

# Projekt zaliczeniowy

Mikołaj Solecki L6

## Utworzenie REST API do obsługi operacji na wybranych danych

*W projekcie wykorzystano technologię Spring Boot.*

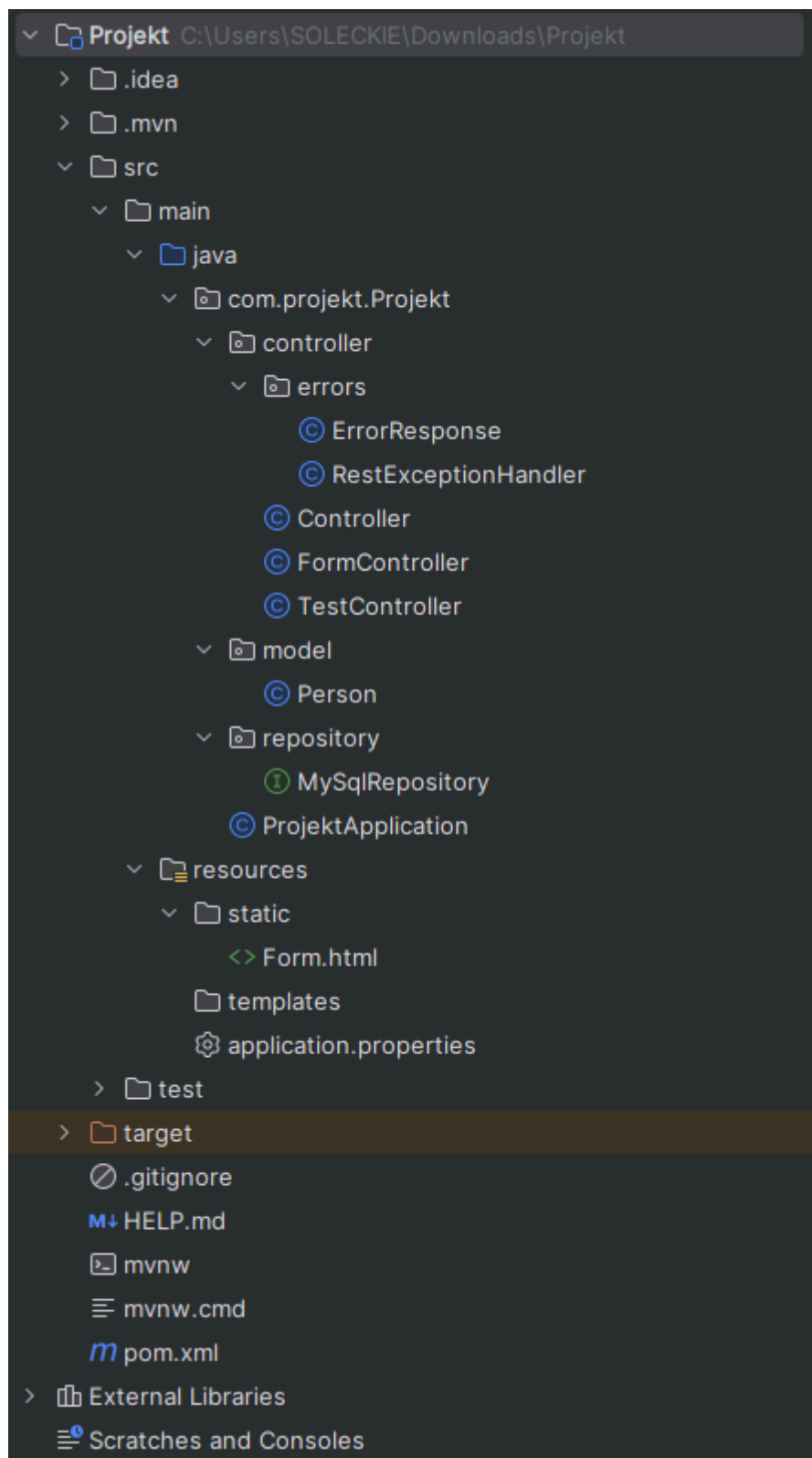
*Zastosowano podział na trzy główne części:*

**Model danych** (klasa Person) - zawiera pola takie jak imię, nazwisko, wiek, a także unikalny klucz API dla każdej osoby.

