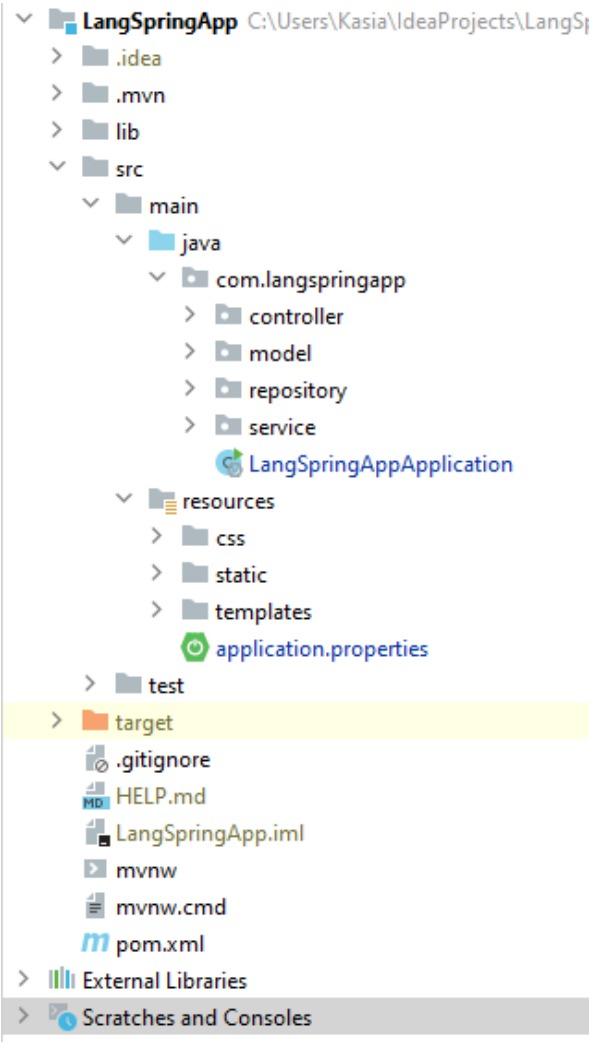
Dokumentacja projektu z przedmiotu *Programowanie aplikacji wielowarstwowych Java EE - technologie Hibernate i Spring*

Katarzyna Pieczonka
nr albumu 38760

semestr VI - letni
rok akademicki 2022/23

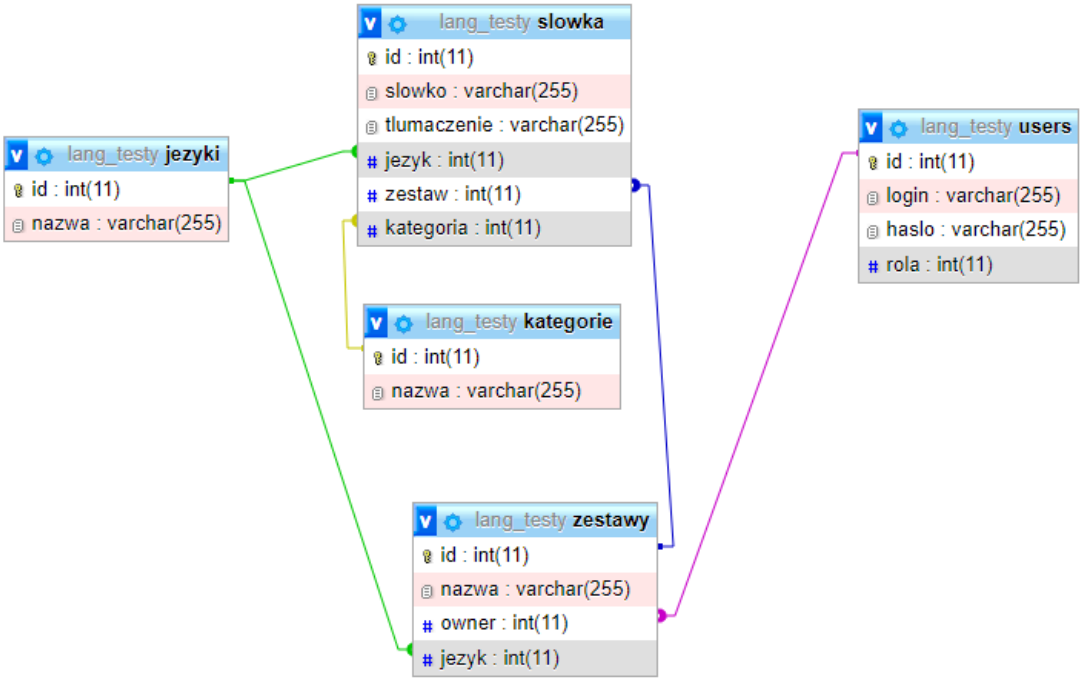
Tytuł projektu: Aplikacja internetowa do nauki języków obcych w języku Java z wykorzystaniem frameworka Spring.

1. Struktura projektu



2. Baza danych

Projekt korzysta z bazy danych MySQL o nazwie `lang_testy` o następującej strukturze:



Konfiguracja połączenia z bazą danych znajduje się w pliku `application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3306/lang_testy
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.database-platform = org.hibernate.dialect.MySQL8Dialect
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.show_sql=true
logging.level.org.hibernate=debug
spring.mvc.view.prefix=/
```

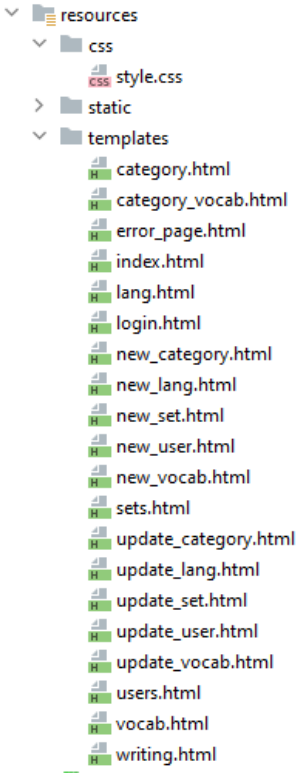
Znajdują się tutaj takie ustawienia, jak adres serwera, na którym znajduje się baza danych (tutaj localhost), dane użytkownika, który uwierzytelnia się do bazy danych, czy sterownik do połączenia z bazą danych.

3. Plik pom.xml

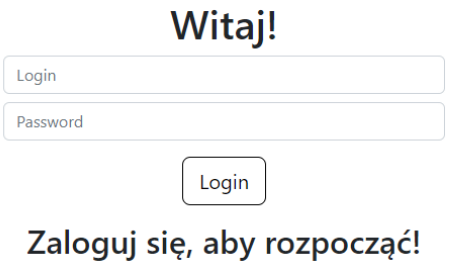
Plik ten odpowiada za implementację wszystkich bibliotek i wtyczek potrzebnych do uruchomienia w projekcie frameworka Spring, obsługi bazy danych za pomocą frameworka Hibernate, a także za podstawowe ustawienia aplikacji takie jak wersja Javy, czy nazwa aplikacji.

4. Pliki html

Za stronę wizualną aplikacji odpowiadają pliki html oraz css. Zlokalizowane są one w katalogu resources.



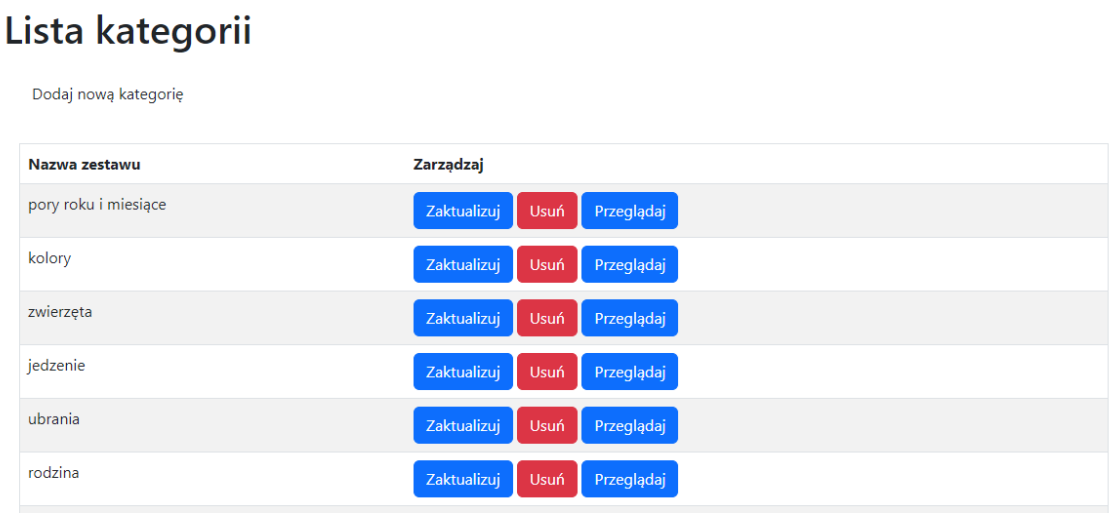
Plikiem odpowiadającym za wygląd strony głównej aplikacji jest plik `index.html`. Z niego użytkownik może wybrać podstrony.
Po uruchomieniu jako pierwsza wyświetla się jednak strona logowania (ścieżka …/login), czyli plik `login.html`, gdzie użytkownik musi wpisać dane logowania, w celu uwierzytelnienia i uzyskania dostępu do pozostałych stron.



W przypadku wystąpienia błędu logowania lub wpisania adresu podstrony, która wymaga uwierzytelnienia, zostanie wyświetlona strona błędu `error_page.html`.

