

Zad.

1. Utwórz aplikację typu **REST API** realizującą logikę biznesową wypożyczania kaset video/ (lub swoją własną). W pierwszej wersji dane będą przechowywane w kolekcji, skupimy się na warstwach aplikacji i możliwościach programowania funkcyjnego.

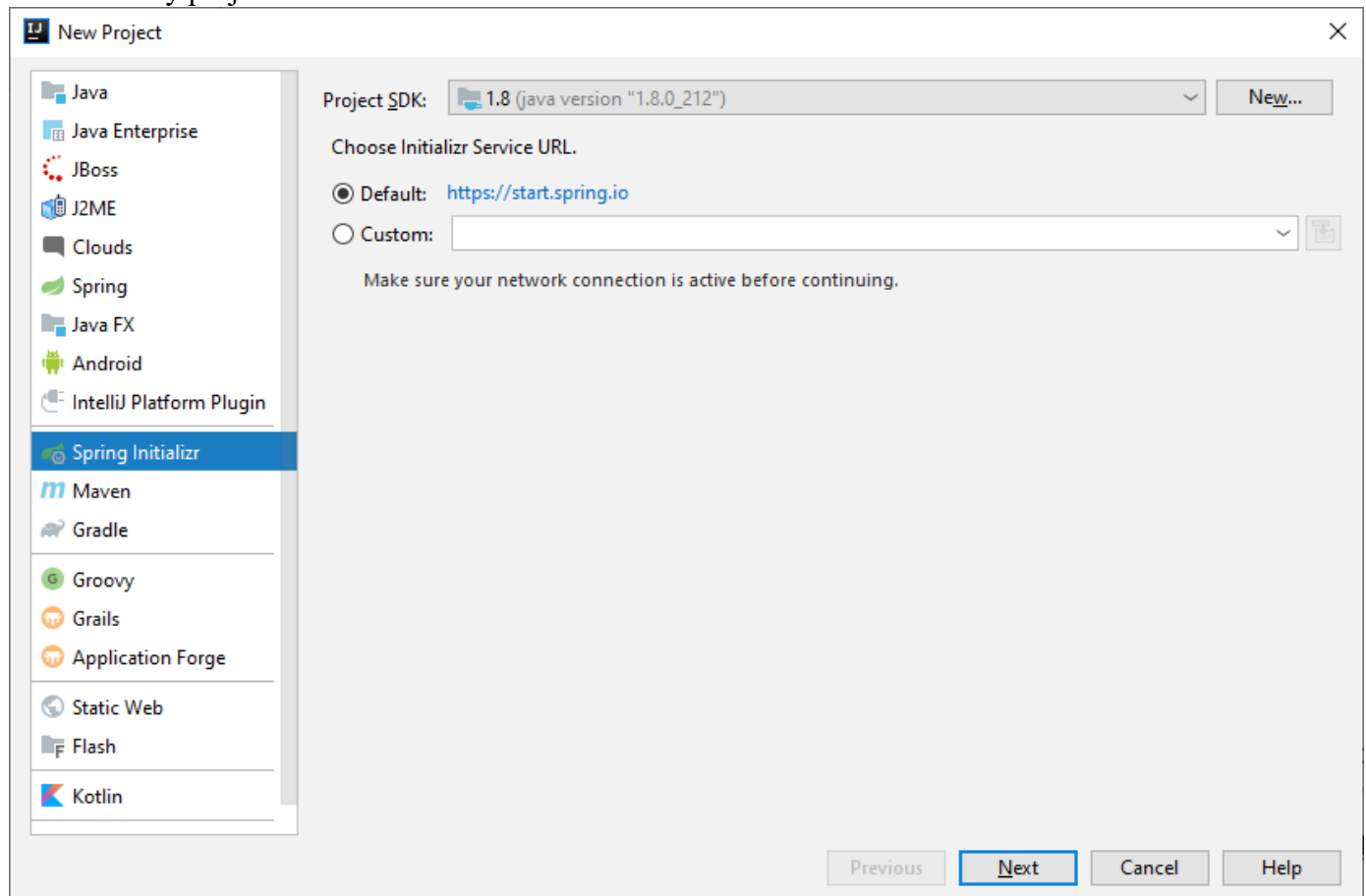
Możliwości API odnośnie przechowywanych w bazie danych kaset wideo (realizacja **CRUD**):

- Dodawanie
- Pobieranie
- Pobieranie wszystkich
- Modyfikowanie
- Usuwanie

2. Aplikacja komunikuje się z **bazą danych**. Przystosowanie aplikacji do współpracy z bazą danych.

3. Po implementacji i testowaniu przy użyciu Postman'a (klient http) wykonasz upload aplikacji na Heroku a potem na serwer CI – Jenkins. (na IO)

Utwórz nowy projekt.



New Project

Project Metadata

Group:

Artifact:

Type:

Language:

Packaging:

Java Version:

Version:

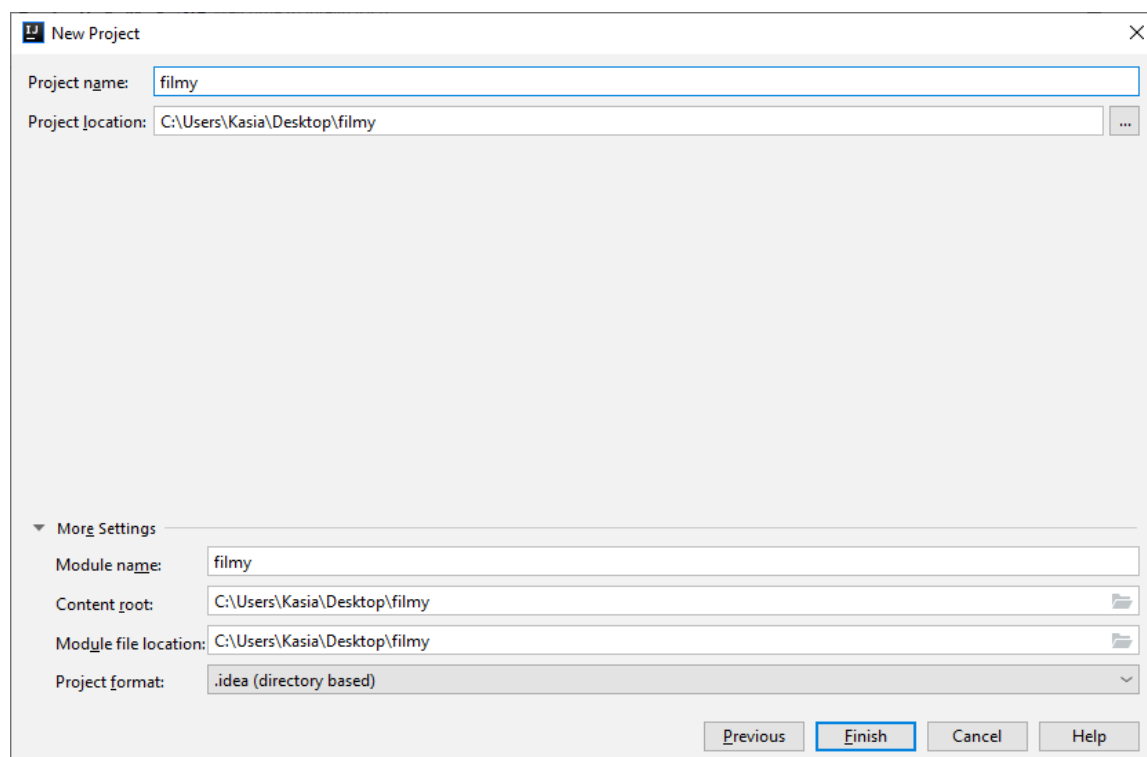
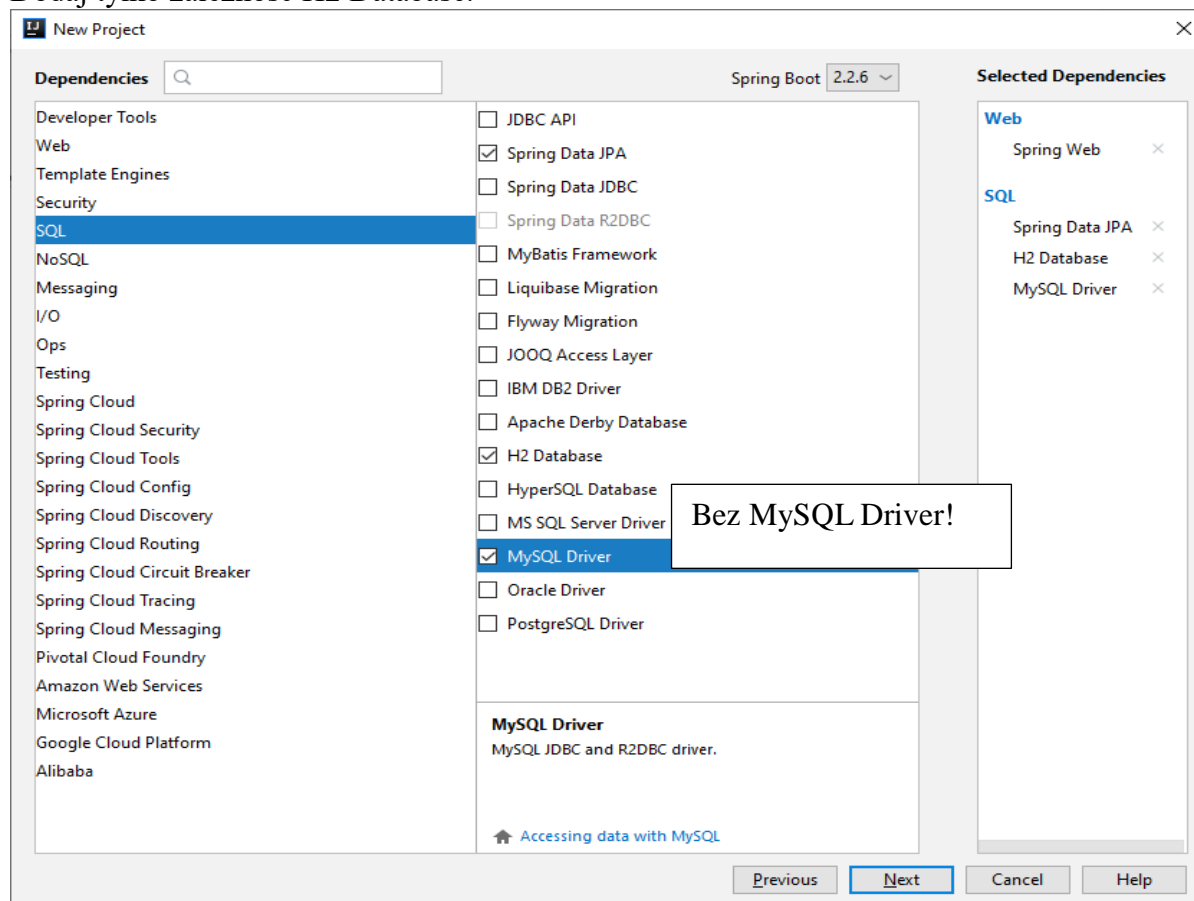
Name:

Description:

Package:

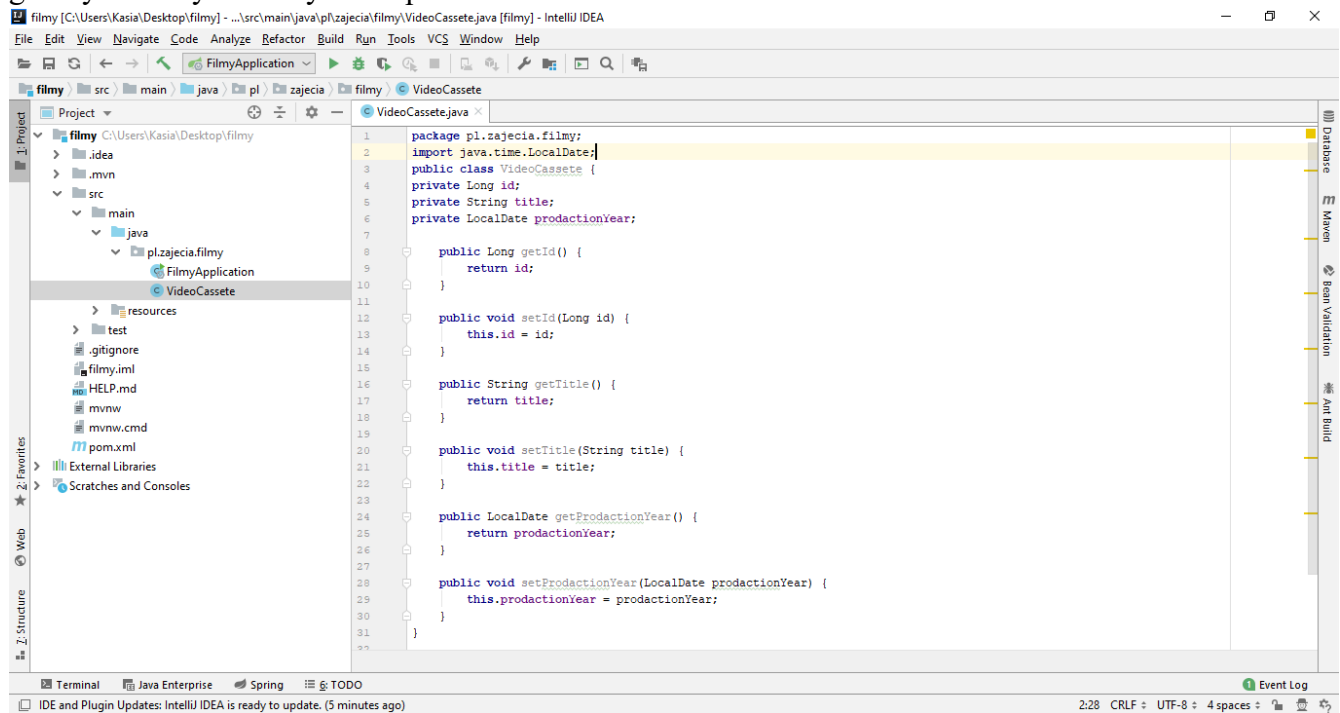
Bedzie to aplikacja webowa, która będzie musiała mieć dostęp do bazy danych.

Dodaj tylko zależność H2 Database.



Kliknij Finish i poczekaj aż projekt się wygeneruje. Uruchom aplikację – sprawdź czy poprawnie się uruchamia.

Utwórz klasę, która będzie reprezentowała obiekt kasyety wideo – VideoCassete z polami: Long identyfikator, String tytuł, LocalDate rokProdukcji (typ LocalDate pochodzi z Api Javy 8, który służy do przechowania informacji na temat daty). Użyj skrótu Alt + Insert aby utworzyć automatycznie gettery i settery do wszystkich pól.



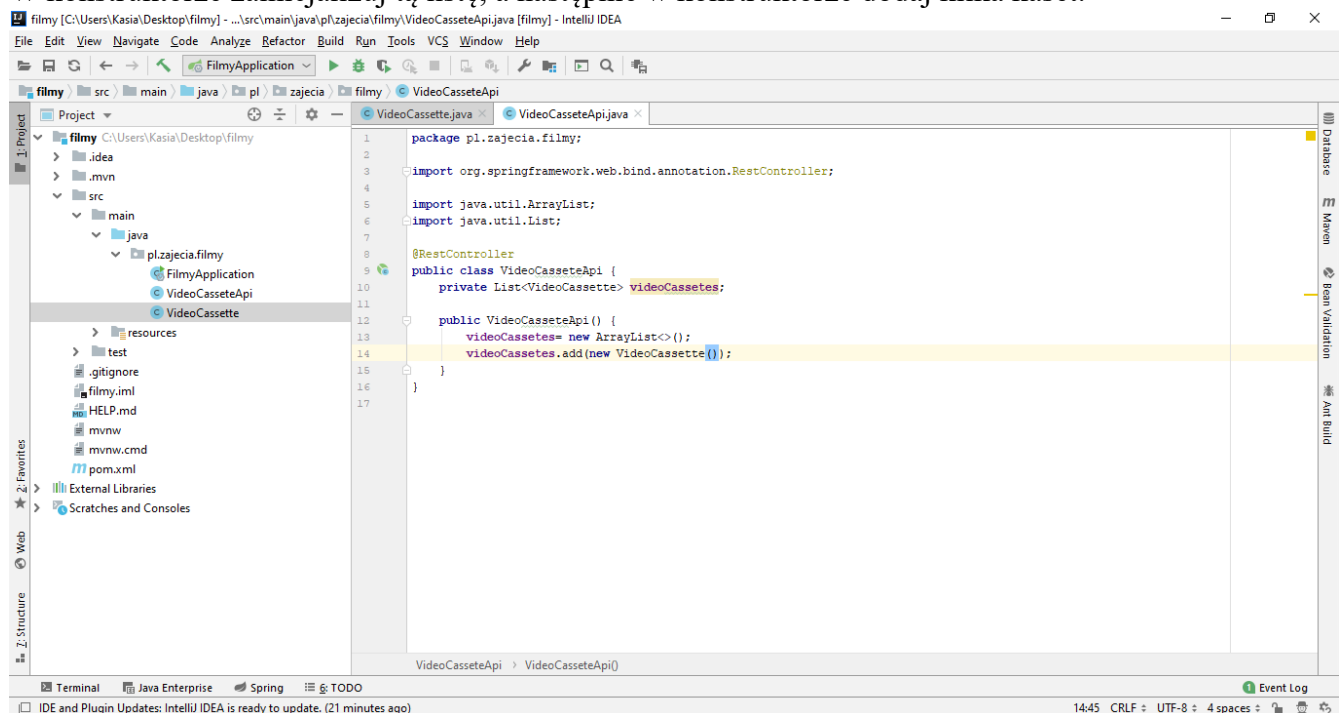
```
1 package pl.zajecia.filmly;
2 import java.time.LocalDate;
3 public class VideoCassete {
4     private Long id;
5     private String title;
6     private LocalDate productionYear;
7
8     public Long getId() {
9         return id;
10    }
11
12    public void setId(Long id) {
13        this.id = id;
14    }
15
16    public String getTitle() {
17        return title;
18    }
19
20    public void setTitle(String title) {
21        this.title = title;
22    }
23
24    public LocalDate getProductionYear() {
25        return productionYear;
26    }
27
28    public void setProductionYear(LocalDate productionYear) {
29        this.productionYear = productionYear;
30    }
31 }
```

Klasa będzie przechowywała kasyety wideo.

W kolejnym kroku utwórz api - klasę VideoCasseteApi, ponieważ będziemy się z tą klasą komunikować z użyciem protokołu http, to musi ona być bean'em z adnotacją RestController.

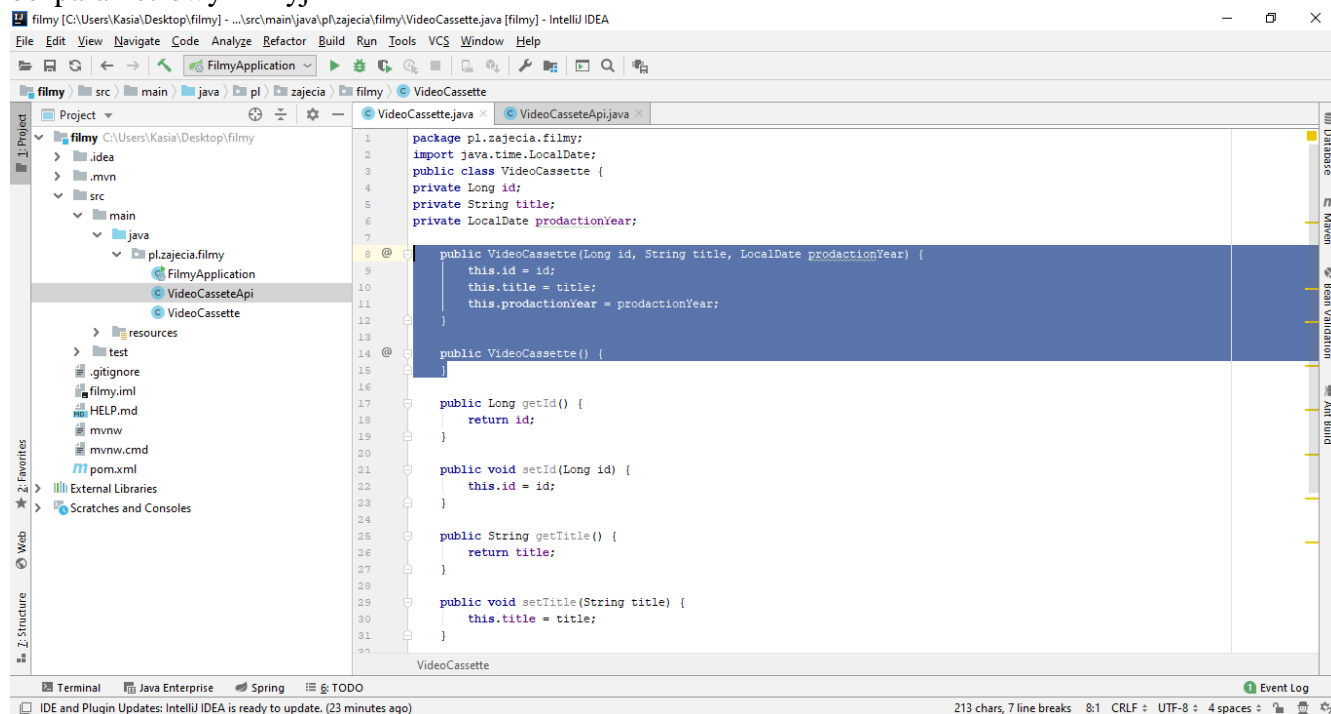
W klasie utwórz listę, która będzie zawierać informacje o kasetach przed utworzeniem bazy danych.

W konstruktorze zainicjalizuj tą listę, a następnie w konstruktorze dodaj kilka kaset.