**Warstwa dostępu do danych** - klasa MySqlRepository jest interfejsem, który dziedziczy po interfejsie JpaRepository dostarczonym przez Spring Data JPA. Repozytorium to jest odpowiedzialne za interakcję z bazą danych MySQL w kontekście encji Person. Dzięki Spring Data JPA i interfejsowi JpaRepository, dostarcza ona gotowe metody do wykonywania operacji CRUD (Create, Read, Update, Delete) na danych Person.

**Kontrolery** (klasy Controller, FormController) - są odpowiedzialne za obsługę żądań HTTP i przetwarzanie danych. W projekcie istnieje klasa Controller, która zawiera metody obsługujące różne operacje, takie jak pobieranie wszystkich osób, pobieranie pojedynczej osoby, dodawanie, aktualizowanie i usuwanie osób. Korzystają z warstwy dostępu do danych, aby wykonywać operacje na danych Person.

Struktura projektu:



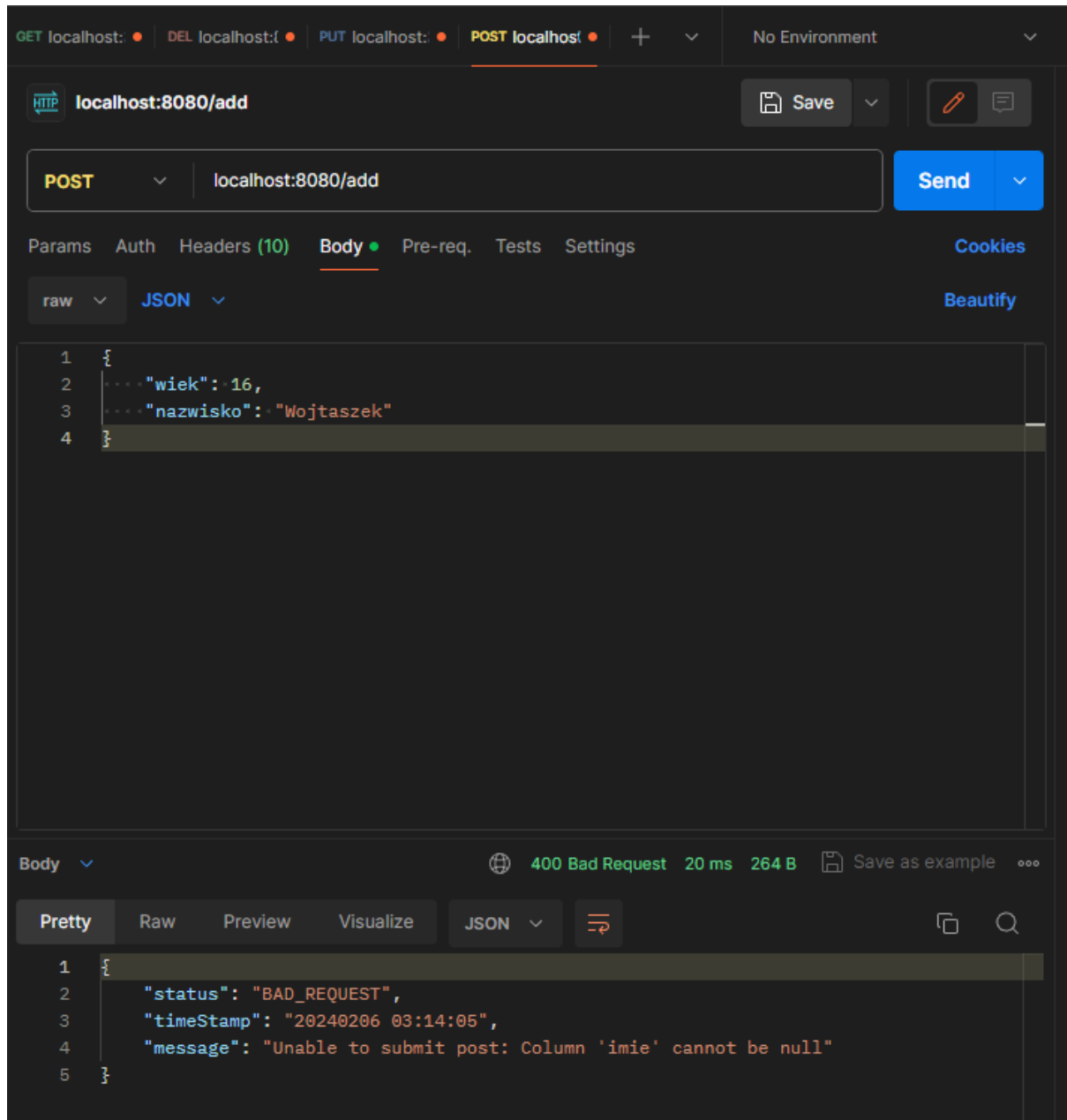
## Walidacja zapytań i zabezpieczenie przed nieprawidłowymi zapytaniami

Klasa **ErrorResponse** definiuje strukturę odpowiedzi błędu, która będzie zwracana w przypadku wystąpienia błędu w kontrolerze. Posiada ona trzy pola:

1. `status`: Status HTTP, np. `NOT_FOUND`.
2. `timeStamp`: Czas wystąpienia błędu.
3. `message`: Komunikat błędu.

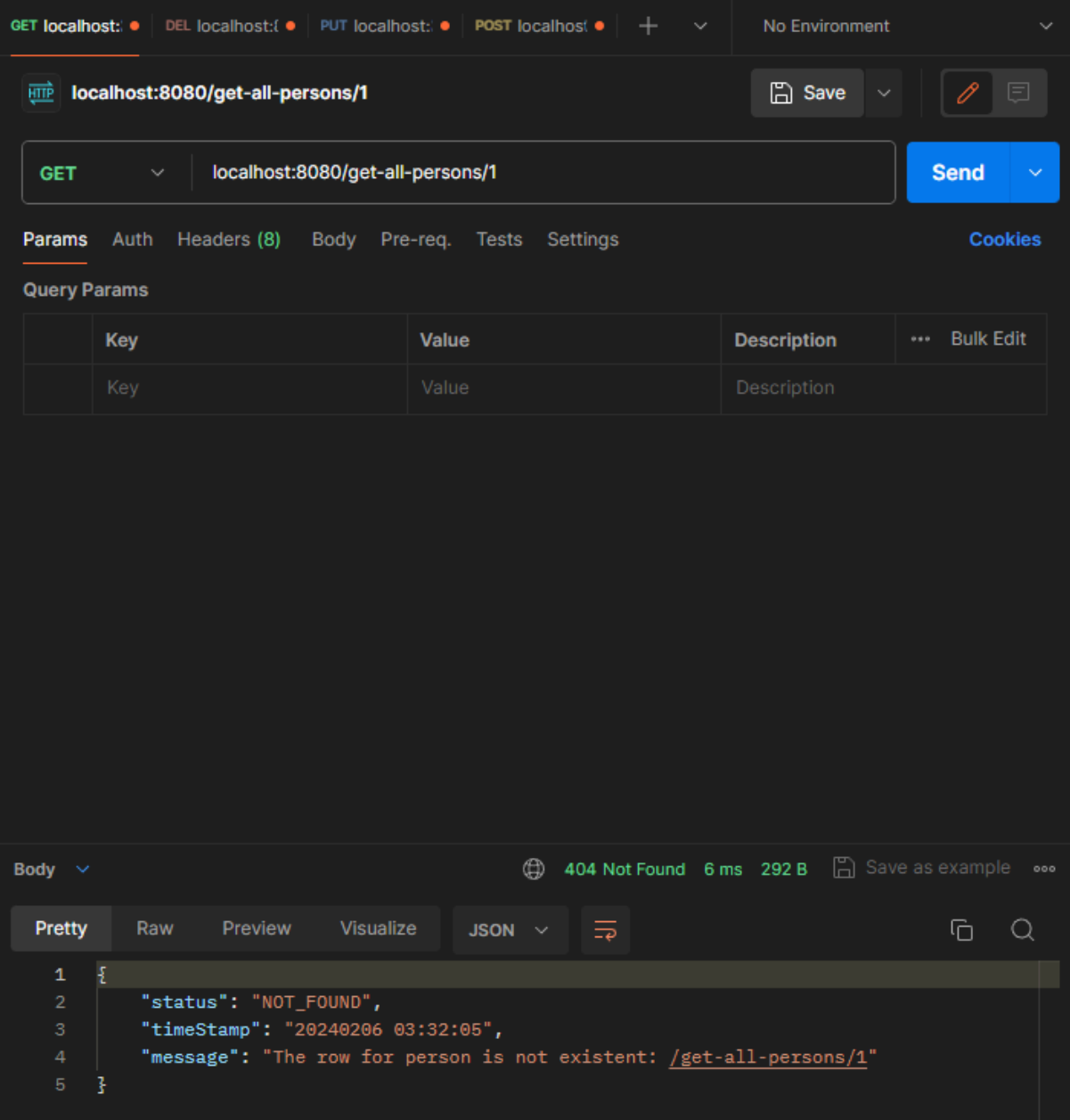
Następnie mamy klasę **RestExceptionHandler**, która jest używana do centralnej obsługi błędów w aplikacji, zapewniając spójne zachowanie w przypadku wystąpienia różnych rodzajów błędów w kontrolerach:

Metoda **handleSqlIntegrityException** obsługuje wyjątek `SQLIntegrityConstraintViolationException`, który może wystąpić, gdy naruszone zostaną ograniczenia integralności w bazie danych, na przykład, gdy próbujemy dodać poniżej przedstawione dane:



Otrzymujemy komunikat błędu, który pojawia się, ponieważ nie zainicjowaliśmy kolumny `imie`, która jest Not Null.

Metoda **handleNoSuchElementException** obsługuje wyjątek `NoSuchElementException`, który występuje, gdy próbujemy uzyskać dostęp do elementu, który nie istnieje.



The screenshot shows a REST client interface with a GET request to `localhost:8080/get-all-persons/1`. The response is a 404 Not Found status, indicating that the resource does not exist. The response body is displayed in JSON format.

Query Params

Key	Value	Description
Key	Value	Description

Body

404 Not Found 6 ms 292 B

Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "status": "NOT_FOUND",
3   "timeStamp": "20240206 03:32:05",
4   "message": "The row for person is not existent: /get-all-persons/1"
5 }
```

Obie metody zwracają odpowiedź zbudowaną na podstawie klasy `ErrorResponse`, zawierająca odpowiedni status HTTP oraz komunikat błędu. Inne przykłady:

Usuwanie danych z bazy:

The screenshot shows a REST client interface with the following details:

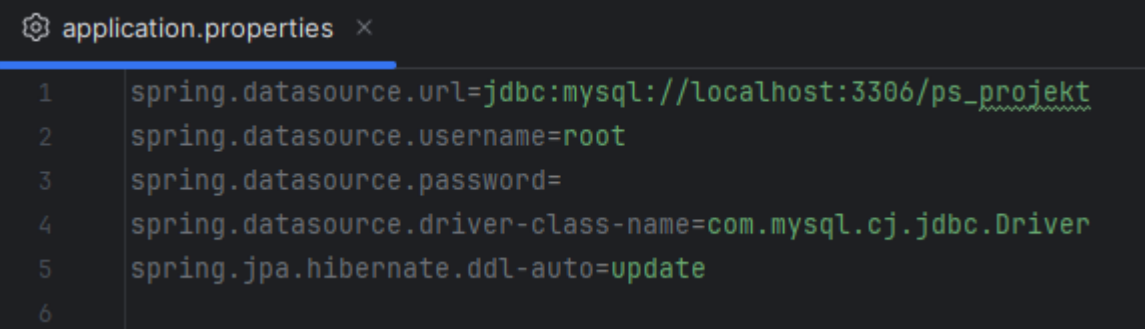
- Method:** DELETE
- URL:** localhost:8080/remove/1
- Headers:** api-key: 13c2df90-56a4-442c-bf35-02f...
- Status:** 200 OK
- Response Time:** 11 ms
- Response Size:** 169 B
- Body:** false

Key	Value	Description
api-key	13c2df90-56a4-442c-bf35-02f...	

Przy próbie usunięcia z bazy osoby która nie istnieje, otrzymamy false. W przeciwnym wypadku - po podaniu id osoby, która znajduje się w bazie, otrzymamy wartość true.

## Dane zapisane w wybranej bazie danych

Do przechowywania danych wybrałem bazę MySQL. Aplikacja Spring Boot łączy się z bazą danych MySQL za pomocą konfiguracji dostępu do bazy danych w pliku `application.properties`:

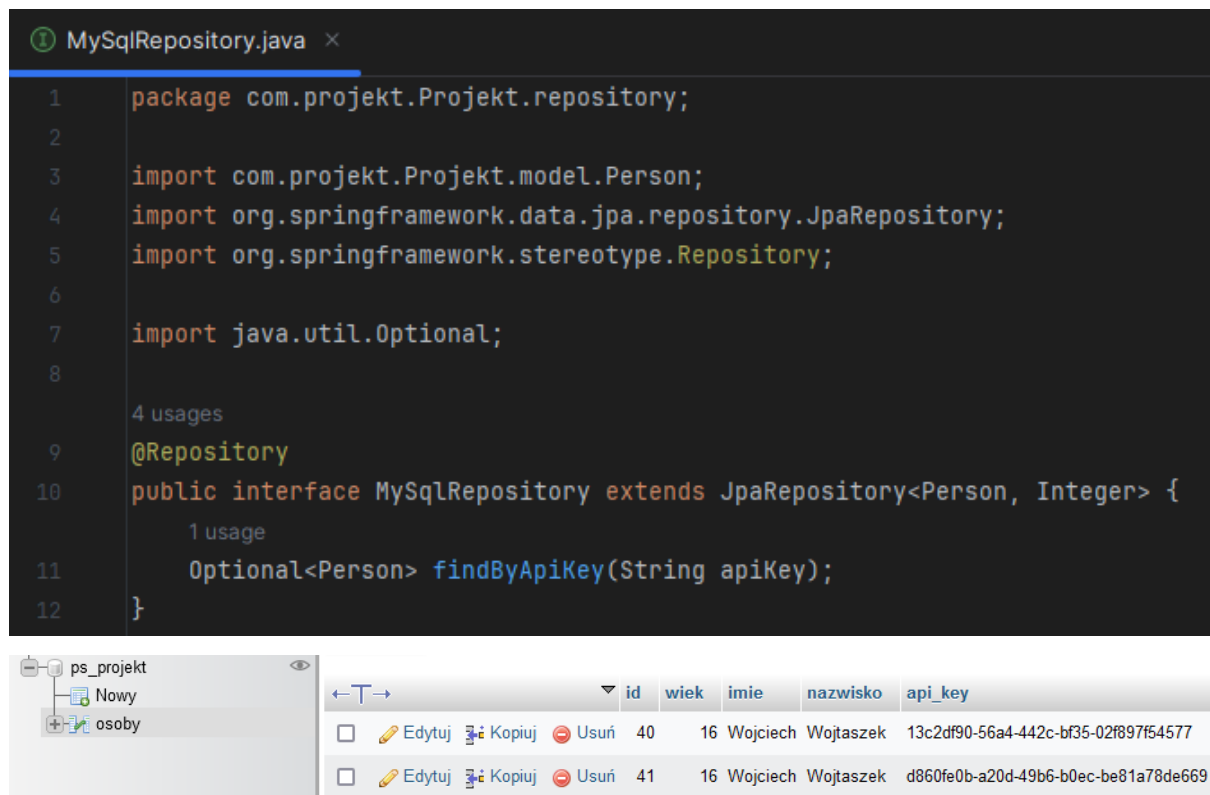
A screenshot of a code editor showing the contents of a file named 'application.properties'. The file has a dark background with light-colored text. The text is as follows:

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/ps_projekt
2 spring.datasource.username=root
3 spring.datasource.password=
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.hibernate.ddl-auto=update
6
```

The first line is highlighted with a blue background. The editor has a tab at the top labeled 'application.properties' with a close button (X) on the right.

- **spring.datasource.url**: Określa URL do bazy danych MySQL, wskazujący na lokalną instancję działającą na porcie 3306.
- **spring.datasource.username** oraz **spring.datasource.password**: Określają nazwę użytkownika i hasło dostępu do bazy danych.
- **spring.datasource.driver-class-name**: Wskazuje na klasę sterownika JDBC, która będzie używana do komunikacji z bazą danych.
- **spring.jpa.hibernate.ddl-auto**: Określa strategię tworzenia lub aktualizacji schematu bazy danych. Update oznacza, że Hibernate będzie aktualizował schemat bazy danych automatycznie na podstawie encji JPA.

Aplikacja może efektywnie komunikować się z bazą danych MySQL i wykonywać operacje CRUD na obiektach klasy Person za pomocą interfejsu repozytorium MySQLRepository:



The screenshot displays an IDE interface. The top pane shows the code for `MySQLRepository.java`. The code defines a package, imports necessary classes, and declares a `@Repository` interface that extends `JpaRepository` for `Person` entities. It includes a `findById` method. The bottom pane shows a database table view for `osoby` with columns `id`, `wiek`, `imie`, `nazwisko`, and `api_key`. Two records are visible, both for a person named Wojciech Wojtaszek.

```
1 package com.projekt.Projekt.repository;
2
3 import com.projekt.Projekt.model.Person;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 import java.util.Optional;
8
9 @Repository
10 public interface MySQLRepository extends JpaRepository<Person, Integer> {
11     Optional<Person> findById(Integer id);
12 }
```

	id	wiek	imie	nazwisko	api_key
<input type="checkbox"/> Edytuj  Kopiuj  Usuń	40	16	Wojciech	Wojtaszek	13c2df90-56a4-442c-bf35-02f897f54577
<input type="checkbox"/> Edytuj  Kopiuj  Usuń	41	16	Wojciech	Wojtaszek	d860fe0b-a20d-49b6-b0ec-be81a78de669

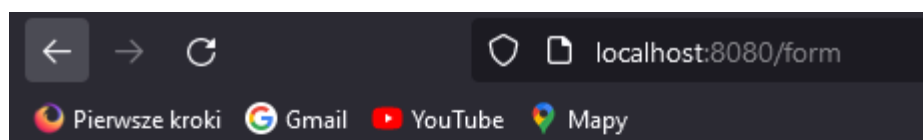
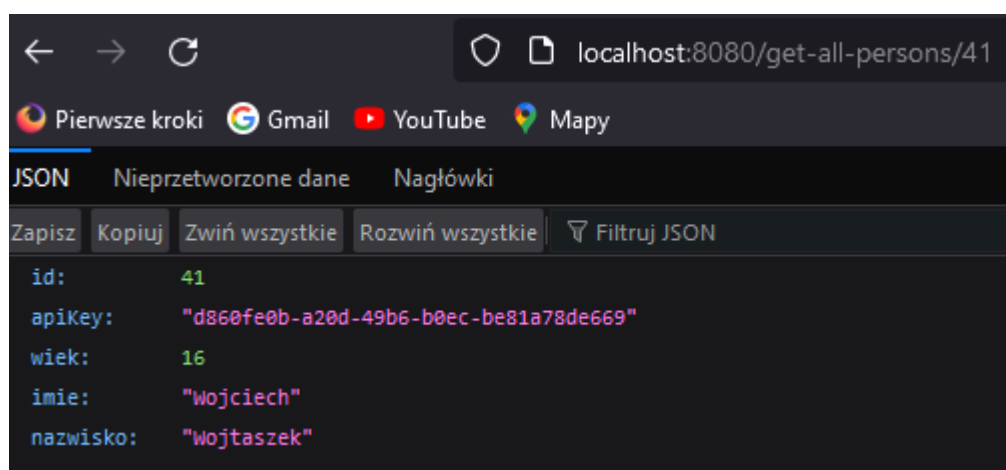
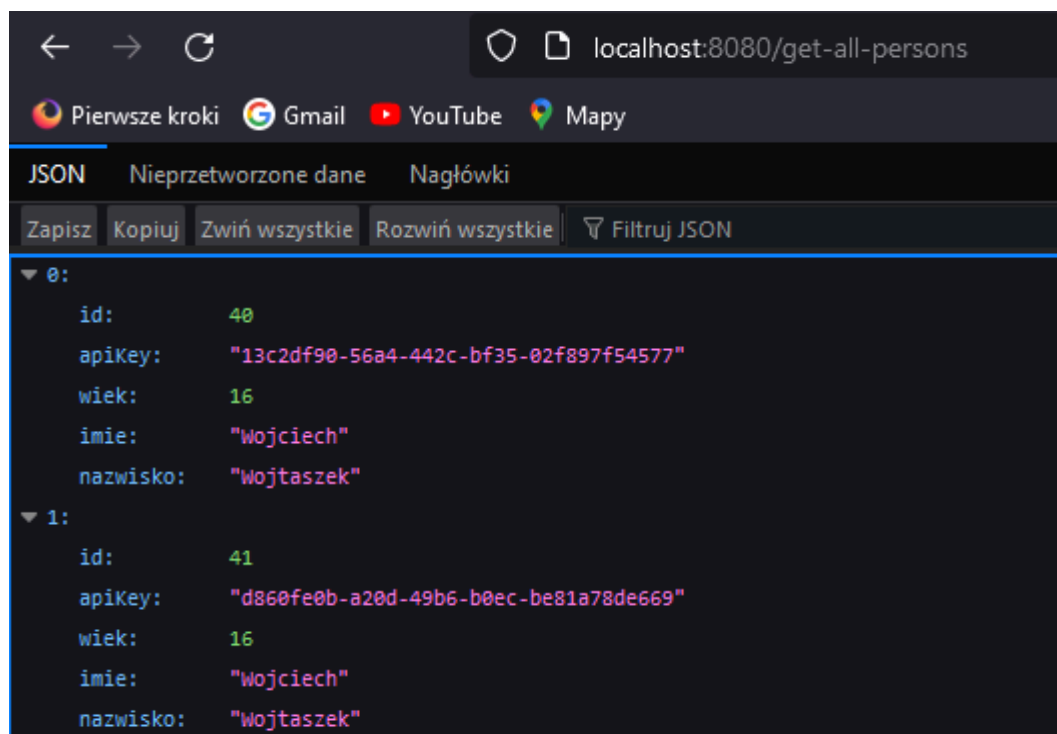


## Utworzenie formularza do dodawania nowych danych

1. **Form.html**: W pliku HTML definiujemy formularz, który będzie używany do wprowadzania danych. Formularz zawiera pola Imię, Wiek i Nazwisko, które odpowiadają polom obiektu `Person`. Każde pole ma określony typ (text dla imienia i nazwiska, number dla wieku) oraz atrybut `required`, co oznacza, że pole jest wymagane.
2. **Akcja Formularza**: Po kliknięciu przycisku "Dodaj osobę", dane z formularza zostaną przesłane do endpointu `/add-person` za pomocą metody HTTP POST.
3. **Klasa Controllera**: W kontrolerze **FormController** mamy dwie metody:
  - o **showForm**: Ta metoda obsługuje żądania GET na endpoint `/form` i zwraca widok formularza HTML. Dodaje również do modelu nowy obiekt `Person`, który zostanie użyty do wiązania danych z formularza.
  - o **addPerson**: Ta metoda obsługuje żądania POST na endpoint `/add-person`. Przyjmuje obiekt `Person`, który został przekazany z formularza jako argument `@ModelAttribute`. Następnie generuje unikalny klucz API za pomocą **UUID.randomUUID().toString()** i ustawia go dla osoby. Ostatecznie zapisuje osobę do bazy danych za pomocą **mySqlRepository.save(person)** i przekierowuje użytkownika na stronę wyświetlającą wszystkie osoby.

Dzięki temu formularzowi użytkownik może wprowadzać nowe dane i dodawać je do bazy danych za pomocą interfejsu użytkownika w przeglądarce.

## Prezentacja danych przez przeglądarkę



## Add New Person

Imię:

Wiek:

Nazwisko:

# Walidacja danych wprowadzanych przez formularz

```
© ProjektApplication.java  <> Form.html x  © FormController.java  application.properties  m pom.x

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Add Person</title>
7  </head>
8  <body>
9      <h2>Add New Person</h2>
10     <form action="/add-person" method="post">
11         <label for="imie">Imię:</label>
12         <input type="text" id="imie" name="imie" pattern="[A-Za-z]+" required><br>
13
14         <label for="wiek">Wiek:</label>
15         <input type="number" id="wiek" name="wiek" min="1" max="150" required><br>
16
17         <label for="nazwisko">Nazwisko:</label>
18         <input type="text" id="nazwisko" name="nazwisko" pattern="[A-Za-z]+" required><br>
19
20         <button type="submit">Dodaj osobę</button>
21     </form>
22 </body>
23 </html>
```

W powyższym kodzie:

- Dla pola "imie" i "nazwisko" używamy atrybutu **pattern="[A-Za-z]+"**, który wymaga, aby pole zawierało tylko litery (bez cyfr ani znaków specjalnych).
- Dla pola "wiek" używamy atrybutów **min="1"** i **max="150"**, aby ograniczyć wiek do wartości większych od 0 i maksymalnie 150 lat.
- Atrybut **required** jest używany dla każdego pola, aby wymusić wprowadzenie danych przed wysłaniem formularza.

W klasie Person:

```
@Entity
@Table(name="osoby")
public class Person {

    1 usage
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    3 usages
    @Column(name = "api_key")
    private String apiKey;

    3 usages
    @NotNull
    @Range(min = 1, max = 150, message = "Age must be between 1 and 150")
    private Integer wiek;

    3 usages
    @NotBlank(message = "Name is mandatory")
    private String imie;

    3 usages
    @NotBlank(message = "Surname is mandatory")
    private String nazwisko;
```

## Rozszerzenie REST API o możliwość dodawania danych przez zapytania

1. **GET /get-all-persons:** To zapytanie służy do pobrania wszystkich istniejących osób z bazy danych. Wywołanie tego endpointu zwraca listę wszystkich osób w formacie JSON.
2. **GET /get-all-persons/{identity}:** Pozwala na pobranie szczegółowych informacji o konkretnej osobie na podstawie jej identyfikatora (ID).
3. **POST /add:** To zapytanie umożliwia dodanie nowej osoby do bazy danych. W ciele żądania należy przekazać dane osoby w formacie JSON. Dane te powinny zawierać imię, nazwisko i wiek.
4. **DELETE /remove/{id}:** Ten endpoint służy do usuwania istniejącej osoby z bazy danych na podstawie jej identyfikatora (ID) - po przekazaniu ID osoby do usunięcia.
5. **PUT /update/{id}:** To zapytanie pozwala na aktualizację istniejącej osoby w bazie danych na podstawie jej identyfikatora (ID). W ciele żądania należy przekazać nowe dane osoby w formacie JSON, które zostaną zaktualizowane.

## Zabezpieczenie API np. przez klucz / token

1. **Mechanizm zabezpieczeń:** W projekcie został zaimplementowany mechanizm zabezpieczeń oparty na kluczu API. Każde zapytanie *oprócz GET* musi zawierać poprawny klucz API w nagłówku *api-key*, który jest weryfikowany przed przetworzeniem żądania. *Wyjątkiem jest POST tylko gdy baza danych jest pusta!*
2. **Weryfikacja klucza API:** Weryfikacja klucza API odbywa się w metodzie **`verifyApiKey()`**, która sprawdza, czy przekazany klucz jest poprawny. W przypadku nieprawidłowego klucza API, zwracany jest błąd **`SecurityException`**.
3. **Generowanie klucza API:** Podczas dodawania nowej osoby do bazy danych, generowany jest unikalny klucz API za pomocą metody **`UUID.randomUUID().toString()`**. Klucz ten jest następnie przypisywany do nowej osoby i zapisywany w bazie danych.
4. **Obsługa błędów:** W przypadku wystąpienia błędów, takich jak brak zasobu lub naruszenie integralności danych, odpowiednie komunikaty błędów są zwracane w formie odpowiedzi HTTP, wraz z odpowiednimi kodami statusu.

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8080/add
- Headers:** 10 headers are listed, including an 'api-key' header with a value starting with '13c2df90-56a4-442c-bf35-02f...'. There is a '9 hidden' indicator.
- Response:** 401 Unauthorized, 411 ms, 286 B. The response body is in JSON format:

```
1 {
2   "status": "UNAUTHORIZED",
3   "timeStamp": null,
4   "message": "Invalid API key provided. Please provide a valid API key."
5 }
```

Klucz przy POST jest wymagany, jeżeli baza danych nie jest pusta. Jeżeli takiego klucza się nie poda, to dostaniemy odpowiedni komunikat błędu.