Zgodnie z ideą odwzorowania obiektowo - relacyjnego dla każdej istniejącej w bazie danych tabeli, istnieje także klasa języka Java (Języki, Kategorie, Slowka, Users, Zestawy). Dla każdej istniejącej w projekcie klasy istnieją trzy pliki html: dotyczące aktualizowania (np. `update_category.html`), dodawania nowego obiektu danej klasy (np. `new_category.html`) oraz wyświetlania wszystkich obiektów tej klasy (np. `category.html`)

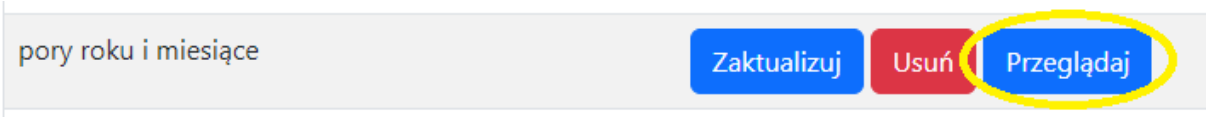
Wygląd strony `category.html`:



Podstrona do aktualizacji rekordów z bazy danych (`update_category.html`):

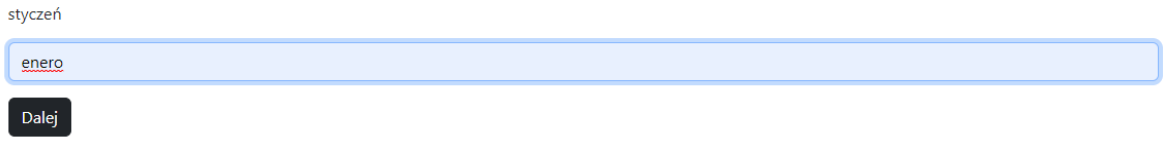
Podstrona do tworzenia nowego rekordu w bazie danych (`new_category.html`):

Aby przeglądać bazę z słówkami (plik `category_vocab.html`) należy na podstronie z kategoriami wybrać przycisk ,,Przeglądaj" przy odpowiedniej kategorii.



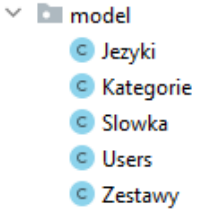
Dostępna jest także opcja ćwiczenia pisowni słówek. W celu jej wybrania należy po wybraniu listy słówek z danej kategorii kliknąć przycisk ,,Ćwicz pisanie słówek”.

Opcja ćwiczenia pisowni:



Po kliknięciu przycisku dalej, program przejdzie do następnego słówka, wyświetlając użytkownikowi informację, czy udzielona odpowiedź jest poprawna, a jeśli nie, to jak wygląda poprawna pisownia danego słówka.

5. Modele danych



Zgodnie z ideą odwzorowania obiektowo - relacyjnego dla każdej istniejącej w bazie danych tabeli, istnieje także klasa języka Java (Języki, Kategorie, Slowka, Users, Zestawy).
Klasy te są opisane w plikach zawartych w folderze src/main/java/com/langspringapp/model.

Przykładowa klasa `Kategorie.java`:

```java
@Entity
@Table(name="kategorie")
public class Kategorie {
    5 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private int id;
    5 usages
    @Basic
    @Column(name = "nazwa")
    private String nazwa;

