LABORATORIUM

ZAAWANSOWANE PROGRAMOWANIE OBIEKTOWE

REST

---

1. Wstęp

Ćwiczenie polega na wykorzystaniu REST endpoint za pomocą, których możliwa jest obsługa żądań http oraz wykonywanie poleceń zrozumiałych dla baz danych MySQL/H2 wykorzystując interfejs JPA (Java Persistance API).

2. Przebieg ćwiczenia

Zapoznaj się z załącznikami do ćwiczenia w których znajdują się fragmenty aplikacji dotyczącej użytkownika:

- User encja reprezentująca Użytkownika
- UserRepository repozytorium realizujące dostęp do danych
- UserService klasa usługowa realizująca logikę biznesową
- UserController kontroler udostępniający końcówki RESTowe
- UserMapper logika umożliwiająca translację pomiędzy encją a klasami DTO (Data Transfer Object)
- CreateUserDTO klasa z danymi potrzebnymi do utworzenia Użytkownika
- UpdateUserDTO klasa z danymi potrzebnymi do modyfikacji Użytkownika
- UserDTO klasa reprezentująca Użytkownika

3. Zadania do wykonania

Na podstawie dostarczonego kodu, rozbuduj aplikację tak, by Użytkownik posiadał Zadania

a. Dodaj encję Task posiadającą dane: (5 pkt.)
   i. identyfikator, tytuł, datę utworzenia, typ oraz status
   ii. typy Zadania: BUG, TASK, FEATURE; możesz rozszerzyć o własne wartości
   iii. statusy Zadania: NEW, ACTIVE, DONE; możesz rozszerzyć o własne wartości
   iv. każde Zadanie należy do jednego Użytkownika
   v. Użytkownik ma wiele Zadań
b. Zaprojektuj relację pomiędzy Użytkownikiem i Zadaniami (2 pkt.)
   i. Zdecyduj, czy relacja będzie jedno, czy dwukierunkowa
c. Dodaj pozostałe komponenty (repozytorium, serwis, kontroler, mapper, klasy DTO) (3 pkt.)
d. Wymagania dotyczące klas DTO (3 pkt.)
   i. pojedyncze Zadanie powinno prezentować informacje o właścicielu (Użytkowniku)
   ii. aby utworzyć Zadanie należy podać identyfikator właściciela (Użytkownika)

iii. przy edycji Zadania nie można zmienić właściciela - posłuży do tego osobna metoda
iv. przy zwracaniu pojedynczego Użytkownika nie ma informacji o jego Zadaniach

e. Wymagania dotyczące kontrolera dla Zadania (5 pkt.)
i. tworzenie nowego Zadania dla podanego Użytkownika
ii. pobieranie Zadania po identyfikatorze
iii. pobieranie wszystkich Zadań danego Użytkownika
iv. pobieranie wszystkich Zadań o podanym statusie danego Użytkownika
v. pobieranie wszystkich Zadań o podanym type danego Użytkownika
vi. pobieranie wszystkich Zadań o podanym statusie i typie danego Użytkownika
vii. edycja danego Zadania
viii. usuwanie danego Zadania
ix. zmiana właściciela danego Zadania

f. Zaproponuj dwie własne dodatkowe metody (2 pkt.)

4. Sprawozdanie

a. Zwięzły opis wykorzystanych technik programistycznych
b. Link do repozytorium GitHub z rozwiązaniami zadań

Liczba punktów do zdobycia: 20 pkt

User.java

```java
@Entity
@Table(name = "USER")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String name;


    public long getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setId(long id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }


    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }

}
```

UserController.java

```
@RestController
public class UserController {

    @Autowired
    private UserService userService;

    @Autowired
    private UserMapper userMapper;

    @GetMapping("/users")
    public List<UserDTO> getUsers() {
        List<User> users = userService.getUsers();

        return userMapper.toDTOs(users);
    }

    @GetMapping("/users/{userId}")
    public UserDTO getUser(@PathVariable long userId) {
        User user = userService.getUser(userId);

        return userMapper.toDTO(user);
    }

    @PostMapping("/users")
    public void createUser(CreateUserDTO userDTO) {
        User user = userMapper.fromDTO(userDTO);

        userService.createUser(user);
    }

    @PutMapping("/users/{userId}")
    public void updateUser(UpdateUserDTO userDTO, @PathVariable long
userId) {
        User userFromDb = userService.getUser(userId);

        if (userFromDb != null) {
            User user = userMapper.fromDTO(userDTO);
            user.setId(userId);
            userService.updateUser(user);
        }
    }

    @DeleteMapping("/users/{userId}")
    public void deleteUser(@PathVariable long userId) {
        userService.deleteUser(userId);
    }
}
```

UserMapper.java

```java
@Component
public class UserMapper {

    public UserDTO toDTO(User user) {
        return new UserDTO(user.getId(), user.getName());
    }

    public List<UserDTO> toDTOs(List<User> users) {
        return users.stream()
                .map(user -> toDTO(user))
                .collect(Collectors.toList());
    }

    public User fromDTO(CreateUserDTO userDTO) {
        User user = new User();
        user.setName(userDTO.getName());
        return user;
    }

    public User fromDTO(UpdateUserDTO userDTO) {
        User user = new User();
        user.setId(userDTO.getId());
        user.setName(userDTO.getName());
        return user;
    }
}
```

UserService.java

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public List<User> getUsers() {
        return userRepository.findAll(Sort.by(Sort.Order.desc("id")));
    }

    public User getUser(long id) {
        return userRepository.getOne(id);
    }

    public void createUser(User user) {
        userRepository.save(user);
    }

    public void updateUser(User user) {
        userRepository.save(user);
    }

    public void deleteUser(long id) {
        userRepository.deleteById(id);
    }
}
```

UserRepository.java

```java
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```