

APLIKACJA WEBOWA DO ZARZĄDZANIA PROJEKTAMI
(REALIZACJA FRONT-ENDU KORZYSTAJĄCEGO Z USŁUGI TYPU REST)

UWAGA! Aplikacja webowa będzie korzystać z usługi typu REST zaimplementowanej w ramach poprzedniego zadania.

Lista projektów

localhost:8081/projektList

Lista projektów

[Lista Studentów](#) [Lista zadań](#)
[Dodaj projekt](#)

Id	Nazwa	Opis	Utworzony	Zmodyfikowany	Data oddania	Edit
203	Projekt testowy 1	Opis projektu testowego 1	2025-03-01 14:34:23	2025-04-09 14:34:23	2025-06-30	Edytuj Usuń
252	Projekt testowy 2	Opis projektu testowego 2	2025-03-06 16:48:36	2025-04-09 16:48:36	2025-06-30	Edytuj Usuń
253	Projekt testowy 3	Opis projektu testowego 3	2025-03-05 16:49:31	2025-04-19 16:49:31	2025-06-30	Edytuj Usuń
254	Projekt testowy 4	Opis projektu testowego 4	2025-03-06 16:49:48	2025-04-11 16:49:48	2025-06-30	Edytuj Usuń
255	Projekt testowy 5	Opis projektu testowego 5	2025-04-02 16:54:02	2025-04-02 16:54:02	2025-06-30	Edytuj Usuń

Edycja projektu

localhost:8081/...

Aktualizuj projekt

AktualizujAnuluj

Id:

203

Nazwa:

Projekt testowy 1

Opis:

Opis projektu testowego 1

Data oddania:

2025-06-30(RRRR-MM-DD)

Utworzony:

2025-03-01 14:34:23.971

Zmodyfikowany:

2025-04-09 14:34:23.971

1. Stwórz projekt aplikacji webowej np. *project-web-app*.

Korzystając ze strony <https://start.spring.io> możesz wygenerować szkielet projektu, ale łatwiej wykorzystać już istniejący, zrealizowany w poprzednim zadaniu tj. *project-rest-api*. W tym celu należy w widoku *Project Explorer* Eclipse'a zaznaczyć główną ikonkę utworzonego wcześniej projektu *project-rest-api*, następnie skopiować projekt wciskając skrót *CTRL + C* i wkleić za pomocą *CTRL + V*. W nowo otwartym okienku należy wpisać nazwę tworzonego projektu tj. *project-web-app*. Na koniec trzeba jeszcze edytować plik *project-web-app\settings.gradle*, zmienić *rootProject.name* na '*project-web-app*' i zapisać zmiany. Jeżeli skorzystasz z tej opcji w kolejnych podpunktach będziesz musiał czasami zmodyfikować lub usunąć kod źródłowy, nawet jeżeli ich opis będzie dotyczył utworzenia nowego rozwiązania.

Edytuj plik *build.gradle* i zmodyfikuj jego elementy, aby odpowiadał poniższej zawartości. Następnie kliknij prawym przyciskiem myszki na głównej ikonce projektu i wybierz *Gradle -> Refresh Gradle Project*.

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.4.3'
    id 'io.spring.dependency-management' version '1.1.7'
}

group = 'com.project'
version = '1.0'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(21)
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.data:spring-data-commons'
    implementation 'com.fasterxml.jackson.datatype:jackson-datatype-jsr310'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
}

compileJava {
    options.encoding = 'UTF-8'
}

compileTestJava {
    options.encoding = 'UTF-8'
}

tasks.named('test') {
    useJUnitPlatform()
    testLogging {
        showStandardStreams = true //ustawia drukowanie komunikatów w konsoli
    }
}
```

2. Przekopiuje pakiet *com.project.model* utworzony w ramach poprzedniego ćwiczenia. W każdej klasie modelu kasujemy wszystkie adnotacje (z *jakarta.persistence.** i *org.hibernate.annotations.**) oraz pozostałe po ich usunięciu zbędne importy (nie trzeba usuwać poszczególnych importów, można użyć skrótu *CTRL + SHIFT + O*, który uporządkuje całą sekcję importów). Następnie w każdej klasie modelu, przed jej nazwą, dodajemy nową adnotację *@JsonIgnoreProperties*, dzięki której podczas tworzenia obiektów ignorowane będą właściwości komunikatów JSON-a niewchodzące w skład mapowanych klas. Poza tym można dodać adnotację *@DateTimeFormat*, określając format prezentowanych dat i czasu.

```
package com.project.model;
...
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import org.springframework.format.annotation.DateTimeFormat;

@JsonIgnoreProperties(ignoreUnknown = true)
public class Projekt {

    ...

    @DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
    private LocalDateTime dataCzasUtworzenia;

    @DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
    private LocalDateTime dataCzasModyfikacji;

    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate dataOddania;
}
```

Uwaga! Podczas tworzenia np. klas modelu/transferowych (DTO – *Data Transfer Object*) i klas encyjnych można korzystać z adnotacji **Lomboka**, dzięki którym zostaną automatycznie wygenerowane m.in. akcesory, mutatory (tzw. gettery i settery) i konstruktory (w tym przypadku nie ma zatem potrzeby ich generowania za pomocą Eclipse'a). Przykładowo – klasa *Zadanie* korzystająca z tych adnotacji:

```
package com.project.model;
import java.time.LocalDateTime;
import org.springframework.data.annotation.Id;
import org.springframework.format.annotation.DateTimeFormat;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder // Dodatkowa adnotacja generująca implementację wzorca projektowego Builder. Dzięki niej można tworzyć obiekty
// korzystając np. z Zadanie.builder().nazwa("Nazwa testowa").opis("Opis testowy").kolejnosc(1).build();

@AllArgsConstructor
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class Zadanie {
    @Id
    private Integer zadanieId;

    @NotNull(message = "Pole nazwa nie może być puste!")
    @Size(min = 3, max = 50, message = "Nazwa musi zawierać od {min} do {max} znaków!")
    private String nazwa;

    private Integer kolejnosc;

    @NotNull(message = "Pole opis nie może być puste!")
    @Size(max = 1000, message = "Pole opis może zawierać maksymalnie {max} znaków!")
    private String opis;

    @DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
    private LocalDateTime dataCzasDodania;

    @JsonIgnoreProperties({ "projekt" })
    private Projekt projekt;
}
```

3. Dodaj pakiet *com.project.service* i przekopiuj do niego wszystkie interfejsy serwisów utworzone w ramach poprzedniego ćwiczenia (z pakietu o takiej samej nazwie co nowo utworzony, patrz realizacja usługi typu REST). Następnie dodaj nowe klasy je implementujące tj. *ProjektServiceImpl*, *ZadanieServiceImpl* i *StudentServiceImpl*, a za pomocą środowiska Eclipse wygeneruj szkielety metod wymagających implementacji.

4. Dodajemy pakiet *com.project.config* i tworzymy w nim klasę *SecurityConfig* z poniższą zawartością. Jej zadaniem jest utworzenie obiektu klasy *RestClient* (jest to specjalna springowa klasa przeznaczona do komunikacji z REST API) oraz ustawienie loginu i hasła wykorzystywanego w uwierzytelnianiu typu *Basic Authentication*, zastosowanym w zaimplementowanej w poprzednim ćwiczeniu usłudze. Utworzony obiekt będzie mógł być wstrzykiwany w dowolnej klasie naszego projektu np. poprzez konstruktor.

```
package com.project.config;

import java.util.Base64;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpHeaders;
import org.springframework.web.client.RestClient;

@Configuration
public class SecurityConfig {
    @Value("${rest.base.url}")
    private String restBaseUrl;    // adres wstrzykiwany z pliku application.properties

    @Value("${rest.user.name}")
    private String restUserName;    // nazwa użytkownika wstrzykiwana z pliku application.properties

    @Value("${rest.user.password}")
    private String restUserPassword; // hasło użytkownika wstrzykiwane z pliku application.properties

    // dzięki adnotacji @Bean Spring uruchomi metodę i zarejestruje w kontenerze obiekt przez nią zwrócony,
    // natomiast adnotacja @Autowired i/lub konstruktor użyte w innej klasie spowodują jego wstrzyknięcie
    @Bean
    public RestClient customRestClient() {
        return RestClient.builder().baseUrl(restBaseUrl)
            .defaultHeader(HttpHeaders.AUTHORIZATION,
                getBasicAuthenticationHeader(restUserName, restUserPassword))
            .build();
    }

    private String getBasicAuthenticationHeader(String username, String password) {
        return "Basic " + Base64.getEncoder().encodeToString((username + ":" + password).getBytes());
    }
}
```

5. W katalogu *project-web-app\src\main\resources* utwórz plik tekstowy *application.properties*. Następnie edytuj go i zdefiniuj port serwera naszej aplikacji webowej. Dodaj też parametry z adresem serwera usługi REST oraz loginu i hasła wykorzystywanego w uwierzytelnianiu typu *Basic Authentication* tj.

```
server.port=8081
rest.base.url=http://localhost:8080
rest.user.name=admin
rest.user.password=admin
```

Spring uwzględni wartość parametru *server.port* i uruchamia aplikację na wskazanym porcie. Poza tym w każdej klasie można pobierać wartości z tego pliku konfiguracyjnego za pomocą adnotacji *@Value* (z pakietu *org.springframework.beans.factory.annotation.**), w której wskazuje się odpowiednią nazwę parametru np.

```
@Value("${rest.base.url}")
private String restBaseUrl;
```

6. W pakiecie *com.project.service* utwórz klasę pomocniczą *RestResponsePage*, będącą uniwersalnym szablonem służącym przekształcaniu komunikatów JSON-owych o specjalnej strukturze stosowanej przy stronicowaniu danych (przykład w punkcie 3.6). Zwracane dane modelu są konwertowane na obiekty i umieszczane w liście *content*, natomiast pozostałe zmienne przechowują parametry stronicowania i sortowania.

```

package com.project.service;

import java.util.ArrayList;
import java.util.List;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.JsonNode;

@JsonIgnoreProperties(ignoreUnknown = true)
public class RestResponsePage<T> extends PageImpl<T> {
    private static final long serialVersionUID = 1L;

    @JsonCreator(mode = JsonCreator.Mode.PROPERTIES)
    public RestResponsePage(@JsonProperty("content") List<T> content,
        @JsonProperty("number") int number,
        @JsonProperty("size") int size,
        @JsonProperty("totalElements") Long totalElements,
        @JsonProperty("pageable") JsonNode pageable,
        @JsonProperty("last") boolean last,
        @JsonProperty("totalPages") int totalPages,
        @JsonProperty("sort") JsonNode sort,
        @JsonProperty("first") boolean first,
        @JsonProperty("numberOfElements") int numberOfElements) {
        super(content, PageRequest.of(number, size), totalElements);
    }

    public RestResponsePage(List<T> content, Pageable pageable, long total) {
        super(content, pageable, total);
    }

    public RestResponsePage(List<T> content) {
        super(content);
    }

    public RestResponsePage() {
        super(new ArrayList<>());
    }
}

```

7. W pakiecie *com.project.service* umieścimy jeszcze jedną klasę pomocniczą – *ServiceUtil*, z metodą generyczną *getPage* wysyłającą żądania typu GET i zwracająca obiekt powyższej klasy *RestResponsePage* (utworzony na podstawie JSON-owej odpowiedzi serwera). Pozostałe metody pomocnicze są wykorzystywane przy budowaniu adresów do zasobów z zadanymi parametrami stronicowania, sortowania i wyszukiwania.

```

package com.project.service;

import java.net.URI;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.web.client.RestClient;
import org.springframework.web.util.UriComponentsBuilder;

public class ServiceUtil {

    public static <T> RestResponsePage<T> getPage(URI uri, RestClient restClient,
        ParameterizedTypeReference<RestResponsePage<T>> responseType) {
        return restClient.get()
            .uri(uri)
            .retrieve()
            .body(responseType);
    }
}

```

```

    public static URI getURI(String resourcePath, Pageable pageable) {
        return getUriComponent(resourcePath)
            .queryParams("page", pageable.getPageNumber())
            .queryParams("size", pageable.getPageSize())
            .queryParams("sort",
                ServiceUtil.getSortParams(pageable.getSort()).build().toUri());
    }

    public static UriComponentsBuilder getUriComponent(String resourcePath, Pageable pageable) {
        return getUriComponent(resourcePath)
            .queryParams("page", pageable.getPageNumber())
            .queryParams("size", pageable.getPageSize())
            .queryParams("sort", ServiceUtil.getSortParams(pageable.getSort()));
    }

    public static UriComponentsBuilder getUriComponent(String resourcePath) {
        return UriComponentsBuilder.fromUriString(resourcePath);
    }

    public static String getSortParams(Sort sort) {
        StringBuilder builder = new StringBuilder();
        if (sort != null) {
            String sep = "";
            for (Sort.Order order : sort) {
                builder.append(sep)
                    .append(order.getProperty())
                    .append(",")
                    .append(order.getDirection());
                sep = "&sort=";
            }
        }
        return builder.toString();
    }
}

```

8. Utwórz pakiet *com.project.exception* i umieść w nim klasę *HttpException*. Za jej pomocą będą zwracane informacje o błędach związanych z przysyłaniem żądań do back-endu.

```

package com.project.exception;

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;

public class HttpException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public HttpException(HttpStatus httpStatusCode, HttpHeaders httpHeaders) {
        super(String.format("Error: statusCode - %s, headers - %s", httpStatusCode, httpHeaders));
    }

    public HttpException(String errorMessage) {
        super(errorMessage);
    }

    public HttpException(String errorMessage, Throwable err) {
        super(errorMessage, err);
    }
}

```

9. Przykładowa klasa serwisowa korzystająca z usługi REST. Zaimplementuj pozostałe klasy – *ZadanieServiceImpl* i *StudentServiceImpl*.

```

package com.project.service;

import java.net.URI;
import java.util.Optional;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

```

```

import org.springframework.core.ParameterizedTypeReference;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestClient;
import com.project.exception.HttpException;
import com.project.model.Projekt;

@Service
public class ProjektServiceImpl implements ProjektService {

    private static final Logger logger = LoggerFactory.getLogger(ProjektServiceImpl.class);

    private final RestClient restClient; // obiekt wstrzykiwany poprzez konstruktor, dzięki adnotacjom
    // @Configuration i @Bean zawartym w klasie SecurityConfig
    // Spring utworzy wcześniej obiekt, a adnotacja @Autowired
    // tej klasy wskaże element docelowy wstrzykiwania
    // (adnotacja @Autowired może być pomijana jeżeli w klasie
    // jest tylko jeden konstruktor)

    public ProjektServiceImpl(RestClient restClient) {
        this.restClient = restClient;
    }

    private String getResourcePath() {
        return "/api/projekty";
    }

    private String getResourcePath(Integer id) {
        return String.format("%s/%d", getResourcePath(), id);
    }

    @Override
    public Optional<Projekt> getProjekt(Integer projektId) {
        String resourcePath = getResourcePath(projektId);
        logger.info("REQUEST -> GET {}", resourcePath);
        Projekt projekt = restClient
            .get()
            .uri(resourcePath) //można też używać .uri("/api/projekty/{projektId}", projektId)
            .retrieve()
            .onStatus(HttpStatus::isError, (req, res) -> {
                throw new HttpException(res.getStatusCode(), res.getHeaders());
            })
            .body(Projekt.class);
        return Optional.ofNullable(projekt);
    }

    @Override
    public Projekt setProjekt(Projekt projekt) {
        if (projekt.getProjektId() != null) { // modyfikacja istniejącego projektu
            String resourcePath = getResourcePath(projekt.getProjektId());
            logger.info("REQUEST -> PUT {}", resourcePath);
            restClient
                .put()
                .uri(resourcePath)
                .accept(MediaType.APPLICATION_JSON)
                .body(projekt)
                .retrieve()
                .onStatus(HttpStatus::isError, (req, res) -> {
                    throw new HttpException(res.getStatusCode(), res.getHeaders());
                })
                .toBodilessEntity();
            return projekt;
        } else { //utworzenie nowego projektu
            // po dodaniu projektu zwracany jest w nagłówku Location - link do utworzonego zasobu
            String resourcePath = getResourcePath();
            logger.info("REQUEST -> POST {}", resourcePath);

```



```

        ResponseEntity<Void> response = restClient
            .post()
            .uri(resourcePath)
            .accept(MediaType.APPLICATION_JSON)
            .body(projekt)
            .retrieve()
            .onStatus(HttpStatus::isError, (req, res) -> {
                throw new HttpException(res.getStatusCode(), res.getHeaders());
            })
            .toBodilessEntity();
        URI location = response.getHeaders().getLocation();
        logger.info("REQUEST (location) -> GET {}", location);
        return restClient
            .get()
            .uri(location)
            .retrieve()
            .onStatus(HttpStatus::isError, (req, res) -> {
                throw new HttpException(res.getStatusCode(), res.getHeaders());
            })
            .body(Projekt.class);
    }

}

@Override
public void deleteProjekt(Integer projektId) {
    String resourcePath = getResourcePath(projektId);
    logger.info("REQUEST -> DELETE {}", resourcePath);
    restClient
        .delete()
        .uri(resourcePath)
        .retrieve()
        .onStatus(HttpStatus::isError, (req, res) -> {
            throw new HttpException(res.getStatusCode(), res.getHeaders());
        })
        .toBodilessEntity();
}

@Override
public Page<Projekt> getProjekty(Pageable pageable) {
    URI uri = ServiceUtil.getURI(getResourcePath(), pageable);
    logger.info("REQUEST -> GET {}", uri);
    return getPage(uri);
}

@Override
public Page<Projekt> searchByNazwa(String nazwa, Pageable pageable) {
    URI uri = ServiceUtil
        .getUriComponent(getResourcePath(), pageable)
        .queryParams("nazwa", nazwa)
        .build().toUri();
    logger.info("REQUEST -> GET {}", uri);
    return getPage(uri);
}

private Page<Projekt> getPage(URI uri) {
    return restClient.get()
        .uri(uri.toString())
        .retrieve()
        .body(new ParameterizedTypeReference<RestResponsePage<Projekt>>(){});
}

}

```

10. Przykładowy, prosty kontroler aplikacji webowej.

```

package com.project.controller;

import jakarta.validation.Valid;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;

```



```

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.client.HttpStatusException;
import com.project.model.Projekt;
import com.project.service.ProjektService;

@Controller
public class ProjectController {
    private ProjektService projektService;

    // @Autowired - przy jednym konstruktorze wstrzykiwanie jest zadaniem domyślnym, adnotacji nie jest potrzebna
    public ProjectController(ProjectService projektService) {
        this.projektService = projektService;
    }

    @GetMapping("/projektList") // np. http://localhost:8081/projektList?page=0&size=10&sort=dataCzasModyfikacji,desc
    public String projektList(Model model, Pageable pageable) {
        model.addAttribute("projekty", projektService.getProjekty(pageable).getContent());
        return "projektList";
    }

    @GetMapping("/projektEdit")
    public String projektEdit(@RequestParam(name="projektId", required = false) Integer projektId, Model model){
        if(projektId != null) {
            model.addAttribute("projekt", projektService.getProjekt(projektId).get());
        } else {
            Projekt projekt = new Projekt();
            model.addAttribute("projekt", projekt);
        }
        return "projektEdit";
    }

    @PostMapping(path = "/projektEdit")
    public String projektEditSave(@ModelAttribute @Valid Projekt projekt, BindingResult bindingResult) {
        // parametr BindingResult powinien wystąpić zaraz za parametrem opatrzonym adnotacją @Valid
        if (bindingResult.hasErrors()) {
            return "projektEdit";
        }
        try {
            projekt = projektService.setProjekt(projekt);
        } catch (HttpStatusException e) {
            bindingResult.rejectValue(Strings.EMPTY, String.valueOf(e.getStatusCode().value()),
                e.getStatusCode().toString());
            return "projektEdit";
        }
        return "redirect:/projektList";
    }

    @PostMapping(params="cancel", path = "/projektEdit")
    public String projektEditCancel() {
        return "redirect:/projektList";
    }

    @PostMapping(params="delete", path = "/projektEdit")
    public String projektEditDelete(@ModelAttribute Projekt projekt) {
        projektService.deleteProjekt(projekt.getProjektId());
        return "redirect:/projektList";
    }
}

```

11. Szablon przykładowego ekranu edycji projektu w *Thymeleaf* (plik z katalogu ...\\src\\main\\resources\\templates).

```

<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<link th:href="@{/css/edit-style.css}" rel="stylesheet" />    <!-- plik z...\\resources\\static\\css\\edit-style.css -->
<title>Edycja projektu</title>
</head>

```

```

<body>
<div class="root" th:with="isDelete=${#strings.equalsIgnoreCase(param.delete, 'true')}">
  <form action="#" th:action="@{/projektEdit}" th:object="${projekt}" method="POST"
    th:with="akcja=${projektId} ? (${isDelete}? 'delete': 'update') : 'create', opis=${projektId} ?
      (${isDelete}? 'Usuń': 'Aktualizuj') : 'Utwórz'" autocomplete="off">

    <h1 th:text="${opis} + ' projekt'">Edytuj projekt</h1>

    <div class="err" th:if="${#fields.hasErrors('*')}">
      BŁĘDY:
      <ul>
        <li th:each="err : ${#fields.errors('*)}" th:text="${err}">Wprowadzone dane są niepoprawne!</li>
      </ul>
    </div>
    <div class="container">
      <div class="btns-panel">
        <input class="btn" type="submit" name="create" value="create" th:name="${akcja}" th:value="${opis}" />
        <input class="btn" type="submit" name="cancel" value="Anuluj" />
      </div>
      <div th:if="${projektId}">
        <label for="projektId" class="lbl">Id:</label>
        <input th:field="${projektId}" class="fld" readonly />
      </div>
      <div>
        <label for="nazwa" class="lbl">Nazwa:</label>
        <input th:field="${nazwa}" class="fld" th:class="${#fields.hasErrors('opis')}? 'err' : 'fld'" size="45" />
        <span class="err" th:if="${#fields.hasErrors('nazwa')}" th:errors="${nazwa}">Error</span>
      </div>
      <div>
        <label for="opis" class="lbl">Opis:</label>
        <textarea class="fld" rows="3" cols="47" th:field="${opis}">Opis</textarea>
      </div>
      <div>
        <label for="dataOddania" class="lbl">Data oddania:</label>
        <input th:field="${dataOddania}" class="fld" type="text" size="10" /><i>(RRRR-MM-DD)</i>
      </div>
      <div th:if="${dataCzasUtworzenia}">
        <label for="dataCzasUtworzenia" class="lbl">Utworzony:</label>
        <input th:field="${dataCzasUtworzenia}" class="fld" type="text" size="23" readonly />
      </div>
      <div th:if="${dataCzasModyfikacji}">
        <label for="dataCzasModyfikacji" class="lbl">Zmodyfikowany:</label>
        <input th:field="${dataCzasModyfikacji}" class="fld" type="text" size="23" readonly />
      </div>
    </div>
  </form>
</div>
</body>
</html>

```

Szablon *projektEdit.html* korzysta z pliku *resources\static\css\edit-style.css*, poniżej jego przykładowa zawartość.

```

.root {
  text-align: center;
}

.container {
  text-align: left;
  display: inline-block;
}

.fld {
  padding: 5px 5px 5px 5px;
  margin: 5px 5px 5px 5px;
}

.btn {
  padding: 5px 10px 5px 10px;
  margin: 5px 5px 5px 5px;
  cursor: pointer;
}

.btns-panel {
  border: 1px solid darkGray;
  background: lightGray;
  margin-bottom: 15px;
  text-align: center;
}

```

```
.lbl {
    display: block;
    margin: 10px 5px 0px 5px;
}

span.err, label.err {
    color: red;
}

div.err {
    background-color: #ffcccc;
    border: 2px solid red;
    text-align: left;
}

textarea.err {
    background-color: #ffcccc;
    padding: 5px 5px 5px 5px;
    margin: 5px 5px 5px 5px;
}
```

12. Po uruchomieniu front-endu oraz usługi REST (back-endu) okno edycji projektów będzie dostępne pod adresem <http://localhost:8081/projektEdit> (patrz adnotacja `@GetMapping` przed metodą `projektEdit` klasy `ProjectController`). Zwróć uwagę, że po udanym utworzeniu nowego lub modyfikacji już istniejącego projektu za pomocą przygotowanego wcześniej okna edycji następuje automatyczne przekierowanie na adres <http://localhost:8081/projektList> (patrz linijka `return "redirect:/projektList"`; w metodzie `projektEditSave` klasy `ProjectController`). W następnym kroku trzeba zatem utworzyć kolejny widok `projektList.html` do prezentowania listy wszystkich projektów i umieścić go w podkatalogu `resources\templates\`. Poniżej została przedstawiona jego przykładowa zawartość. Widok zawiera daty utworzenia i modyfikacji projektu, które będzie trzeba usunąć jeżeli Twoja wersja aplikacji ich nie uwzględnia.

```

<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">

<head>
<meta charset="UTF-8">
<link th:href="@{/css/List-style.css}" rel="stylesheet" />
<!-- plik z ...\resources\static\css\list-style.css -->
<title>Lista projektów</title>
</head>

<body>
<div class="root">
<h1>Lista projektów</h1>
<p>
<a th:href="@{/studentList}">Lista Studentów</a>
<a th:href="@{/zadanieList}">Lista zadań</a>
</p>
<p>
<a th:href="@{/projektEdit}">Dodaj projekt</a>
</p>
<table>
<thead>
<tr>
<th>Id</th>
<th>Nazwa</th>
<th>Opis</th>
<th>Utworzony</th>
<th>Zmodyfikowany</th>
<th>Data oddania</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr th:each="projekt : ${projekty}">
<td th:text="${projekt.projektId}">Id</td>
<td th:text="${projekt.nazwa}">Nazwa</td>
<td th:text="${projekt.opis}">Opis</td>
<td th:text="${#temporals.format(projekt.dataCzasUtworzenia, 'yyyy-MM-dd HH:mm:ss')}">Utworzony</td>
<td th:text="${#temporals.format(projekt.dataCzasModyfikacji, 'yyyy-MM-dd HH:mm:ss')}">Zmodyfikowany</td>
<td th:text="${#temporals.format(projekt.dataOddania, 'yyyy-MM-dd')}">Data oddania</td>
<td><a th:href="@{/projektEdit(projektId=${projekt.projektId})}">Edytuj</a><br>
<a th:href="@{/projektEdit(projektId=${projekt.projektId},delete='true')}">Usuń</a>
</td>
</tr>
</tbody>
</table>
</div>
</body>
</html>

```

Szablon *projektList.html* korzysta z pliku *resources\static\css\list-style.css*, poniżej jego przykładowa zawartość.

```

.root {
    text-align: center;
}

table {
    margin-left: auto;
    margin-right: auto;
    border: 1px solid darkGray;
    border-collapse: collapse;
}

th {
    background: lightGray;
    color: black;
    border: 1px solid darkGray;
    padding: 10px;
    font-weight: normal;
    font-size: 1.1em;
}

td {
    border: 1px solid darkGray;
    padding: 5px 10px;
}

.search-block {
    width: 80%;
}

```

13. ZADANIE OBLIGATORYJNE

- Rozszerz funkcjonalność aplikacji webowej o możliwość dodawania, edytowania i usuwania danych studentów.

14. ZADANIA DODATKOWE

- Zaimplementowanie funkcjonalności zarządzania zadaniami tj. dodawanie, modyfikowanie i usuwanie zadań w systemie.
- Rozszerzenie systemu o przypisywanie zadań oraz studentów do konkretnych projektów.
- Zaimplementowanie dla listy studentów mechanizmów stronicowania i wyszukiwania, które ułatwią dostęp danych w systemie.

15. KONFIGURACJA MECHANIZMU REJESTRACJI

W podkatalogu `src\main\resources` dodaj plik `logback-spring.xml` z przedstawioną poniżej zawartością.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="true">
  <property name="LOG_FILE" value="project-application" />
  <property name="LOG_DIR" value="logs" />
  <property name="LOG_ARCHIVE" value="${LOG_DIR}/archive" />

  <!-- Send messages to System.out -->
  <appender name="STDOUT"
    class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36}.%M\(%line\) - %msg%n</pattern>
    </encoder>
  </appender>

  <!-- Save messages to a file -->
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${LOG_DIR}/${LOG_FILE}.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <!-- daily rollover -->
      <fileNamePattern>${LOG_ARCHIVE}/%d{yyyy-MM-dd}${LOG_FILE}.log.zip
      </fileNamePattern>
      <!-- keep 30 days' worth of history capped at 30MB total size -->
      <maxHistory>30</maxHistory>
      <totalSizeCap>30MB</totalSizeCap>
    </rollingPolicy>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36}.%M\(%line\) - %msg%n</pattern>
    </encoder>
  </appender>

  <!-- For the 'com.project' package and all its subpackages -->
  <logger name="com.project" level="INFO" additivity="false">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="FILE" />
  </logger>

  <!-- By default, the level of the root level is set to INFO -->
  <root level="INFO">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

Zamiast korzystać z `System.out.println(...)`; używaj mechanizmu rejestracji, który oprócz standardowego drukowania komunikatów w konsoli będzie zapisywał również ich zawartość w plikach podkatalogu `logs`, a także automatycznie je archiwizował. Pamiętaj, że we wszystkich klasach, które mają korzystać z mechanizmu rejestracji trzeba tworzyć zmienną za pomocą statycznej metody `LoggerFactory.getLogger` przekazując w jej parametrze odpowiednią klasę. Poniżej przedstawione zostały przykłady prezentujące korzystanie z mechanizmu rejestracji.

```
package ...
...
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
...
public class JakasKlasa {
    private static final Logger logger = LoggerFactory.getLogger(JakasKlasa.class);
    ...

    logger.info("Uruchamianie programu ...");
    ...
    logger.info("Wersja aplikacji: {}", 1.9);

    logger.warn("Uaktualnij aplikację. Najnowsza dostępna wersja: {}", 2.0);
    ...

} catch (SQLException e) {
    logger.error("Błąd podczas zapisywania projektu!", e);
    ...
    int kodBledu = 7;
    logger.error("Błąd podczas zapisywania projektu (kod błędu: {})", kodBledu, e);
}
```

LITERATURA

- Herbert Schildt, *Java. Kompendium programisty*. Wydanie X, 2018, Helion
- Craig Walls, *Spring w akcji*. Wydanie V, 2019, Helion
- Christian Bauer, Gavin King, Gary Gregory, *Java Persistence. Programowanie aplikacji bazodanowych w Hibernate*. Wydanie II, 2016, Helion

PRZYDATNE SKRÓTY

CTRL + SHIFT + L – pokazuje wszystkie dostępne skróty

CTRL + SHIFT + F – formatowanie kodu

SHIFT + ALT + R – zmiana nazwy klasy, metody lub zmiennej itp., trzeba wcześniej ustawić kursor na nazwie

SHIFT + ALT + L – utworzenie zmiennej z zaznaczonego fragmentu kodu

SHIFT + ALT + M – utworzenie metody z zaznaczonego fragmentu kodu

CTRL + ALT + STRZAŁKA W GÓRĘ – skopiowanie linii i wklejenie w bieżącym wierszu

CTRL + ALT + STRZAŁKA W DÓŁ – skopiowania bieżącej linii i wklejenie poniżej

CTRL + SHIFT + O – automatyczne dodawanie i porządkowanie sekcji importów

CTRL + 1 – „zrób to co chcę zrobić”, m.in. sugestie rozwiązań bieżącego problemu

CTRL + Q – przejście do miejsca ostatniej modyfikacji

F11 – debugowanie aplikacji

CTRL + F11 – uruchomienie aplikacji

CTRL + M – powiększenie/zmniejszenie widoku w perspektywie

Ustawienie kursora np. na wywołaniu metody, typie zmiennej, klasie importu itp. i wciśnięcie **F3** powoduje przejście do kodu źródłowego wywoływanej metody, klasy zmiennej, klasy importu itd.

Wpisanie `sysout` i naciśnięcie skrótu **CTRL + SPACJA** spowoduje wstawienie `System.out.println();`