    Katarzyna Pieczonka
    public int getId() { return id; }

    Katarzyna Pieczonka
    public void setId(int id) { this.id = id; }

    Katarzyna Pieczonka
    public String getNazwa() { return nazwa; }

    Katarzyna Pieczonka
    public void setNazwa(String nazwa) { this.nazwa = nazwa; }
```

Każda z klas, oprócz pól i odpowiednich adnotacji umożliwiających mapowanie obiektowo-relacyjne przez Hibernate posiada także metody `get()` i `set()` dla każdego z pól.
Adnotacje w definicji klasy pozwalają na odwzorowanie klasy na tabelę w bazie danych, lub odwrotnie, dlatego oznaczona jest w klasie wartość generowana automatycznie, a pola klasy opisane są nazwami kolumn z tabeli.

Klasa `Zestawy.java`:

```
@Entity
@Table(name="zestawy")
public class Zestawy {
    5 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private int id;
    5 usages
    @Basic
    @Column(name = "nazwa")
    private String nazwa;
    5 usages
    @Basic
    @Column(name = "owner")
    private int owner;
    5 usages
    @Basic
    @Column(name = "jezyk")
    private int jezyk;
```

Klasa `Users.java`:

```
@Entity
@Table(name="users")
public class Users {
    5 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private int id;
    5 usages
    @Basic
    @Column(name = "login")
    private String login;
    5 usages
    @Basic
    @Column(name = "haslo")
    private String haslo;
    5 usages
    @Basic
    @Column(name = "rola")
    private int rola;
```

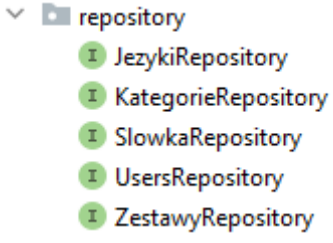
Klasa `Slowka.java`:

```
@Entity
@Table(name="users")
public class Users {
    5 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private int id;
    5 usages
    @Basic
    @Column(name = "login")
    private String login;
    5 usages
    @Basic
    @Column(name = "haslo")
    private String haslo;
    5 usages
    @Basic
    @Column(name = "rola")
    private int rola;
```

Klasa `Jezyki.java`:

```
@Entity
@Table(name="jezyki")
public class Jezyki {
    5 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    //@Column(name = "id")
    private int id;
    5 usages
    @Basic
    @Column(name = "nazwa")
    private String nazwa;
```

6. Repozytoria.



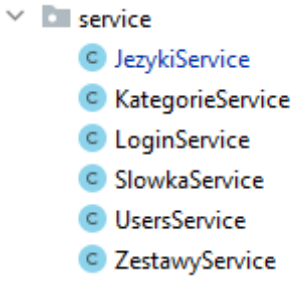
Dla każdej z klas języka Java w projekcie istnieją repozytoria, czyli specjalne klasy tworzone przy pomocy adnotacji `@Repository`. Ich implementację umożliwia framework Spring, który znacznie ułatwia operacje CRUD na bazie danych, oferując gotowe funkcje właśnie do podstawowych operacji na tabelach. W tym przypadku jest to JpaRepository.

```
package com.langspringapp.repository;

import com.langspringapp.model.Jezyki;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

6 usages  Katarzyna Pieczonka
@Repository
public interface JezykiRepository extends JpaRepository<Jezyki, Integer> {
}
```

7. Services



Również dla każdej z klas istnieje klasa Service, tworzona przy pomocy adnotacji `@Service`, pochodzącej ze Springa.
Klasy te mają za zadanie wprowadzenie dodatkowych funkcji pozwalających na łatwiejsze operacje na danych z repozytoriów, które są tutaj wstrzykiwane za pomocą adnotacji `@Autowired`.

```
@Service
public class JezykiService {

    4 usages
    @Autowired
    private JezykiRepository jezykiRepository;

    Katarzyna Pieczonka
    public List<Jezyki> listAll() { return (List<Jezyki>) jezykiRepository.findAll(); }
```

Dla każdej z klas istnieje więc funkcja pozwalająca wyświetlić wszystkie dane z tabeli w bazie danych (powyższa funkcja `listAll()`), odnaleźć rekord o danym id, a także usunąć

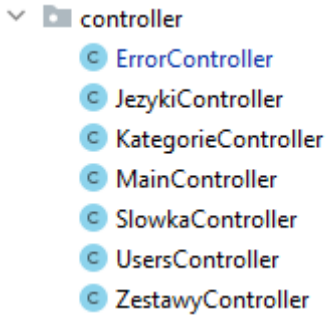
rekord o id podanym przez użytkownika (`getLangById()` i `deleteLangById()`). Istnieje także funkcja do zapisywania do bazy nowo utworzonego obiektu danej klasy (`save()`).

