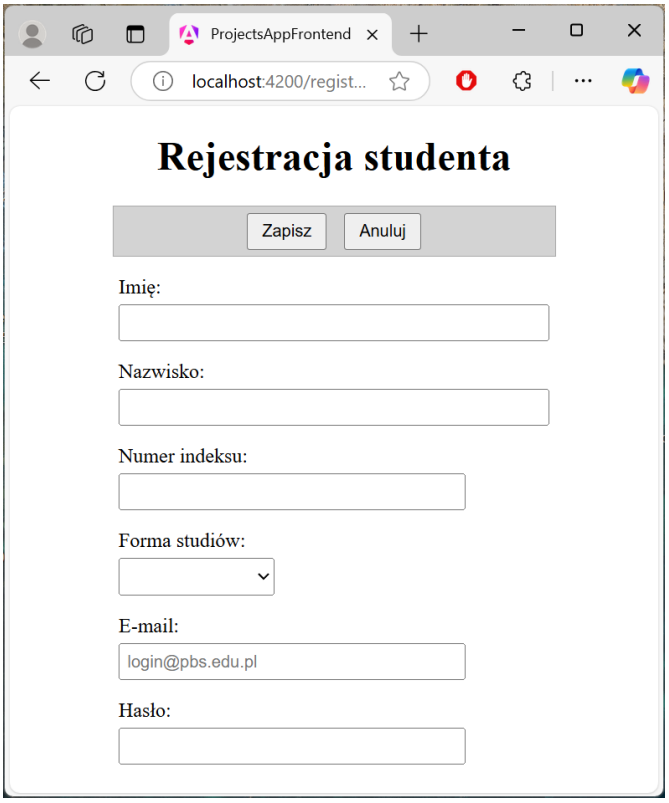
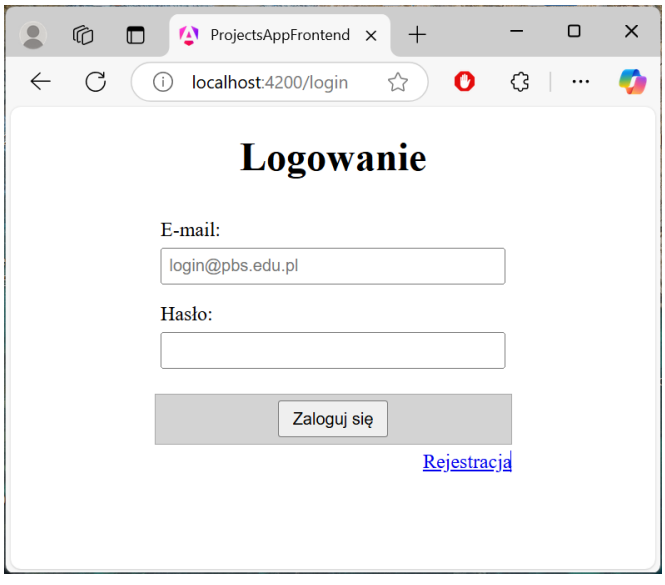
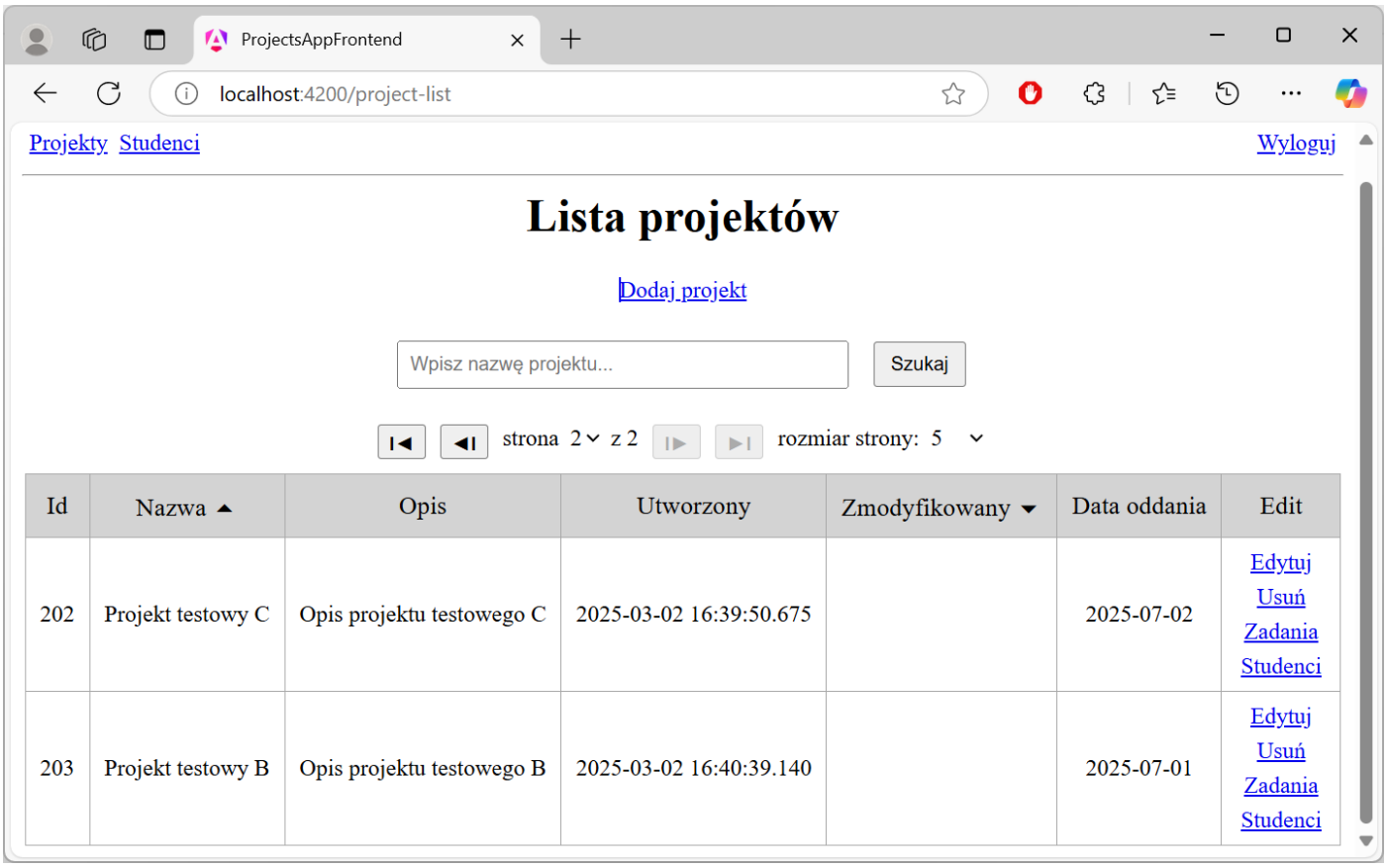
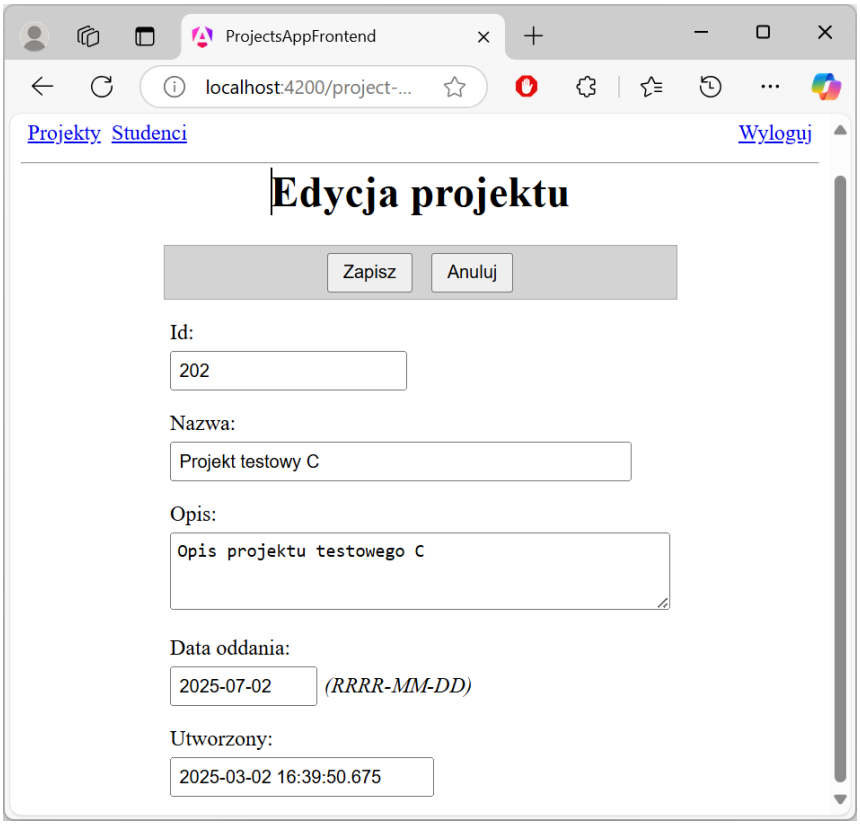
INSTRUKCJA LABORATORYJNA Z PRZEDMIOTU PROGRAMOWANIE ZWINNE

W ramach zajęć laboratoryjnych każdy zespół będzie realizował aplikację webową do zarządzania projektami, zgodnie z zasadami programowania zwinnego i ze szczególnym uwzględnieniem metodyki Scrum. Aplikacja webowa będzie ewoluować podczas realizacji kolejnych ćwiczeń laboratoryjnych. W zdecydowanej większości zadań z nią związanych będzie wykorzystywany framework Spring Boot i narzędzie Gradle lub Maven. Oprogramowanie zostanie rozdzielone na back-end i front-end, będzie korzystać z REST API, Thymeleafa i RestClienta lub opcjonalnie innego rozwiązania do tworzenia interfejsu użytkownika (np. Angular, React itp.). Ponadto oprogramowanie będzie testowane za pomocą bibliotek JUnit, MockMVC i Mockito. Aplikacja zostanie też zabezpieczona przy użyciu modułu Spring Security, początkowo za pomocą podstawowego uwierzytelniania (ang. *basic authentication*), a później tokenów dostępowych i odświeżania. Utworzona w ramach zajęć laboratoryjnych aplikacja powinna umożliwiać wyświetlanie listy wszystkich projektów przechowywanych w bazie danych, a także ich tworzenie, modyfikowanie i usuwanie. Trzeba również zapewnić możliwość przeglądania zadań wybranego projektu, a także przypisywania do niego nowych pozycji. Poza tym należy uwzględnić przypisywanie użytkowników aplikacji do poszczególnych projektów, opcjonalnie również do zadań. Utworzony graficzny interfejs użytkownika powinien umożliwiać stronicowanie danych oraz posiadać mechanizm wyszukiwania i filtrowania.









Realizacja powinna uwzględniać m.in.:

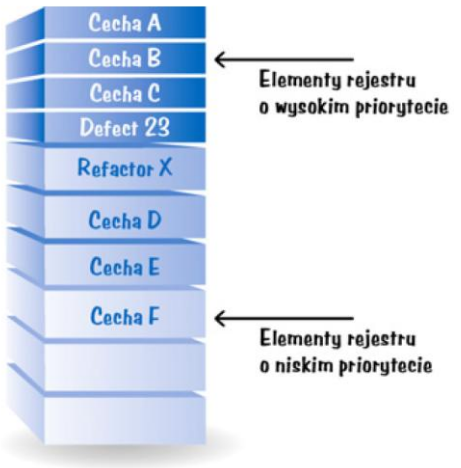
- zabezpieczenie danych i aplikacji przed nieupoważnionym dostępem,
- odrębne uprawnienia dla prowadzącego i studentów,
- testy jednostkowe, integracyjne i akceptacyjne (koniecznie serwisów i kontrolerów, zalecane JUnit5, Mockito i MockMVC),
- pełną funkcjonalność systemu pozwalającą dodawać, modyfikować i usuwać dane projektów, zadań i studentów. Powinna istnieć możliwość stronicowania i wyszukiwania, opcjonalnie sortowania, danych projektów i studentów.

- możliwość przesyłania na serwer i pobierania plików przypisywanych do danego projektu lub zadania,
- ogólnodostępny chat korzystający z dwukierunkowego kanału websocketowego (można przy tym użyć frameworku Atmosphere lub skorzystać ze Springa tworząc kanał websocketowy z wykorzystaniem protokołu STOMP). Dla chętnych funkcjonalność komunikacji w obrębie grupy projektowej i możliwość przesyłania plików do wybranych użytkowników,
- aplikacja powinna używać mechanizmu rejestracji np. biblioteki Logback i rozwiązania SLF4J w roli abstrakcyjnej fasady.

Kolejne zadania i technologie do rozważenia:

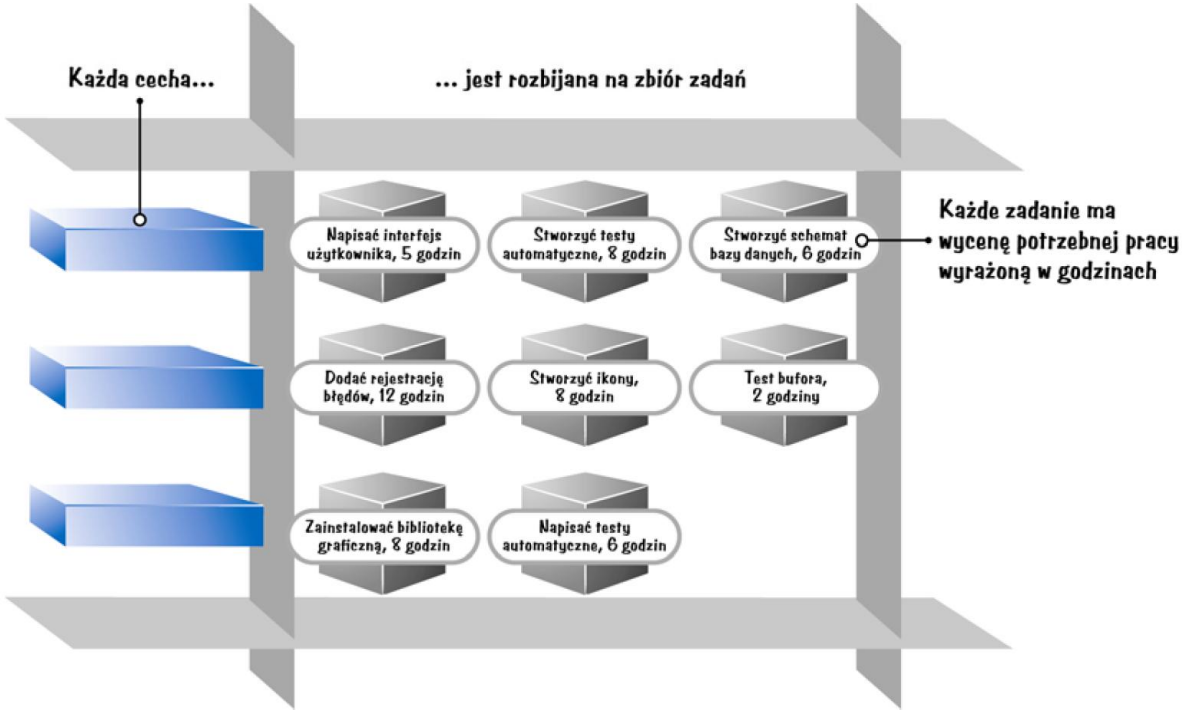
- wykorzystanie OAuth 2.0 - standardowego protokołu autoryzacji dostępu (logowanie za pomocą np. Google'a czy Facebooka),
- aplikacja reaktywna z użyciem Spring WebFlux (R2DBC - Reactive Relational Database Connectivity, Reactive Transactions, Backpressure, RSocket) lub korzystająca z wątków wirtualnych (wymagana Java 21 lub nowsza),
- implementacja tablicy scrumowej,
- implementacja tablicy kanbanowej z ustawianymi limitami prac w każdej kolumnie, a także skumulowanego wykresu przepływu z nanoszonymi liniami trendu dla tempa przybywania i liczby elementów w systemie,
- aplikacja springowa z wykorzystaniem programowania funkcyjnego zastępującego większość adnotacji,
- GraphQL z użyciem Spring Boota.
  Trzeba będzie definiować m.in.:
  - tzw. *Input* czyli prostą klasę (POJO) dla przyjmowania danych z edycji w GraphQL,
  - *QueryResolver*, klasę obsługująca zapytania w GraphQL,
  - *MutationResolver*, klasę obsługująca modyfikacje w GraphQL,
  - plik schema dla GraphQL, opisujący strukturę bazy danych i dostępne metody,
- Elasticsearch w Spring Boot,
- SOAP (ang. Simple Object Access Protocol) - usługa opisywana przez udostępniany plik WSDL (nazwa operacji, jej dane wejściowe, ich typ itp.) zabezpieczona za pomocą SAML-a (ang. Security Assertion Markup Language),
- mechanizm bazodanowych triggerów do automatycznego archiwizowania zmian projektu i jego zadań,
- utworzenie Springowej aplikacji natywnej przy użyciu kompilatora GraalVM.

Przed przystąpieniem do realizacji aplikacji webowej zespół powinien przedstawić swoją wizję oprogramowania, wyjaśnić dokładnie co i w jaki sposób chce osiągnąć, a także określić główne cele projektu. Wizja celu może być również uzupełniona przez historie użytkownika, które pomagają definiować specyficzne funkcje i potrzeby, które aplikacja ma spełniać.
Zespół powinien przygotować i prowadzić rejestr produktu (ang. *product backlog*), który przechowuje listę zadań funkcji i wymagań, zazwyczaj wyrażonych za pomocą historii użytkownika. Każdy element tego rejestru powinien być odpowiednio opisany i mieć przypisany priorytet (np. za pomocą liczb z zakresu od 1 do 10, gdzie 1 oznacza najwyższy priorytet). Często też dokonuje się szacowania czasu realizacji poszczególnych elementów rejestru produktu np. w tak zwanych dniach idealnych (przede wszystkim tych o najwyższych priorytetach).



REJESTR PRODUKTU (źródło: Kenneth S. Rubin, *Scrum. Praktyczny przewodnik po najpopularniejszej metodyce Agile*)

Przed realizacją zespół powinien przygotować niezbędne narzędzia i środowiska do pracy, takie jak system kontroli wersji, narzędzia do zarządzania projektami, środowisko deweloperskie itp.
Ponadto należy opracować plan na pierwsze iteracje - sprinty, w tym ich cele, spotkania planistyczne i cotygodniowe krótkie rozmowy.
Przed przystąpieniem do implementacji zespół powinien wybrać z rejestru produktu podgrupę historii użytkownika (zwracając przy tym uwagę na ich priorytet) możliwą do zrealizowania podczas jednego sprintu. Następnie wybrane historie użytkownika należy rozbić na listę zadań cząstkowych i oszacować czas ich realizacji (np. w godzinach).



REJESTR SPRINTU (źródło: Kenneth S. Rubin, *Scrum. Praktyczny przewodnik po najpopularniejszej metodyce Agile*)

Poszczególne zadania nie powinny być z góry przypisywane, lecz kolejno wybierane przez każdego członka zespołu, zawsze po zakończeniu poprzedniego. Aktualna wersja listy zadań (z oznaczeniem zrealizowanych i pozycji do implementacji) powinna być zawsze dostępna dla całego zespołu. Ponadto wygenerowany projekt/kod wzorcowy musi być umieszczony w zdalnym repozytorium. Każdy użytkownik przed rozpoczęciem realizacji danego zadania powinien uaktualnić swoją lokalną wersję projektu, a po zaimplementowaniu nowej funkcji oprogramowania przesłać zmiany na serwer.

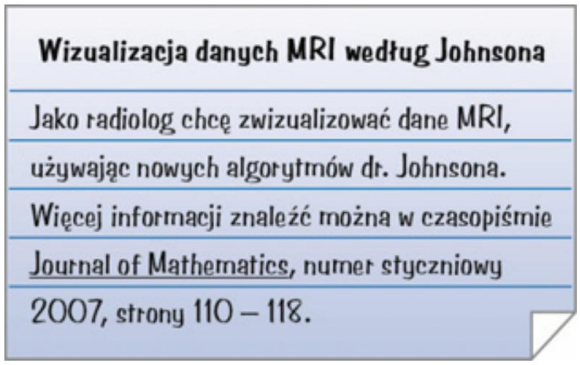
Zespół nie powinien poświęcać dużych ilości czasu na wytworzenie szczegółowych wymagań na samym początku projektu. Zamiast obszernego zbioru szczegółowych wymagań tworzymy na początku coś w rodzaju wskaźników wymagań, które nazywane są historyjkami rejestru produktu. Każda historyjka rejestru produktu reprezentuje sobą oczekiwaną wartość biznesową. Początkowo historyjki rejestru produktu mają duży rozmiar, czyli obejmują duże obszary wartości biznesowej, ale przez to nie są szczegółowe. Wraz z upływem czasu takie historyjki są być może nawet wielokrotnie omawiane i uszczegóławiane przez członków zespołu i przy tym najczęściej rozbijane na mniejsze, bardziej szczegółowe elementy rejestru produktu. Po pewnym czasie stają się wystarczająco małe i szczegółowe, aby można było wybrać je do kolejnej iteracji – sprintu (trafiają do tzw. rejestru sprintu), podczas którego zostaną zaprojektowane, zbudowane i przetestowane. Jednak nawet i w trakcie sprintu mogą być ujawniane kolejne szczegóły danego wymagania. Wzorzec i przykład historyjki użytkownika przedstawiono na poniższych rysunkach.

HISTORYJKA UŻYTKOWNIKA - karta, rozmowa i potwierdzenie

(źródło: Kenneth S. Rubin, Scrum. Praktyczny przewodnik po najpopularniejszej metodyce Agile)

KARTA (używamy małych kart, aby zachować zwięzłość. Karta powinna zawierać kilka zdań wyrażających sedno lub intencję danego wymogu)

ROZMOWA (historyjki użytkownika mogą i w zasadzie powinny być uzupełniane o dodatkowe informacje w formie pisemnej, jeżeli tylko są w stanie lepiej wyjaśnić jak dokładnie ma wyglądać ostateczne rozwiązanie)



POTWIERDZENIE Przednia część karty z historyjką użytkownika zawiera zazwyczaj kilka wierszy opisujących historyjkę, natomiast tylna część może specyfikować tzw. warunki zadowolenia. Są to kryteria akceptacji, które opisują spodziewane zachowanie.

Na początku zajęć laboratoryjnych będą prowadzone krótkie rozmowy – tzw. codzienne spotkania scrumowe (ang. *daily scrum / stand-up*). Celem tych spotkań jest szybkie omówienie postępów w realizacji zadań, zidentyfikowanie potencjalnych problemów oraz zaplanowanie działań na najbliższy czas. Podczas spotkania scrumowego każdy członek zespołu odpowiada na trzy pytania:

- ***Jakie zadania zrealizowałem od czasu ostatniego spotkania?***
- ***Co planuję zrobić do czasu następnego spotkania?***
- ***Czy napotkałem jakieś przeszkody, które uniemożliwiają postęp?***

Spotkania scrumowe trwają około 10 minut (maksymalnie 15 min.) i mają na celu tylko identyfikowanie problemów, a nie ich rozwiązywanie. Jeśli dany problem nie został rozwiązany w wyniku np. dwuminutowej dyskusji, trzeba zaplanować spotkanie uzupełniające z odpowiednimi osobami z grupy lub prowadzącym zajęcia. Wiele tego typu spotkań uzupełniających dotyczy tego, kto czym powinien się zająć. W ten sposób przebiega samodzielne organizowanie się zespołu. Większość zadań można wybrać z listy sobie samemu, jednak niektóre z nich wymagają jeszcze dodatkowego omówienia.

Obowiązkiem każdego zespołu będzie przygotowywanie krótkich protokołów z przeprowadzonych spotkań scrumowych (za każdym razem jeden protokół powinien być wypełniany, zgodnie z przedstawionym poniżej wzorem, przez wszystkich członków zespołu), dzięki czemu łatwiej będzie monitorować postępy, śledzić jakie decyzje podjęto oraz jakie były zadania w toku. Protokół zatem będzie stanowił punkt odniesienia, do którego można będzie wrócić w razie potrzeby. Ponadto osoby, które nie będą mogły uczestniczyć w spotkaniu, będą miały możliwość zapoznania się z postępami oraz ustaleniami i kontynuować pracę bez utraty kluczowych informacji.

W trakcie całego semestru zajęć laboratoryjnych powinny zostać zaplanowane i zrealizowane co najmniej trzy sprinty, a po zakończeniu każdego z nich zespół będzie zobowiązany do przygotowania sprawozdania. Ponadto ostatnie z tych opracowań powinno dodatkowo zawierać opis produktu końcowego i podsumowanie realizacji wszystkich iteracji.

PROTOKÓŁ ZE SPOTKANIA SCRUMOWEGO

Data: ……

Uczestnicy: ……

……

1. Co zostało zrealizowane od ostatniego spotkania:

……

……

……

……

……

……

……

……

……

2. Nad czym obecnie pracujemy:

……

……

……

……

……

……

……

……

3. Napotkane przeszkody:

……

……

……

……

……

4. Data kolejnego spotkania: ……

LITERATURA

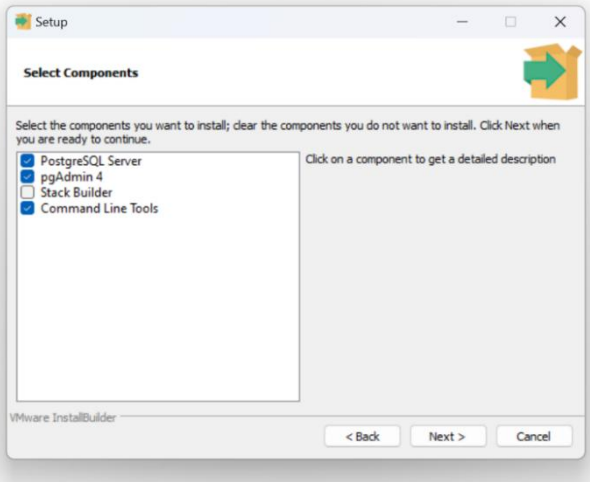
- Robert C. Martin, *Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów.* 2018, Helion
- Andrew Stellman, Jennifer Greene. *Agile. Przewodnik po zwinnych metodykach programowania.* 2015, Helion.
- Robert C. Martin, *Zwinne wytwarzanie oprogramowania. Najlepsze zasady, wzorce i praktyki.* 2015, Helion
- Kenneth S. Rubin, *Scrum. Praktyczny przewodnik po najpopularniejszej metodyce Agile.* 2013, Helion
- Craig Walls, *Spring w akcji. Wydanie V*, 2019, Helion

MATERIAŁY POMOCNICZE DLA OSÓB BEZ DOŚWIADCZENIA W TECHNOLOGIACH I NARZĘDZIACH UŻYWANYCH PODCZAS ZAJĘĆ LABORATORYJNYCH

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 *PostgresSQL 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:\ProgramFiles\PostgreSQL\17\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*.

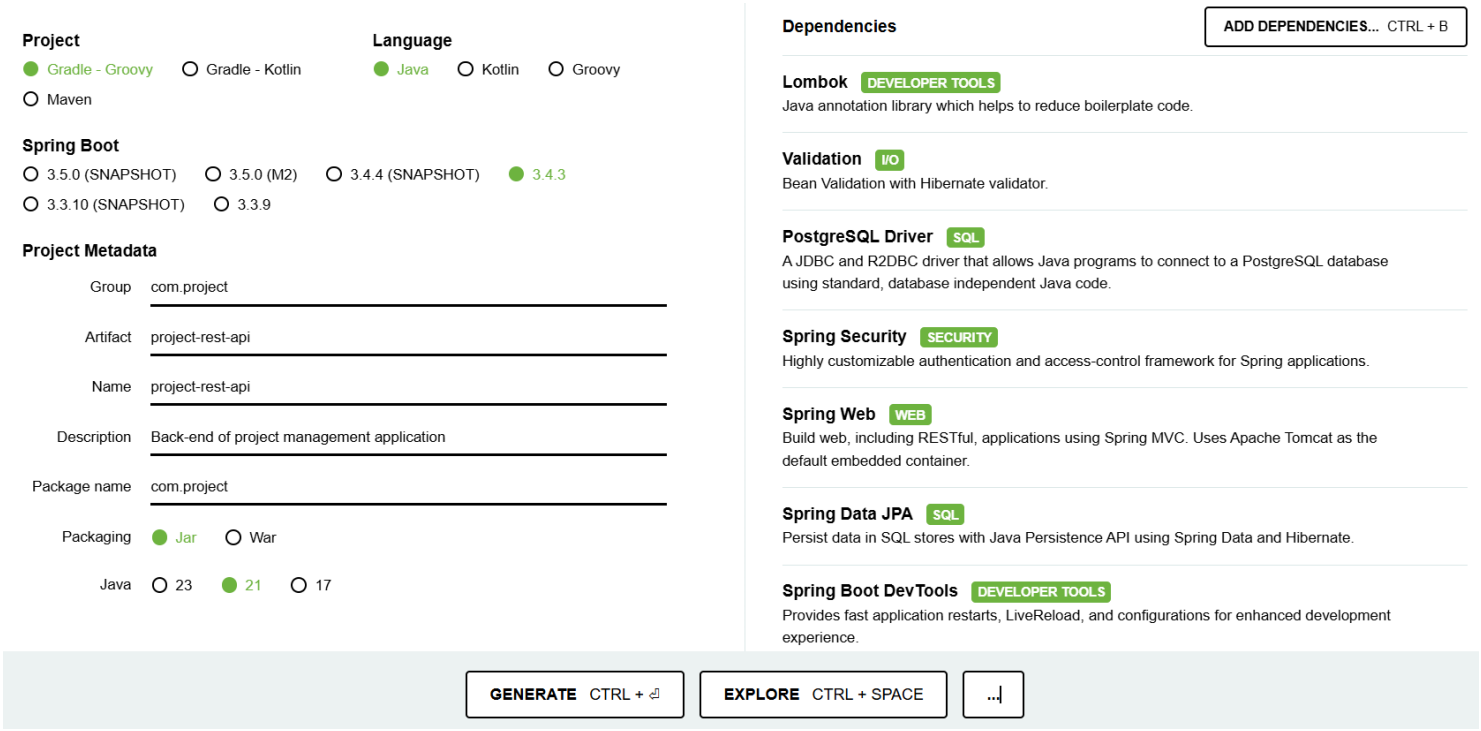
1.1. Utwórz bazę danych o nazwie *projekty*. Możesz skorzystać z instalowanego wraz z bazą PostgreSQL programu *PgAdmin4* lub wpisać w windowsowym *Wierszu polecenia* (użycie konsoli wymaga, aby w zmiennej środowiskowej *Path* była dodana ścieżka do katalogu z narzędziami PostgreSQL-a):

```
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

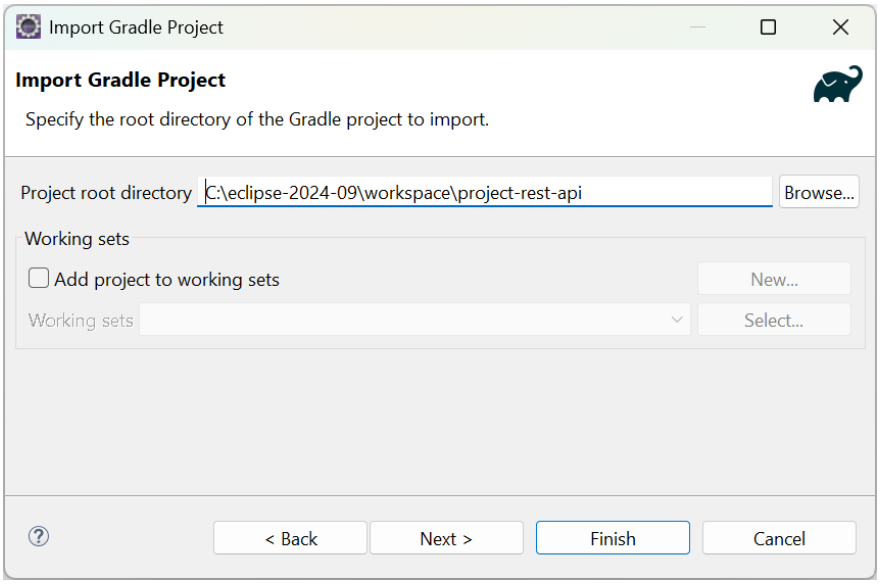
2.1. Otwórz stronę https://start.spring.io i wygeneruj szkielet projektu. Użyj przedstawionych poniżej nazw, ustawień i zależności.



2.2. Najnowszą wersję oprogramowania wykorzystywanego na zajęciach laboratoryjnych (Eclipse 2024-09, JavaFX SDK 23, Scene Builder) można ściągnąć z

https://utpedupl-my.sharepoint.com/:u:/g/personal/dszczeg_o365_pbs_edu_pl/EfcqMZaTR9pDsIW0xeVeWUIBwE6joSRvknnYAQnTnNQ7vw?e=ERhp8y

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 (rozwinięcie protokołu SSL) nie zapewnia ochrony przekazywanych danych (można np. podejrzeć 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
        @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 *@CreatedDate* i *@LastModifiedDate*, które pozwalają na automatyczne przypisywanie dat i czasu podczas tworzenia lub modyfikacji rekordu.

```
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;

 // ...

 @CreatedDate
 @Column(name = "dataczas_utworzenia", nullable = false, updatable = false)
 private LocalDateTime createdDate;


 @LastModifiedDate
 @Column(name = "dataczas_modyfikacji", insertable = false)
 private LocalDateTime lastModifiedDate;
```

- 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
        @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.
        */

}
```

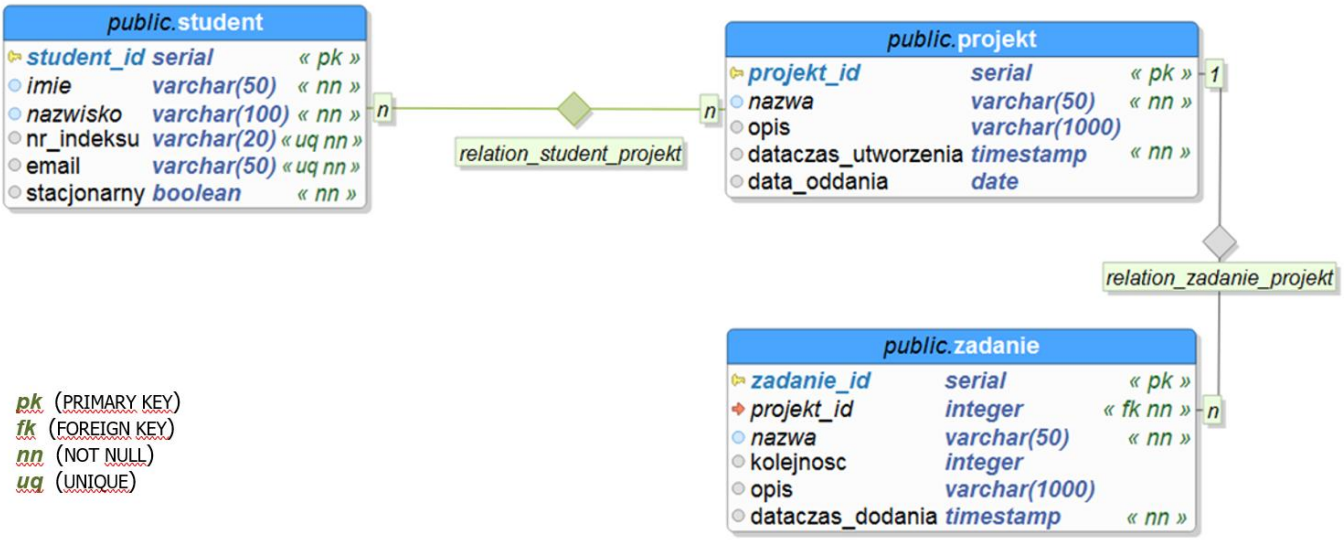
- Realizacja dwukierunkowej relacji jeden do wielu. W klasie *Zadanie* dodaj zmienną *projekt* oraz adnotację *@ManyToOne*. Możesz też użyć adnotacji *@JoinColumn*. Wygeneruj akcesory i mutatory dla nowo utworzonej zmiennej.

```
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*.


```
{
  "projektId": 8,
    "nazwa": "Rozpoznawanie znaków drogowych",
    "opis": "Projekt i implementacja wielowarstowej 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": {
             "projektId": 8,
             "nazwa": "Rozpoznawanie znaków drogowych",
             "opis": "Projekt i implementacja wielowarstowej sieci neuronowej.",
             "dataCzasUtworzenia": "2020-04-17T22:25:35.258",
             "dataCzasModyfikacji": "2020-04-17T22:25:35.258",
             "dataOddania": "2020-06-15",
             "zadania": [
                {
                   "zadanieId": 9,
                      ......
```

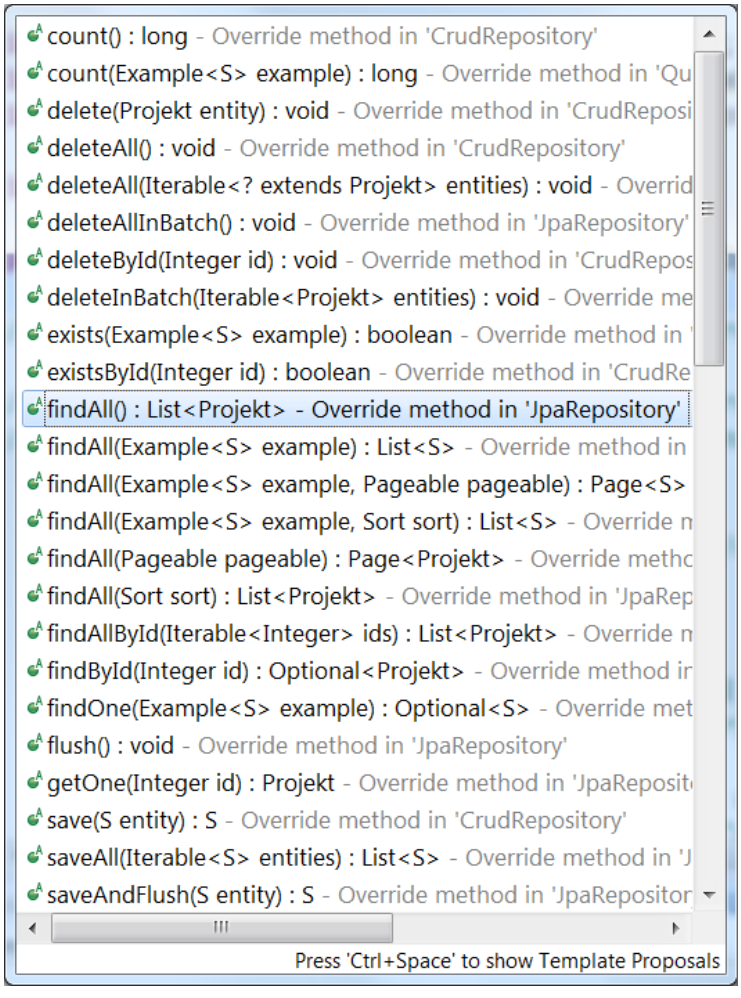
Poniżej przykład zmodyfikowanej klasy *Projekt*. Dodaj w pozostałych klasach encyjnych analogiczne adnotacje.

```
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. Utworzenie 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/reference/jpa/query-methods.html. 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 zarządzania przez kontener Springa, pełni taką samą rolę co adnotacja *@Component* z dodatkowymi 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 dziać 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 "fkaupfb9wjo0o9fu7bm2s5tifid" 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(ProjektRepository 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: *getProjekt*)	GET	.../api/projekty/{projektId}
POBIERANIE PROJEKTÓW (metoda kontrolera: *getProjekty*)	GET	.../api/projekty[?page=0&size=10&sort=nazwa]
WYSZUKANIE PROJEKTÓW (metoda kontrolera: *getProjektyByNazwa*)	GET	.../api/projekty?nazwa=*wartosc*[&page=0&size=10&sort=nazwa]
UTWORZENIE PROJEKTU (metoda kontrolera: *createProjekt*)	POST	.../api/projekty
MODYFIKACJA PROJEKTU (metoda kontrolera: *updateProjekt*)	PUT	.../api/projekty/{projektId}
USUWANIE PROJEKTU (metoda kontrolera: *deleteProjekt*)	DELETE	.../api/projekty/{projektId}

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

```
package com.project.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;
import io.swagger.v3.oas.annotations.tags.Tag;

                        // dzięki adnotacji @RestController klasa jest traktowana jako zarządzany
@RestController         // przez kontener Springa REST-owy kontroler obsługujący sieciowe żądania
@RequestMapping("/api")             // adnotacja @RequestMapping umieszczona w tym miejscu pozwala definiować
                                    // cześć wspólną adresu, wstawianą przed wszystkimi poniższymi ścieżkami
@Tag(name = "Projekt") // zmiana nazwy, uwzględniania m.in. przy generowaniu specyfikacji za pomocą OpenAPI
public class ProjektRestController {

   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
```

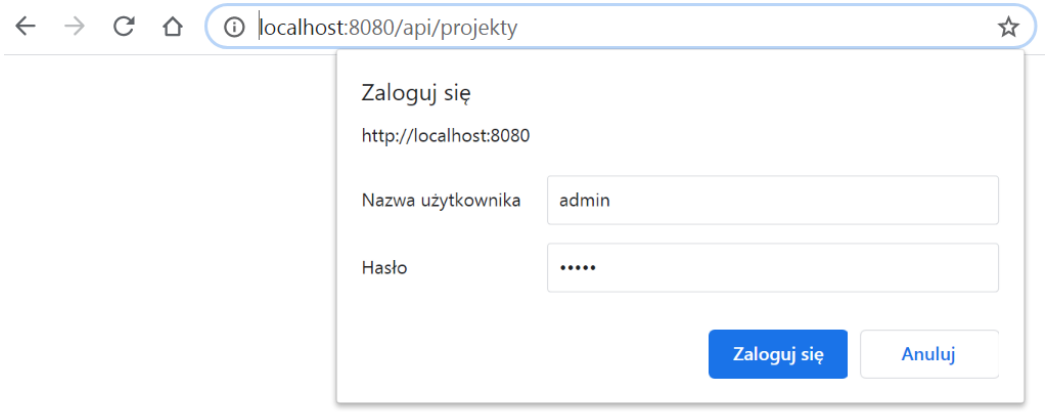
```
    //Przykład żądania wywołującego metodę: GET http://localhost:8080/api/projekty/1
    @GetMapping("/projekty/{projektId}")
    ResponseEntity<Projekt> getProjekt(@PathVariable("projektId") Integer projektId){// @PathVariable oznacza,
        return ResponseEntity.of(projektService.getProjekt(projektId));              // że wartość 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
        Projekt createdProjekt = projektService.setProjekt(projekt);         // przekazywane w ciele żądania

        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("projektId") 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("projektId") 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 przesyłana wartość parametru nazwa.
    @GetMapping(value = "/projekty", params="nazwa")
    Page<Projekt> getProjektyByNazwa(@RequestParam(name="nazwa") String nazwa, Pageable pageable) {
        return projektService.searchByNazwa(nazwa, pageable);
    }
}
```

3.6. Próba uruchomienia aplikacji za pomocą klasy *ProjectRestApiApplication* może zakończyć się błędem, jeżeli ta klasa nie będzie na najwyższym poziom w strukturze pakietów. Spring podczas inicjalizacji poszukuje adnotacji w pakiecie klasy uruchomieniowej oraz wszystkich jego podpakietach. Chociaż w parametrze adnotacji *@SpringBootApplication* można wskazywać pakiety zewnętrzne to w przypadku błędu najprostszym rozwiązaniem będzie zmiana nazwy pakietu przenosząca klasę *ProjectRestApiApplication* na najwyższy poziom w strukturze pakietów tj. *com.project*. Jeżeli musisz zmienić nazwę pakietu to w widoku *Project Explorer* zaznacz główną ikonkę pakietu zawierającego klasę uruchomieniową (*ProjectRestApiApplication*), następnie wciśnij prawy przycisk myszy i wybierz z menu *Refactor -> Rename*. W polu tekstowym okienka wpisz *com.project* i kliknij *OK*. Aby uruchomić aplikację kliknij prawym przyciskiem myszki wewnątrz okna z kodem źródłowym lub na ikonce 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.

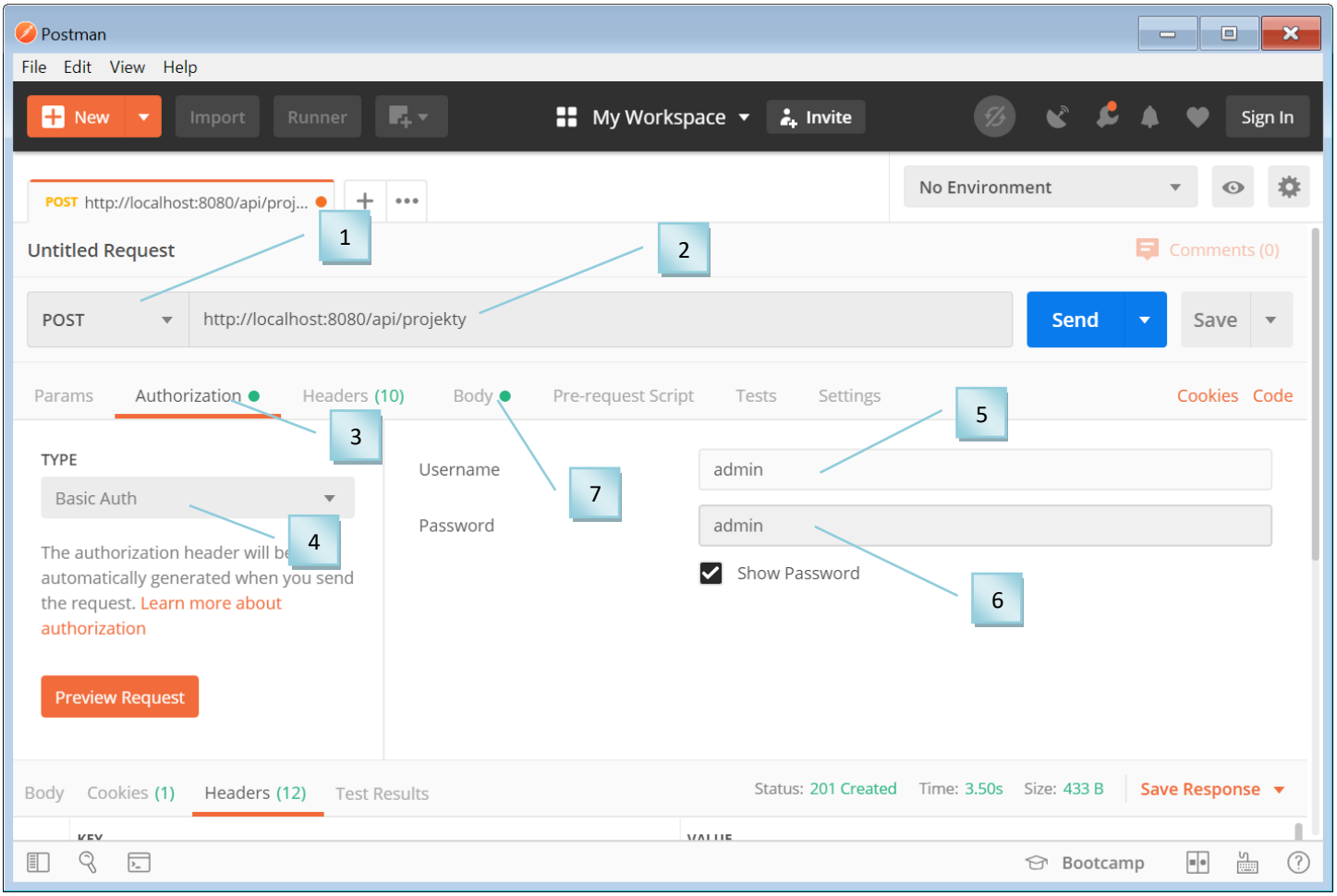
```
@GetMapping(value = "/projekty")
List<Projekt> getProjekty() {
  return projektService.getProjekty(PageRequest.of(0, 100)).getContent();
}
```

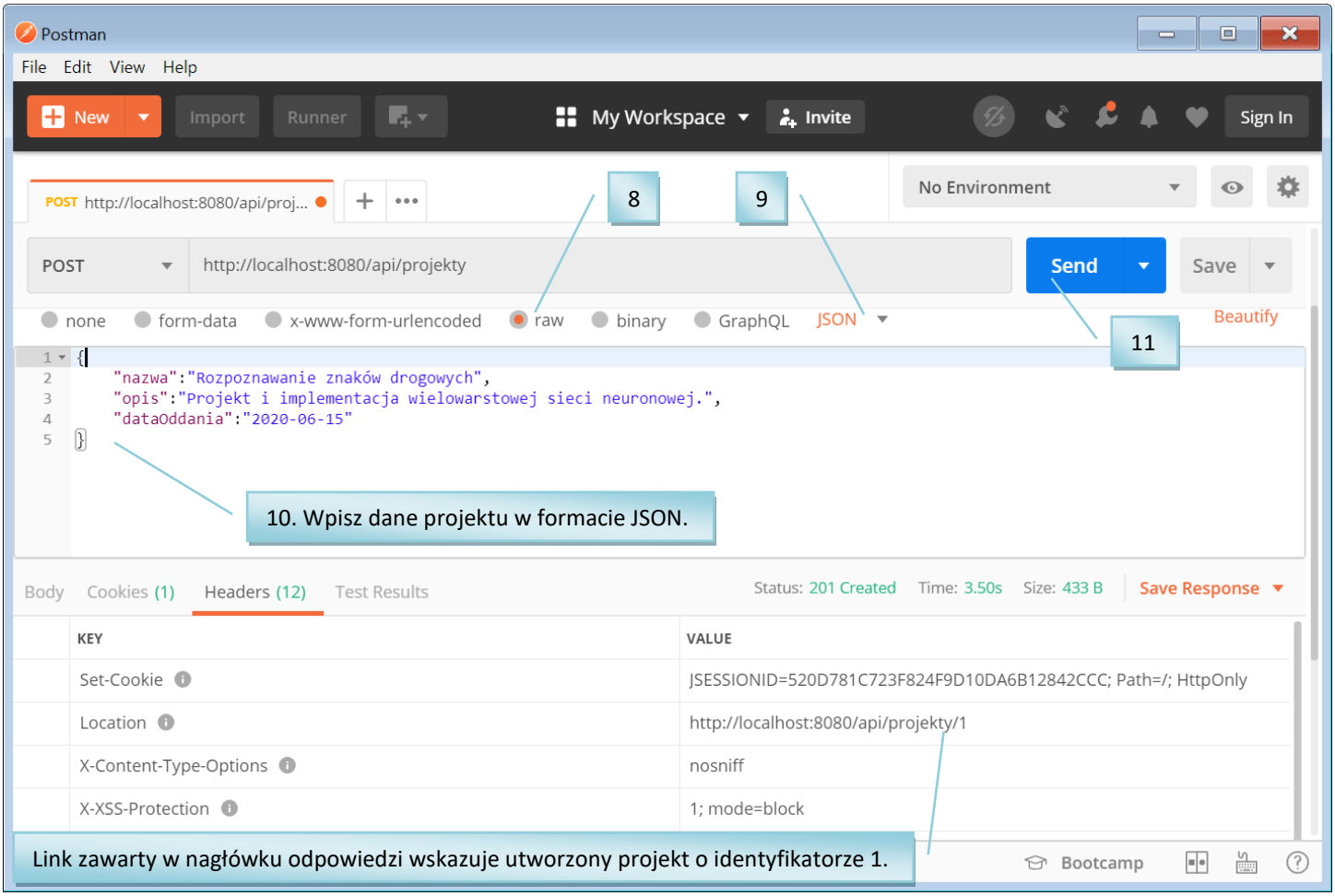
4. TESTOWANIE REST API

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 kilka 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.

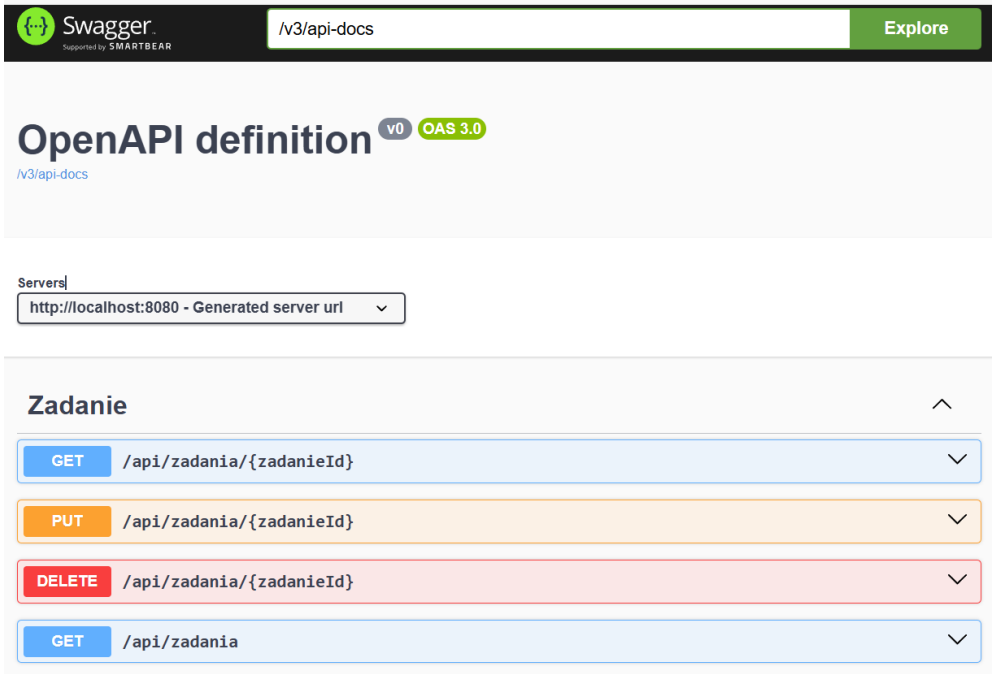

Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
Sign In
POST http://localhost:8080/api/proj...
No Environment
Untitled Request
Comments (0)
POST
http://localhost:8080/api/projekty
Send
Save
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
Code
TYPE
Basic Auth
The authorization header will be automatically generated when you send the request. Learn more about authorization
Preview Request
Username
Password
admin
admin
Show Password
1
2
3
4
5
6
7
Body
Cookies (1)
Headers (12)
Test Results
Status: 201 Created
Time: 3.50s
Size: 433 B
Save Response
Bootcamp


Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
Sign In
POST http://localhost:8080/api/proj...
No Environment
POST
http://localhost:8080/api/projekty
Send
Save
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"nazwa":"Rozpoznawanie znaków drogowych",
"opis":"Projekt i implementacja wielowarstwowej sieci neuronowej.",
"dataOddania":"2020-06-15"
}
8
9
11
10. Wpisz dane projektu w formacie JSON.
Body
Cookies (1)
Headers (12)
Test Results
Status: 201 Created
Time: 3.50s
Size: 433 B
Save Response
KEY
VALUE
Set-Cookie
JSESSIONID=520D781C723F824F9D10DA6B12842CCC; Path=/; HttpOnly
Location
http://localhost:8080/api/projekty/1
X-Content-Type-Options
nosniff
X-XSS-Protection
1; mode=block
Link zawarty w nagłówku odpowiedzi wskazuje utworzony projekt o identyfikatorze 1.
Bootcamp

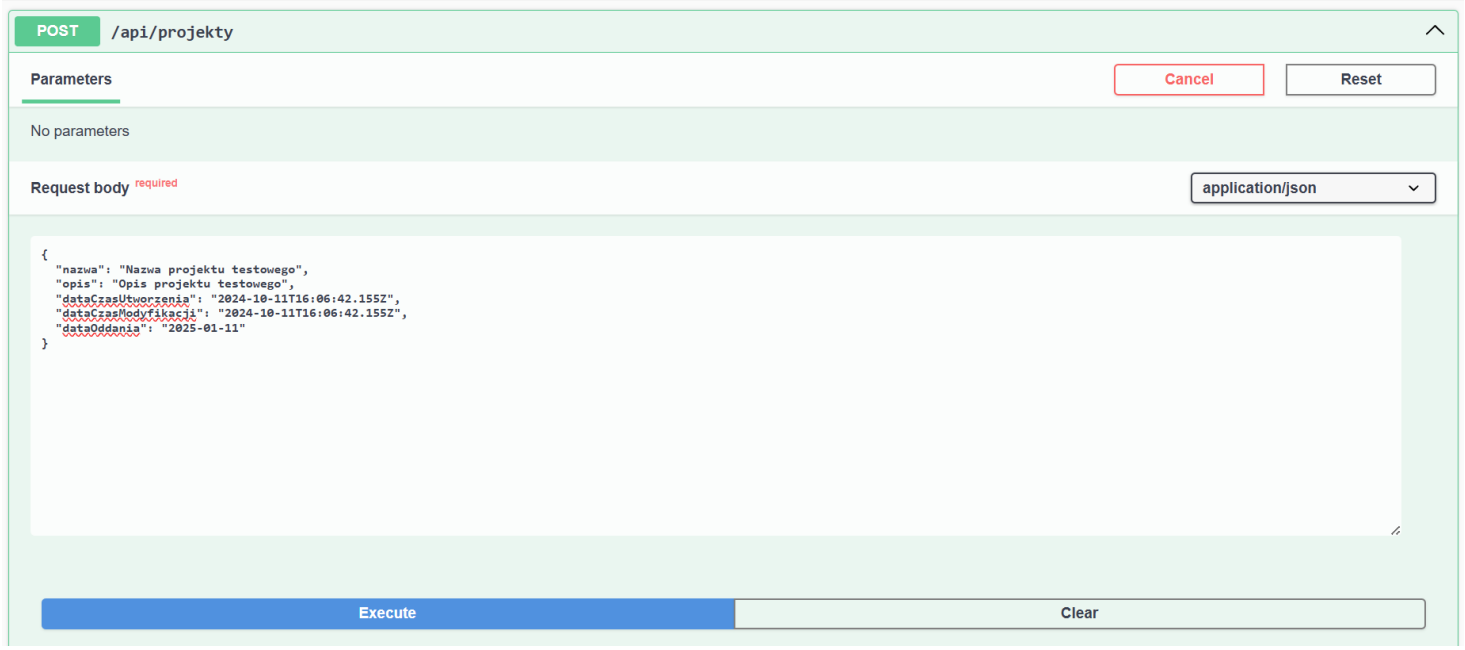
4.2. Możesz też skorzystać z biblioteki, która automatyzuje w aplikacjach REST-owych Spring Boota generowanie dokumentacji na podstawie specyfikacji OpenAPI 3. W tym celu **trzeba do pliku *project-rest-api\build.gradle* w sekcji *dependencies* dodać poniższą zależność**. Następnie należy kliknąć prawym przyciskiem myszki na głównej ikonce projektu i wybrać *Gradle -> Refresh Gradle Project*.

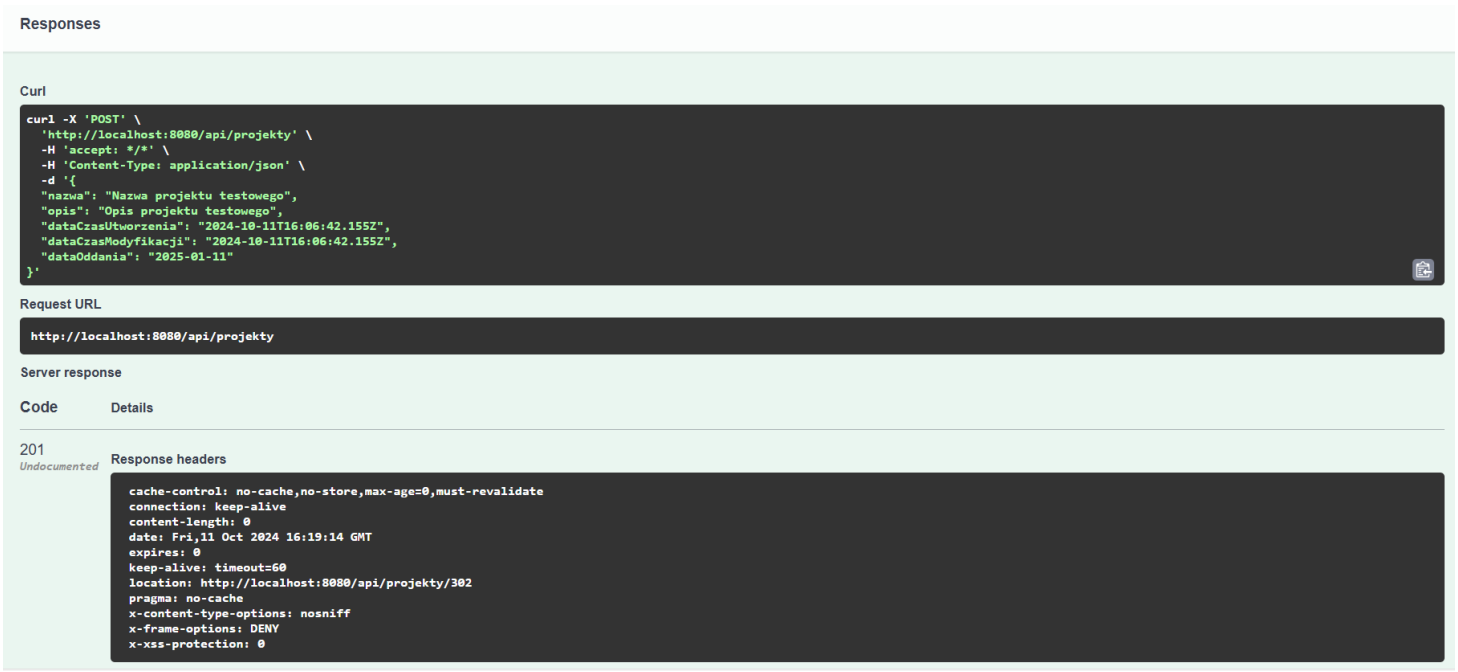
```
dependencies {
  ...
  implementation group: 'org.springdoc', name: 'springdoc-openapi-starter-webmvc-ui', version: '2.8.5'
  ...
}
```

Po uruchomieniu aplikacji wpisz w przeglądarce adres *http://localhost:8080/swagger-ui/index.html*, a później sprawdź adres *http://localhost:8080/v3/api-docs*.



Zwróć uwagę, że za pomocą wygenerowanej strony można również testować poszczególne punkty końcowe usługi REST-owej.





4.3. Testowanie kontrolerów można zautomatyzować za pomocą testów korzystających z bibliotek MockMVC i Mockito. Na początek edytuj plik *project-rest-api\build.gradle* i w bloku *test*, 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 ikonce 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;
```

```java
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.getProjekt(projekt.getProjektId()))
                .thenReturn(Optional.of(projekt));
        mockMvc.perform(get(apiPath + "/{projektId}", projekt.getProjektId()).accept(MediaType.APPLICATION_JSON))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.projektId").value(projekt.getProjektId()))
                .andExpect(jsonPath("$.nazwa").value(projekt.getNazwa()));
        verify(mockProjektService, times(1)).getProjekt(projekt.getProjektId());
        verifyNoMoreInteractions(mockProjektService);
    }

    @Test
    public void getProject_whenInvalidId_shouldReturnNotFound() throws Exception {
        Integer projektId = 2;
        when(mockProjektService.getProjekt(projektId)).thenReturn(Optional.empty());
        mockMvc.perform(get(apiPath + "/{projektId}", projektId).accept(MediaType.APPLICATION_JSON))
        .andDo(print())
        .andExpect(status().isNotFound());
        verify(mockProjektService, times(1)).getProjekt(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("$.content[0].projektId").value(projekt1.getProjektId()))
            .andExpect(jsonPath("$.content[1].projektId").value(projekt2.getProjektId()));
    verify(mockProjektService, times(1)).getProjekty(any(Pageable.class));
    verifyNoMoreInteractions(mockProjektService);
}

@Test
public void createProject_whenValidData_shouldReturnCreatedStatusWithLocation() throws Exception {
    Projekt projekt = new Projekt("Nazwa3", "Opis3",  LocalDate.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.getProjektId())));
}



@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 + "/{projektId}", 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 żądanie 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.getProjekt(projekt.getProjektId())).thenReturn(Optional.of(projekt));

		ResponseEntity<Projekt> responseEntity = projectController.getProjekt(projekt.getProjektId());

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

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

		ResponseEntity<Projekt> responseEntity = projectController.getProjekt(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.getProjekty(pageable)).thenReturn(page);

		Page<Projekt> pageWithProjects = projectController.getProjekty(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(2024, 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()));
    }

}
```

4.4. 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 JRE np.
set path="C:\eclipse-2024-09\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.4.v20240802-1551\jre\bin"
::można też wskazać katalog z plikiem JAR, jeżeli plik BAT jest przechowywany w innym miejscu
::cd C:\eclipse-2024-09\workspace\project-rest-api\build\libs
java -jar project-rest-api-0.0.1-SNAPSHOT.jar
```

FRONT-END APLIKACJI DO ZARZĄDZANIA PROJEKTAMI

Aplikacja webowa będzie wykorzystywać usługę typu REST zaimplementowaną w ramach poprzedniego zadania.

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

Uwaga! Nie trzeba tworzyć nowego projektu gradle'owskiego, można wykorzystać już istniejący np. *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
     }
}
```

5.2. Przekopiuj 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 adnotacje *@DateTimeFormat*, określające 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 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;
}
```

5.3. 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.

5.4. 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,
	@Bean		// natomiast adnotacja @Autowired i/lub konstruktor użyte w innej klasie spowodują jego wstrzyknięcie
	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());
	}
}
```

5.5. W kataloru *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 uwzględnia 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;
```

5.6. 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<>());
	}
}
```

5.7. 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)
                        .queryParam("page", pageable.getPageNumber())
                        .queryParam("size", pageable.getPageSize())
                        .queryParam("sort",
                                        ServiceUtil.getSortParams(pageable.getSort())).build().toUri();
	}
```

```
	public static UriComponentsBuilder getUriComponent(String resourcePath, Pageable pageable) {
		return getUriComponent(resourcePath)
				.queryParam("page", pageable.getPageNumber())
				.queryParam("size", pageable.getPageSize())
				.queryParam("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();
	}
}
```

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

```
package com.project.exception;

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatusCode;

public class HttpException extends RuntimeException {

	private static final long serialVersionUID = 1L;

	public HttpException(HttpStatusCode 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);
	}
}
```

5.9. 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.HttpStatusCode;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
```

```java
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(HttpStatusCode::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(HttpStatusCode::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(HttpStatusCode::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(HttpStatusCode::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(HttpStatusCode::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)
                                .queryParam("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>>(){});
        }

}
```

5.10. 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.HttpStatusCodeException;
import com.project.model.Projekt;
import com.project.service.ProjektService;
```

```java
@Controller
public class ProjectController {
   private ProjektService projektService;

   //@Autowired – przy jednym konstruktorze wstrzykiwanie jest zadaniem domyślnym, adnotacji nie jest potrzebna
   public ProjectController(ProjektService 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 (HttpStatusCodeException 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";
   }
}
```

5.11. Przykładowy szablon (*Thymeleaf*) ekranu edycji projektu (plik z katalogu *...\src\main\resources\templates*).

```html
<!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;
}
```

```css
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;
}
```

5.12. 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.
Zaimplementuj pełną funkcjonalność systemu pozwalającą dodawać, modyfikować i usuwać dane projektów, zadań i studentów, a także przypisywać zadania i studentów do projektów. Pamiętaj też o realizacji mechanizmów stronicowania i wyszukiwania.

```html
<!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%;
}
```

6. KONFIGURACJA MECHANIZMU REJESTRACJI

6.1. 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>
```

6.2. 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);
}
```

7.1. W pliku *build.gradle* dodaj zależności wskazujące na biblioteki do generowania i weryfikacji tokenów JWT oraz obsługi mechanizmu adnotacji Lomboka. Następnie w Eclipse'ie kliknij prawym przyciskiem myszki na głównej ikonce projektu i wybierz *Gradle -> Refresh Gradle Project*. Uwaga! Jeśli korzystasz z innej wersji Eclipse'a niż ta udostępniona, to powinieneś zainstalować wtyczkę umożliwiającą obsługę Lomboka w tym środowisku.

```
...
 dependencies {
       ...
       annotationProcessor 'org.projectlombok:lombok'
       compileOnly 'org.projectlombok:lombok'

       implementation group: 'io.jsonwebtoken', name: 'jjwt-api', version: '0.12.6'
       runtimeOnly group: 'io.jsonwebtoken', name: 'jjwt-impl', version: '0.12.6'
       runtimeOnly group: 'io.jsonwebtoken', name: 'jjwt-jackson', version: '0.12.6'
  }
```

7.2. W klasach modelu będą wykorzystywane adnotacje *@CreatedDate* i *@LastModifiedDate*, które pozwalają na automatyczne przypisywanie dat i czasu podczas tworzenia lub modyfikacji rekordu.

```
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;

 // ...

 @CreatedDate
 @Column(name = "dataczas_utworzenia", nullable = false, updatable = false)
 private LocalDateTime dataczasUtworzenia;


 @LastModifiedDate
 @Column(name = "dataczas_modyfikacji", insertable = false)
 private LocalDateTime dataczasModyfikacji;
```

Uwaga! Aby włączyć mechanizm *Spring Data JPA Auditing*, który automatycznie wypełnia pola takie jak te opatrzone adnotacjami *@CreatedDate* i *@LastModifiedDate*, należy użyć adnotacji *@EnableJpaAuditing*. Najprostszym sposobem jest umieszczenie jej w jakiejś istniejącej klasie konfiguracyjnej (opatrzonej adnotacją *@Configuration*) lub w głównej klasie aplikacji (tej, która posiada już adnotację @SpringBootApplication). Alternatywnie, w celu lepszej organizacji kodu, można utworzyć dedykowaną klasę konfiguracyjną (np. *JpaConfig* w pakiecie *com.project.config*) i tam umieścić adnotację *@EnableJpaAuditing*. Tworzenie nowej klasy nie jest więc obligatoryjne.

```
package com.project.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@Configuration
@EnableJpaAuditing
public class JpaConfig {

}
```

Poza tym w klasach encyjnych używających adnotacji *@CreatedDate* i *@LastModifiedDate* trzeba też dodać przed definicją klasy adnotację *@EntityListeners(AuditingEntityListener.class)*, tak jak to zostało pokazane poniżej.

```
package com.project.model;

...
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import jakarta.persistence.EntityListeners;
...

@Entity
@Table(name="role")
@EntityListeners(AuditingEntityListener.class)
public class Role {
   ...

 @CreatedDate
 @Column(name = "dataczas_utworzenia", nullable = false, updatable = false)
 private LocalDateTime dataczasUtworzenie;


 @LastModifiedDate
 @Column(name = "dataczas_modyfikacji", insertable = false)
 private LocalDateTime dataczasModyfikacji;


   ...
}
```

7.3. Utwórz klasę *Role*, która będzie odwzorowywać tabelę bazodanową do przechowywania ról (uprawnień) użytkownika.

```
package com.project.model;

import java.time.LocalDateTime;
import java.util.Set;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

/*
 Konfiguracja globalna audytowania przy użyciu @EnableJpaAuditing nie wymaga dodawania
@EntityListeners(AuditingEntityListener.class) do każdej encji. Spring automatycznie dodaje odpowiednich
listenerów do wszystkich encji, które mają adnotacje takie jak @CreatedDate, @LastModifiedDate itp.
Zatem jej użycie poprawia jedynie czytelność kodu, ułatwia zrozumienie intencji programisty.
 */
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "role")
@EntityListeners(AuditingEntityListener.class)
public class Role {
	@Id
	@GeneratedValue
	@Column(name="role_id")
	private Integer roleId;
```

```
        @Column(unique = true)
        private String name;

        @JsonIgnore
        @ManyToMany(mappedBy = "roles")
        private Set<Student> studenci;
        @CreatedDate
        @Column(name = "dataczas_utworzenia", nullable = false, updatable = false)
        private LocalDateTime dataczasUtworzenia;

        @LastModifiedDate
        @Column(name = "dataczas_modyfikacji", insertable = false)
        private LocalDateTime dataczasModyfikacji;
}
```

7.4. Proces rejestracji i weryfikacji uprawnień użytkownika wymaga obecności odpowiednich ról w bazie danych. Dlatego przed pierwszym uruchomieniem aplikacji należy je utworzyć. Można to zrobić ręcznie, dodając do bazodanowej tabeli *role* odpowiednie rekordy (np. *USER* i *ADMIN*), lub zastosować automatyczną inicjalizację – czyli klasę, która po uruchomieniu aplikacji webowej sprawdzi, czy wymagane role istnieją, i w razie potrzeby je utworzy. Poniżej przedstawiono przykład klasy *DataInitializer* (utworzonej w pakiecie *com.project.init*) inicjalizującej role użytkownika. Automatyczne uruchomienie metody *init()* zapewnia adnotacja *@EventListener(ApplicationReadyEvent.class)*

```
package com.project.init;

import java.util.List;
import org.springframework.boot.context.event.ApplicationReadyEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import com.project.model.Role;
import com.project.repository.RoleRepository;
import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
@Component
public class DataInitializer {
	private final RoleRepository roleRepository;

	@EventListener(ApplicationReadyEvent.class)
	public void init() {
		List.of("USER", "ADMIN")
		.forEach(role -> roleRepository
			.findByName(role)
			.orElseGet(() -> roleRepository.save(Role.builder().name(role).build())));
	}
}
```

7.5. W klasie *Student* dodaj dwa pola - dla hasła i ról użytkownika.

```
//...
public class Student {
    //...

    @JsonProperty(access = Access.WRITE_ONLY)
    @Size(min=8, max=64, message="Hasło musi składać się z przynajmniej {min} i nie przekraczać {max} znaków")
    private String password;

                                        // dzięki FetchType.EAGER dane związane z tą relacją
    @ManyToMany(fetch = FetchType.EAGER) // będą natychmiast wczytywane, wraz z głównym obiektem tj. studentem
    @JoinTable(name = "student_role",
            joinColumns = {@JoinColumn(name="student_id")},
            inverseJoinColumns = {@JoinColumn(name = "role_id")})
    private Set<Role> roles;

    //...
}
```

Poza tym z klasy *Student* usuń wszystkie akcesory i mutatory (tzw. *gettery* i *settery)* oraz konstruktory, a później wstaw adnotacje Lomboka je zastępujące tj.:

```
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Student {
   //...
}
```

Ponadto pole *email* musi być teraz obligatoryjne, a przechowywane wartości w kolumnie bazodanowej unikatowe, niepowtarzalne, ponieważ adres e-mail będzie wykorzystywany do logowania.

```
@NotEmpty(message = "Nie podano adresu e-mail")
@Email(message = "Niepoprawny format adresu e-mail")
@Column(length = 100, nullable = false, unique = true)
private String email;
```

Na koniec załóż indeksy bazodanowe, które będą przyśpieszać wyszukiwanie studentów za pomocą ich adresu e-mail, nazwiska lub numeru indeksu:

```
@Table(name = "student",
     indexes = { @Index(name = "idx_nazwisko", columnList = "nazwisko", unique = false),
                 @Index(name = "idx_email", columnList = "email", unique = true),
                 @Index(name = "idx_nr_indeksu", columnList = "nr_indeksu", unique = true)})
public class Student {
    //...
}
```

7.6. Do klasy *Student* dodaj implementację interfejsu *UserDetails*. W tym interfejsie znajduje się również abstrakcyjna metoda *getPassword()*, ale jej implementacja jest realizowana dzięki zastosowaniu adnotacji *@Getter* (Lomboka). Interfejs *UserDetails* posiada też domyślne implementacje metod *isEnabled()*, i*sAccountNonLocked()*, *isAccountNonExpired()* i *isCredentialsNonExpired()*. W razie potrzeby można je również nadpisać, aby spełnić wymagania specyficzne dla tworzonej aplikacji.

```
//...
public class Student implements UserDetails {
      //...

      @Override
      public Collection<? extends GrantedAuthority> getAuthorities() {
             return this.roles
                    .stream()
                    .map(r -> new SimpleGrantedAuthority(r.getName()))
                    .collect(Collectors.toList());
      }

      @Override
      public String getUsername() {
            return this.email;
      }
}
```

7.7. Ponieważ adres e-mail będzie używany do logowania, należy zaimplementować metody umożliwiające pobieranie danych użytkownika z bazy danych na podstawie tego adresu. W tym celu dodaj metodę *Optional<Student> findByEmail(String email)* do *StudentRepository*, a także definicję metody *Optional<Student> findByEmail(String email)* do interfejsu *StudentService* i jej implementację do *StudentServiceImpl*.

7.8. Opracuj mechanizm zapewniający globalną obsługę błędów walidacji danych, generowanych we wszystkich kontrolerach aplikacji. W tym celu utwórz pakiet *com.project.validation* i dodaj do niego następujące elementy:

- Rekord *Violation* służący do przekazywania szczegółowych informacji o błędach.

```
package com.project.validation;

public record Violation(String fieldName, String errorMessage){}
```

- Serwis *ValidationService* odpowiedzialny za realizację walidacji danych za pomocą metody *validate*, która będzie generować wyjątek typu *ConstraintViolationException* w przypadku nieprawidłowych danych w przekazanym do tej metody obiekcie.

```
package com.project.validation;

import org.springframework.stereotype.Service;
import jakarta.validation.ConstraintViolationException;
import jakarta.validation.Validator;
import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class ValidationService<T> {

	private final Validator validator;

	public void validate(T object) {
		var violations = validator.validate(object);
		if (!violations.isEmpty())
			throw new ConstraintViolationException(violations);
	}

}
```

- Klasę *ValidationExceptionHandler* opatrzoną adnotacją *@RestControllerAdvice,* której metody oznaczone adnotacją *@ExceptionHandler* będą obsługiwać wyjątki określonych w tej adnotacji typów.

```
package com.project.validation;

import java.time.format.DateTimeParseException;
import java.util.Set;
import java.util.stream.Collectors;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import jakarta.validation.ConstraintViolationException;

@RestControllerAdvice
public class ValidationExceptionHandler {

	@ExceptionHandler(ConstraintViolationException.class)
	public ResponseEntity<?> handleConstraintValidationException(ConstraintViolationException e) {
		Set<Violation> violations = e.getConstraintViolations()
				.stream()
				.map(v -> new Violation(v.getPropertyPath().toString(), v.getMessage()))
				.collect(Collectors.toSet());
		return ResponseEntity
				.badRequest()
				.body(violations);
	}
}
```

7.9. W pakiecie *com.project.config* utwórz/zmodyfikuj klasę konfiguracyjną *SecurityConfig*. Poniżej przedstawiono jej zawartość.

```
package com.project.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import com.project.service.StudentService;
import lombok.RequiredArgsConstructor;

@Configuration
@RequiredArgsConstructor
public class SecurityConfig {
	private final StudentService studentService;

	@Bean
	public UserDetailsService userDetailsService() {
		return userName -> studentService
				.searchByEmail(userName)
				.orElseThrow(() -> new UsernameNotFoundException(
						String.format("User '%s' not found!", userName)));
	}

	@Bean
	public PasswordEncoder passwordEncoder() {
		return new BCryptPasswordEncoder();
	}

	@Bean
	public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
			throws Exception {
		return config.getAuthenticationManager();
	}

}
```

7.10. W pliku *resources\application.properties* dodaj cztery parametry - klucze i czasy ważności tokenów (dostępowego i odświeżania).

```
# Testowy klucz można wygenerować np. za pomocą https://acte.ltd/utils/randomkeygen (jako Encryption key 256-bit)
jwt.access-secret-key=Xn2r5u8x/A?D(G-KaPdSgVkYp3s6v9y$
jwt.access-token-validity-in-min=15
jwt.refresh-secret-key=2s5v8y/B?E(G+KbPeShVmYq3t6w9z$C&
jwt.refresh-token-validity-in-min=10
```

7.11. Utwórz *JwtService* - klasę do m.in. generowania i weryfikacji tokenów JWT.

```
package com.project.config;

import java.nio.charset.StandardCharsets;
import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.Date;
import java.util.Map;
import java.util.function.Function;
import javax.crypto.SecretKey;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
```

```java
@Service
public class JwtService {

    // W zadaniu tylko dwa klucze służą do generowania i weryfikacji tokenów dla wszystkich
    // użytkowników, jest to częste rozwiązanie w praktycznych zastosowaniach. Jednak te dwa
    // klucze mogą być również dla każdego użytkownika odrębne, unikatowe i przechowywane w bazie.
    private final SecretKey accessSecretKey;
    private final Integer accessTokenValidityInMin;
    private final SecretKey refreshSecretKey;
    private final Integer refreshTokenValidityInMin;

    // Wartości zdefinowane w application.properties są automatycznie wstrzykiwane dzięki @Value
    public JwtService(@Value("${jwt.access-secret-key}") String accesSecretKey,
            @Value("${jwt.access-token-validity-in-min}") Integer accessTokenValidityInMin,
            @Value("${jwt.refresh-secret-key}") String refreshSecretKey,
            @Value("${jwt.refresh-token-validity-in-min}") Integer refreshTokenValidityInMin) {
      this.accessSecretKey = Keys.hmacShaKeyFor(accesSecretKey.getBytes(StandardCharsets.UTF_8));
      this.accessTokenValidityInMin = accessTokenValidityInMin;
      this.refreshSecretKey=Keys.hmacShaKeyFor(refreshSecretKey.getBytes(StandardCharsets.UTF_8));
      this.refreshTokenValidityInMin = refreshTokenValidityInMin;
    }

    private String generateToken(Map<String, Object> extraClaims, UserDetails userDetails,
                                 SecretKey secretKey, Integer tokenValidityInMin) {
       final Instant now = Instant.now();
       final Instant expiration = now.plus(tokenValidityInMin, ChronoUnit.MINUTES);
       final var authorities = userDetails.getAuthorities()
                   .stream()
                   .map(GrantedAuthority::getAuthority)
                   .toList();
       return Jwts.builder()
                .claims(extraClaims)
                .subject(userDetails.getUsername())
                .issuedAt(Date.from(now))
                .expiration(Date.from(expiration))
               .claim("authorities", authorities)
                .signWith(secretKey, Jwts.SIG.HS256)
                .compact();
    }
    // Claims-y to tzw. deklaracje, będące zbiorem par klucz-wartość
    public Claims extractAllClaims(String token, SecretKey secretKey) {
       // Jeżeli zmodyfikowany token zostanie przesłany w żądaniu HTTP to dostaniemy wyjątek
       // przy próbie odczytu jego zawartości (SignatureException: JWT signature does not match
       // locally computed signature. JWT validity cannot be asserted and should not be trusted.)
       return
           Jwts.parser()
              .verifyWith(secretKey)
              .build()
              .parseSignedClaims(token)
              .getPayload();
    }

    //Ostatnim parametrem może być referencja do metody np. Claims::getSubject
    private <T> T extractClaim(String token, SecretKey secretKey,
                               Function<Claims, T> claimsResolver) {
       final Claims claims = extractAllClaims(token, secretKey);
       return claimsResolver.apply(claims);
    }

    private String extractUserName(String token, SecretKey secretKey) {
       return extractClaim(token, secretKey, Claims::getSubject);
    }

    public String extractUserNameFromAccessToken(String token) {
       return extractUserName(token, this.accessSecretKey);
    }
```

```
    public String extractUserNameFromRefreshToken(String token) {
        return extractUserName(token, this.refreshSecretKey);
    }

    private boolean isTokenValid(String token, SecretKey secretKey, UserDetails userDetails) {
        final Claims claims = extractAllClaims(token, secretKey);
        final Date expiration = claims.getExpiration();
        final String userName = claims.getSubject();
        boolean valid = !expiration.before(Date.from(Instant.now()));
        return userName.equals(userDetails.getUsername()) && valid;
    }

    public boolean isAccessTokenValid(String token, UserDetails userDetails) {
        return isTokenValid(token, this.accessSecretKey, userDetails);
    }

    public boolean isRefreshTokenValid(String token, UserDetails userDetails) {
        return isTokenValid(token, this.refreshSecretKey, userDetails);
    }

    public String generateAccessToken(UserDetails userDetails) {
        return generateAccessToken(Map.of(), userDetails);
    }

    public String generateAccessToken(Map<String, Object> extraClaims, UserDetails userDetails) {
        return generateToken(extraClaims, userDetails,
                                        this.accessSecretKey, this.accessTokenValidityInMin);
    }

    public String generateRefreshToken(UserDetails userDetails){
        return generateRefreshToken(Map.of(), userDetails);
    }

    public String generateRefreshToken(Map<String, Object> extraClaims, UserDetails userDetails) {
        return generateToken(extraClaims, userDetails,
                                       this.refreshSecretKey, this.refreshTokenValidityInMin);
    }
}
```

7.12. Dodaj klasę *JwtAuthFilter* dziedzicząca z *OncePerRequestFilter*. Następnie nadpisz metodę *doFilterInternal* filtrującą każde przesyłane na serwer żądanie HTTP. W jej wnętrzu należy weryfikować przesyłany w nagłówku żądania token.

```
package com.project.config;

import java.io.IOException;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import io.jsonwebtoken.ExpiredJwtException;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;

@Component
@RequiredArgsConstructor
public class JwtAuthFilter extends OncePerRequestFilter {

  private final JwtService jwtService;
```

```java
// automatycznie wstrzykiwany (zdefiniowany jako @Bean w SecurityConfig)
private final UserDetailsService userDetailsService;


 // gdy przychodzi żądanie HTTP to filtr zdefiniowany w metodzie doFilterInternal(...)
 // je przechwytuje i sprawdza, czy zawiera poprawny token JWT w nagłówku Authorization
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                FilterChain filterChain) throws ServletException, IOException {

    final String authHeader = request.getHeader(HttpHeaders.AUTHORIZATION); //np. Bearer eyJhb...

    final String prefix = "Bearer ";
    if (authHeader == null || !authHeader.startsWith(prefix)) {
       // brak tokenu w nagłówku, tylko przetwarzanie przez kolejne filtry w łańcuchu
       filterChain.doFilter(request, response);
       // i przerwanie realizacji metody
       return;
    }


    final String token = authHeader.substring(prefix.length());

    try {
       final String userName = jwtService.extractUserNameFromAccessToken(token);
       if (userName != null //druga cześć warunku sprawdza czy użytkownik po raz pierwszy jest uwierzytelniany
                        && SecurityContextHolder.getContext().getAuthentication() == null) {

          UserDetails userDetails = this.userDetailsService.loadUserByUsername(userName);

          if (jwtService.isAccessTokenValid(token, userDetails)) {

             // Tworzenie obiektu Authentication (authToken). Klasa UsernamePasswordAuthenticationToken
             // ze Spring Security jest jedną z implementacji interfejsu Authentication i służy do
             // w reprezentowania procesu uwierzytelnienia użytkownika na podstawie nazwy użytkownika
             // i hasła. Pierwszy parametr konstruktora reprezentuje tożsamość użytkownika, np. login
             // lub obiekt użytkownika załadowany z bazy danych. Drugi zawiera dane uwierzytelnienia,
             // najczęściej jest to hasło użytkownika. Po uwierzytelnieniu zazwyczaj dla bezpieczeństwa
             // wartość tego pola zostaje wyczyszczona. W naszym przypadku od razu przekazujemy null.
             // Ostatni parametr to kolekcja ról (uprawnień) przypisanych do użytkownika.

              UsernamePasswordAuthenticationToken authToken =
                              new UsernamePasswordAuthenticationToken(userDetails, null,
                                                                      userDetails.getAuthorities());
               authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

              // Obiekt Authentication (authToken) jest zapisywany w SecurityContext, który jest
              // zarządzany przez SecurityContextHolder. Dzięki temu inne komponenty aplikacji
              // mogą łatwo uzyskać dostęp do danych uwierzytelnienia.
              SecurityContextHolder.getContext().setAuthentication(authToken);
              // SecurityContextHolder udostępnia dane o uwierzytelnionym użytkowniku i
              // w obrębie jednego wątku zawsze zwróci ten sam SecurityContext. Domyślnie
              // używa obiektu ThreadLocal do przechowywania kontekstu bezpieczeństwa,
              // co oznacza, że kontekst bezpieczeństwa jest zawsze dostępny dla metod w
              // tym samym wątku wykonania.
          }
       }
    } catch(ExpiredJwtException e) {
       response.setStatus(HttpStatus.UNAUTHORIZED.value());
       return;
    }
    // o tym trzeba pamiętać, aby umożliwić przetwarzanie przez kolejne filtry w łańcuchu
    filterChain.doFilter(request, response);
 }
```

7.13. Zdefiniuj klasę *SecurityWebConfig* z metodą *securityFilterChain*.

```
package com.project.config;

import static org.springframework.security.config.Customizer.withDefaults;
import java.util.Arrays;
import java.util.Collections;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpHeaders;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;
import lombok.RequiredArgsConstructor;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityWebConfig {

private final JwtAuthFilter jwtAuthFilter;

	@Bean
	public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
		return http
			.cors(withDefaults())
			.csrf(csrf -> csrf.disable()) //cross-site request forgery
			.authorizeHttpRequests(auth -> auth
				.requestMatchers(
					"/api/register",
					"/api/login",
					"/api/refresh",        // Kolejne adresy tego bloku powiązane są ze
					"/v2/api-docs",        // Swaggerem (OpenAPI), przeznaczone tylko do
					"/v3/api-docs",        // testów. Jeśli Swagger jest dostępny
					"/v3/api-docs/**",     // publicznie, może ujawniać szczegóły API
					"/swagger-resources",  // osobom nieuprawionym!
					"/swagger-resources/**",
					"/configuration/ui",
					"/configuration/security",
					"/swagger-ui/**",
					"/webjars/**",
					"/swagger-ui.html").permitAll()



				.requestMatchers("/api/studenci/**").hasAuthority("ADMIN")
				.anyRequest().authenticated())
				.sessionManagement(session -> session
						.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
			.addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
			.build();
	}
```

Jeżeli masz już utworzony front-end np. w Angularze lub React to możesz również w tej klasie zdefiniować kolejną metodę konfigurującą tzw. CORS-y. Pamiętaj przy tym o wpisaniu właściwego adresu w metodzie *setAllowedOrigins(...)*.

```
	@Bean
	public CorsFilter corsFilter() {
		final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
		final CorsConfiguration config = new CorsConfiguration();
		config.setAllowCredentials(true);
		 //--> TODO Wpisz właściwy adres
		config.setAllowedOrigins(Collections.singletonList("http://localhost:4200")); //<--
		config.setAllowedHeaders(Arrays.asList(
				HttpHeaders.ORIGIN,
				HttpHeaders.CONTENT_TYPE,
				HttpHeaders.ACCEPT,
				HttpHeaders.AUTHORIZATION
		));
		config.setAllowedMethods(Arrays.asList(
				"GET",
				"POST",
				"DELETE",
				"PUT",
				"PATCH"
		));
		source.registerCorsConfiguration("/**", config);
		return new CorsFilter(source);
	}
```

7.14. Utworz pakiet *com.project.auth* z klasą *Credentials* do przesyłania danych logowania.

```
package com.project.auth;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Credentials {
   @NotBlank(message = "Nie podano adresu e-mail")
   @Email(message = "Niepoprawny format adresu e-mail")
   private String email;

   @NotNull(message = "Nie podano hasła")
   @Size(min=8, max=25, message="Hasło musi składać się z przynajmniej {min} znaków i nie przekraczać {max}")
   private String password;
}
```

Ponadto w tym pakiecie dodaj klasę *Tokens* do przesyłania tokenów.

```
package com.project.auth;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Tokens {
	private String accessToken;
	private String refreshToken;
}
```

7.15. W pakiecie *com.project.auth* utwórz interfejs *AuthService* z metodami *register*, *authenticate* i *refreshTokens*,

```
package com.project.auth;
import com.project.model.Student;

public interface AuthService {
	void register(Student student);
	Tokens authenticate(Credentials credentials);
	Tokens refreshTokens(Tokens tokens);
}
```

a jego implementację w klasie *AuthServiceImpl*. Z tego serwisu będzie korzystał kontroler *AuthController* obsługujący żądania związane z rejestracją i uwierzytelnianiem użytkowników.

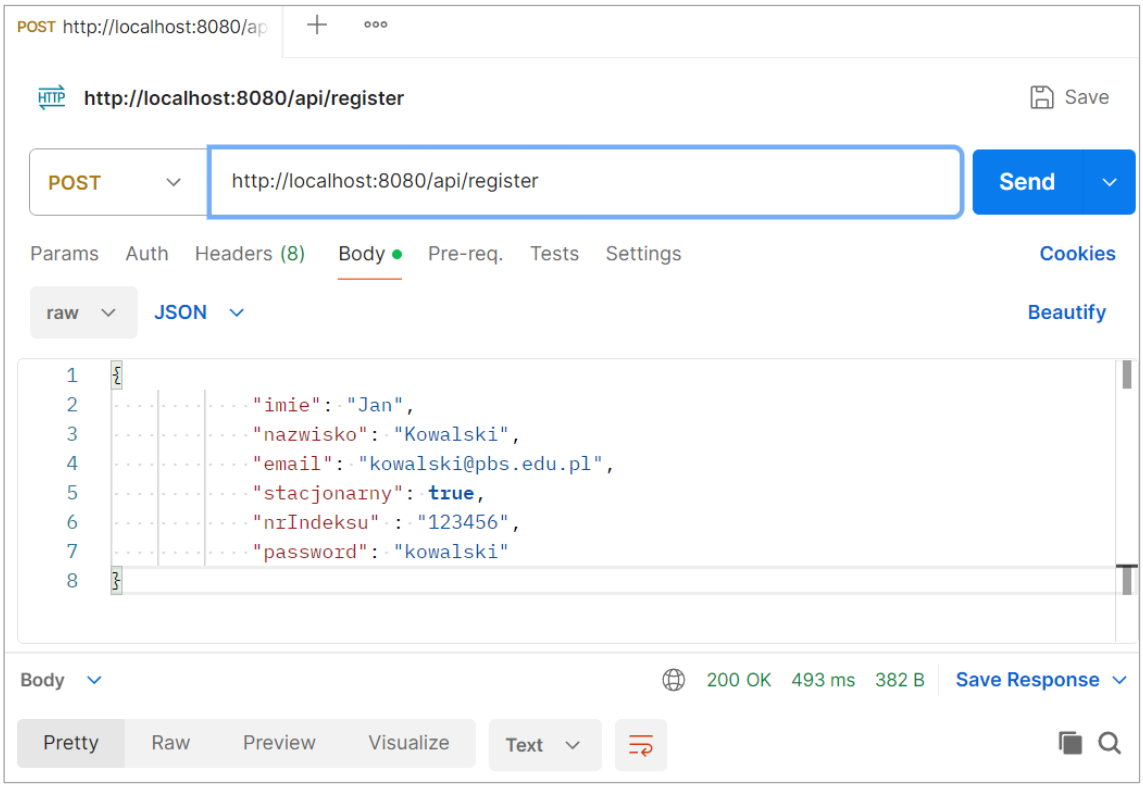
```
package com.project.auth;

import java.util.Set;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;
import com.project.config.JwtService;
import com.project.model.Student;
import com.project.repository.RoleRepository;
import com.project.service.StudentService;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class AuthServiceImpl implements AuthService {
	private final StudentService studentService;
	private final RoleRepository roleRepository;
	private final PasswordEncoder passwordEncoder;
	private final JwtService jwtService;
	private final AuthenticationManager authenticationManager;

	@Override
	public void register(Student student) {
	   student.setPassword(passwordEncoder.encode(student.getPassword()));
	   var role = roleRepository
	               .findByName("USER")
	               .orElseThrow(() -> new IllegalStateException("Role USER was not initiated"));
	   student.setRoles(Set.of(role));
	   studentService.setStudent(student);
	}
```

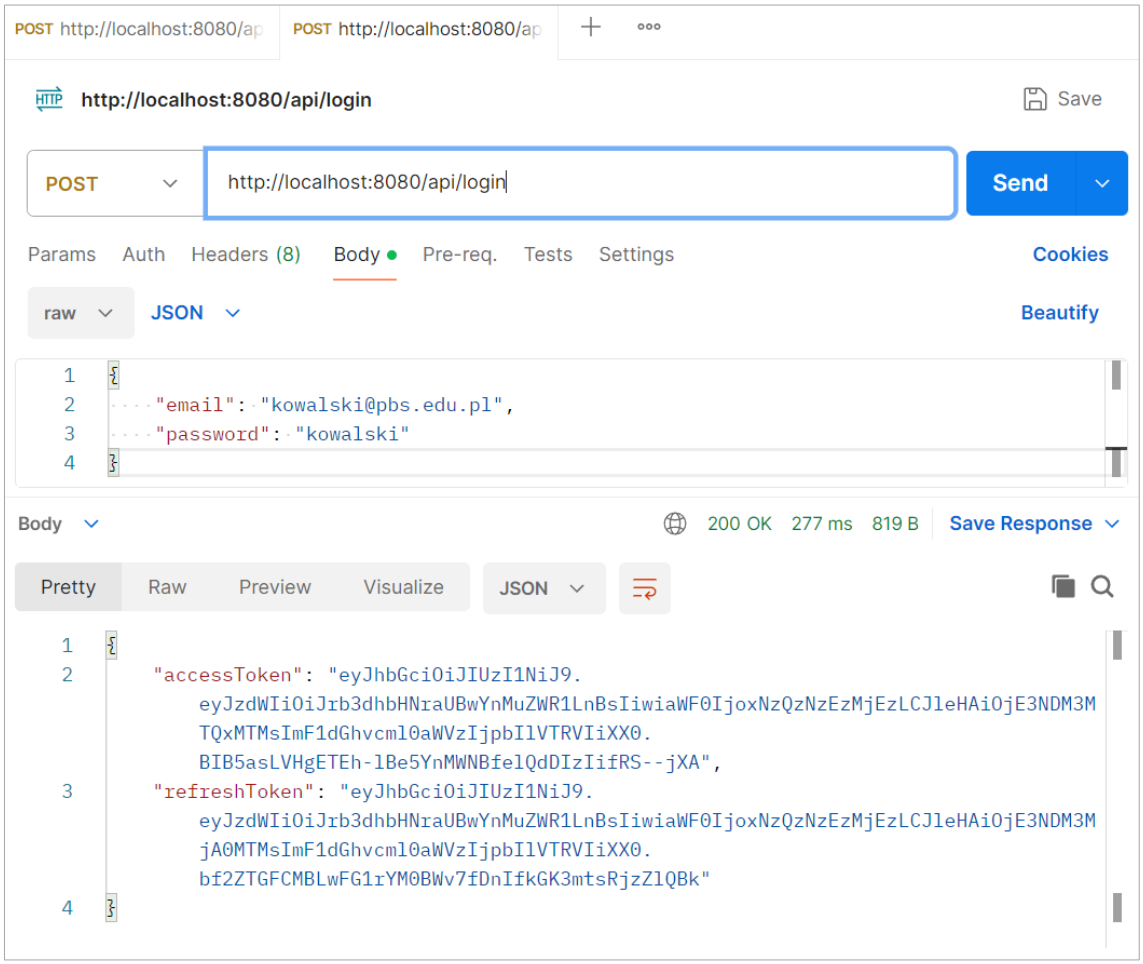
```
    @Override
    public Tokens authenticate(Credentials credentials) {
        return authenticate(credentials.getEmail(), credentials.getPassword());
    }

    private Tokens authenticate(@NonNull String email, @NonNull String password) {
        // uwierzytelnianie użytkownika na podstawie podanego adresu e-mail i hasła,
        // poprzez delegację weryfikacji do komponentu AuthenticationManager
        // skonfigurowanego w kontekście Spring Security
        authenticationManager
                        .authenticate(new UsernamePasswordAuthenticationToken(email, password));
        var user = studentService
                    .searchByEmail(email)
                    .orElseThrow(() ->
                      new UsernameNotFoundException(String.format("User '%s' not found!", email)));
        var accessToken = jwtService.generateAccessToken(user);
        var refreshToken = jwtService.generateRefreshToken(user);
        return Tokens.builder()
                  .accessToken(accessToken)
                  .refreshToken(refreshToken)
                  .build();
    }

    @Override
    public Tokens refreshTokens(Tokens tokens) {
        final String refreshToken = tokens.getRefreshToken();
        if (refreshToken == null || refreshToken.isBlank())
             throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Brak tokenu odświeżania");
        final String prefix = "Bearer ";
        final String token = refreshToken.startsWith(prefix) ?
                                          refreshToken.substring(prefix.length()) : refreshToken;
        final String email = jwtService.extractUserNameFromRefreshToken(token);
        if (email == null || email.isBlank())
             throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
                                                    "Niepoprawny format tokenu odświeżania");
        var user = studentService.searchByEmail(email)
                        .orElseThrow(() -> new UsernameNotFoundException(
                                                  String.format("User '%s' not found!", email)));
        if (!jwtService.isRefreshTokenValid(token, user))
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED,
                                                       "Token odświeżania stracił ważność");
        var accessToken = jwtService.generateAccessToken(user);
        var newRefreshToken = jwtService.generateRefreshToken(user);
        return Tokens.builder()
                  .accessToken(accessToken)
                  .refreshToken(newRefreshToken)
                  .build();
    }
}
```

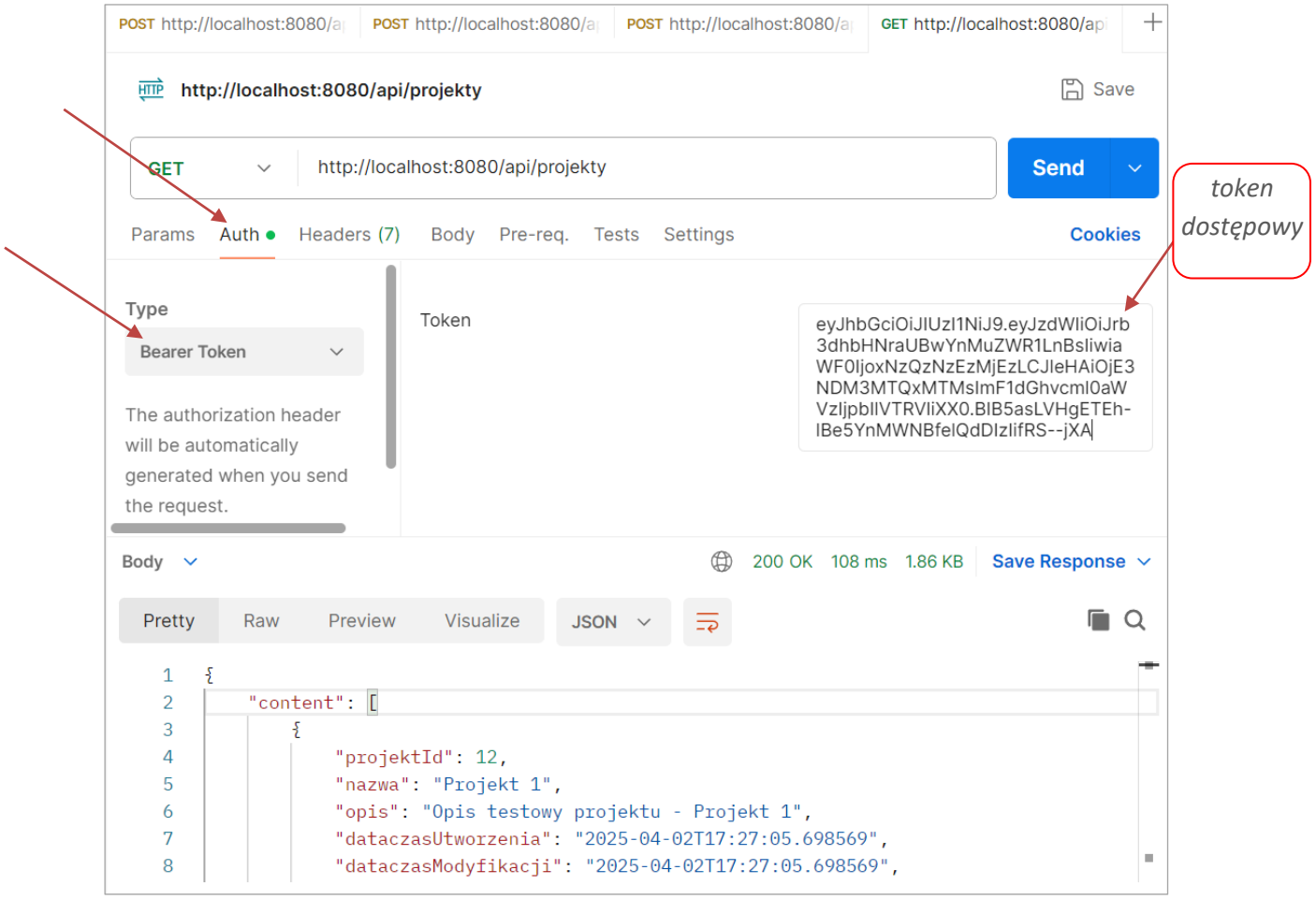
7.16. W pakiecie *com.project.auth* zaimplementuj kontroler *AuthController* obsługujący rejestrację, logowanie i odświeżanie tokenu.

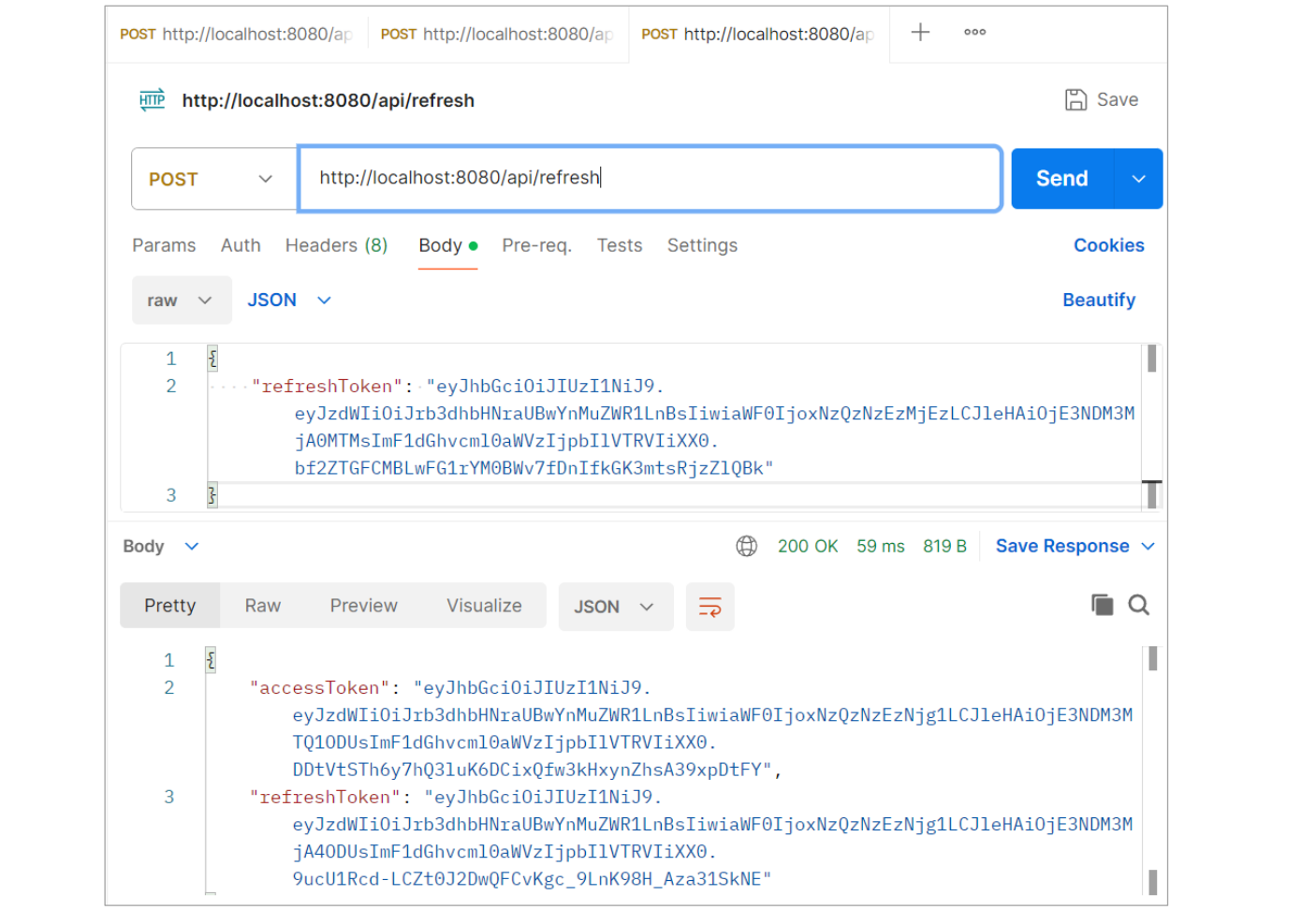
```
package com.project.auth;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.project.model.Student;
import com.project.validation.ValidationService;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api")
@Tag(name = "Auth")
public class AuthController {

	private final AuthService authService;
	private final ValidationService<Student> studentValidator;

	@PostMapping("/register")
	public ResponseEntity<Void> register(@RequestBody Student student){
	   studentValidator.validate(student);
	   authService.register(student);
	   return new ResponseEntity<Void>(HttpStatus.OK);
	}

	//Sprawdź zawartość zwracanego tokenu np. na https://jwt.io/
	@PostMapping("/login")
	public ResponseEntity<Tokens> login(@RequestBody Credentials credentials){
	   return ResponseEntity.ok(authService.authenticate(credentials));
	}

	@PostMapping("/refresh")
	public ResponseEntity<Tokens> refreshToken(@RequestBody Tokens tokens) {
	   return ResponseEntity.ok(authService.refreshTokens(tokens));
	}
}
```

7.17. Przetestuj działanie utworzonego rozwiązania w Postmanie .






POST http://localhost:8080/a
POST http://localhost:8080/a
POST http://localhost:8080/a
GET http://localhost:8080/ap
http://localhost:8080/api/projekty
Save
GET
http://localhost:8080/api/projekty
Send
token dostępowy
Params
Auth
Headers (7)
Body
Pre-req.
Tests
Settings
Cookies
Type
Bearer Token
The authorization header will be automatically generated when you send the request.
Token
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJrb3dhbHNraUBwYnMuZWR1LnBsIiwiaWF0IjoxNzQzNzEzMjEzLCJleHAiOjE3NDM3MTQxMTMsImF1dGhvcml0aWVzIjpbIlVTRVIiXX0.BIB5asLVHgETEh-lBe5YnMWNBfelQdDIzlifRS--jXA
Body
200 OK
108 ms
1.86 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"content": [
{
"projektId": 12,
"nazwa": "Projekt 1",
"opis": "Opis testowy projektu - Projekt 1",
"dataczasUtworzenia": "2025-04-02T17:27:05.698569",
"dataczasModyfikacji": "2025-04-02T17:27:05.698569",


POST http://localhost:8080/ap
POST http://localhost:8080/ap
POST http://localhost:8080/ap
http://localhost:8080/api/refresh
Save
POST
http://localhost:8080/api/refresh
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"refreshToken": "eyJhbGciOiJIUzI1NiJ9.
eyJzdWIiOiJrb3dhbHNraUBwYnMuZWR1LnBsIiwiaWF0IjoxNzQzNzEzMjEzLCJleHAiOjE3NDM3MjA0MTMsImF1dGhvcml0aWVzIjpbIlVTRVIiXX0.
bf2ZTGFCMBLwFG1rYM0BWv7fDnIfkGK3mtsRjzZlQBk"
}
Body
200 OK
59 ms
819 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"accessToken": "eyJhbGciOiJIUzI1NiJ9.
eyJzdWIiOiJrb3dhbHNraUBwYnMuZWR1LnBsIiwiaWF0IjoxNzQzNzEzNjg1LCJleHAiOjE3NDM3MTQ1ODUsImF1dGhvcml0aWVzIjpbIlVTRVIiXX0.
DDtVtSTh6y7hQ3luK6DCixQfw3kHxynZhsA39xpDtFY",
"refreshToken": "eyJhbGciOiJIUzI1NiJ9.
eyJzdWIiOiJrb3dhbHNraUBwYnMuZWR1LnBsIiwiaWF0IjoxNzQzNzEzNjg1LCJleHAiOjE3NDM3MjA4ODUsImF1dGhvcml0aWVzIjpbIlVTRVIiXX0.
9ucU1Rcd-LCZt0J2DwQFCvKgc_9LnK98H_Aza31SkNE"

7.18. Prawdopodobnie podczas uruchamiania aplikacji będzie można zauważyć drukowane na konsoli ostrzeżenie:

> *Serializing PageImpl instances as-is is not supported, meaning that there is no guarantee about the stability of the resulting JSON structure!*
> *For a stable JSON structure, please use Spring Data's PagedModel or Spring HATEOAS and Spring Data's PagedResourcesAssembler.*

Komunikat ten informuje, że bezpośrednia serializacja (czyli przekształcenie obiektu Javy do formatu JSON) klasy *PageImpl* nie jest zalecana, ponieważ jej struktura może ulec zmianie w przyszłych wersjach biblioteki, co może skutkować niespodziewanymi błędami aplikacji klienckich korzystających z naszego API (ponieważ używana jest stała struktura JSON). Aby rozwiązać ten potencjalny problem i zapewnić stabilność we współpracy z API, Spring rekomenduje użycie dedykowanej klasy *PagedModel*. Jest to klasa zaprojektowana specjalnie do reprezentowania stronicowanych danych w odpowiedziach API. Automatyzację procesu konwersji z *PageImpl* na *PagedModel* zapewnia użycie adnotacji:

@EnableSpringDataWebSupport*(pageSerializationMode =*
*EnableSpringDataWebSupport.PageSerializationMode.**VIA_DTO**)*

Dodanie tej adnotacji do klasy konfiguracyjnej naszej aplikacji (np. do głównej klasy ze *@SpringBootApplication*) zmienia zwracany komunikat JSON z


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


na


```
{
  "content":[ … ],
  "page":{
      "size":20,
      "number":0,
      "totalElements":0,
      "totalPages":0
  }
}
```


W tej nowej strukturze wszystkie metadane dotyczące stronicowania są zgrupowane w zagnieżdżonym obiekcie *page*, co czyni odpowiedź bardziej czytelną i gwarantuje jej stabilność, niezależnie od ewentualnych zmian w wewnętrznej implementacji *PageImpl*.

W celu lepszej organizacji kodu, można też utworzyć dedykowaną klasę konfiguracyjną (np. *JacksonConfig* w pakiecie *com.project.config*) i tam umieścić adnotację *@EnableSpringDataWebSupport* tj.:

```
package com.project.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.web.config.EnableSpringDataWebSupport;

@Configuration
@EnableSpringDataWebSupport(pageSerializationMode =
                                    EnableSpringDataWebSupport.PageSerializationMode.VIA_DTO)
public class JacksonConfig {

}
```

Uwaga! Jeżeli z usługi REST korzysta aplikacja front-endowa, konieczne będzie również wprowadzenie w niej odpowiednich modyfikacji.

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/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();