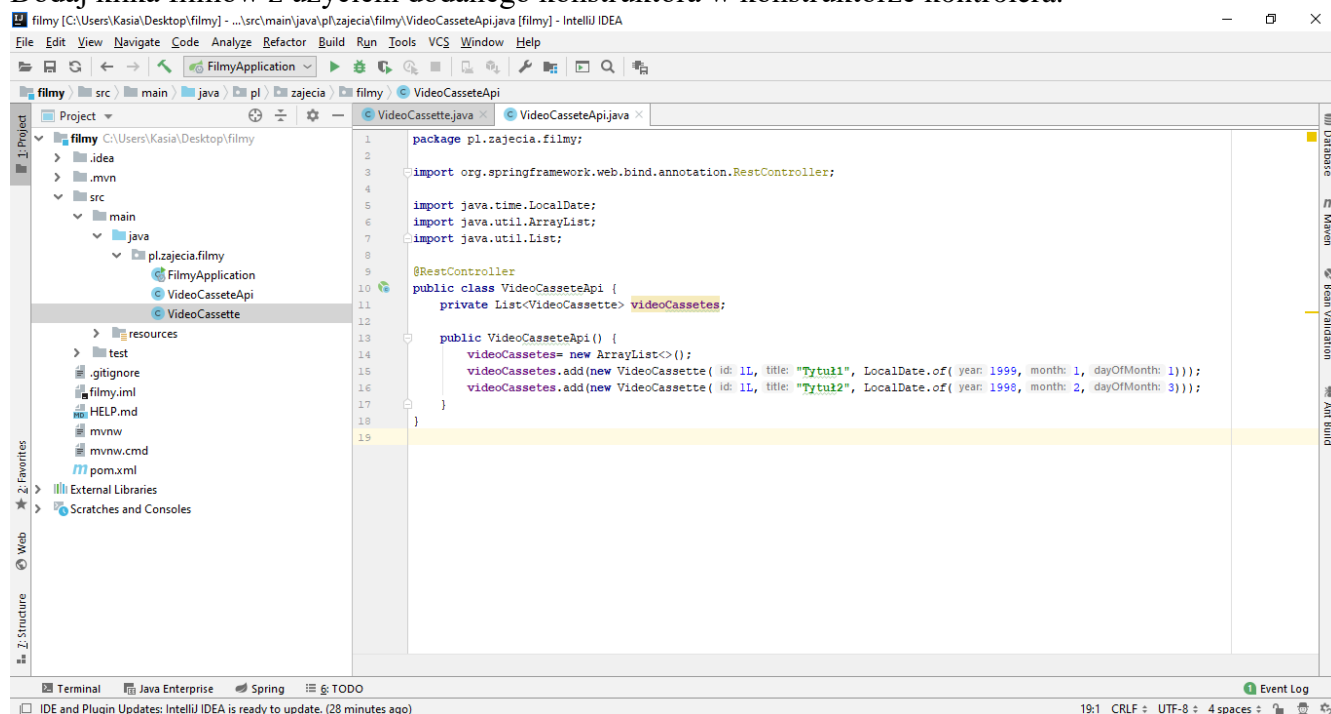
```
1 package pl.zajecia.filmly;
2
3 import org.springframework.web.bind.annotation.RestController;
4
5 import java.util.ArrayList;
6 import java.util.List;
7
8 @RestController
9 public class VideoCasseteApi {
10     private List<VideoCassete> videoCassetes;
11
12     public VideoCasseteApi() {
13         videoCassetes = new ArrayList<>();
14         videoCassetes.add(new VideoCassete());
15     }
16 }
17
```

W klasie VideoCassette dostępny jest tylko konstruktor bezparametrowy, dlatego dodaj do niej konstruktor do tworzenia obiektu, który przyjmuje wszystkie parametry oraz jawnie podaj konstruktor bezparametrowy – użyj skrótu Alt+Insert.



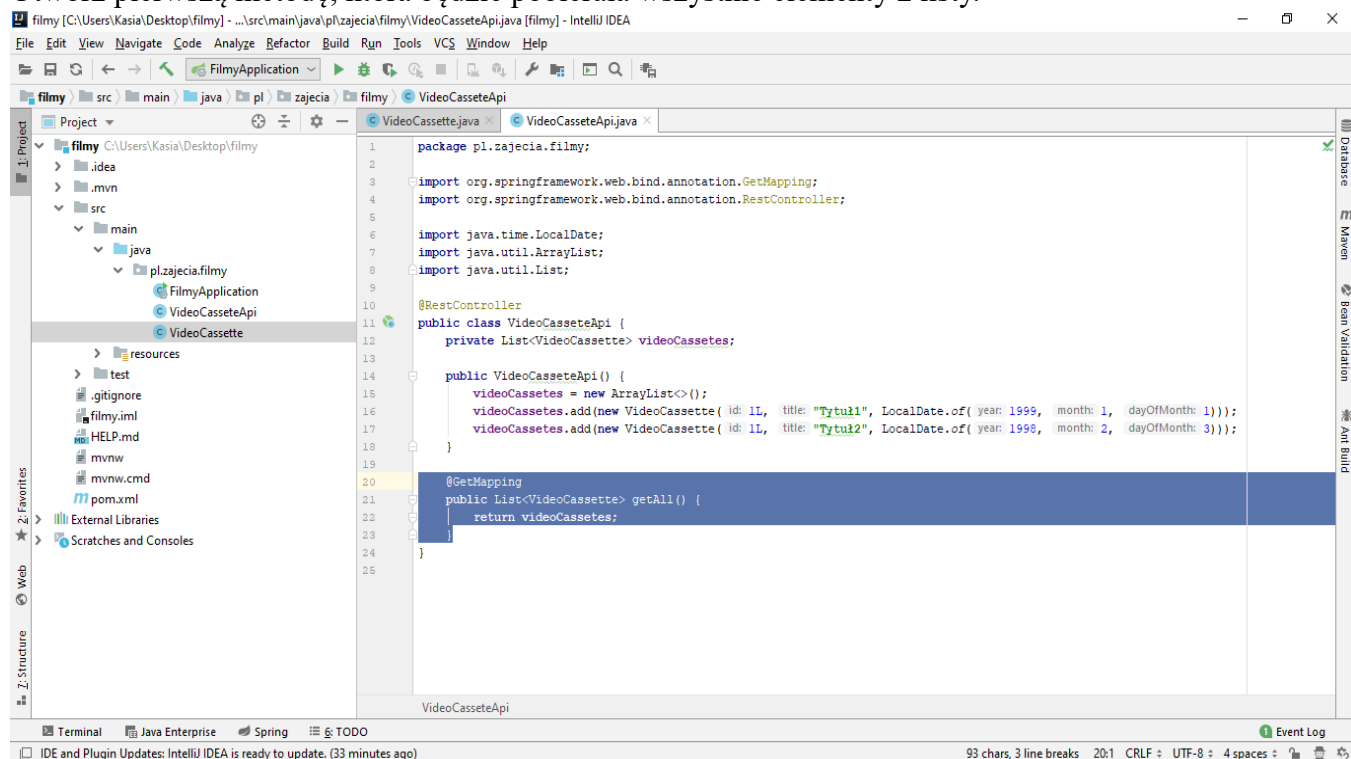
```
1 package pl.zajecia.filmly;
2 import java.time.LocalDate;
3 public class VideoCassette {
4     private Long id;
5     private String title;
6     private LocalDate productionYear;
7
8     @param
9     public VideoCassette(Long id, String title, LocalDate productionYear) {
10         this.id = id;
11         this.title = title;
12         this.productionYear = productionYear;
13     }
14
15     @param
16     public VideoCassette() {
17
18     }
19
20     public Long getId() {
21         return id;
22     }
23
24     public void setId(Long id) {
25         this.id = id;
26     }
27
28     public String getTitle() {
29         return title;
30     }
31
32     public void setTitle(String title) {
33         this.title = title;
34     }
35 }
```

Dodaj kilka filmów z użyciem dodanego konstruktora w konstruktorze kontrolera.



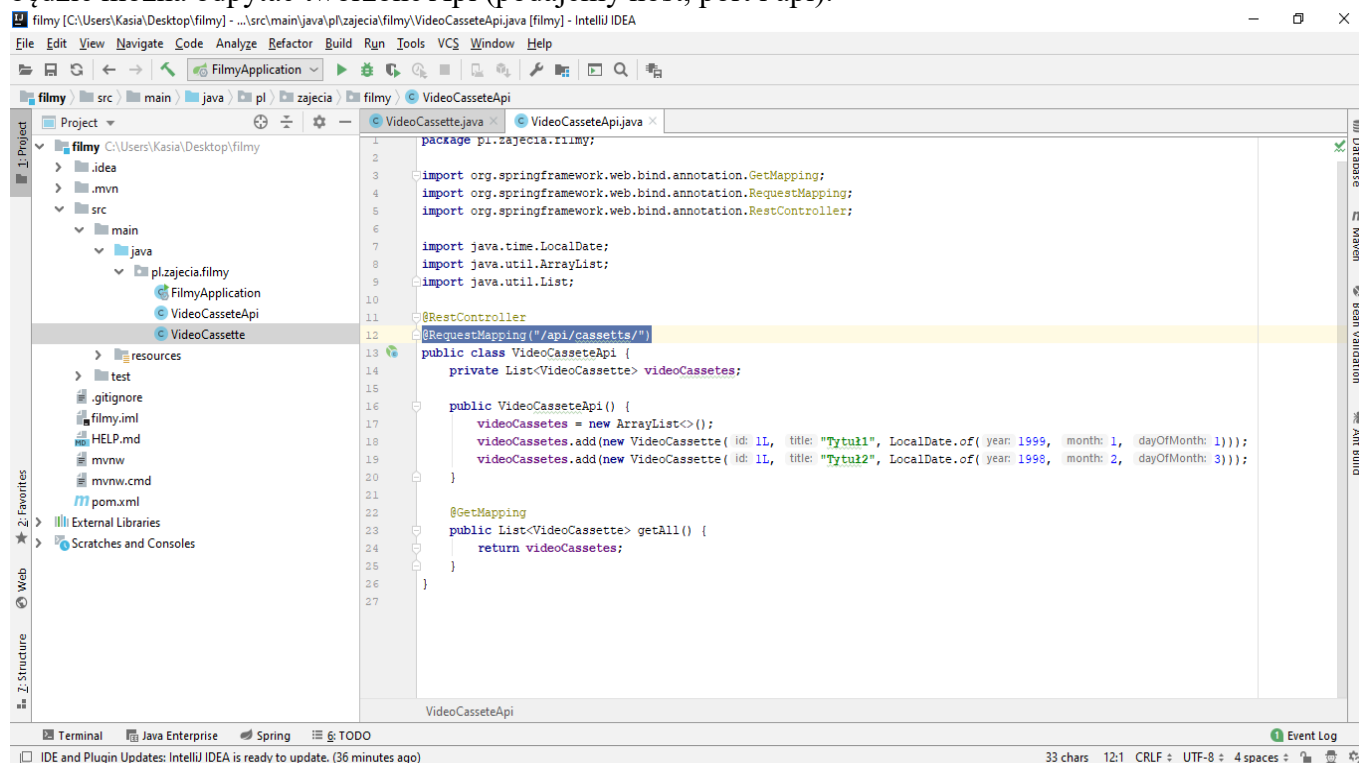
```
1 package pl.zajecia.filmly;
2
3 import org.springframework.web.bind.annotation.RestController;
4
5 import java.time.LocalDate;
6 import java.util.ArrayList;
7 import java.util.List;
8
9 @RestController
10 public class VideoCassetteApi {
11     private List<VideoCassette> videoCassettes;
12
13     public VideoCassetteApi() {
14         videoCassettes = new ArrayList<>();
15         videoCassettes.add(new VideoCassette(1L, title: "Tytuł1", LocalDate.of( year: 1999, month: 1, dayOfMonth: 1)));
16         videoCassettes.add(new VideoCassette( id: 1L, title: "Tytuł2", LocalDate.of( year: 1998, month: 2, dayOfMonth: 3)));
17     }
18 }
```

Utwórz pierwszą metodę, która będzie pobierała wszystkie elementy z listy.

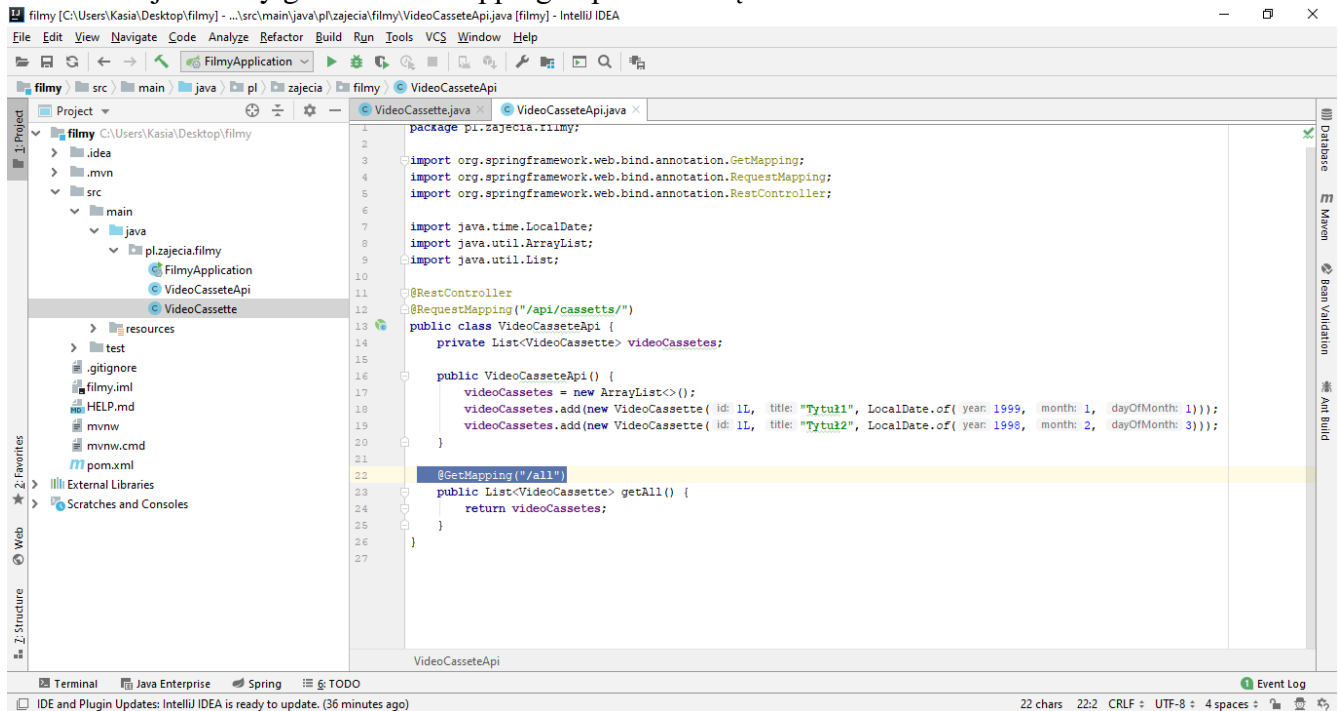


Do metody dodaj adnotację `GetMapping` ponieważ metoda http GET odpowiada za pobieranie elementów z API, zwracane będą wszystkie elementy z listy.

Nad klasą dodaj adnotację `RequestMapping` z parametrem, którym będzie adres za pomocą którego będzie można odpytać tworzone Api (podajemy host, port i api).



Do adnotacji metody getAll - GetMapping dopisz ścieżkę /all.



```
package pl.zajecia.film;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

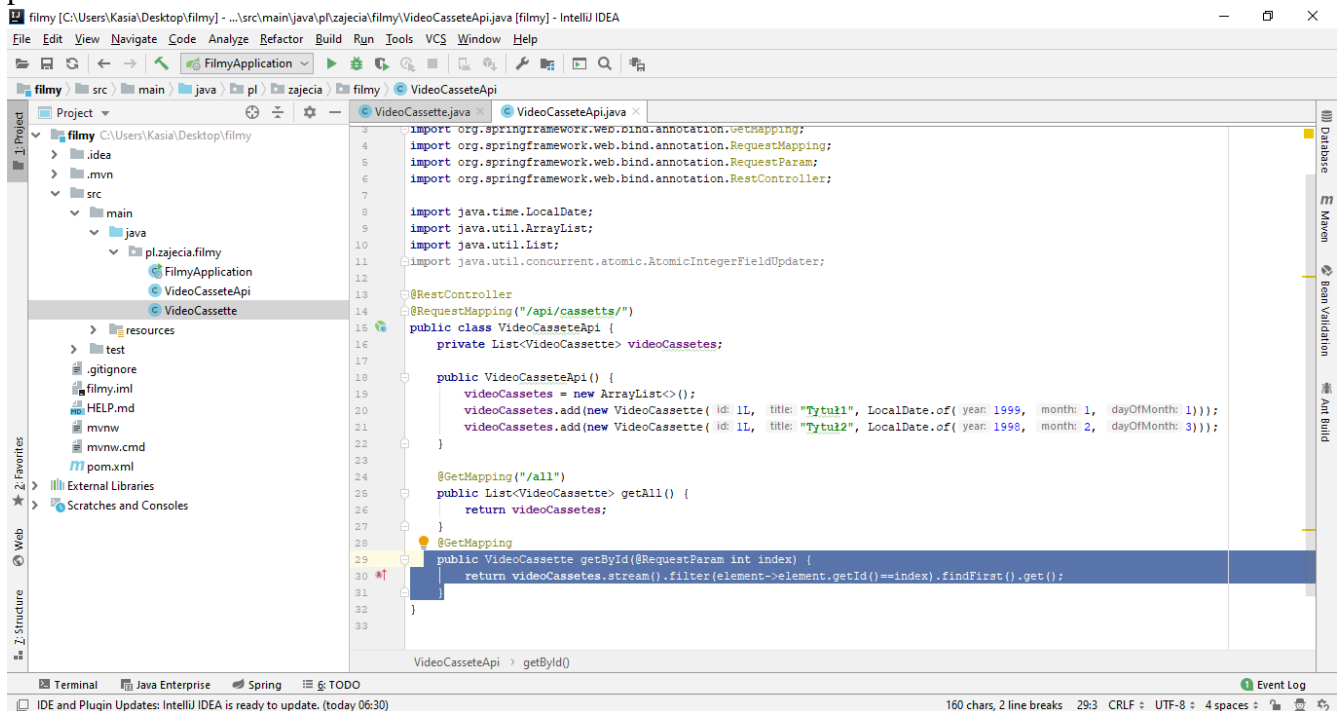
@RestController
@RequestMapping("/api/cassette/")
public class VideoCassetteApi {
    private List<VideoCassette> videoCassettes;

    public VideoCassetteApi() {
        videoCassettes = new ArrayList<>();
        videoCassettes.add(new VideoCassette(1L, "Tytuł1", LocalDate.of(1999, 1, 1)));
        videoCassettes.add(new VideoCassette(2L, "Tytuł2", LocalDate.of(1998, 2, 3)));
    }

    @GetMapping("/all")
    public List<VideoCassette> getAll() {
        return videoCassettes;
    }
}
```

VideoCassette

Po zaimplementowaniu pobierania wszystkich elementów z listy, dodaj pobieranie pojedynczego elementu z listy po id. Żeby odwołać się do konkretnego elementu z listy użyj adnotacji RequestParam przed pobieranym parametrem, którym będzie indeks elementu. Dany element będzie pobierany z listy na podstawie podanego parametru typu int. Użyjemy do tego elementów programowanie funkcyjnego - metody stream i filter, gdzie podasz jakiego elementu szukasz – który ma takie samo id jak id podane w parametrze.



```
package pl.zajecia.film;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.atomic.AtomicIntegerFieldUpdater;

@RestController
@RequestMapping("/api/cassette/")
public class VideoCassetteApi {
    private List<VideoCassette> videoCassettes;

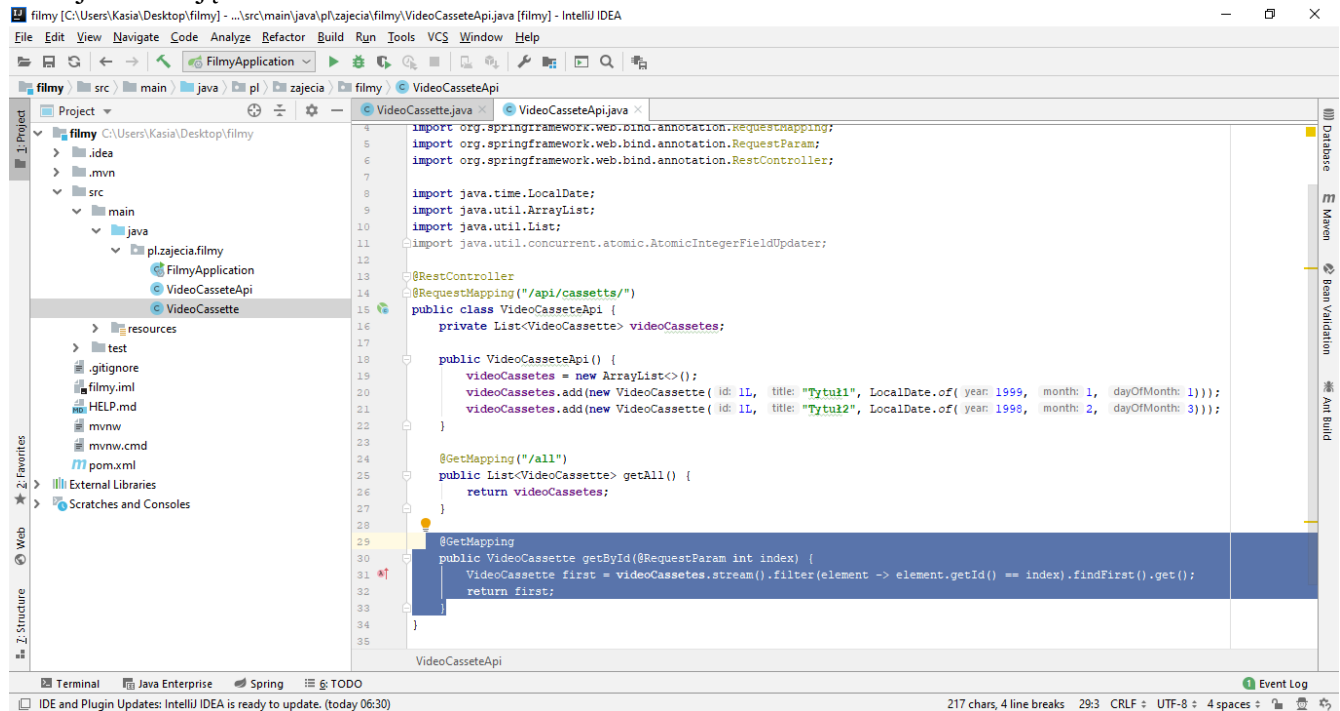
    public VideoCassetteApi() {
        videoCassettes = new ArrayList<>();
        videoCassettes.add(new VideoCassette(1L, "Tytuł1", LocalDate.of(1999, 1, 1)));
        videoCassettes.add(new VideoCassette(2L, "Tytuł2", LocalDate.of(1998, 2, 3)));
    }

    @GetMapping("/all")
    public List<VideoCassette> getAll() {
        return videoCassettes;
    }

    @GetMapping
    public VideoCassette getById(@RequestParam int index) {
        return videoCassettes.stream().filter(element->element.getId()==index).findFirst().get();
    }
}
```

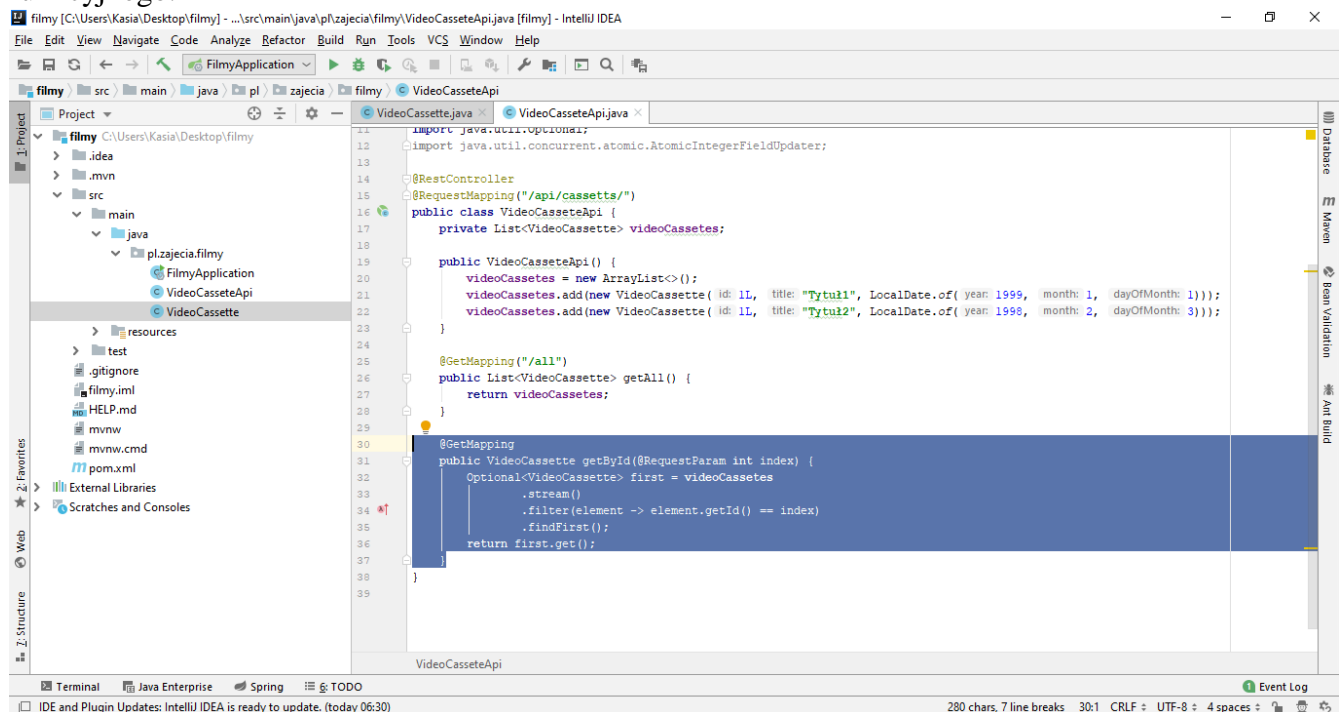
VideoCassetteApi : getById()

Dodaj instrukcję return.



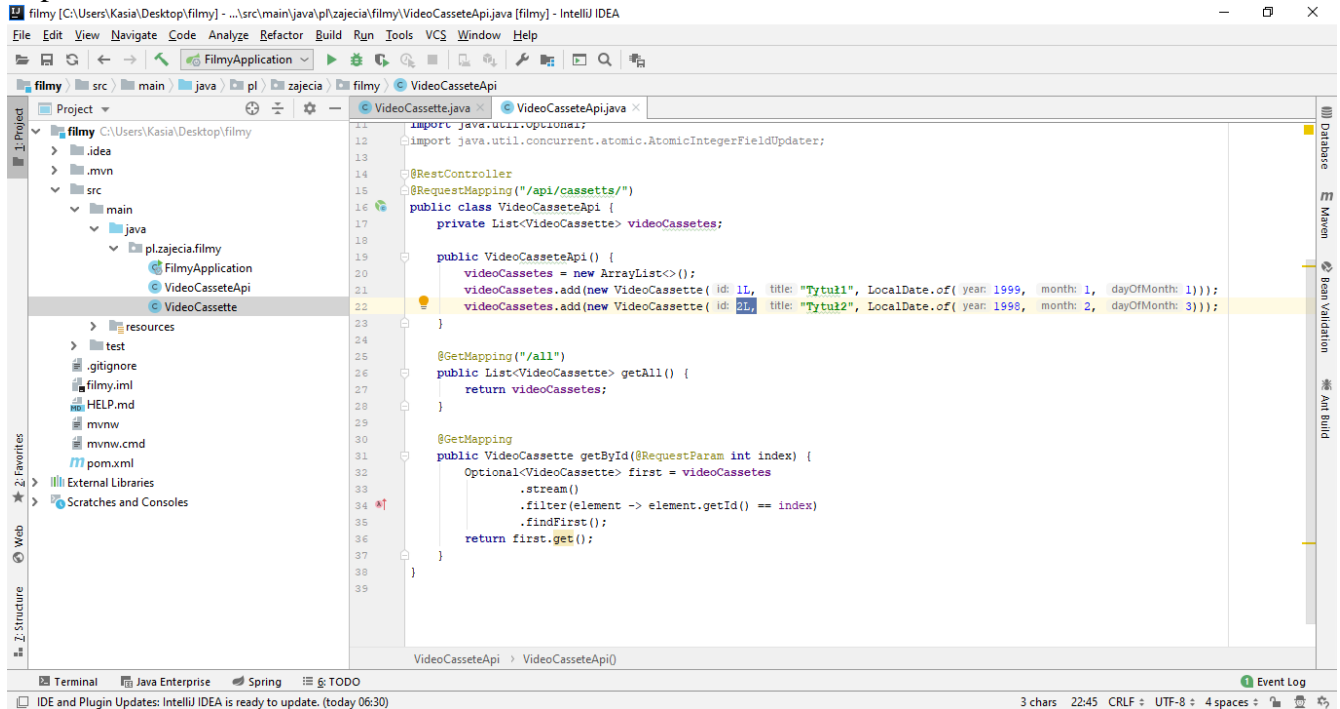
```
1 import org.springframework.web.bind.annotation.RequestMapping;
2 import org.springframework.web.bind.annotation.RequestParam;
3 import org.springframework.web.bind.annotation.RestController;
4
5 import java.time.LocalDate;
6 import java.util.ArrayList;
7 import java.util.List;
8 import java.util.concurrent.atomic.AtomicIntegerFieldUpdater;
9
10 @RestController
11 @RequestMapping("/api/cassetts/")
12 public class VideoCassetteApi {
13     private List<VideoCassette> videoCassettes;
14
15     public VideoCassetteApi() {
16         videoCassettes = new ArrayList<>();
17         videoCassettes.add(new VideoCassette(1L, "Tytuł1", LocalDate.of(1999, 1, 1)));
18         videoCassettes.add(new VideoCassette(2L, "Tytuł2", LocalDate.of(1998, 2, 3)));
19     }
20
21     @GetMapping("/all")
22     public List<VideoCassette> getAll() {
23         return videoCassettes;
24     }
25
26     @GetMapping
27     public VideoCassette getById(@RequestParam int index) {
28         VideoCassette first = videoCassettes.stream().filter(element -> element.getId() == index).findFirst().get();
29         return first;
30     }
31 }
```

Zmodyfikuj wyrażenie aby użyć typu Optional, który zabezpiecza przed wyjątkiem związanym z wystąpieniem wartości typu null. Sformatuj odpowiednio kod zgodnie z konwencją programowania funkcyjnego.



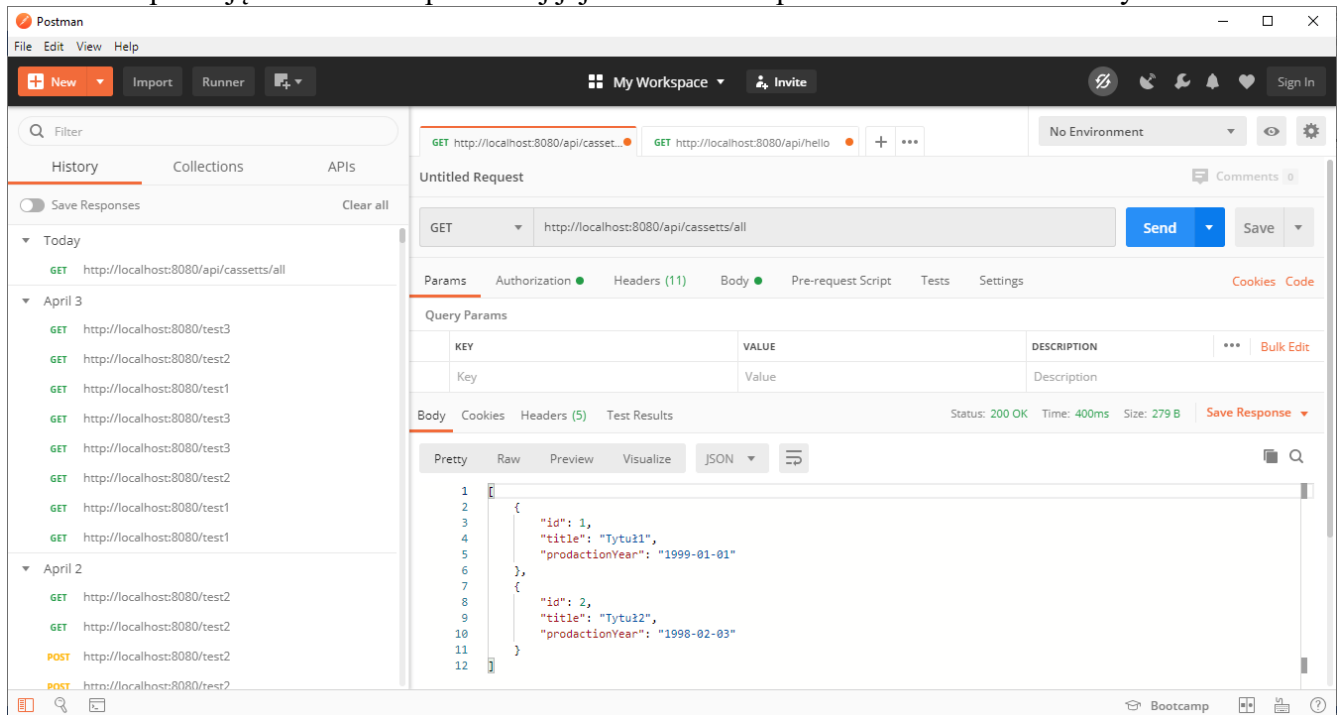
```
1 import java.util.Optional;
2 import java.util.concurrent.atomic.AtomicIntegerFieldUpdater;
3
4 @RestController
5 @RequestMapping("/api/cassetts/")
6 public class VideoCassetteApi {
7     private List<VideoCassette> videoCassettes;
8
9     public VideoCassetteApi() {
10         videoCassettes = new ArrayList<>();
11         videoCassettes.add(new VideoCassette(1L, "Tytuł1", LocalDate.of(1999, 1, 1)));
12         videoCassettes.add(new VideoCassette(2L, "Tytuł2", LocalDate.of(1998, 2, 3)));
13     }
14
15     @GetMapping("/all")
16     public List<VideoCassette> getAll() {
17         return videoCassettes;
18     }
19
20     @GetMapping
21     public Optional<VideoCassette> getById(@RequestParam int index) {
22         Optional<VideoCassette> first = videoCassettes
23             .stream()
24             .filter(element -> element.getId() == index)
25             .findFirst();
26         return first;
27     }
28 }
```

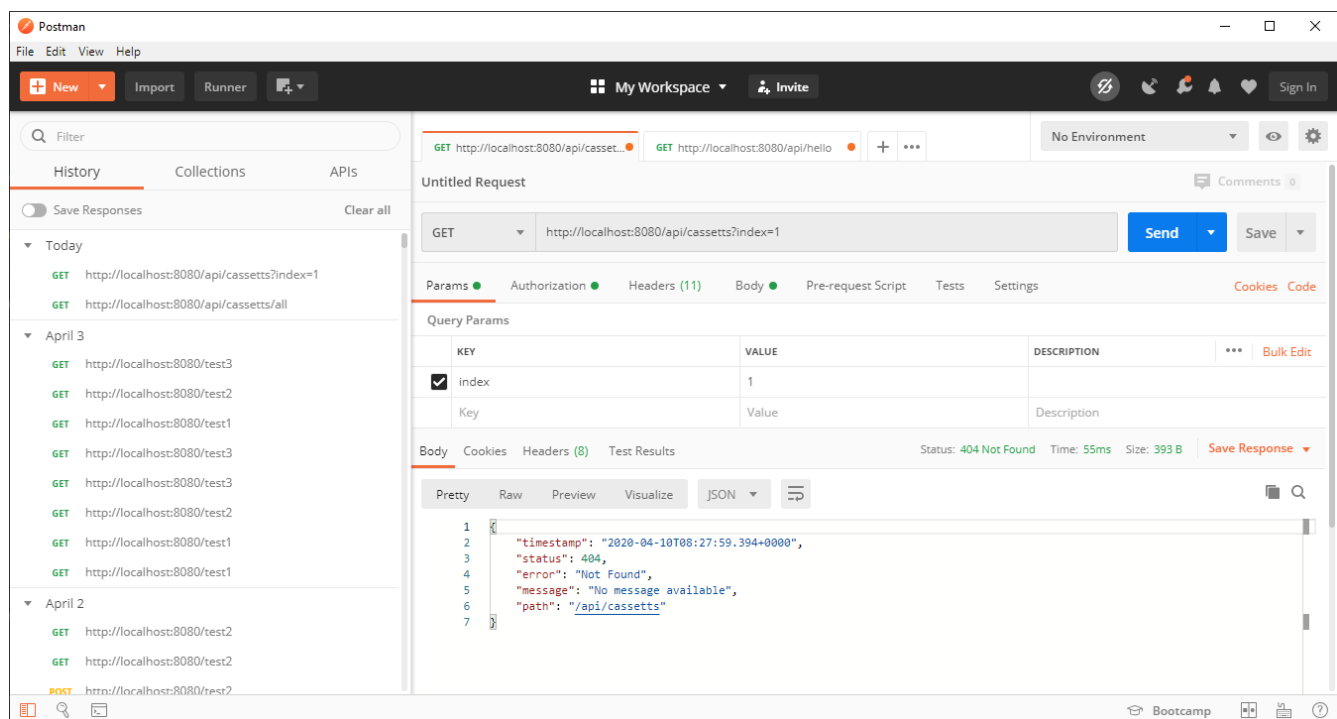

Popraw na 2L.



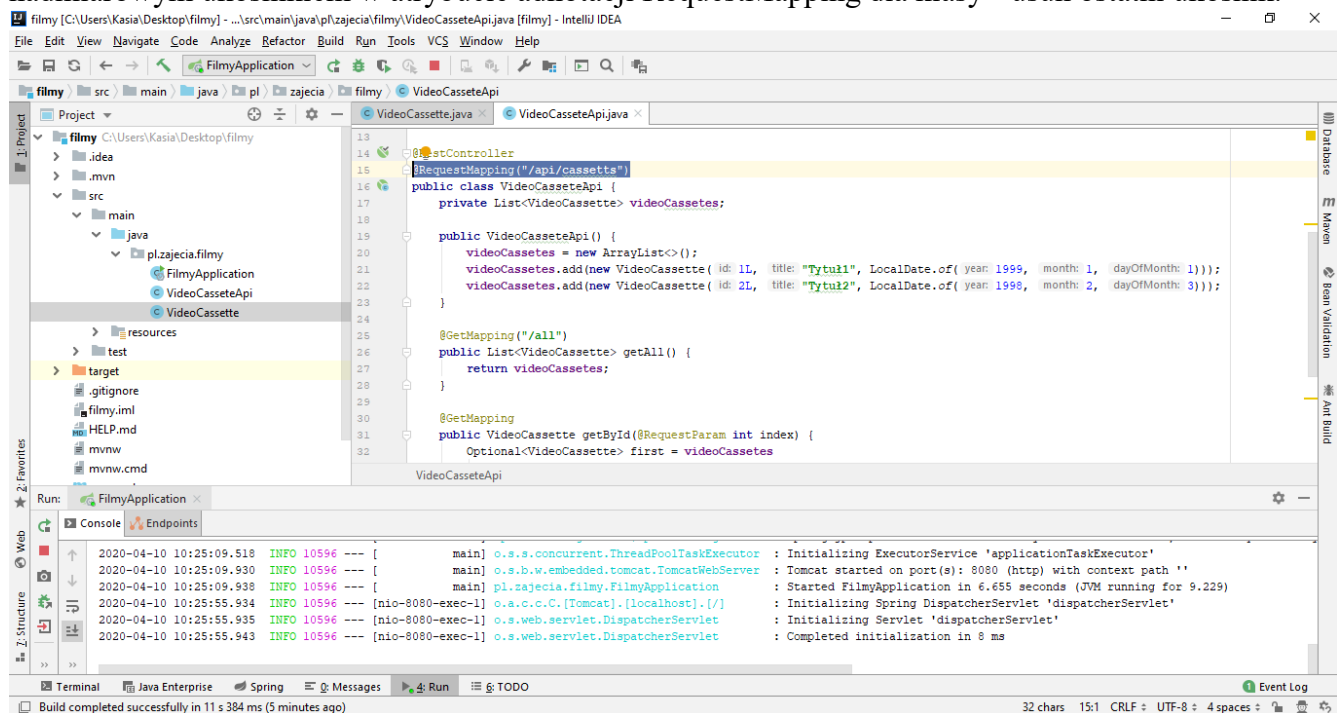
W programowaniu funkcyjnym zamiast używać pętli foreach do przechodzenia po elementach listy w celu sprawdzenia czy dany element ma id zgodne z podanym przez użytkownika, używamy wyrażenia lambda, w którym pobieramy z listy element o id zgodnym z indeksem podanym przed użytkownika odpytującego aplikację.

Uruchom aplikację i Postmana i przetestuj jej działanie. Do pobierania elementów służy metoda GET.





Komunikaty 400 oznaczają błędy po stronie klienta np. że klient coś źle wpisał i nie można odnaleźć danego zasobu. Przy próbie pobrania elementu o zadanym indeksie otrzymujemy błąd spowodowany nadmiarowym ukośnikiem w atrybucie anotacji RequestMapping dla klasy - usuń ostatni ukośnik.



Uruchom jeszcze raz.

Postman interface showing a GET request to `http://localhost:8080/api/cassetts?index=1`. The response is a JSON object:

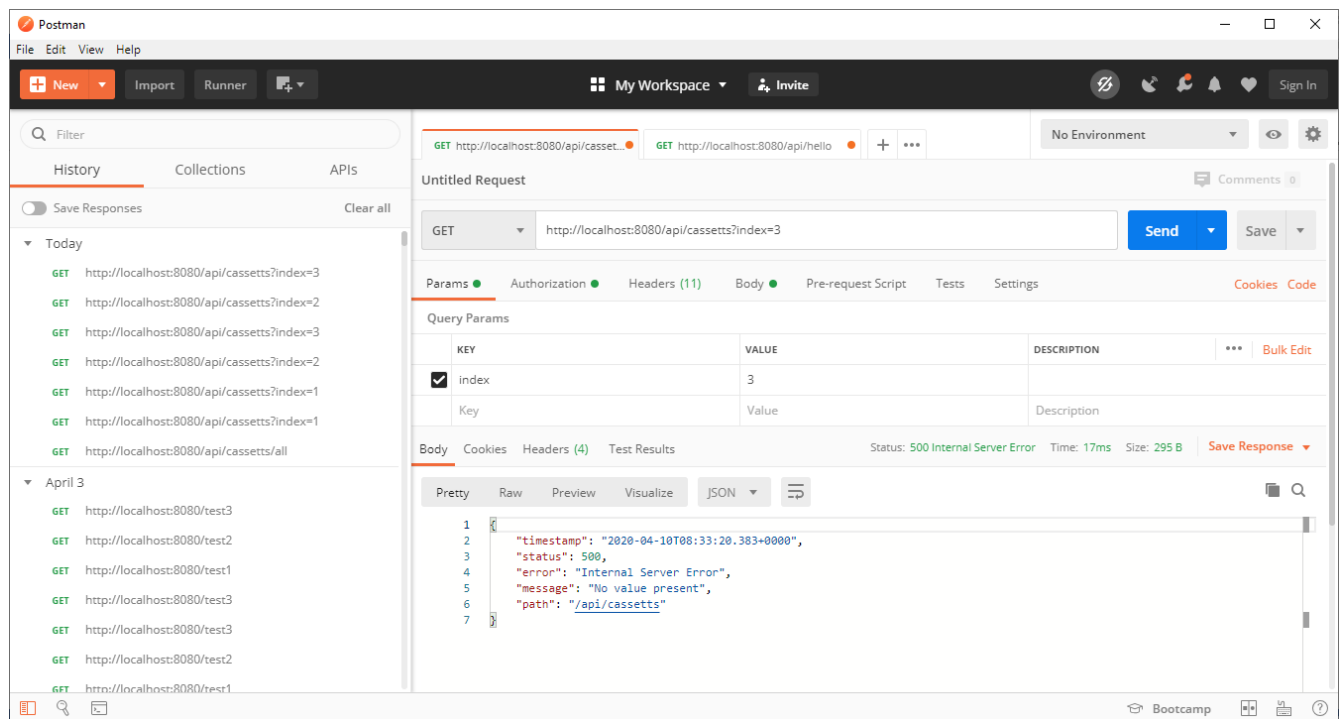
```
{
  "id": 1,
  "title": "Tytuł1",
  "productionYear": "1999-01-01"
}
```

The interface includes a sidebar with a history of requests, a main workspace for editing requests, and a bottom section for viewing response details like status (200 OK), time (323ms), and size (220 B).

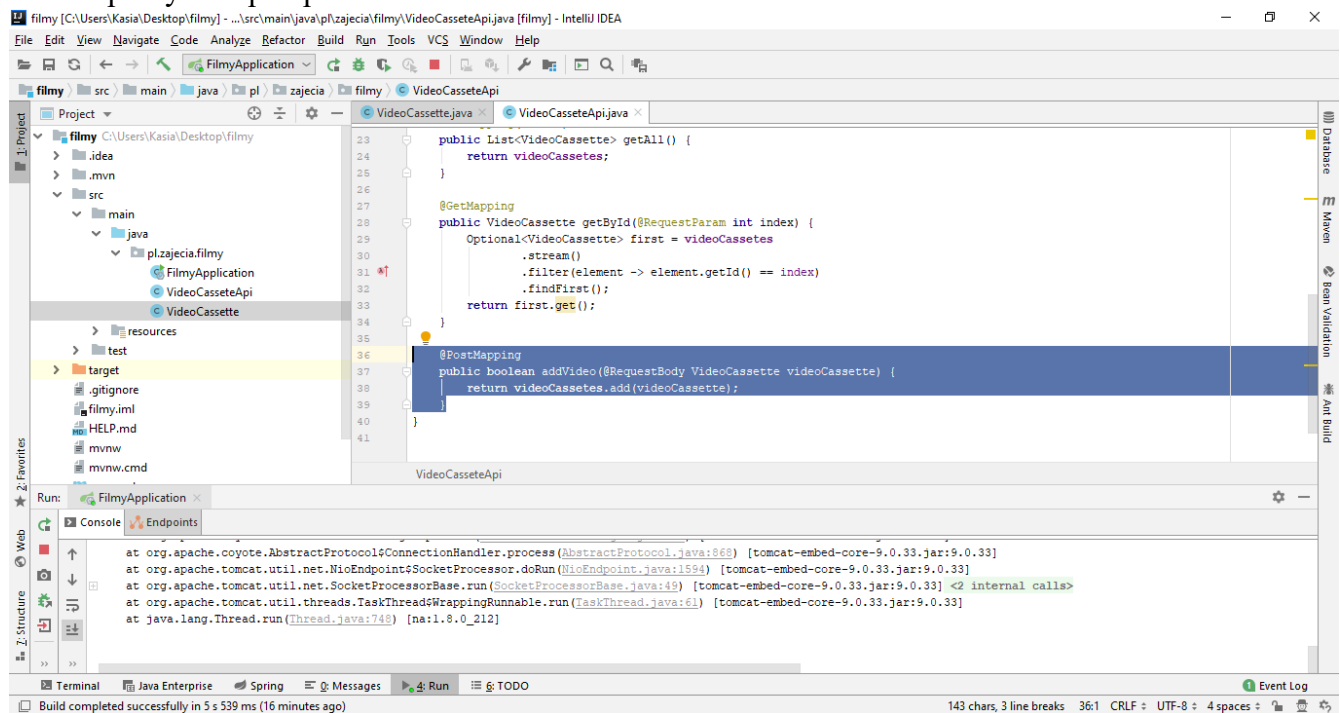
Postman interface showing a GET request to `http://localhost:8080/api/cassetts?index=2`. The response is a JSON object:

```
{
  "id": 2,
  "title": "Tytuł2",
  "productionYear": "1998-02-03"
}
```

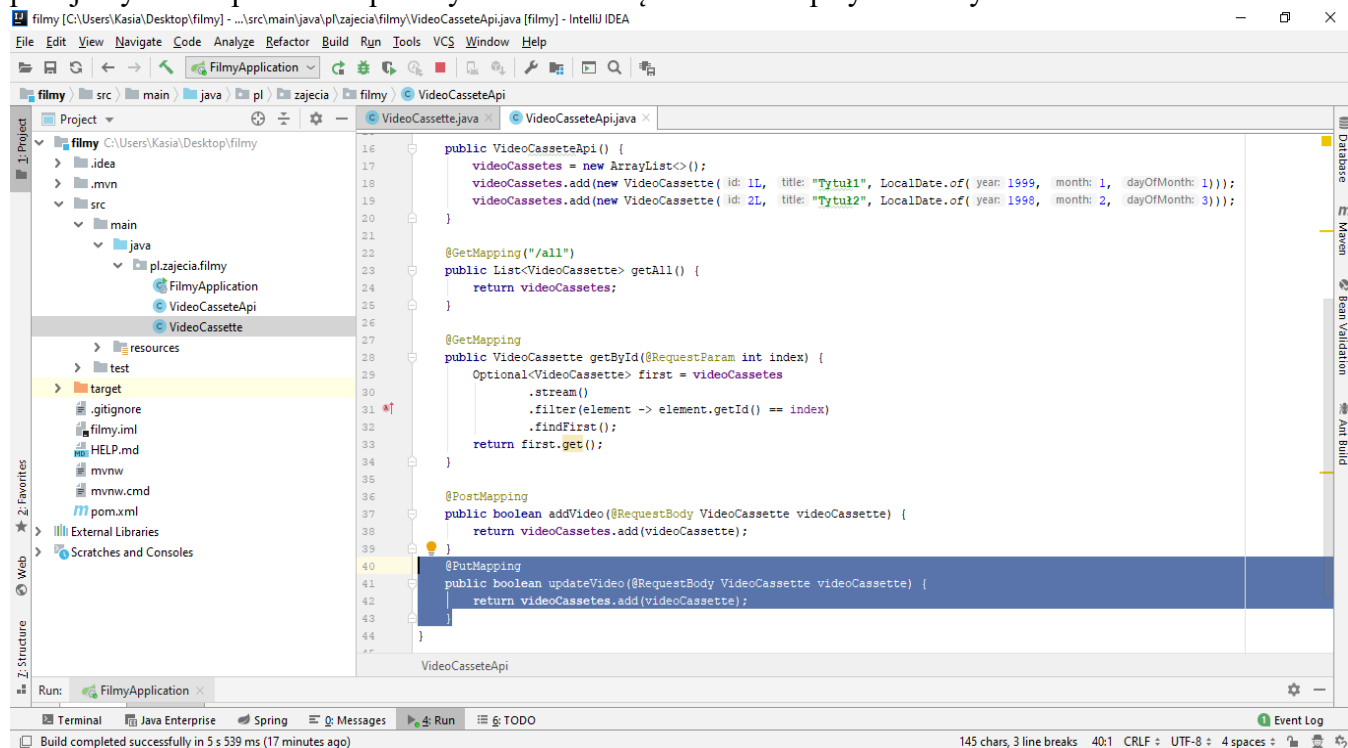
The interface includes a sidebar with a history of requests, a main workspace for editing requests, and a bottom section for viewing response details like status (200 OK), time (14ms), and size (220 B).



Przejdź do implementacji metody `addVideo` realizującej dodawanie elementów, przy użyciu metody `http POST` - użyj adnotacji `PostMapping`. Elementem dodawanym będzie obiekt klasy `VideoCassettes`. Będzie to metoda webowa więc do parametru przekazanego metodzie `add` adnotację `RequestBody`, co oznacza, że ten obiekt zostanie przesłany do aplikacji w postaci serializowanej. <https://www.samouczekprogramisty.pl/serializacja-w-jezyku-java/> Metoda ma zwracać typ `boolean`, co będzie informacją o tym czy wykonanie dodawania obiektu zostało pomyślnie przeprowadzone.

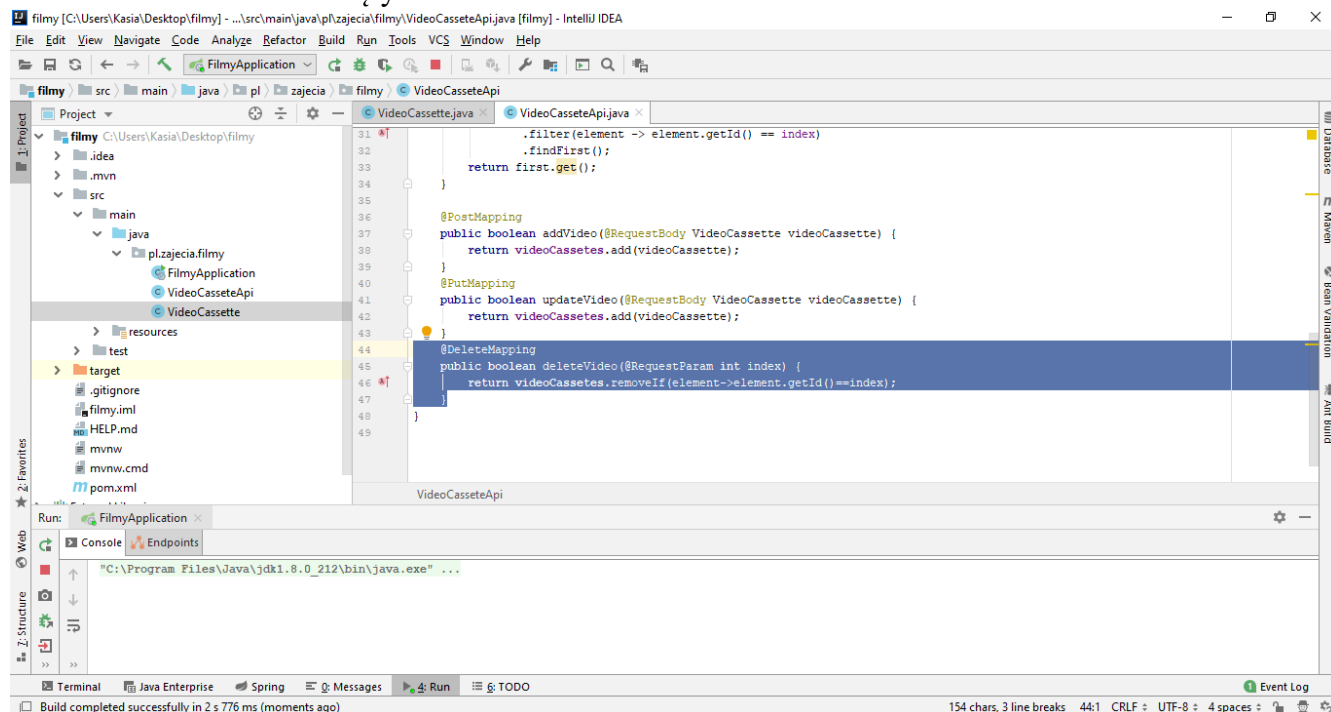


Następnie zaimplementuj metodę do modyfikacji obiektu - użyj adnotacji PutMapping. Metodzie podajemy id i na podstawie podanych wartości będzie ona nadpisywać dany element.



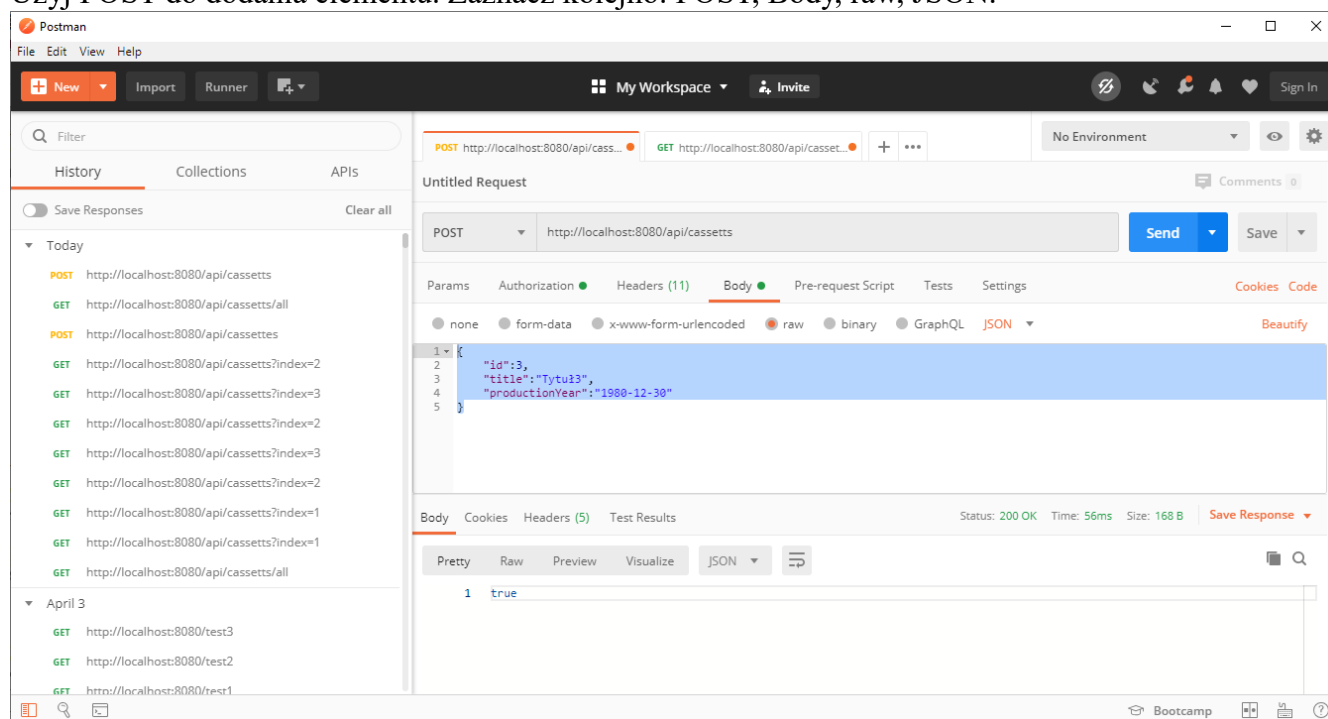
Przy implementacji ostatniej metody, która będzie służyła do usuwania obiektu z listy – użyj adnotacji DeleteMapping.

W ciele metody w ramach instrukcji return użyj metody removeIf w ramach wyrażenia lambda zapisz warunek usunięcia – jeżeli id danego elementu jest równe podanej w parametrze wartości index wówczas zostanie on usunięty.



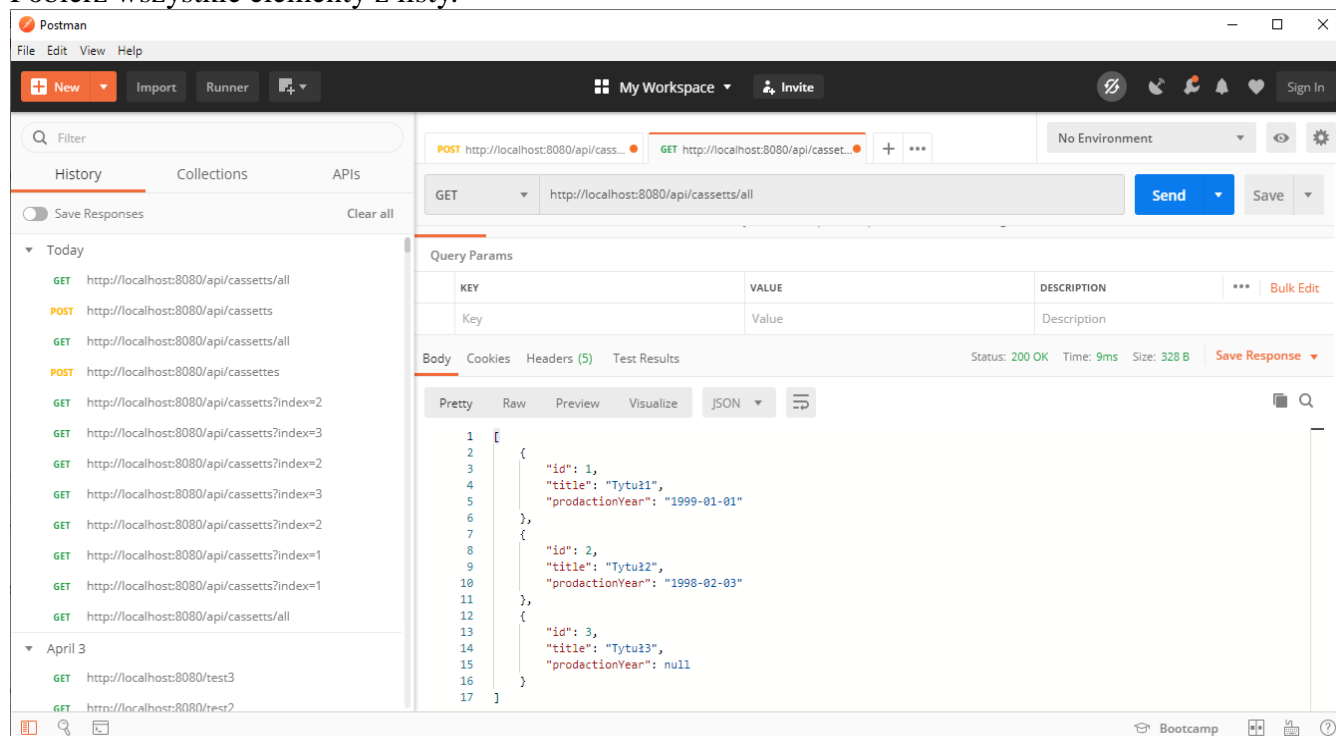
Uruchom aplikację i przetestuj jej działanie w Postmanie.

Użyj POST do dodania elementu. Zaznacz kolejno: POST, Body, raw, JSON.

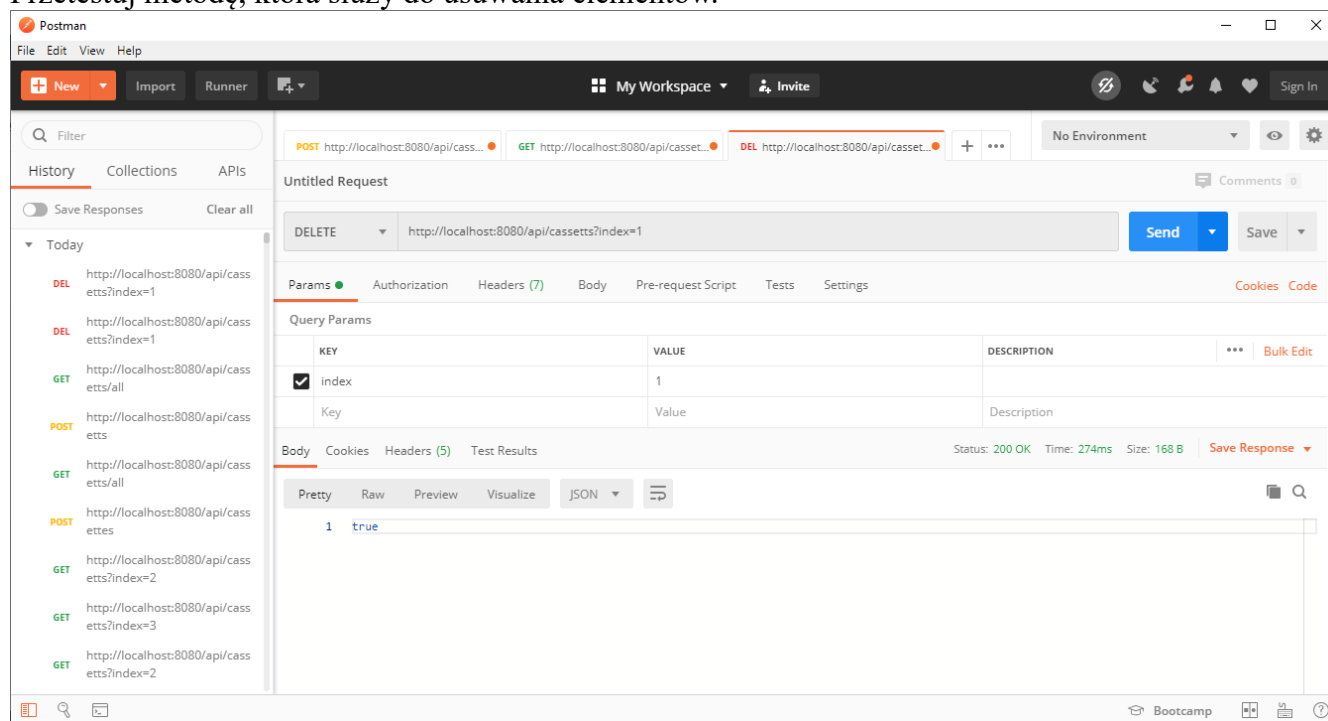


Status odpowiedzi to 200, metoda zwróciła wartość true – co oznacza że dodanie elementu do listy zostało przeprowadzone pomyślnie.

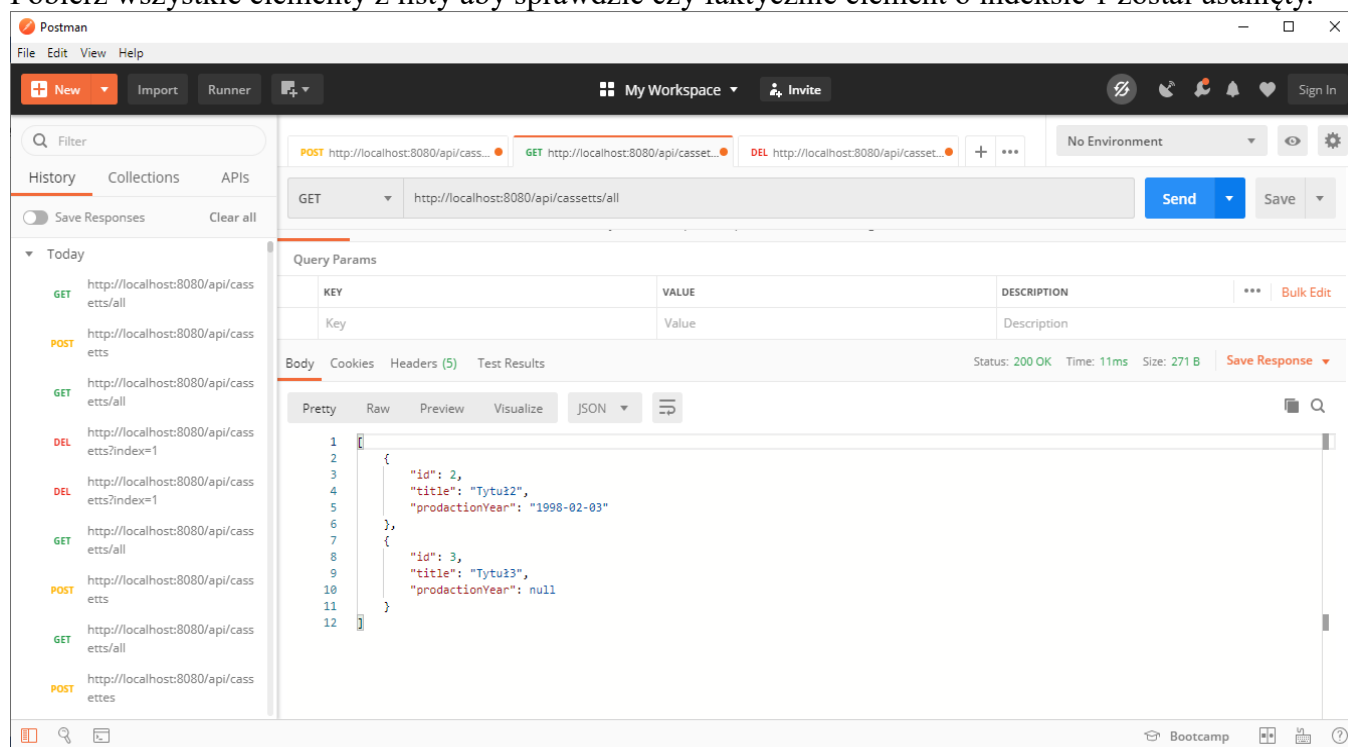
Pobierz wszystkie elementy z listy.



Przetestuj metodę, która służy do usuwania elementów.



Pobierz wszystkie elementy z listy aby sprawdzić czy faktycznie element o indeksie 1 został usunięty.



Różnica między PUT a PATCH. PUT służy do modyfikacji/update całego elementu natomiast PATCH do np. przesłania jednego pola edytowanego obiektu, czyli jeśli będziemy chcieli zamienić cały obiekt nadpisać title i productionYear to należy użyć do tego PUT, jeśli natomiast chcemy zmienić tylko jedno pole np. title wówczas lepiej posłużyć się PATCH'em.