SPRAWOZDANIE

Imię Nazwisko: Klaudia Kromołowska

Temat zajęć laboratoryjnych: Implementacja API w oparciu o wzorzec architektoniczny REST w języku JAVA

1. Zwięzły opis aplikacji realizowanej podczas zajęć laboratoryjnych.

W ramach zajęć przygotowano aplikację udostepniającą prosty interfejs REST w oparciu o Spring Boost i język JAVA

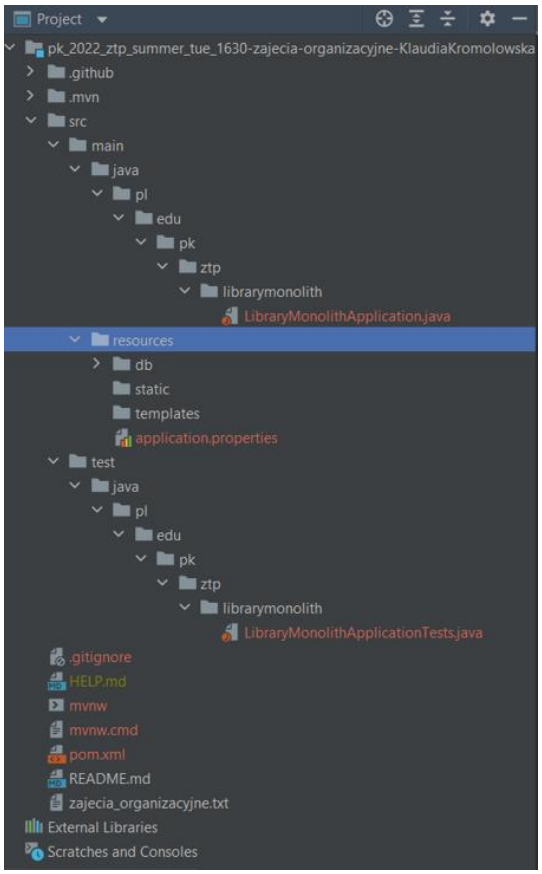
2. Użyte narzędzia.

(język programowania, najistotniejsze biblioteki, moduły baza danych itp. Nie opisujemy, tylko wypisujemy w postaci listy)

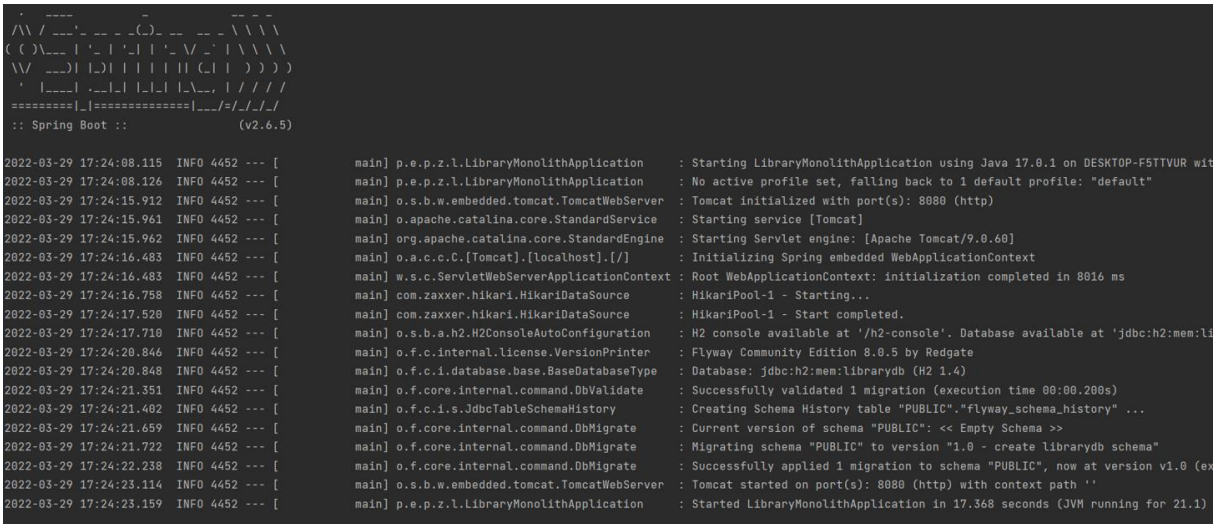
Spring, Java, Swagger, Yaml, Postman, baza danych H2, moduł Flyway

3. Sprawozdanie zgodne z plikiem README

1. Zrzut ekranu struktury projektu po zaimportowaniu do IDE



2. **Wynik uruchomienia z komentarzem wyjaśniającym poszczególne etapy inicjalizacji aplikacji oraz źródło błędu.**



Powyższy zrzut ekranu nie zawiera już błędu. Był on spowodowany brakiem konfiguracji połączenia do bazy danych.
Kolejne logi: start, informacja o braku czytania profili (używamy np. gdy środowisko testowe i to na którym pracujemy wymagają różnych baz danych) – w projekcie jest on domyślny. Następnie uruchamiany jest serwer Tomcat, który obsługuje zapytania restowe. Później inicjalizowana jest konfiguracja aplikacji webowej. Kolejno występuje połączenie do bazy danych (właśnie tu pojawiał się błąd). Za pomocą Flyway tworzony jest schemat bazy danych, po czym aplikacja zostaje uruchomiona na porcie 8080.

