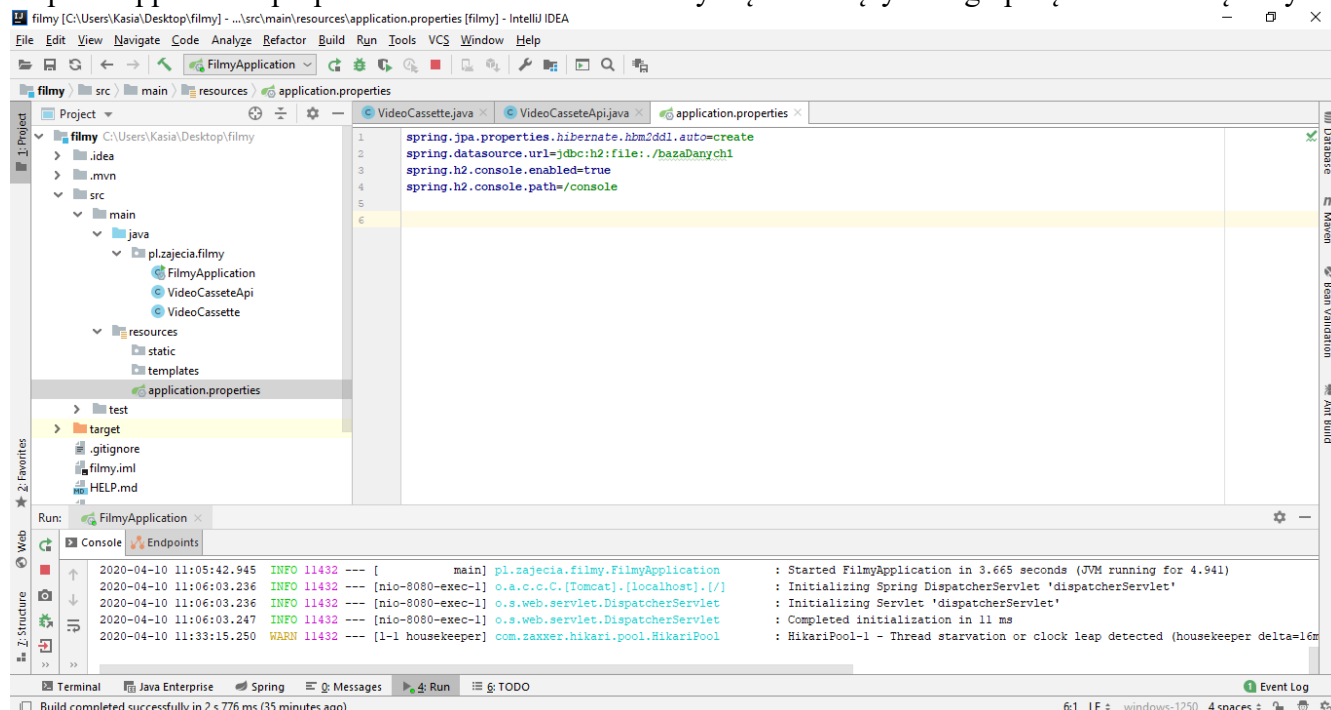


Po przetestowaniu API aplikacji przystąpimy do realizacji aplikacji zapisującej dane w bazie danych. W pliku `application.properties` ustaw właściwości dotyczące nawiązywanego połączenia z bazą danych.