```
    Katarzyna Pieczonka
    public void save(Jezyki jezyki) { jezykiRepository.save(jezyki); }

    1 usage  Katarzyna Pieczonka
    public Jezyki getLangById(Integer id) {
        return jezykiRepository.findById(id).get();
    }

    1 usage  Katarzyna Pieczonka
    public void deleteLangById(Integer id) { jezykiRepository.deleteById(id); }
}
```

8. Kontrolery



Kontrolery to klasy odpowiadające za wywoływanie funkcji i obsługę danych w zależności od tego, jaki adres zostanie wywołany w przeglądarce.
Dla każdej klasy istnieje osobny kontroler, oprócz tego kontroler `MainController.java` odpowiedzialny jest za obsługę logowania i uwierzytelniania użytkowników, a `ErrorController.java` za obsługę błędów.

Plik `ErrorController.java`:

```
@Controller
public class ErrorController implements org.springframework.boot.web.servlet.error.ErrorController {

    Katarzyna Pieczonka
    @RequestMapping("/error")
    public String errorPage(HttpServletRequest request)
    {
        return "error_page";
    }

}
```

Plik `MainController.java`:

```java
@Controller
@RequiredArgsConstructor
public class MainController {
    1 usage
    private final LoginService loginService;
    1 usage
    @Autowired
    UsersRepository usersRepository;

    Katarzyna Pieczonka
    @GetMapping("/welcome")
    private String viewHomePage(Model model) { return "login"; }

    Katarzyna Pieczonka
    @GetMapping("/")
    private String viewIndex(Model model) {return "index";}

    Katarzyna Pieczonka
    @PostMapping("/check_user")
    private String check_user(Users user, HttpSession session, Model model) {...}

    Katarzyna Pieczonka
    @GetMapping("/login")
    public String login(HttpSession session, Model model){...}

    Katarzyna Pieczonka
    @GetMapping("/logout")
```

Każdy z tych kontrolerów posiada adnotacje Springa `@Controller`, oraz wykorzystuje repozytoria poprzez mechanizm wstrzykiwania.

Każda z funkcji posiada adnotacje mapujące `@GetMapping` albo `@PostMapping` wraz ze ścieżką. Kiedy w aplikacji zostanie wywołana odpowiednia ścieżka, na serwer zostaje wysyłane żądanie HTTP i uruchamiana jest przez Springa przypisana do ścieżki metoda w kontrolerze.

Przykładowy kontroler `UsersController.java`:

```java
@Controller
public class UsersController {
    4 usages
    @Autowired(required=true)
    private UsersService usersService;

    Katarzyna Pieczonka
    @GetMapping("/users")
    public String viewUsersPage(Model model) {...}

    Katarzyna Pieczonka
    @GetMapping("/users/add")
    public String showNewUserForm(Model model) {...}

    Katarzyna Pieczonka +1
    @PostMapping("/users/save")
    public String saveUser(@ModelAttribute("jezyk") Users user) {...}

    Katarzyna Pieczonka
    @GetMapping("/users/update/{id}")
    public String showFormForUpdate(@PathVariable(value="id") int id, Model model) {...}

    Katarzyna Pieczonka +1
    @GetMapping("/users/delete/{id}")
    public String deleteUser(@PathVariable (value = "id") int id) {...}
}
```

9. Logowanie do aplikacji

Za uwierzytelnianie użytkownika odpowiedzialny jest kontroler `MainController.java`, a także `LoginService.java`, który sprawdza, czy użytkownik o podanych danych istnieje w bazie, a potem ustawia odpowiednie atrybuty dla sesji:

```java
@Service
public class LoginService {

    1 usage  Katarzyna Pieczonka
    public String exisingUser(Users user, HttpSession session, Model model) {
        if(user != null) {
            session.setAttribute( s: "user", user);
            return "redirect:/";
        }
        else {
            model.addAttribute( attributeName: "login_error",  attributeValue: "true");
            return "login";
        }
    }
}
```

Jeśli użytkownik nie istnieje, wyświetlany jest błąd, a jeśli istnieje, jego dane przypisywane są do sesji.

10. Klasa główna programu

Znajduje się ona w pliku `LangSpringAppApplication.java`.

```java
package com.langspringapp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

1 usage  Katarzyna Pieczonka
@SpringBootApplication
public class LangSpringAppApplication {

    Katarzyna Pieczonka
    public static void main(String[] args) {
        SpringApplication.run(LangSpringAppApplication.class, args);
    }

}
```