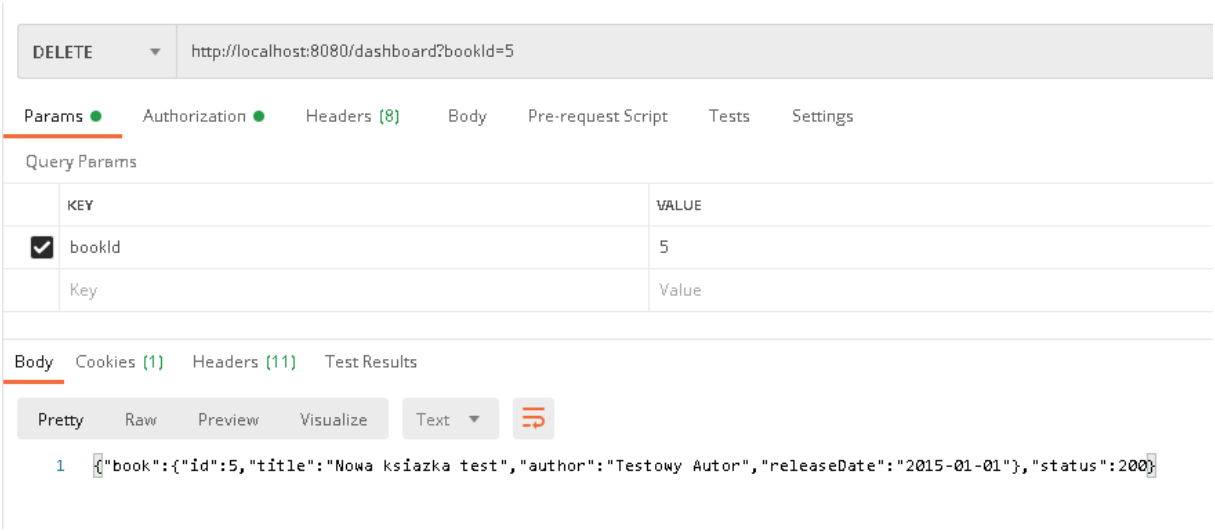
By możliwa była taka funkcja dodano generator ID dla książek.



Dodanie książki:



Usunięcie książki:



Book

Obiekt książki nie uległ zmianie i nadal występuje w postaci:

```
public Book(String title, String author, String releaseDate) {
    this.id = createBookId();
    this.title = title;
    this.author = author;
    this.releaseDate = releaseDate;
}
```

Generator id

```
public Integer createBookId()
{
    return idCounter++;
}
```

Zwracany przez system obiekt jest w postaci JSON (widoczny na poprzednich screenshotach z programu Postman).

Exceptions

Nowością w projekcie jest obsługa błędów zwracanych w trakcie działania aplikacji.

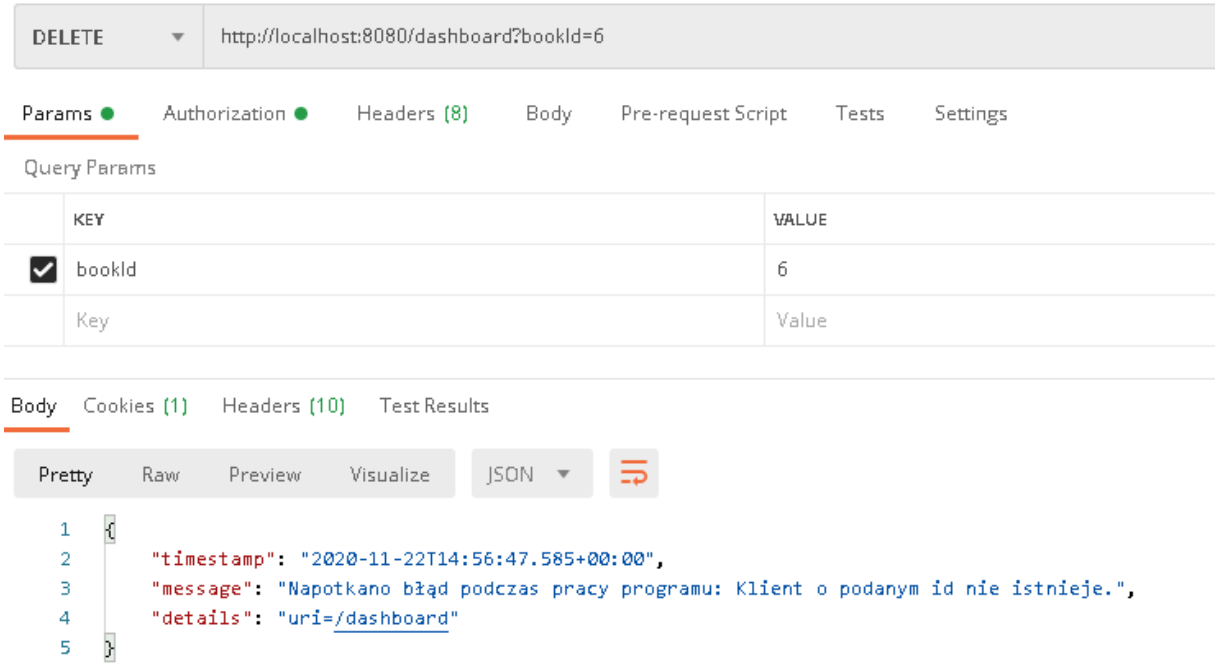
```
@ControllerAdvice
@RestController
public class GlobalDefaultExceptionHandler {

    @ExceptionHandler(Exception.class)
    public final ResponseEntity<ExceptionResponse>
handleAllExceptions(Exception ex, WebRequest request) {
        String message = "Napotkano błąd podczas pracy programu";
        if (ex.getMessage() != null) {
            message += ": " + ex.getMessage();
        }
        ExceptionResponse exceptionResponse = new ExceptionResponse(new
Date(), message, request.getDescription(false));
        return new ResponseEntity<>(exceptionResponse,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Podczas wystąpienia błędu np. przy próbie usunięcie rekordu, który nie istnieje zrzucany jest błąd. Następnie system przechwytuje i formatuje odpowiedź tego błędu.

W tym celu stworzono klasę :

```
public class ExceptionResponse {
    private Date timestamp;
    private String message;
    private String details;

    public ExceptionResponse(Date timestamp, String message, String
details) {
        super();
        this.timestamp = timestamp;
        this.message = message;
        this.details = details;
    }
```

Przykład:



Przykład 2.

Próba dodania drugi raz tego samego obiektu.