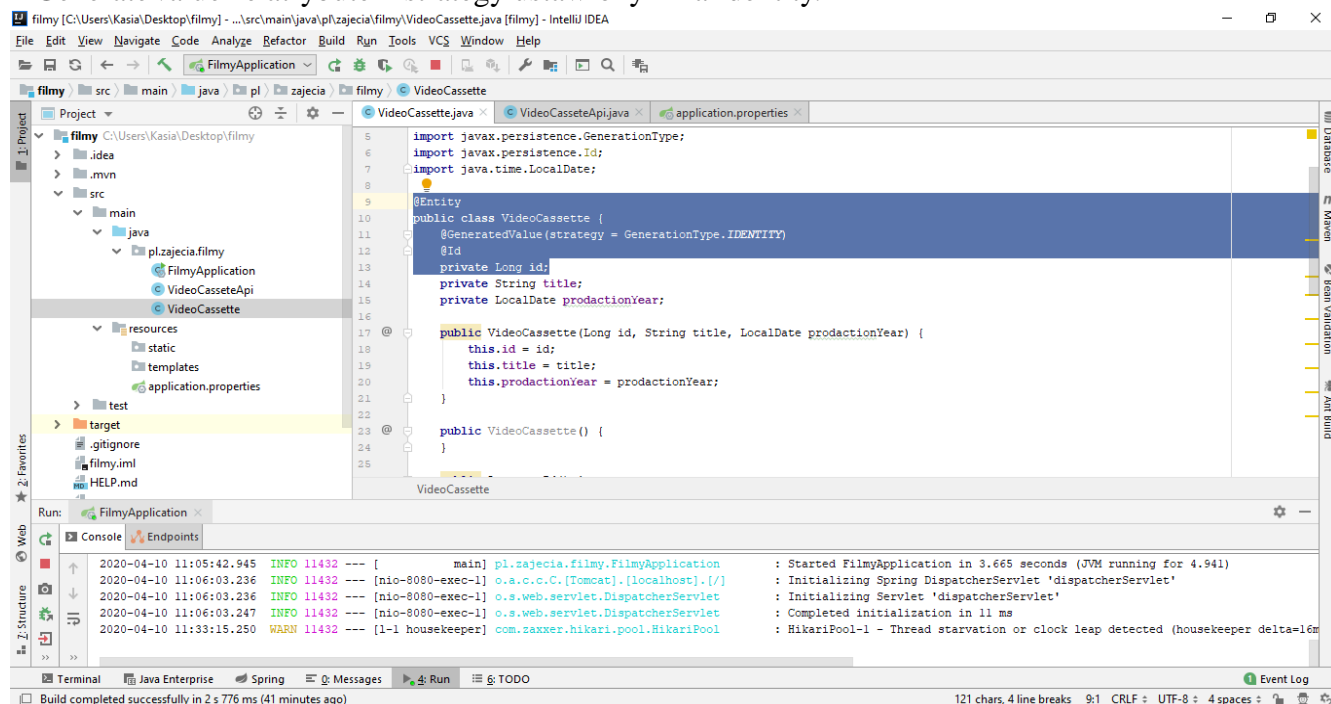


W momencie, gdy chcemy obiekty klasy przechowywać w bazie danych należy klasę zamienić na encję.

Do klasy dodaj adnotację `Entity`, a do pola które będzie kluczem głównym dwie adnotacje:

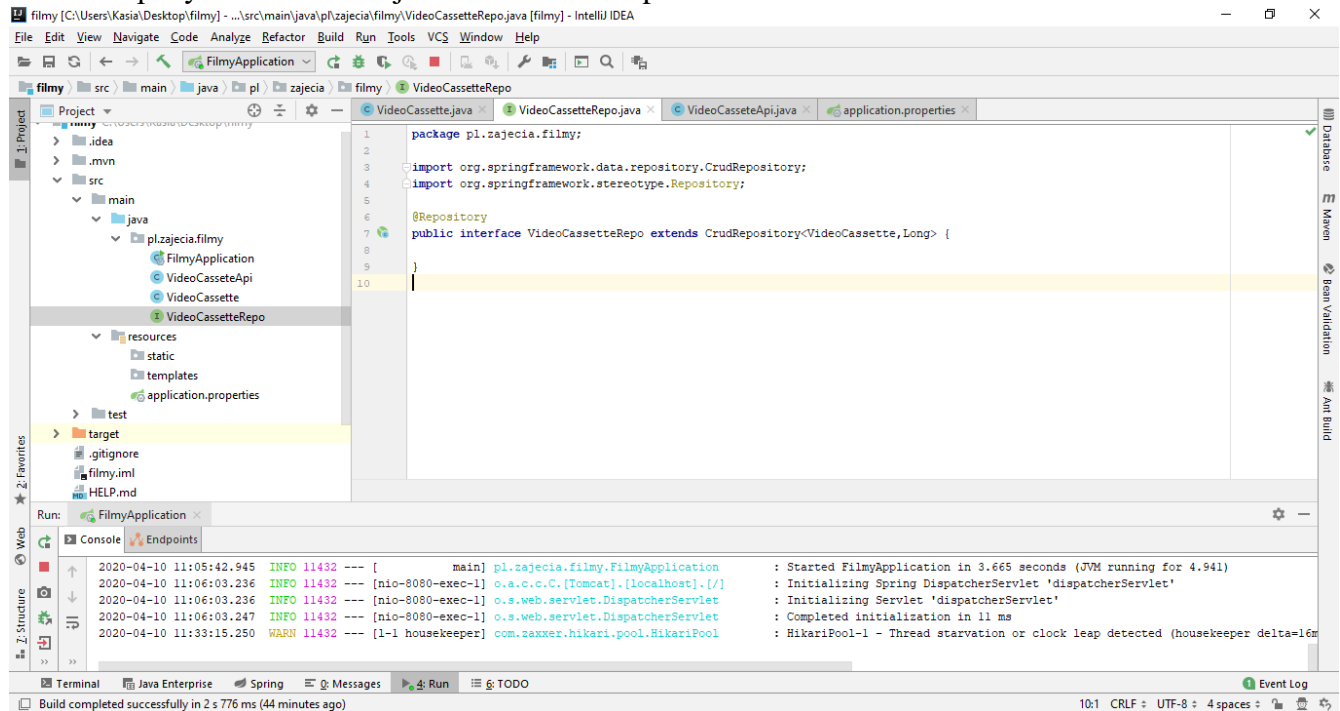
– `Id` oraz

– `GeneratedValue` ze atrybutem strategy ustawionym na `Identity`.



Klasa powinna mieć również gettery i settery do wszystkich pól oraz bezparametrowy konstruktor – upewnij się że je ma.

Utwórz repozytorium - interfejs VideoCassetteRepo.



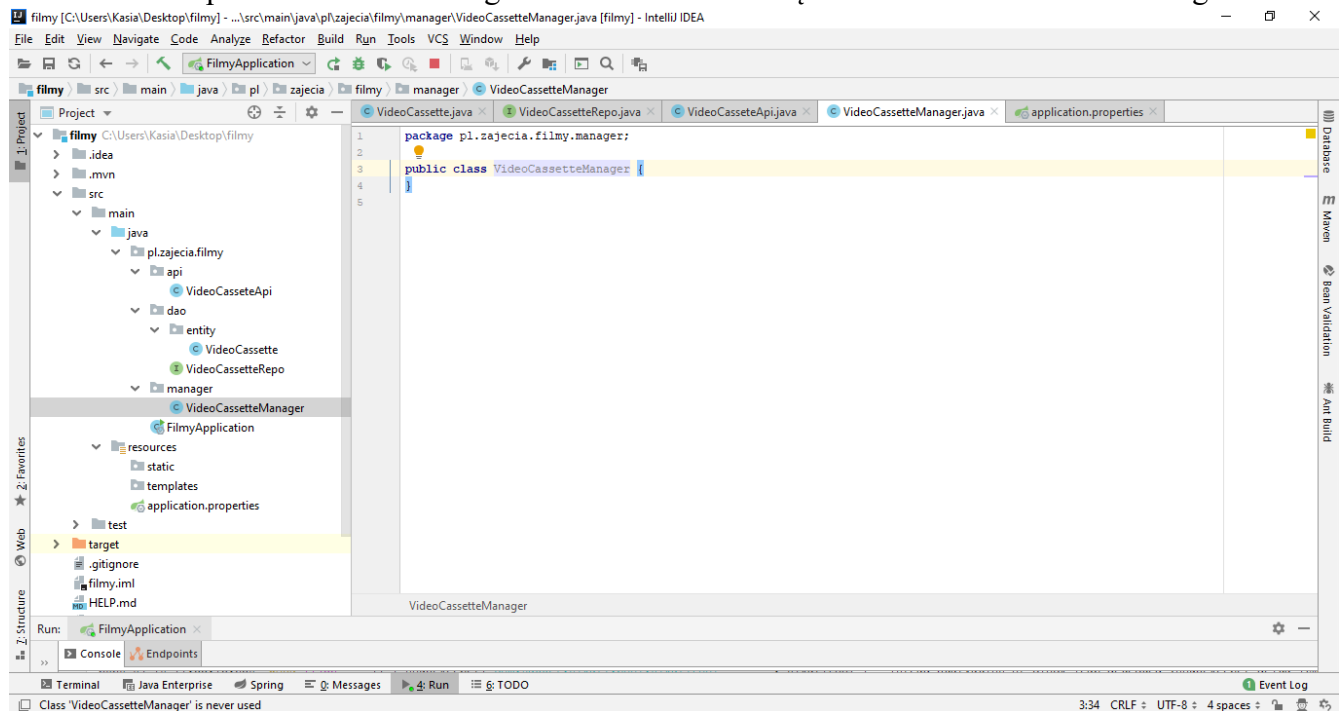
Uporządkuj kod tworzonego projektu, aby miał strukturę warstwową, co ułatwi zadanie utrzymania aplikacji, jej rozwoju i poprawy błędów.

Utwórz paczkę dao – warstwa dostępu do danych.

Do dao wrzuć repozytorium i utwórz w nim pakiet, który będzie przechowywał encje.

W projekcie utwórz jeszcze jeden pakiet api i umieść w nim VideoCassetteApi.

Utwórz ostatni pakiet o nazwie manager a w nim utwórz klasę o nazwie VideoCassetteManager.



Zadaniem klasy menagera będzie separacja warstwy dostępu do danych od warstwy api aplikacji.

Nie powinno się wiązać bezpośrednio logiki aplikacji, która odpowiada za dostęp do danych z api, ponieważ stanowi to złą praktykę, która naraża aplikację na błędy.