3. **Wyjaśnienie znaczenia poszczególnych opcji konfiguracyjnych**

```
spring.datasource.url=jdbc:h2:mem:librarydb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=admin
spring.datasource.password=password
spring.h2.console.enabled=true
```

Są to opcje konfiguracji bazy danych: url zawiera nazwę, driver odpowiedni sterownik (umożliwia tłumaczenie SQL na komunikaty zrozumiałe przez bazę danych), kolejne wiersze to nazwa użytkownika i hasło. Ostatni wiersz pozwala na dostęp do konsoli przeglądarkowej H2.

4. **Wyjaśnienie celu i działania adnotacji @Repository oraz @Autowired**
`@Repository` (z frameworku Spring) pozwala oznaczyć klasę aplikacji jako element umożliwiający dostęp do bazy danych
`@Autowired` - wstrzykiwanie zależności – zamiast ręcznie tworzyć obiekty i przekazywać konfigurację, ta adnotacja pozwala na automatyczną konfigurację w oparciu o application.properities.

5. **`UserController` i `BookController` – elementy konfiguracji zgodnie z dokumentacją oraz realizacja obsługi błędów**

W obu wspominanych klasach zdefiniowane są metody, które będą pozwalały na wykonywanie różnych akcji na aplikacji.

`@RestController` – informacja o reprezentacji kontrolera, umożliwiającego wykonanie akcji.

USER CONTROLLER:

```
@GetMapping
List<UserDTO> getAllUsers()
```

Zwraca wszystkich użytkowników z bazy danych

```
@GetMapping("/{id}")
UserDTO getUserById(final Integer userID)
```

Zwraca informacje o użytkowniku znajdujące się w rekordzie o podanym ID. W przypadku braku wskazanego ID w bazie danych zwraca błąd 404

```
@DeleteMapping("/{id}")
void deleteUser(final Integer userID)
```

Usuwa użytkownika o podanym numerze ID. Jeżeli nie ma takiego użytkownika zwraca błąd 404, jeżeli z jakiegoś powodu nie da się go usunąć, błąd 403

```
@PostMapping
UserDTO postUser(final UserDTO user)
```

Tworzy nowego użytkownika w bazie danych.

BOOKS CONTROLLER:

```
@GetMapping
List<BookDTO> getAllBooks(final boolean showOnlyAvailable)
```

Zwraca informacje o wszystkich książkach znajdujących się w bazie danych

```
@GetMapping("/{id}")
BookDTO getBookRentals(final Integer bookID)
```

Zwraca informację o książce o podanym numerze ID uwzględniając listę jej wypożyczeń. W przypadku, kiedy nie ma w bazie danych książki o podanym ID zwraca błąd 404.

```
@PatchMapping("/return/{id}")
BookDTO patchRentBook(final Integer bookID, final Integer userID)
```

Usuwa informację o wypożyczeniu książki o podanym ID (następuje „zwrot"). Jeżeli nie podano numeru ID użytkownika lub nie ma go w bazie – błąd 401. Jeżeli nie można zwrócić książki (np. nie było wypożyczonej o tym numerze ID) – 409.

```
@PatchMapping("/rent/{id}")
BookDTO patchReturnBook(final Integer bookID, final Integer userID)
```

Dodaje informację o wypożyczeniu książki o podanym ID. Jeżeli nie podano numeru ID użytkownika lub nie ma go w bazie – błąd 401. Jeżeli wskazana książka nie jest dostępna lub nie można jej wypożyczyć – błąd 409.

6. BookRepository – opis publicznych metod

```
List<BookDTO> findAll
```

W zależności od podanego parametru funkcja zwróci wszystkie książki lub jedynie je które są dostępne do wypożyczenia.

```
boolean isBookAvailable
```

Funkcja sprawdza czy jest możliwe wypożyczenie książki – czy ilość książek wypożyczonych o podanym numerze ID jest mniejsza od ilości książek o wskazanym ID w całej bazie.

`BookDTO findBookById`

Zwraca informacje o książce o podanym numerze ID z tabeli tbl_books. W przypadku, kiedy nie występuje książka o takim ID, zwracany jest błąd 400.

`List<BookRentalDTO> findRentalByBookId`

Zwraca informacje o wypożyczeniach książki o podanym numerze ID

`public boolean returnBook`

Sprawdza czy możliwe jest wykonanie operacji zwracania książki

`public boolean rentBook`

Sprawdza czy możliwe jest wypożyczenie książki

7. Wyniki testów: