

# USŁUGA SIECIOWA TYPU REST

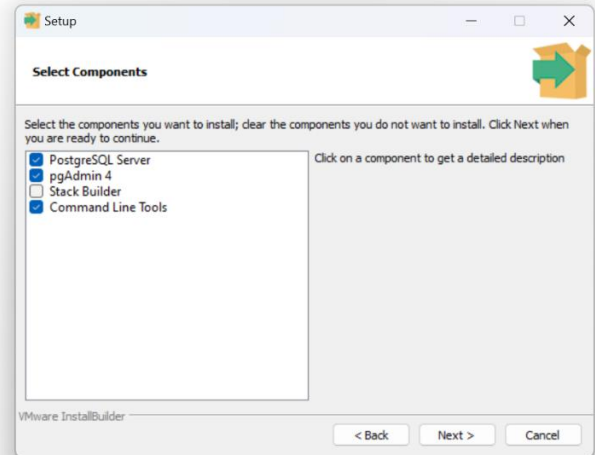
## 1. INSTALACJA BAZY POSTGRESQL (instalują tylko osoby korzystające na zajęciach z własnych laptopów)

Zainstaluj najnowszą wersję bazy PostgreSQL (<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>). Wybierając komponenty do instalacji należy zaznaczyć *PostgreSQL Server*, *Command Line Tools* i *PgAdmin 4*, natomiast z dodatku *Stack Builder* można zrezygnować. Podczas instalacji w systemie Windows zakładany jest użytkownik o nazwie *postgres*, który uruchamia PostgreSQL-a jako serwis. Zakładany jest również wewnętrzny użytkownik baz danych o tej samej nazwie, który ma prawa administratora dotyczące dostępu do baz. Jako hasło dla tego użytkownika można wpisać *postgres* (we wszystkich prezentowanych przykładach będzie używane takie hasło).

Do zmiennej środowiskowej *Path* można też dodać ścieżkę do katalogu *bin* z narzędziami PostgreSQL-a (przykładowa ścieżka

`C:\Program Files\PostgreSQL\15\bin`).

Pamiętaj, że dodawana ścieżka musi być oddzielona średnikiem od już istniejących wpisów w zmiennej *Path*. W systemie Windows 10 / 11 podczas definiowania ścieżki do podkatalogu *bin* nie należy dodawać średnika (ze względu na sposób edycji zmiennej środowiskowej w tym systemie). Natomiast we wszystkich starszych systemach trzeba podczas wpisywania oddzielić ścieżkę średnikiem od już istniejących wpisów. Po każdej modyfikacji zmiennej środowiskowej zalecane jest ponownie uruchomienie komputera. Ścieżka wskazującą na podkatalog *bin* PostgreSQL-a została poprawnie dodana do zmiennej środowiskowej *Path*, jeżeli komenda *psql* jest rozpoznawana w windowsowym *Wierszu polecenia*.



**1.1.** Utwórz bazę danych o nazwie *projekty*. Możesz skorzystać z instalowanego wraz z bazą PostgreSQL programu *PgAdmin4* lub wpisać w windowsowym *Wierszu polecenia* (użycie konsoli wymaga, aby w zmiennej środowiskowej *Path* była dodana ścieżka do katalogu z narzędziami PostgreSQL-a):

```
createdb --username=postgres projekty
```

(Zamiast `--username=postgres` można używać `-U postgres`, a także pomijać wpisywanie użytkownika i jego hasła, jeżeli zdefiniujesz zmienne systemowe `PGUSER` i `PGPASSWORD`. Pamiętaj, że w środowisku produkcyjnym korzystanie z takiego rozwiązania jest niedopuszczalne.)

## 2. GENEROWANIE PROJEKTU

**2.1.** Otwórz stronę <https://start.spring.io> i wygeneruj szkielet projektu. Użyj przedstawionych poniżej nazw, ustawień i zależności.

Project

☒ Gradle - Groovy
 ☐ Gradle - Kotlin
 ☐ Maven

Language

☒ Java
 ☐ Kotlin
 ☐ Groovy

Spring Boot

☐ 3.3.0 (SNAPSHOT)
 ☐ 3.3.0 (M1)
 ☐ 3.2.4 (SNAPSHOT)
 ☒ 3.2.3
 ☐ 3.1.10 (SNAPSHOT)
 ☐ 3.1.9

Project Metadata

Group

com.project

Artifact

project-rest-api

Name

project-rest-api

Description

Back-end of project management application

Package name

com.project.rest

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Security

SECURITY

Highly customizable authentication and access-control framework for Spring applications.

PostgreSQL Driver

SQL

A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

Validation

UO

Bean Validation with Hibernate validator.

GENERATE CTRL + G

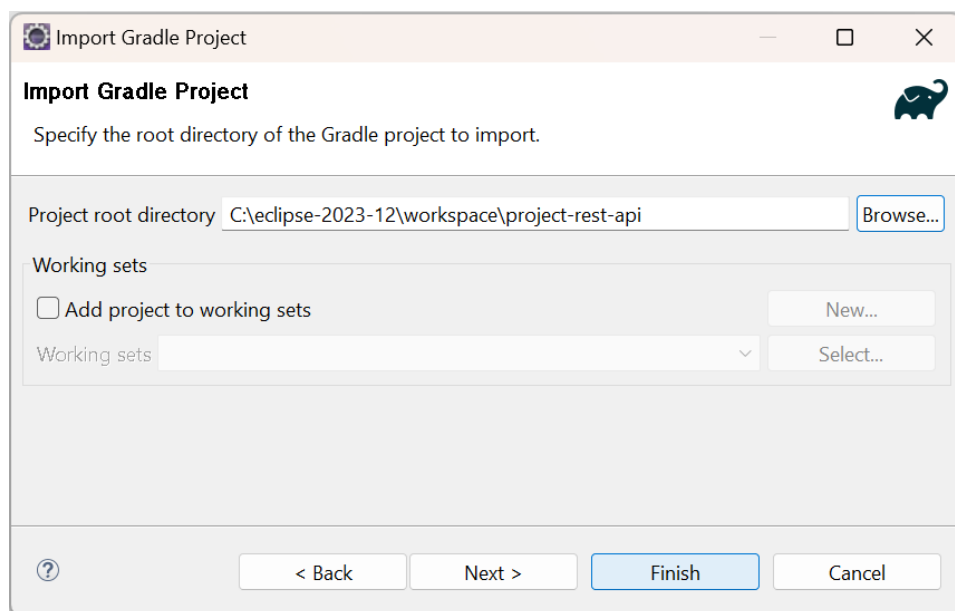
EXPLORE CTRL + SPACE

SHARE...

**2.2.** Najnowszą wersję oprogramowania wykorzystywanego na zajęciach laboratoryjnych (Eclipse 2023-12, JDK 17 i 21, JavaFX SDK 21, Scene Builder) można ściągnąć z

[https://utpedupl-my.sharepoint.com/:u:/g/personal/dszczeg\\_o365\\_pbs\\_edu\\_pl/EYbayJ1nEpFEk927YI9D2KcB\\_w8nqzyGXP0tbiTykltNQw?e=YDmnLs](https://utpedupl-my.sharepoint.com/:u:/g/personal/dszczeg_o365_pbs_edu_pl/EYbayJ1nEpFEk927YI9D2KcB_w8nqzyGXP0tbiTykltNQw?e=YDmnLs)

Archiwum wygenerowanego projektu rozpakuj bezpośrednio w tzw. przestrzeni projektów (*workspace*) - folderze z projektami Eclipse'a (możesz sprawdzić lokalizację wybierając z menu *File* -> *Switch Workspace* -> *Other...*). Następnie w środowisku Eclipse wybierz z menu *File* -> *Import...* -> *Gradle / Existing Gradle Project*, wskaż główny katalog rozpakowanego projektu i naciśnij *Finish*.



**2.3.** Edytuj plik *project-rest-api/src/main/resources/application.properties* i dodaj poniższe parametry konfiguracyjne.

```
# Spring DataSource
spring.datasource.url=jdbc:postgresql://localhost:5432/projekty
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.datasource.driver-class-name=org.postgresql.Driver

# Spring JPA
# The SQL dialect makes Hibernate generate better SQL for the chosen database
# (Postgres 9.5 and later)
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

```
# Validation or export of schema DDL to the database (create, create-drop, validate, update, none)
spring.jpa.hibernate.ddl-auto=update
# Logging JPA Queries
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# Spring Security
# HTTP authentication credentials
spring.security.user.name=admin
spring.security.user.password=admin
```

### 3. IMPLEMENTACJA USŁUGI

**3.1.** Użyjemy uwierzytelniania typu *Basic Authentication* – prostego zabezpieczenia usługi przed niepożądanym dostępem. Zastosowanie uwierzytelniania tego typu bez użycia protokołu szyfrowania np. TLS (rozwiązanie protokołu SSL) nie zapewnia ochrony przekazywanych danych (można np. podejrzec przesyłane w *Base64* login i hasło). Innym często wykorzystywanym i bardziej zaawansowanym mechanizmem zabezpieczającym jest *JWT* (*JSON Web Token*), który również można by zastosować w naszej usłudze.

Login i hasło wykorzystywane w *Basic Authentication* zostały zdefiniowane w pliku *application.properties* jako wartości parametrów *spring.security.user.name* i *spring.security.user.password*. Pozostaje jeszcze utworzyć w pakiecie *com.project.config* klasę *SecurityConfig*. Należy jedynie zdefiniować metodę *filterChain*, tak jak to zostało przedstawione poniżej. Proszę zwrócić uwagę, na adnotację *@Configuration*, która wskazuje Springowi klasę konfiguracyjną. Podczas uruchamiania aplikacji framework Spring skanuje pliki poszukując w nich różnych adnotacji i na ich podstawie podejmuje odpowiednie działania m.in. tworzy obiekty, wstrzykuje zależności, konfiguruje mechanizmy itd.

```
package com.project.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {
        return httpSecurity
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth.anyRequest().authenticated())
            .httpBasic(Customizer.withDefaults())
            .build();
    }
}
```

**3.2.** Utworzenie klas encyjnych odwzorowujących bazodanowe tabele.

- W pakiecie *com.project.model* utwórz klasę *Projekt*.

```
package com.project.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="projekt") //TODO Indeksować kolumny, które są najczęściej wykorzystywane do wyszukiwania projektów
public class Projekt {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="projekt_id") //tylko jeżeli nazwa kolumny w bazie danych ma być inna od nazwy zmiennej
    private Integer projektId;
```

```

@Column(nullable = false, length = 50)
private String nazwa;

/*TODO Uzupełnij kod o zmienne reprezentujące pozostałe pola tabeli projekt (patrz rys. 3.1),
.      następnie wygeneruj dla nich tzw. akcesory i mutatory (Source -> Generate Getters and Setters),
.      ponadto dodaj pusty konstruktor oraz konstruktor ze zmiennymi nazwa i opis.
*/
}

```

W klasach modelu można też korzystać z adnotacji spoza pakietu *jakarta.persistence* np. *@CreationTimestamp* i *@UpdateTimestamp*, które pozwalają na automatyczne przypisywanie dat i czasu podczas tworzenia lub modyfikacji rekordu.

```

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

// ...

@CreationTimestamp
@Column(name = "dataczas_utworzenia", nullable = false, updatable = false)
private LocalDateTime dataCzasUtworzenia;

// ...

@UpdateTimestamp
@Column(name = "dataczas_modyfikacji", nullable = false)
private LocalDateTime dataCzasModyfikacji;

```

- W pakiecie *com.project.model* utwórz klasę *Zadanie*.

```

package com.project.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="zadanie")
public class Zadanie {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="zadanie_id")
    private Integer zadanieId;

    /*TODO Uzupełnij kod o zmienne reprezentujące pozostałe pola tabeli zadanie (patrz rys. 3.1),
    .      następnie wygeneruj dla nich akcesory i mutatory (Source -> Generate Getters and Setters),
    .      ponadto dodaj pusty konstruktor oraz konstruktor ze zmiennymi nazwa, opis i kolejnosc.
    */
}

```

- Realizacja dwukierunkowej relacji jeden do wielu. W klasie *Zadanie* dodaj zmienną *projekt* oraz adnotację *@ManyToOne*. Możesz też użyć adnotacji *@JoinColumn*. Wygeneruj akcesory i mutatory dla nowo utworzonej zmiennej.

```

import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;

...

@ManyToOne
@JoinColumn(name = "projekt_id")
private Projekt projekt;

```

W klasie *Projekt* dodaj listę *zadania* z adnotacją *@OneToMany* i parametrem *mappedBy*, którego wartość wskazuje zmienną po drugiej stronie relacji tj. *projekt* z klasy *Zadanie*. Pamiętaj o wygenerowaniu akcesorów i mutatorów.

```

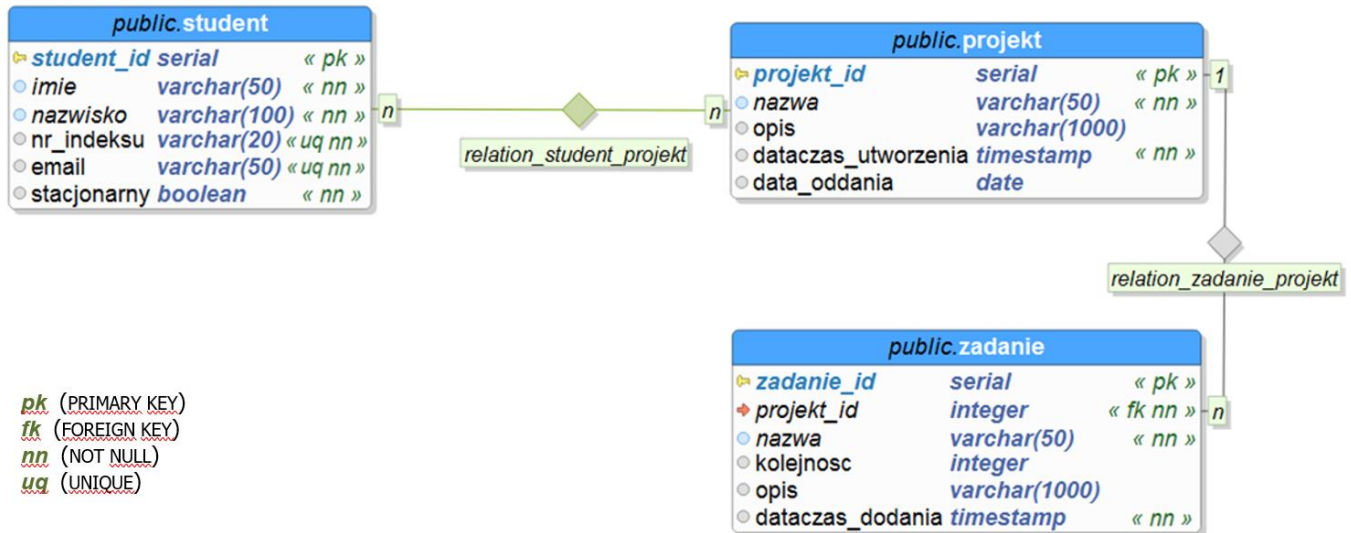
import jakarta.persistence.OneToMany;

...

@OneToMany(mappedBy = "projekt")
private List<Zadanie> zadania;

```

Rys. 3.1. Model bazy danych *projekty*



- W pakiecie *com.project.model* utwórz klasę *Student*.

```
...
import jakarta.persistence.Index;

@Entity //Indeksujemy kolumny, które są najczęściej wykorzystywane do wyszukiwania studentów
@Table(name = "student",
    indexes = { @Index(name = "idx_nazwisko", columnList = "nazwisko", unique = false),
        @Index(name = "idx_nr_indeksu", columnList = "nr_indeksu", unique = true) })
public class Student {

    /* TODO Uzupełnij kod o zmienne reprezentujące pola tabeli student (patrz rys. 3.1),
     * następnie wygeneruj dla nich akcesory i mutatory (Source -> Generate Getters and Setters)
     */

    public Student() {}

    public Student(String imie, String nazwisko, String nrIndeksu, Boolean stacjonarny) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrIndeksu = nrIndeksu;
    }

    public Student(String imie, String nazwisko, String nrIndeksu, String email, Boolean stacjonarny) {
        this.imie = imie;
        this.nazwisko = nazwisko;
        this.nrIndeksu = nrIndeksu;
        this.email = email;
        this.stacjonarny = stacjonarny;
    }

    //...
}
```

Do nowo utworzonej klasy *Student* dodaj zmienną *projekty* z adnotacją *@ManyToMany* (*mappedBy* wskazuje na zmienną w klasie *Projekt*).

```
import jakarta.persistence.ManyToMany;
...
@ManyToMany(mappedBy = "studenci")
private Set<Projekt> projekty;
```

Do istniejącej klasy *Projekt* dodaj zmienną *studenci* z adnotacją *@ManyToMany* i w każdej z tych klas wygeneruj akcesory i mutatory dla utworzonych zmiennych.

```
import jakarta.persistence.ManyToMany;
import jakarta.persistence.JoinTable;
...
@ManyToMany
@JoinTable(name = "projekt_student",
```

```

        joinColumns = {@JoinColumn(name="projekt_id")},
        inverseJoinColumns = {@JoinColumn(name="student_id")})
private Set<Student> studenci;

```

- W powyższych klasach dodamy też kilka nowych adnotacji – do automatycznej walidacji (*jakarta.validation.constraints.\**) oraz określającą sposób odwzorowania w JSON-ie dwukierunkowych relacji zastosowanych w modelu. Bez tej ostatniej adnotacji próba pobrania projektu z zadaniami spowodowałaby zapętlenie podczas mapowania obiektów do formatu JSON, tak jak przedstawiono poniżej. Adnotacja *@JsonIgnoreProperties({"projekt"})* przed listą zadań pozwoli pominąć podczas mapowania obiektów zmienną *projekt* klasy *Zadanie* i tym samym rozwiąże problem. Istnieje jeszcze kilka innych adnotacji eliminujących zapętlenia m.in. *@JsonManagedReference* i *@JsonBackReference*, *@JsonView* oraz *@JsonIdentityReference* i *@JsonIdentityInfo*.

```

{
  "projectId": 8,
  "nazwa": "Rozpoznawanie znaków drogowych",
  "opis": "Projekt i implementacja wielowarstwowej sieci neuronowej.",
  "dataCzasUtworzenia": "2020-04-17T22:25:35.258",
  "dataCzasModyfikacji": "2020-04-17T22:25:35.258",
  "dataOddania": "2020-06-15",
  "zadania": [
    {
      "zadanieId": 9,
      "nazwa": "Przygotowanie zbioru testowego.",
      "opis": "Utworzenie obrazków niskiej rozdzielczości w formacie PNG.",
      "kolejnosc": 1,
      "dataCzasDodania": "2020-04-17T22:25:35.319",
      "projekt": {
        "projectId": 8,
        "nazwa": "Rozpoznawanie znaków drogowych",
        "opis": "Projekt i implementacja wielowarstwowej sieci neuronowej.",
        "dataCzasUtworzenia": "2020-04-17T22:25:35.258",
        "dataCzasModyfikacji": "2020-04-17T22:25:35.258",
        "dataOddania": "2020-06-15",
        "zadania": [
          {
            "zadanieId": 9,
            .....

```



Poniżej przykład zmodyfikowanej klasy *Projekt*. Dodaj w pozostałych klasach encyjnych analogiczne adnotacje.

```
package com.project.model;

...
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.NotBlank;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
@Table(name = "projekt")
public class Projekt {

    ...    //lepszym rozwiązaniem jest przechowywanie komunikatów poza kodem źródłowym np. w plikach *.properties
    @NotBlank(message = "Pole nazwa nie może być puste!")
    @Size(min = 3, max = 50, message = "Nazwa musi zawierać od {min} do {max} znaków!")
    @Column(nullable = false, length = 50)
    private String nazwa;

    ...

    @OneToMany(mappedBy = "projekt")
    @JsonIgnoreProperties({"projekt"})
    private List<Zadanie> zadania;

    ...
}
```

**3.3. Tworzenie repozytoriów.** Podstawowych metod bazodanowych nie trzeba samodzielnie implementować, wystarczy tylko utworzenie interfejsu dziedziczącego z *JpaRepository<T, ID>* (gdzie: *T* – klasa encyjna, *ID* – typ identyfikatora).

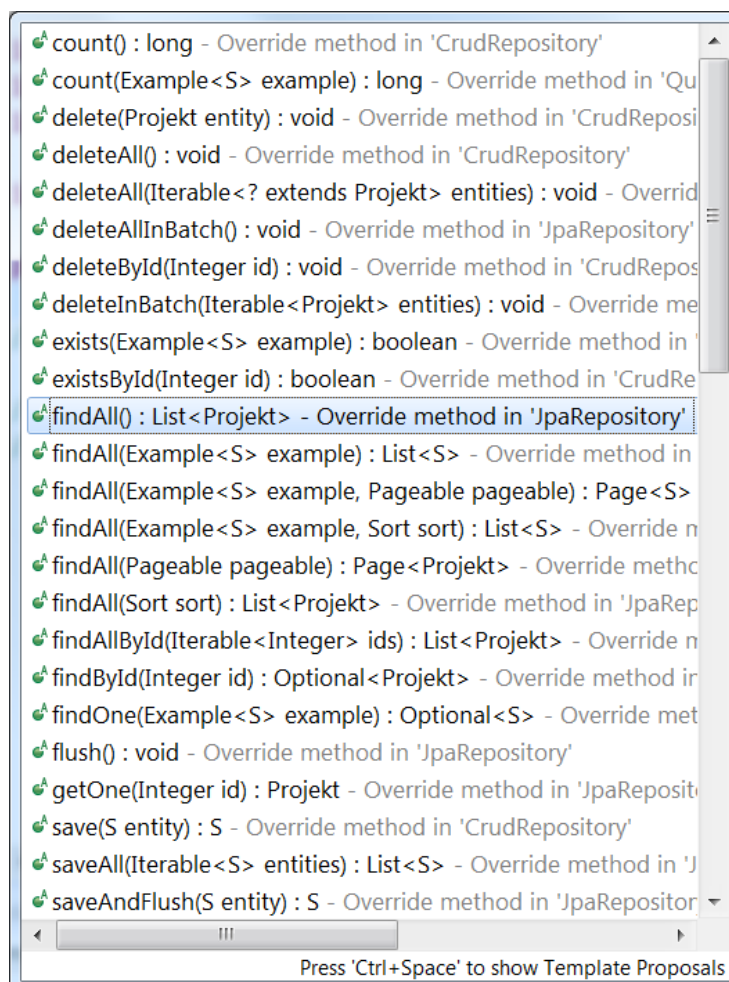
```
package com.project.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.project.model.Projekt;

public interface ProjektRepository extends JpaRepository<Projekt, Integer> {

}
```

Samo zdefiniowanie powyższego interfejsu pozwala na korzystanie ze wszystkich przedstawionych na obrazku metod.



W przypadku, gdy potrzebne są bardziej zaawansowane metody bazodanowe możemy użyć zapytań wbudowanych w nazwę metody – np. `findBy{query}` lub adnotacji `@Query(...)`. Więcej informacji na ten temat można znaleźć na stronie: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods>. Utwórz w projekcie pakiet `com.project.repository` i dodaj do niego trzy poniższe interfejsy – `ProjektRepository`, `ZadanieRepository` i `StudentRepository`.

```
package com.project.repository;
import java.util.List;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import com.project.model.Projekt;

public interface ProjektRepository extends JpaRepository<Projekt, Integer> {
    Page<Projekt> findByNazwaContainingIgnoreCase(String nazwa, Pageable pageable);
    List<Projekt> findByNazwaContainingIgnoreCase(String nazwa);

    // Metoda findByNazwaContainingIgnoreCase definiuje zapytanie
    // SELECT p FROM Projekt p WHERE upper(p.nazwa) LIKE upper(:%nazwa%)
}

package com.project.repository;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import com.project.model.Zadanie;

public interface ZadanieRepository extends JpaRepository<Zadanie, Integer> {
    //dwukropkiem oznacza się parametry zapytania
    @Query("SELECT z FROM Zadanie z WHERE z.projekt.projektId = :projektId")
    Page<Zadanie> findZadaniaProjektu(@Param("projektId") Integer projektId, Pageable pageable);

    @Query("SELECT z FROM Zadanie z WHERE z.projekt.projektId = :projektId")
    List<Zadanie> findZadaniaProjektu(@Param("projektId") Integer projektId);
}

package com.project.repository;
import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import com.project.model.Student;

public interface StudentRepository extends JpaRepository<Student, Integer> {
    Optional<Student> findByNrIndeksu(String nrIndeksu);
    Page<Student> findByNrIndeksuStartsWith(String nrIndeksu, Pageable pageable);
    Page<Student> findByNazwiskoStartsWithIgnoreCase(String nazwisko, Pageable pageable);

    // Metoda findByNrIndeksuStartsWith definiuje zapytanie
    // SELECT s FROM Student s WHERE s.nrIndeksu LIKE :nrIndeksu%

    // Metoda findByNazwiskoStartsWithIgnoreCase definiuje zapytanie
    // SELECT s FROM Student s WHERE upper(s.nazwisko) LIKE upper(:nazwisko%)
}
```

**3.4. Utworzenie serwisów.** Powyższe repozytoria definiują bezpośredni dostęp do danych poprzez bazodanowe zapytania, są podstawowymi elementami, a w zasadzie interfejsami tzw. warstwy persystencji. Zwykle chcemy opakować naszą interakcję z bazą danych w warstwę pośrednią – serwisy, które tworzą tzw. warstwę logiki biznesowej. Serwis zwykle korzysta z niższej warstwy persystencji, ale może także używać innych klas z tej samej warstwy. W naszej aplikacji utworzymy serwisy domenowe, których metody będą wywoływane przez kontrolery. Oczywiście wszystkie implementacje serwisów będą również hermetyzowane przy pomocy interfejsów. Utwórz interfejs `ProjektService` w pakiecie `com.project.service`, następnie dodaj klasę `ProjektServiceImpl` z jego implementacją. W podobny sposób utwórz pozostałe serwisy dla zadań i studentów.



```

package com.project.service;

import java.util.Optional;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import com.project.model.Projekt;

public interface ProjektService {

    Optional<Projekt> getProjekt(Integer projektId);
    Projekt setProjekt(Projekt projekt);
    void deleteProjekt(Integer projektId);
    Page<Projekt> getProjekty(Pageable pageable);
    Page<Projekt> searchByNazwa(String nazwa, Pageable pageable);
}

```

W poniższej klasie mamy kilka adnotacji wymagających wyjaśnienia np.:

- *@Service* - adnotacja oznacza, że klasa będzie zarządzana przez kontener Springa, pełni taką samą rolę co adnotacja *@Component* z dodatkowym wskazaniem na klasę warstwy logiki biznesowej.
- *@Autowired* – oznacza, że Spring zajmie się wstrzyknięciem instancji klasy *ProjektRepository* do zmiennej *projektRepository* (wcześniej oczywiście też sam utworzy obiekt tej klasy). W tym przypadku jest to tzw. wstrzykiwanie zależności poprzez konstruktor. Parametr wejściowy konstruktora określa klasę obiektu, który przeznaczony jest do wstrzyknięcia. Jeżeli Springowi nie uda się utworzyć żadnego obiektu, który ma być wstrzyknięty lub będzie możliwość utworzenia kilku takich obiektów to pojawi się błąd podczas uruchamiania aplikacji (tego typu problemy można również rozwiązywać za pomocą specjalnych adnotacji). W najnowszych wersjach frameworku adnotacja przed konstruktorem może być pomijana, ponieważ wstrzykiwanie przez konstruktor jest działaniem domyślnym (o ile nie zdefiniowano kilku różnych konstruktorów).

```

package com.project.service;

import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import com.project.model.Projekt;
import com.project.repository.ProjektRepository;

@Service
public class ProjektServiceImpl implements ProjektService {

    private ProjektRepository projektRepository;

    @Autowired //adnotację można pomijać, jeżeli nie ma kilku wersji konstruktora
    public ProjektServiceImpl(ProjektRepository projektRepository) {
        this.projektRepository = projektRepository;
    }

    @Override
    public Optional<Projekt> getProjekt(Integer projektId) {
        return projektRepository.findById(projektId);
    }

    @Override
    public Projekt setProjekt(Projekt projekt) {
        //TODO
        return null;
    }

    @Override
    public void deleteProjekt(Integer projektId) {
        //TODO
    }

    @Override
    public Page<Projekt> getProjekty(Pageable pageable) {
        //TODO
        return null;
    }
}

```

```

@Override
public Page<Projekt> searchByNazwa(String nazwa, Pageable pageable) {
    //TODO
    return null;
}
}

```

Jeżeli w jakiejś metodzie serwisu modyfikującej dane wykonywanych jest kilka operacji bazodanowych i chcemy mieć gwarancję, że w przypadku niepowodzenia którejkolwiek z nich pozostałe zmiany zostaną wycofane to musimy skorzystać z adnotacji *@Transactional*. Dzięki niej wszystkie operacje uruchamiane wewnątrz metody zostaną wykonane w jednej transakcji bazodanowej (adnotację można wstawiać też przed nazwą klasy, wtedy wszystkie jej metody zostaną uwzględnione).

W naszym modelu nie ustawialiśmy kaskadowości operacji, która określa co ma się dzieć z zależnymi encjami podczas modyfikacji obiektu głównego np. możemy określić co zrobimy z zadaniami projektu w momencie jego usunięcia (mamy do dyspozycji kilka opcji m.in.: *CascadeType.PERSIST*, *CascadeType.MERGE*, *CascadeType.REMOVE*, *CascadeType.ALL*). W obecnej wersji podczas próby usunięcia projektu z zadaniami dostaniemy błąd (nie jest to oczywiście błąd, który trzeba naprawiać, często nawet unika się kaskadowego usuwania danych) np.:

[org.postgresql.util.PSQLException](#): BŁĄD: modyfikacja lub usunięcie na tabeli "projekt" narusza klucz obcy "fkauptb9wjo0o9fu7bm2s5tifid" tabeli "zadanie"  
Szczegóły: Klucz (projekt\_id)=(7) ma wciąż odwołanie w tabeli "zadanie".

...

Zatem jeżeli nie dodamy odpowiedniego ustawienia *CascadeType* do adnotacji *@OneToMany* to będziemy musieli podczas usuwania projektu zadbać o wcześniejsze usunięcie jego zadań. W takim przypadku zwykle chcemy mieć gwarancję, że usuniemy wszystko albo nic. Poniższa, przykładowa metoda to zapewnia dzięki przetwarzaniu transakcyjnemu.

```

@Service
public class ProjektServiceImpl implements ProjektService {

    private ProjektRepository projektRepository;
    private ZadanieRepository zadanieRepository;

    @Autowired // w tej wersji konstruktora Spring wstrzyknie dwa repozytoria
    public ProjektServiceImpl(PjektRepository projektRepository, ZadanieRepository zadanieRepo) {
        this.projektRepository = projektRepository;
        this.zadanieRepository = zadanieRepo;
    }

    ...

    @Override
    @Transactional
    public void deleteProjekt(Integer projektId) {
        for (Zadanie zadanie : zadanieRepository.findZadaniaProjektu(projektId)) {
            zadanieRepository.delete(zadanie);
        }
        projektRepository.deleteById(projektId);
    }
}

```

Do realizacji transakcji wykorzystywany jest springowy moduł programowania aspektowego, powyższy kod jest równoważny fragmentowi:

```

EntityTransaction etx = entityManager.getTransaction();
try {
    etx.begin();
    deleteProjekt(projektId); //wywołanie metody oznaczonej adnotacją @Transactional
    etx.commit();
} catch (Exception e) {
    etx.rollback();
    throw e;
}

```

**3.5. Utworzenie kontrolerów.** Proszę zwrócić uwagę w poniższej tabeli na adresy tzw. endpointów, jak widać do różnych akcji używamy zawsze tego samego głównego adresu, ale różnych metod HTTP oraz parametrów dołączanych do adresu (wartość identyfikatora zasobu umieszczana jest bezpośrednio w adresie, na jego końcu, a po znaku zapytania w formacie *klucz=wartość* definiuje się dodatkowe parametry np. stronicowania, sortowania, wyszukiwania). Takie podejście nie jest obligatoryjne, ale jest praktyką rekomendowaną.

ADRES ZASOBU: http://localhost:8080/api/projekty		
AKCJA	METODA HTTP	ADRES ENDPOINTA
POBIERANIE PROJEKTU (metoda kontrolera: <i>getProjekt</i> )	GET	.../api/projekty/{projektId}
POBIERANIE PROJEKTÓW (metoda kontrolera: <i>getProjekty</i> )	GET	.../api/projekty[?page=0&size=10&sort=nazwa]
WYSZUKANIE PROJEKTÓW (metoda kontrolera: <i>getProjektyByNazwa</i> )	GET	.../api/projekty?nazwa=wartosc[&page=0&size=10&sort=nazwa]
UTWORZENIE PROJEKTU (metoda kontrolera: <i>createProjekt</i> )	POST	.../api/projekty
MODYFIKACJA PROJEKTU (metoda kontrolera: <i>updateProjekt</i> )	PUT	.../api/projekty/{projektId}
USUWANIE PROJEKTU (metoda kontrolera: <i>deleteProjekt</i> )	DELETE	.../api/projekty/{projektId}

Przeanalizuj poniższy kod źródłowy kontrolera, zwróć uwagę na adnotacje i komentarze. Utwórz pakiet *com.project.rest.controller* i zdefiniuj w nim trzy kontrolery dla projektu, zadania i studenta.

```
package com.project.rest.controller;

import java.net.URI;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
import com.project.model.Projekt;
import com.project.service.ProjektService;

// dzięki adnotacji @RestController klasa jest traktowana jako zarządzany
// przez kontener Springa REST-owy kontroler obsługujący sieciowe żądania
@RestController
@RequestMapping("/api") // adnotacja @RequestMapping umieszczona w tym miejscu pozwala definiować
public class ProjektRestController { // część wspólną adresu, wstawianą przed wszystkimi poniższymi ścieżkami

    private ProjektService projektService; //serwis jest automatycznie wstrzykiwany poprzez konstruktor

    @Autowired
    public ProjektRestController(ProjektService projektService) {
        this.projektService = projektService;
    }

    // PRZED KAŻDĄ Z PONIŻSZYCH METOD JEST UMIESZCZONA ADNOTACJA (@GetMapping, PostMapping, ... ), KTÓRA OKREŚLA
    // RODZAJ METODY HTTP, A TAKŻE ADRES I PARAMETRY ŻĄDANIA

    //Przykład żądania wywołującego metodę: GET http://localhost:8080/api/projekty/1
    @GetMapping("/projekty/{projektId}")
```

```

ResponseEntity<Projekt> getProjekt(@PathVariable Integer projektId) { // @PathVariable oznacza, że wartość
    return ResponseEntity.of(projektService.getProjekt(projektId)); // parametru przekazywana jest w ścieżce
}

// @Valid włącza automatyczną walidację na podstawie adnotacji zawartych
// w modelu np. NotNull, Size, NotEmpty itd. (z jakarta.validation.constraints.*)
@PostMapping(path = "/projekty")
ResponseEntity<Void> createProjekt(@Valid @RequestBody Projekt projekt) { // @RequestBody oznacza, że dane
    // projektu (w formacie JSON) są
    // przekazywane w ciele żądania
    Projekt createdProjekt = projektService.setProjekt(projekt);
    URI location = ServletUriComponentsBuilder.fromCurrentRequest() // link wskazujący utworzony projekt
        .path("/{projektId}").buildAndExpand(createdProjekt.getProjektId()).toUri();
    return ResponseEntity.created(location).build(); // zwracany jest kod odpowiedzi 201 - Created
    // z linkiem location w nagłówku
}

@PutMapping("/projekty/{projektId}")
public ResponseEntity<Void> updateProjekt(@Valid @RequestBody Projekt projekt,
                                           @PathVariable Integer projektId) {
    return projektService.getProjekt(projektId)
        .map(p -> {
            projektService.setProjekt(projekt);
            return new ResponseEntity<Void>(HttpStatus.OK); // 200 (można też zwracać 204 - No content)
        })
        .orElseGet(() -> ResponseEntity.notFound().build()); // 404 - Not found
}

@DeleteMapping("/projekty/{projektId}")
public ResponseEntity<Void> deleteProjekt(@PathVariable Integer projektId) {
    return projektService.getProjekt(projektId).map(p -> {
        projektService.deleteProjekt(projektId);
        return new ResponseEntity<Void>(HttpStatus.OK); // 200
    }).orElseGet(() -> ResponseEntity.notFound().build()); // 404 - Not found
}

//Przykład żądania wywołującego metodę: http://localhost:8080/api/projekty?page=0&size=10&sort=nazwa,desc
@GetMapping(value = "/projekty")
Page<Projekt> getProjekty(Pageable pageable) { // @RequestHeader HttpHeaders headers - jeżeli potrzebny
    return projektService.getProjekty(pageable); // byłby nagłówek, wystarczy dodać drugą zmienną z adnotacją
}

// Przykład żądania wywołującego metodę: GET http://localhost:8080/api/projekty?nazwa=webowa
// Metoda zostanie wywołana tylko, gdy w żądaniu będzie przesłana wartość parametru nazwa.
@GetMapping(value = "/projekty", params="nazwa")
Page<Projekt> getProjektyByNazwa(@RequestParam String nazwa, Pageable pageable) {
    return projektService.searchByNazwa(nazwa, pageable);
}
}

```

**3.6.** Próba uruchomienia aplikacji za pomocą klasy *ProjectRestApiApplication* zakończy się błędem, ponieważ Spring podczas inicjalizacji poszukuje adnotacji w pakiecie klasy uruchomieniowej oraz wszystkich jego podpakietach. Chociaż w parametrze adnotacji *@SpringBootApplication* można wskazywać pakiety zewnętrzne to w naszym przypadku najprostszym rozwiązaniem problemu będzie zmiana nazwy pakietu przenosząca klasę *ProjectRestApiApplication* poziom wyżej. W widoku *Project Explorer* zaznacz główną ikonkę pakietu *com.project.rest*, następnie wciśnij prawy przycisk myszy i wybierz z menu *Refactor -> Rename*. W polu tekstowym okienka pozostaw *com.project* i kliknij *OK*. Aby uruchomić aplikację kliknij prawym przyciskiem myszki wewnątrz okna z kodem źródłowym lub na ikonke klasy *ProjectRestApiApplication* i wybierz *Run As -> Java Application*.

```

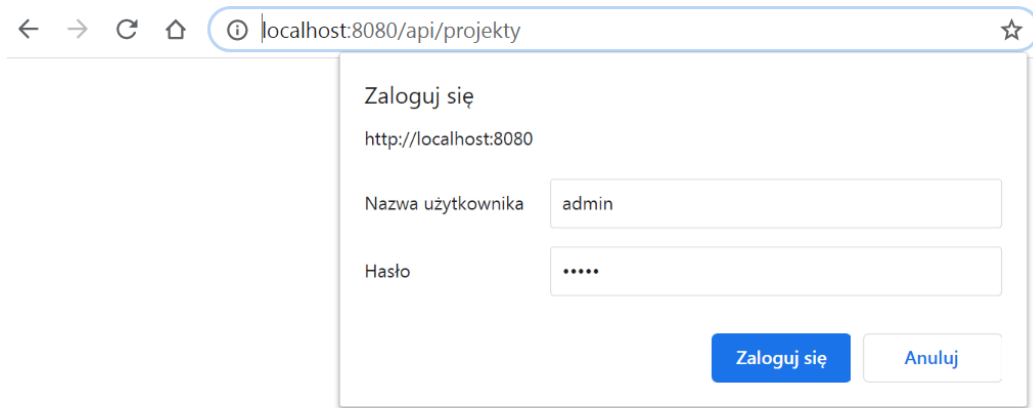
package com.project;

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

@SpringBootApplication
public class ProjectRestApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProjectRestApiApplication.class, args);
    }
}

```

Sprawdź, czy w konsoli środowiska Eclipse nie zostały wyświetlone jakieś błędy, powinna być też informacja o uruchomieniu serwera np. *Tomcat started on port(s): 8080 (http) with context path "*. Jeżeli aplikacja została poprawnie uruchomiona otwórz przeglądarkę i wpisz adres, który wywoła metodę pobierającą dane wszystkich projektów tj. <http://localhost:8080/api/projekty>.



Po podaniu nazwy i hasła (patrz p. 3.1) otrzymamy odpowiedź:

```
{"content": [], "pageable": {"sort": {"sorted": false, "unsorted": true, "empty": true}, "offset": 0, "pageNumber": 0, "pageSize": 20, "unpaged": false, "paged": true}, "totalPages": 0, "totalElements": 0, "last": true, "size": 20, "number": 0, "sort": {"sorted": false, "unsorted": true, "empty": true}, "numberOfElements": 0, "first": true, "empty": true}
```

W bazie nie są jeszcze przechowywane żadne dane, więc składnik *content* jest pusty. Struktura i pozostałe elementy odpowiedzi służą do stronicowania. Przekazywane wartości są ustawiane przez użyte w metodach kontrolera obiekty klas *Page* i *Pageable*. Jeżeli zmienimy, tak jak poniżej, zwracany typ metody kontrolera to w odpowiedzi otrzymamy tylko puste nawiasy prostokątne.

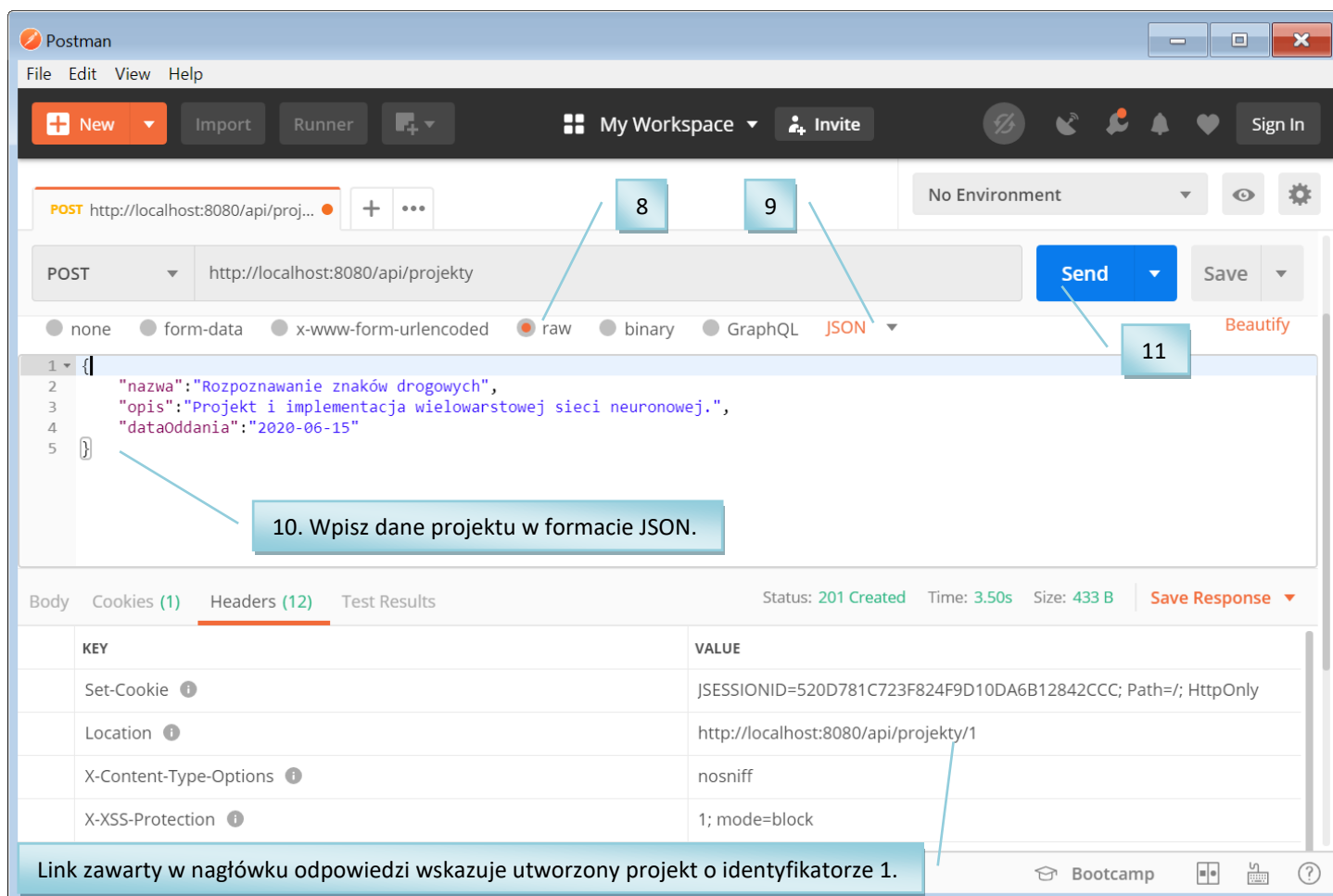
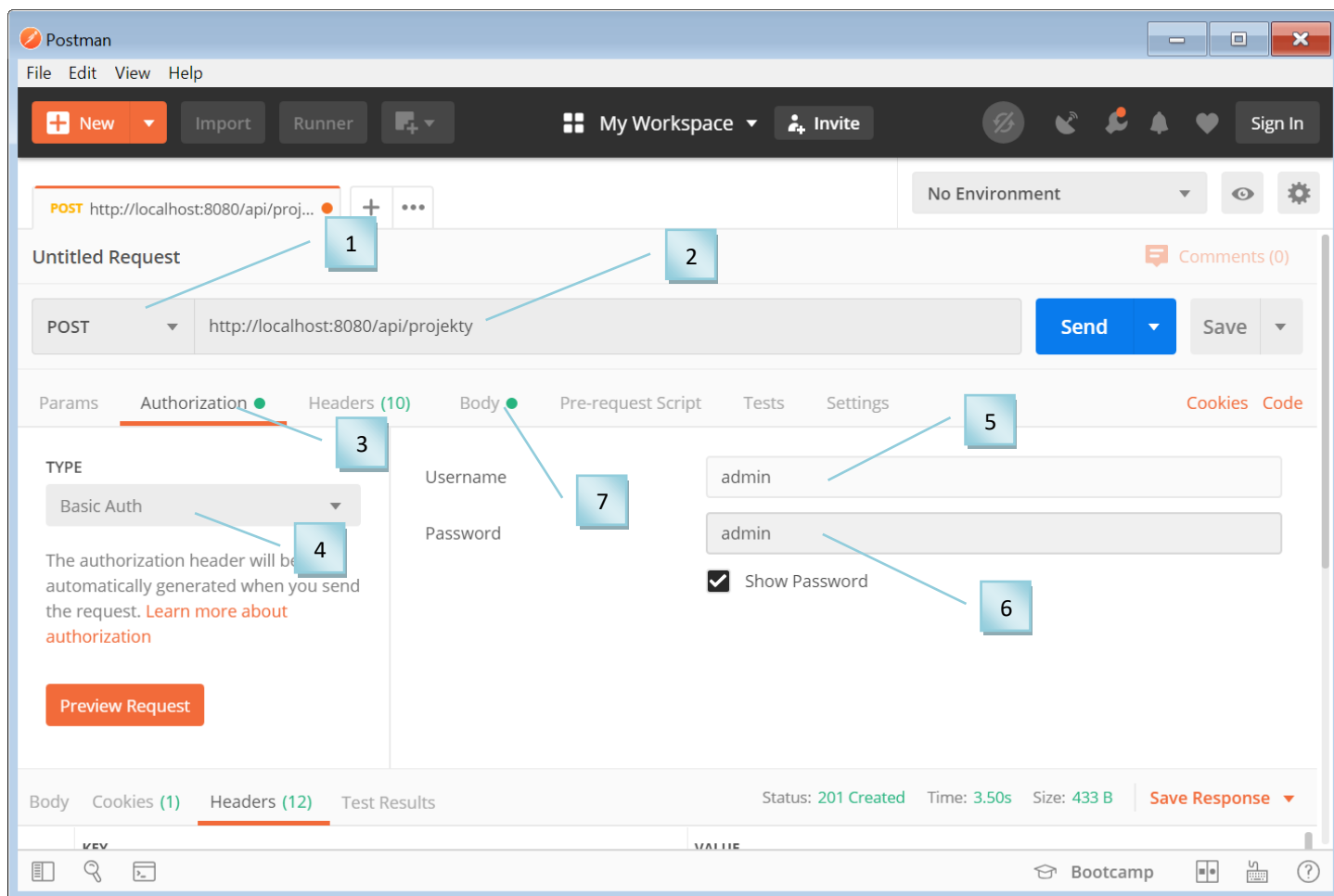
```
@GetMapping(value = "/projekty")
List<Projekt> getProjekty() {
    return projektService.getProjekty(PageRequest.of(0, 100)).getContent();
}
```

## 4. TESTOWANIE REST API

**4.1.** Ściągnij darmową wersję aplikacji *Postman* (<https://www.postman.com/downloads/>), za pomocą której przetestujesz endpointy kontrolerów. Na początek utwórz kilka projektów w bazie danych. Dla wybranego projektu utwórz też parę zadań. Później sprawdź pobieranie danych z dodatkowymi parametrami stronicowania i wyszukiwania dołączając np.

...?nazwa=jakieś\_wyszukiwane\_słowo\_lub\_jego\_fragment&page=0&size=10&sort=nazwa,desc

Na koniec usuń projekt z przypisanymi zadaniami.





**4.2.** Testowanie kontrolerów można zautomatyzować za pomocą testów korzystających ze springowej biblioteki MockMVC. Na początek edytuj plik *project-rest-api\build.gradle* i w bloku *test* i jeśli trzeba dodaj fragment włączający drukowanie komunikatów w konsoli. Dodaj też bloki *compileJava* i *compileTestJava* jeżeli w środowisku Eclipse ustawione jest windowsowe kodowanie znaków. Następnie kliknij prawym przyciskiem myszki na głównej ikonke projektu i wybierz *Gradle -> Refresh Gradle Project*.

```
...
compileJava {
    options.encoding = 'UTF-8'
}
// lub 'windows-1250' gdy edytor kodu źródłowego
// środowiska Eclipse ma ustawione windowsowe kodowanie

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

...
tasks.named('test') {
    useJUnitPlatform() //aktywacja natywnego wsparcia JUnit 5 (od wersji 4.6)
    testLogging {
        showStandardStreams = true //ustawia drukowanie komunikatów w konsoli
    }
}
```

W *src\test\java* utwórz pakiet *com.project.rest* i dodaj do niego klasy *ProjektRestControllerIntegrationTest* *ProjektRestControllerUnitTest*. Następnie przekopiuj do nich odpowiednie, przedstawione poniżej przykładowe zawartości. Przeanalizuj ich metody, uruchom testy (zwróć uwagę na czasy ich wykonania) i sprawdź wydruki w konsoli środowiska Eclipse.

## A. Test integracyjny

```
package com.project.rest;

import static org.hamcrest.Matchers.containsString;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.verifyNoMoreInteractions;
import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.header;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInfo;
import org.mockito.ArgumentCaptor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.json.JacksonTester;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.http.MediaType;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.web.bind.MethodArgumentNotValidException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import com.project.model.Projekt;
```

```

import com.project.service.ProjektService;

@SpringBootTest
@AutoConfigureMockMvc
@WithMockUser(username = "admin", password = "admin")
public class ProjektRestControllerIntegrationTest {
    // Uwaga! Test wymaga poniższego konstruktora w klasie Projekt, dodaj jeżeli nie został jeszcze zdefiniowany.
    // public Projekt(Integer projektId, String nazwa, String opis, LocalDateTime dataCzasUtworzenia, LocalDate
    // dataOddania){
    // ...
    //}

    // --- URUCHAMIANIE TESTÓW ---
    // ABY URUCHOMIĆ TESTY KLIKNIJ NA NAZWIE KLASY PRAWYM PRZYCISKIEM
    // MYSZY I WYBIERZ Z MENU 'Run As' -> 'Gradle Test' LUB PO USTAWIENIU
    // KURSORA NA NAZWIE KLASY WCIŚNIJ SKRÓT 'CTRL+ALT+X' A PÓŹNIEJ 'G'
    // MOŻNA RÓWNIEŻ ANALOGICZNIE URUCHAMIAĆ POJEDYNCZE METODY KLIKAJĄC
    // WCZEŚNIEJ NA ICH NAZWĘ

    private final String apiPath = "/api/projekty";
    @MockBean
    private ProjektService mockProjektService; // tzw. mock (czyli obiekt, którego używa się zamiast rzeczywistej
                                                // implementacji) serwisu wykorzystywany przy testowaniu kontrolera

    @Autowired
    private MockMvc mockMvc;
    private JacksonTester<Projekt> jacksonTester;

    @Test
    public void getProject_whenValidId_shouldReturnGivenProject() throws Exception {
        Projekt projekt = new Projekt(2, "Nazwa2", "Opis2", LocalDateTime.now(), LocalDate.of(2024, 6, 7));
        when(mockProjektService.getProject(projekt.getProjectId()))
            .thenReturn(Optional.of(projekt));
        mockMvc.perform(get(apiPath +("/{projektId}", projekt.getProjectId()).accept(MediaType.APPLICATION_JSON))
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.projektId").value(projekt.getProjectId()))
            .andExpect(jsonPath("$.nazwa").value(projekt.getNazwa()));
        verify(mockProjektService, times(1)).getProject(projekt.getProjectId());
        verifyNoMoreInteractions(mockProjektService);
    }

    @Test
    public void getProject_whenInvalidId_shouldReturnNotFound() throws Exception {
        Integer projektId = 2;
        when(mockProjektService.getProject(projektId)).thenReturn(Optional.empty());
        mockMvc.perform(get(apiPath +("/{projektId}", projektId).accept(MediaType.APPLICATION_JSON))
            .andDo(print())
            .andExpect(status().isNotFound());
        verify(mockProjektService, times(1)).getProject(projektId);
        verifyNoMoreInteractions(mockProjektService);
    }

    @Test
    public void getProjects_whenTwoAvailable_shouldReturnContentWithPagingParams() throws Exception {
        Projekt projekt1 = new Projekt(1, "Nazwa1", "Opis1", LocalDateTime.now(), LocalDate.of(2024, 6, 1));
        Projekt projekt2 = new Projekt(2, "Nazwa2", "Opis2", LocalDateTime.now(), LocalDate.of(2024, 6, 2));
        Projekt projekt1 = new Projekt(1, "Nazwa1", "Opis1", LocalDateTime.now(), LocalDate.of(2024, 6, 1));
        Projekt projekt2 = new Projekt(2, "Nazwa2", "Opis2", LocalDateTime.now(), LocalDate.of(2024, 6, 2));
        Page<Projekt> page = new PageImpl<>(List.of(projekt1, projekt2));
        when(mockProjektService.getProjekty(any(Pageable.class))).thenReturn(page);
        mockMvc.perform(get(apiPath).contentType(MediaType.APPLICATION_JSON))
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.content[*]").exists()) //content[*] - oznacza całą zawartość tablicy content
            .andExpect(jsonPath("$.content.length()").value(2))
            .andExpect(jsonPath("$.content[0].projektId").value(projekt1.getProjectId()))
            .andExpect(jsonPath("$.content[1].projektId").value(projekt2.getProjectId()));
        verify(mockProjektService, times(1)).getProjekty(any(Pageable.class));
        verifyNoMoreInteractions(mockProjektService);
    }

    @Test
    public void createProject_whenValidData_shouldReturnCreatedStatusWithLocation() throws Exception {
        Projekt projekt = new Projekt("Nazwa3", "Opis3", LocalDateTime.of(2024, 6, 7));
        String jsonProjekt = jacksonTester.write(projekt).getJson();
        projekt.setProjektId(3);
        when(mockProjektService.setProjekt(any(Projekt.class))).thenReturn(projekt);
        mockMvc.perform(post(apiPath).content(jsonProjekt).contentType(MediaType.APPLICATION_JSON)
            .accept(MediaType.ALL))
            .andDo(print())
            .andExpect(status().isCreated())
            .andExpect(header().string("location", containsString(apiPath + "/" + projekt.getProjectId())));
    }

    @Test

```

```

public void createProject_whenEmptyName_shouldReturnNotValidException() throws Exception {
    Projekt projekt = new Projekt( "", "Opis4", LocalDate.of(2024, 6, 7));
    MvcResult result = mockMvc.perform(post(apiPath)
        .content(jacksonTester.write(projekt).getJson())
        .contentType(MediaType.APPLICATION_JSON)
        .accept(MediaType.ALL))
        .andDo(print())
        .andExpect(status().isBadRequest())
        .andReturn();
    verify(mockProjektService, times(0)).setProjekt(any(Projekt.class));
    Exception exception = result.getResolvedException();
    assertNotNull(exception);
    assertTrue(exception instanceof MethodArgumentNotValidException);
    System.out.println(exception.getMessage());
}

@Test
public void updateProject_whenValidData_shouldReturnOkStatus() throws Exception {
    Projekt projekt = new Projekt(5, "Nazwa5", "Opis5", LocalDateTime.now(), LocalDate.of(2024, 6, 7));
    String jsonProjekt = jacksonTester.write(projekt).getJson();
    when(mockProjektService.getProjekt(projekt.getProjektId())).thenReturn(Optional.of(projekt));
    when(mockProjektService.setProjekt(any(Projekt.class))).thenReturn(projekt);
    mockMvc.perform(put(apiPath + "/" + projekt.getProjektId())
        .content(jsonProjekt)
        .contentType(MediaType.APPLICATION_JSON)
        .accept(MediaType.ALL))
        .andDo(print())
        .andExpect(status().isOk());
    verify(mockProjektService, times(1)).getProjekt(projekt.getProjektId());
    verify(mockProjektService, times(1)).setProjekt(any(Projekt.class));
    verifyNoMoreInteractions(mockProjektService);
}

/**
 * Test sprawdza czy ządanie o danych parametrach stronicowania i sortowania
 * spowoduje przekazanie do serwisu odpowiedniego obiektu Pageable, wcześniej
 * wstrzykniętego do parametru wejściowego metody kontrolera
 */
@Test
public void getProjectsAndVerifyPagingParams() throws Exception {
    Integer page = 5;
    Integer size = 15;
    String sortProperty = "nazwa";
    String sortDirection = "desc";
    mockMvc.perform(get(apiPath)
        .param("page", page.toString())
        .param("size", size.toString())
        .param("sort", String.format("%s,%s", sortProperty, sortDirection)))
        .andExpect(status().isOk());
    ArgumentCaptor<Pageable> pageableCaptor = ArgumentCaptor.forClass(Pageable.class);
    verify(mockProjektService, times(1)).getProjekty(pageableCaptor.capture());
    PageRequest pageable = (PageRequest) pageableCaptor.getValue();
    assertEquals(page, pageable.getPageNumber());
    assertEquals(size, pageable.getPageSize());
    assertEquals(sortProperty, pageable.getSort().getOrderFor(sortProperty).getProperty());
    assertEquals(Sort.Direction.DESC, pageable.getSort().getOrderFor(sortProperty).getDirection());
}

@BeforeEach
public void before(TestInfo testInfo) {
    System.out.printf("-- METODA -> %s%n", testInfo.getTestMethod().get().getName());
    ObjectMapper mapper = new ObjectMapper();
    mapper.registerModule(new JavaTimeModule());
    mapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    JacksonTester.initFields(this, mapper);
}

@AfterEach
public void after(TestInfo testInfo) {
    System.out.printf("<- KONIEC -- %s%n", testInfo.getTestMethod().get().getName());
}
}

```

## B. Test jednostkowy

```
package com.project.rest;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.hasSize;
import static org.hamcrest.Matchers.is;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;
import java.time.LocalDate;
import java.util.List;
import java.util.Optional;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.PageRequest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.mock.web.MockHttpServletRequest;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;
import com.project.controller.ProjektController;
import com.project.model.Projekt;
import com.project.service.ProjektService;

@ExtendWith(MockitoExtension.class)
public class ProjektRestControllerUnitTest {

    @Mock
    private ProjektService mockProjektService;

    @InjectMocks
    private ProjektController projectController;

    @Test
    void getProject_whenValidId_shouldReturnGivenProject() {
        Projekt projekt = new Projekt(1, "Nazwa1", "Opis1", LocalDate.of(2024, 6, 7));
        when(mockProjektService.getProject(projekt.getProjectId())).thenReturn(Optional.of(projekt));

        ResponseEntity<Projekt> responseEntity = projectController.getProject(projekt.getProjectId());

        assertAll(() -> assertEquals(responseEntity.getStatusCode().value(), HttpStatus.OK.value()),
            () -> assertEquals(responseEntity.getBody(), projekt));
    }

    @Test
    void getProject_whenInvalidId_shouldReturnNotFound() {
        Integer projektId = 2;
        when(mockProjektService.getProject(projektId)).thenReturn(Optional.empty());

        ResponseEntity<Projekt> responseEntity = projectController.getProject(projektId);

        assertEquals(responseEntity.getStatusCode().value(), HttpStatus.NOT_FOUND.value());
    }

    @Test
    //@DisplayName("Should return the page containing projects")
    void getProjects_shouldReturnPageWithProjects() {
        List<Projekt> list =
            List.of(new Projekt(1, "Nazwa1", "Opis1", LocalDate.of(2024, 6, 7)),
                new Projekt(2, "Nazwa2", "Opis2", LocalDate.of(2024, 6, 7)),
                new Projekt(3, "Nazwa3", "Opis3", LocalDate.of(2024, 6, 7)));
        PageRequest pageable = PageRequest.of(1, 5);
        Page<Projekt> page = new PageImpl<>(list, pageable, 5);
        when(mockProjektService.getProjecty(pageable)).thenReturn(page);

        Page<Projekt> pageWithProjects = projectController.getProjecty(pageable);

        assertNotNull(pageWithProjects);
        List<Projekt> projects = pageWithProjects.getContent();
        assertNotNull(projects);
        assertThat(projects, hasSize(3));
        assertAll(() -> assertTrue(projects.contains(list.get(0))),
            () -> assertTrue(projects.contains(list.get(1))),
            () -> assertTrue(projects.contains(list.get(2))));
        // W przypadku assertAll wszystkie asercje przekazane jako argumenty zostaną
```

```

        // wykonane, nawet gdy jedna z pierwszych da wynik negatywny, a jeśli choć
        // jedna zakończy się wyjątkiem, to cały test zakończy się błędem.
    }

    @Test
    void createProject_whenValidData_shouldCreateProject() {
        MockHttpServletRequest request = new MockHttpServletRequest();
        RequestContextHolder.setRequestAttributes(new ServletRequestAttributes(request));
        Projekt projekt = new Projekt(1, "Nazwa1", "Opis1", LocalDate.of(2023, 6, 7));
        when(mockProjektService.setProjekt(any(Projekt.class))).thenReturn(projekt);

        ResponseEntity<Void> responseEntity = projectController.createProjekt(projekt);

        assertThat(responseEntity.getStatusCode().value(), is(HttpStatus.CREATED.value()));
        assertThat(responseEntity.getHeaders().getLocation().getPath(), is("/") +
                                                                projekt.getProjektId());
    }

    @Test
    void updateProject_whenValidData_shouldUpdateProject() {
        Projekt projekt = new Projekt(1, "Nazwa1", "Opis1", LocalDate.of(2024, 6, 7));

        when(mockProjektService.getProjekt(projekt.getProjektId())).thenReturn(Optional.of(projekt));

        ResponseEntity<Void> responseEntity = projectController.updateProjekt(projekt,
                                                                projekt.getProjektId());

        assertThat(responseEntity.getStatusCode().value(), is(HttpStatus.OK.value()));
    }

    @Test
    void deleteProject_whenValidId_shouldDeleteProject() {
        Projekt projekt = new Projekt(1, "Nazwa1", "Opis1", LocalDate.of(2024, 6, 7));
        when(mockProjektService.getProjekt(projekt.getProjektId())).thenReturn(Optional.of(projekt));

        ResponseEntity<Void> responseEntity = projectController.deleteProjekt(projekt.getProjektId());

        assertThat(responseEntity.getStatusCode().value(), is(HttpStatus.OK.value()));
    }

    @Test
    void deleteProject_whenInvalidId_shouldReturnNotFound() {
        Integer projektId = 1;

        ResponseEntity<Void> responseEntity = projectController.deleteProjekt(projektId);

        assertThat(responseEntity.getStatusCode().value(), is(HttpStatus.NOT_FOUND.value()));
    }
}

```

**4.3.** Możesz też wygenerować wersję uruchomieniową usługi REST – plik JAR. W widoku *Gradle Tasks* kliknij prawym przyciskiem myszki na *assemble* (z *project-rest-api/build*) i wybierz *Run Gradle Tasks*. Utworzony plik można znaleźć w katalogu *project-rest-api\build\libs*. Usługę najlepiej uruchamiać za pomocą konsoli, aby można łatwo zamykać proces bezokienkowej aplikacji. W tym celu można utworzyć plik wsadowy np. *run-project-rest-api.bat* z poniższą zawartością:

```

::można wskazać lokalizację prywatnego JRE np.
set path="C:\eclipse-2022-12\jdk-19.0.2\bin"
::można też wskazać katalog z plikiem JAR, jeżeli plik BAT jest przechowywany w innym miejscu
::cd C:\eclipse-2022-12\workspace\project-rest-api\build\libs
java -jar project-rest-api-0.0.1-SNAPSHOT.jar

```

## 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 linijki i wklejenie w bieżącym wierszu

**CTRL + ALT + STRZAŁKA W DÓŁ** – skopiowania bieżącej linijki 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 okna

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();`