Tworzenie warstwy pośredniej między warstwą dostępu do danych a api stanowi dobrą praktykę - będzie nią klasa VideoCassetteManager.

Przy implementowaniu logiki biznesowej gdy stworzymy api i stworzymy dostęp do danych to nie powinniśmy bezpośrednio tych dwóch rzeczy wiązać, powinna być między nimi jeszcze jedna warstwa i tu jest nią klasa menagera i jest to praktyką w tworzeniu tego typu aplikacji.

Klasa menagera będzie pełniła rolę serwisu i będzie realizowała podobne operacje co repozytorium.

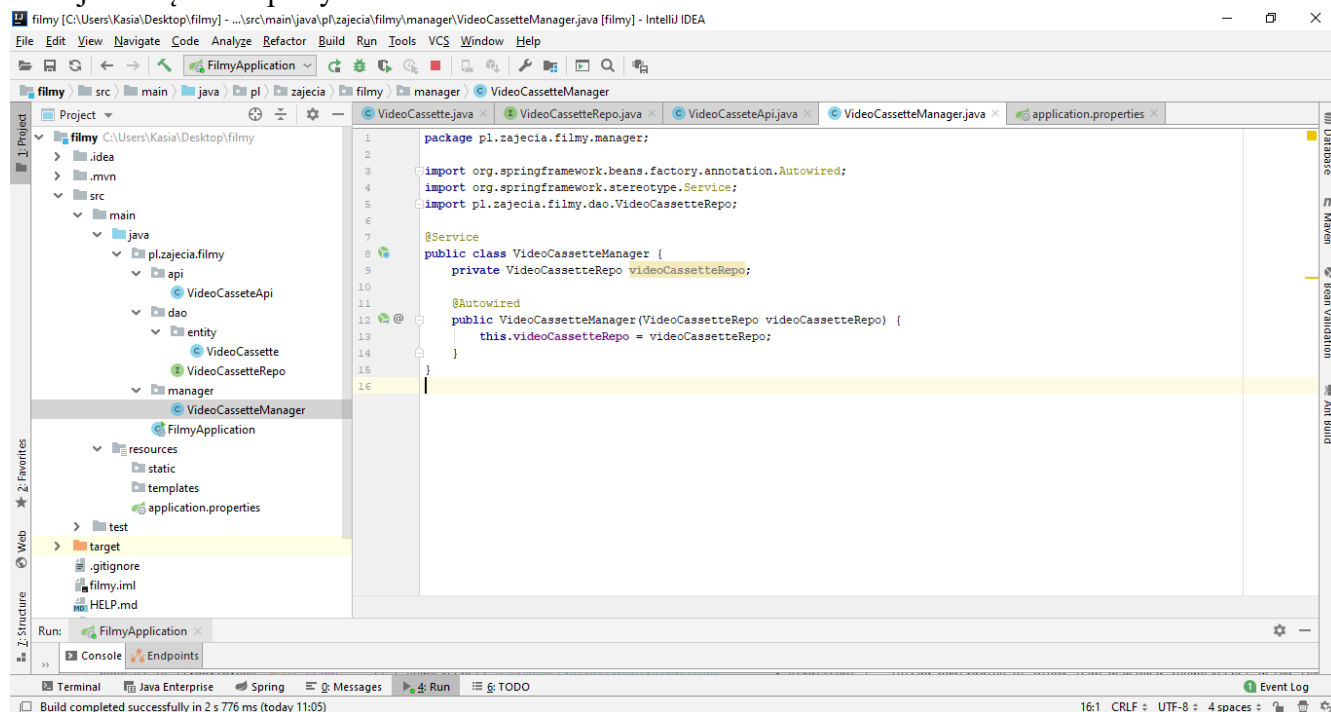
Do klasy dodaj adnotację Service.

Utwórz w niej zmienną, która będzie przechowywała informację na temat repozytorium.

Dodaj konstruktor w ramach którego wstrzyknij instancję klasy repozytorium.

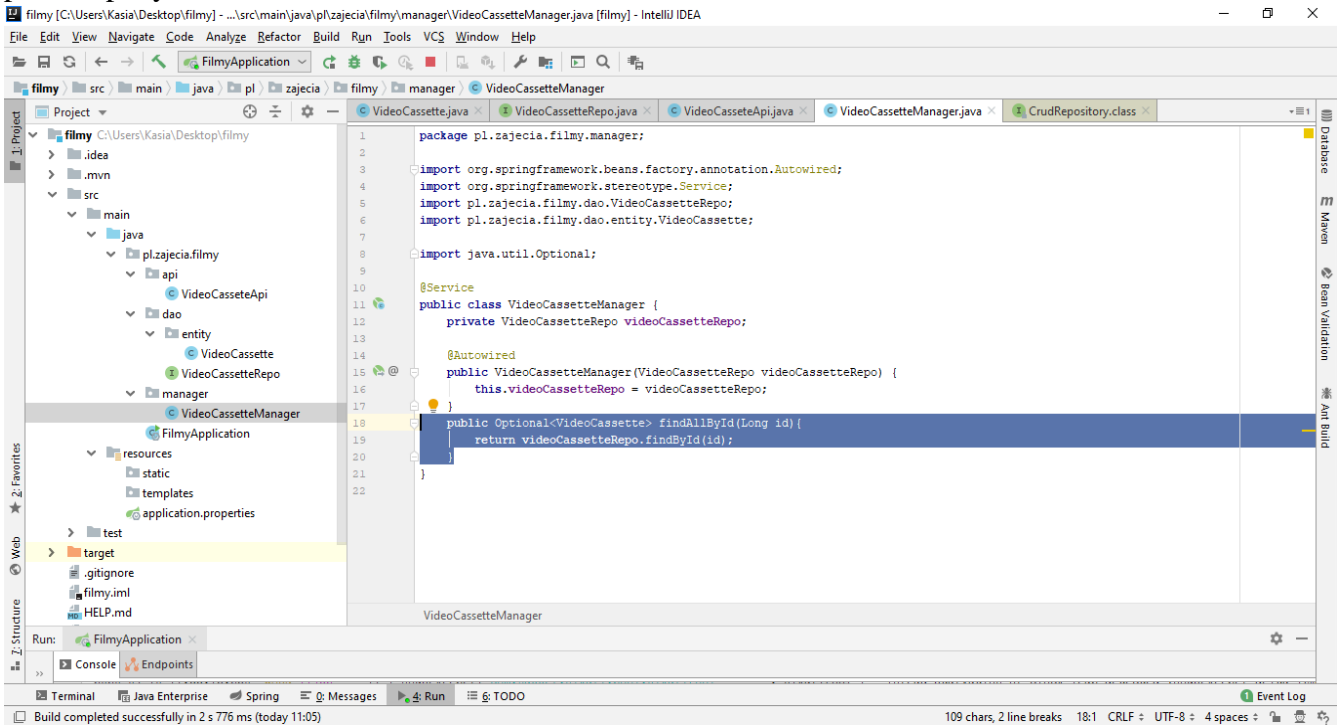
Nad konstruktorem umieść adnotację Autowired.

Dodaj niezbędne importy.

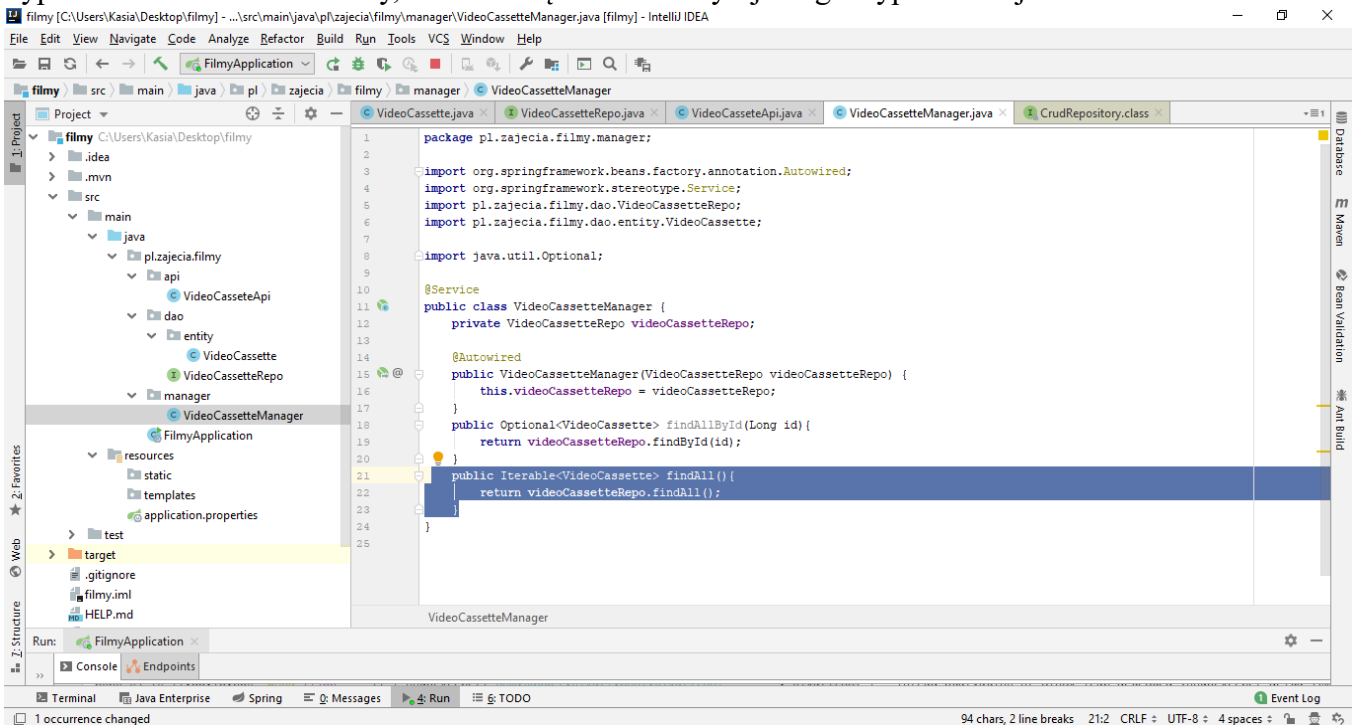


Utwórz metody, które będą odpowiadały metodom znajdującym się w repozytorium i będą wywoływane w ramach implementacji metod klasy api aplikacji.

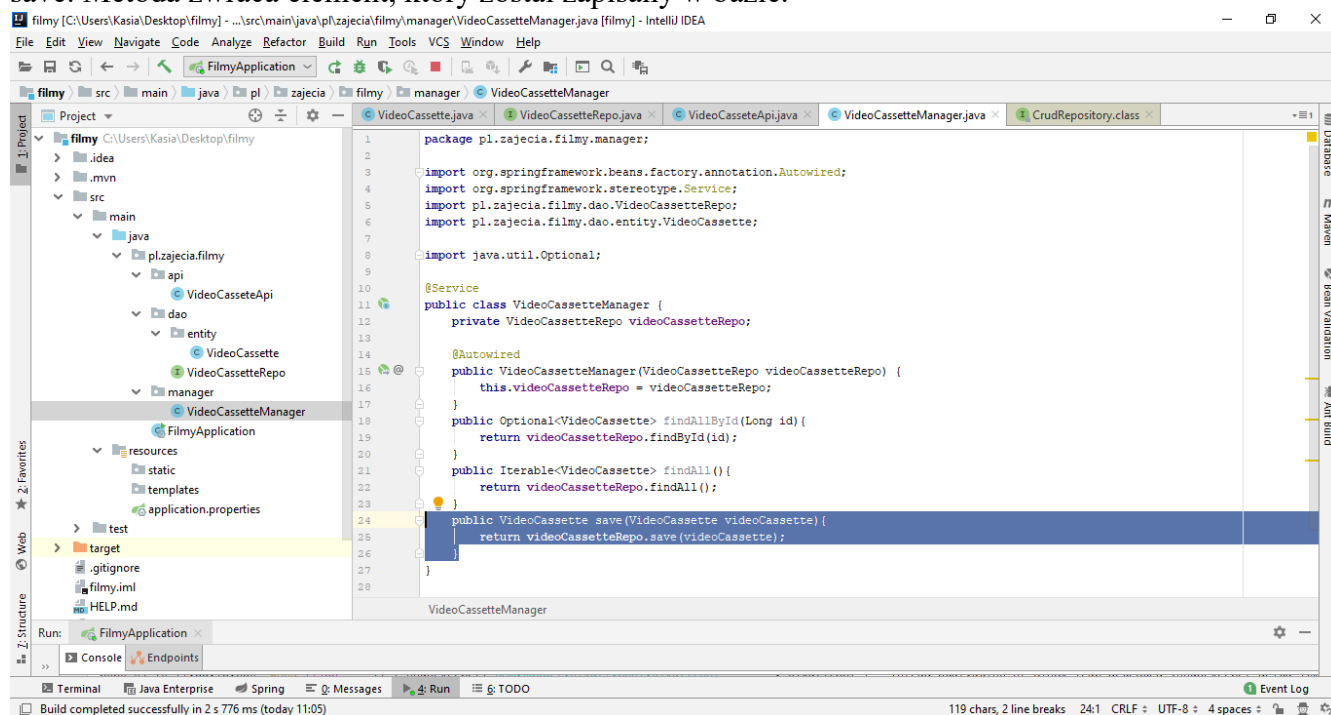
Pierwsza metoda będzie odpowiadała za pobieranie danych - utwórz metodę, która na podstawie podanego id dostarcza informacje o konkretnej kasecie wideo.
Typ Optional zabezpiecza przed wyjątkiem związanym z null. Metoda wywołuje metodę dostępną przez repozytorium.



Druga metoda o nazwie findAll ma pobierać wszystkie elementy z bazy danych.
Typ Iterable oznacza elementy, które da się iterować czyli jakiegoś typu kolekcję.



Zaimplementuj metodę realizującą dodawanie elementu do bazy – repozytorium udostępnia metodę save. Metoda zwraca element, który został zapisany w bazie.



The screenshot shows the IntelliJ IDEA interface with the 'VideoCassetteManager.java' file open. The code defines a service class that interacts with a repository. The 'save' method is highlighted, showing it takes a 'VideoCassette' object and returns the saved object from the repository.

```
package pl.zajecia.filmly.manager;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import pl.zajecia.filmly.dao.VideoCassetteRepo;
import pl.zajecia.filmly.dao.entity.VideoCassette;
import java.util.Optional;

@Service
public class VideoCassetteManager {
    private VideoCassetteRepo videoCassetteRepo;

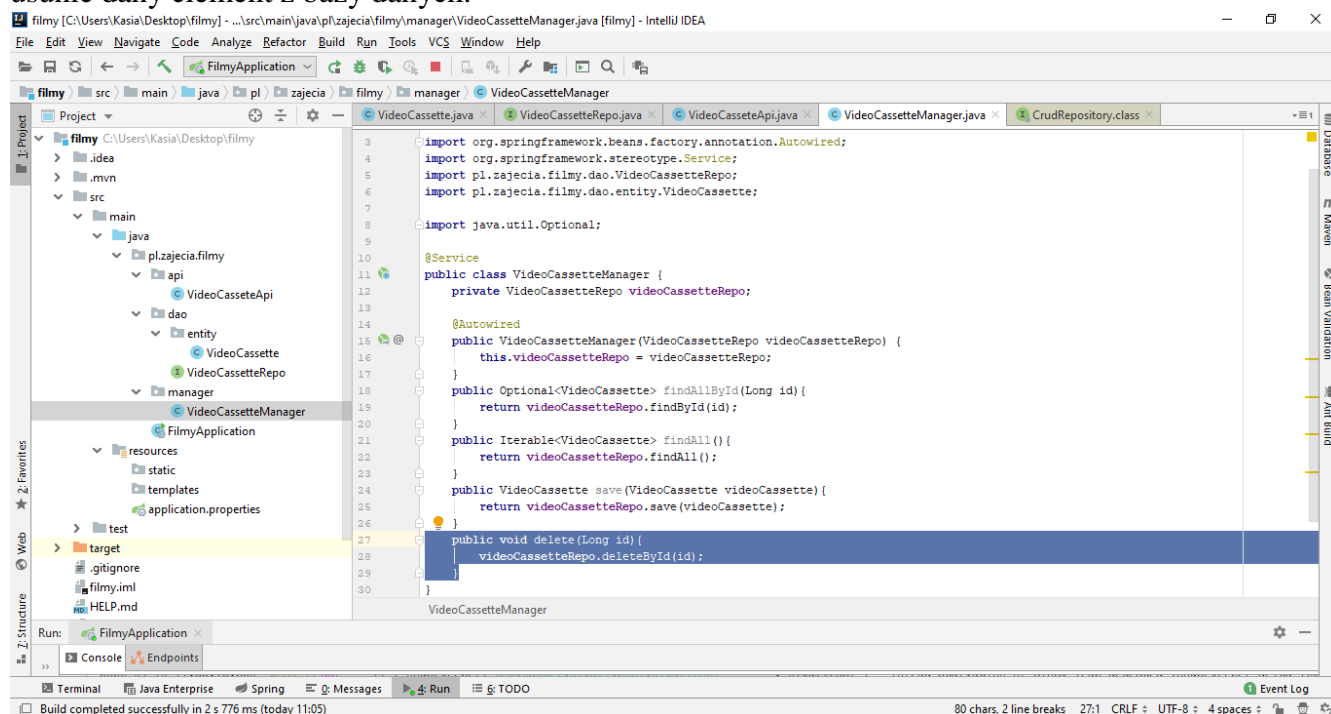
    @Autowired
    public VideoCassetteManager(VideoCassetteRepo videoCassetteRepo) {
        this.videoCassetteRepo = videoCassetteRepo;
    }

    public Optional<VideoCassette> findById(Long id) {
        return videoCassetteRepo.findById(id);
    }

    public Iterable<VideoCassette> findAll() {
        return videoCassetteRepo.findAll();
    }

    public VideoCassette save(VideoCassette videoCassette) {
        return videoCassetteRepo.save(videoCassette);
    }
}
```

Zaimplementuj w serwisie metodę do usuwania elementów, która na podstawie dostarczonego id usunie dany element z bazy danych.



The screenshot shows the IntelliJ IDEA interface with the 'VideoCassetteManager.java' file open. The code defines a service class that interacts with a repository. The 'delete' method is highlighted, showing it takes a 'Long id' and calls the 'deleteById' method on the repository.

```
package pl.zajecia.filmly.manager;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import pl.zajecia.filmly.dao.VideoCassetteRepo;
import pl.zajecia.filmly.dao.entity.VideoCassette;
import java.util.Optional;

@Service
public class VideoCassetteManager {
    private VideoCassetteRepo videoCassetteRepo;

    @Autowired
    public VideoCassetteManager(VideoCassetteRepo videoCassetteRepo) {
        this.videoCassetteRepo = videoCassetteRepo;
    }

    public Optional<VideoCassette> findById(Long id) {
        return videoCassetteRepo.findById(id);
    }

    public Iterable<VideoCassette> findAll() {
        return videoCassetteRepo.findAll();
    }

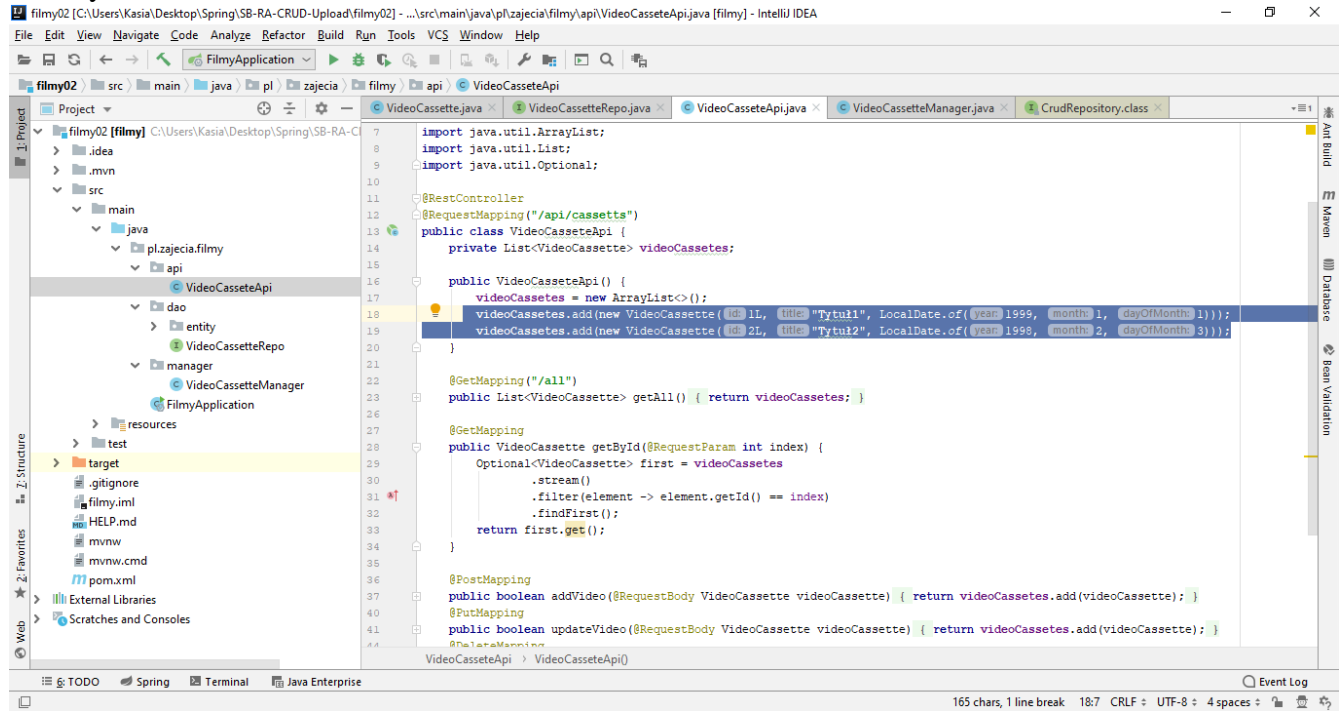
    public VideoCassette save(VideoCassette videoCassette) {
        return videoCassetteRepo.save(videoCassette);
    }

    public void delete(Long id) {
        videoCassetteRepo.deleteById(id);
    }
}
```

Po zaimplementowaniu serwisu należy teraz przejść do klasy z api aplikacji i wstrzyknąć w niej instancję klasy menagera.

Aplikację należy zmodyfikować tak, aby informacje o filmie zapisywały się w bazie danych a nie na liście.

Skopiuj z konstruktora klasy api tworzenie dwóch obiektów i wklej w klasie menagera w ramach metody fillDB.



```
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/cassetts")
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; }

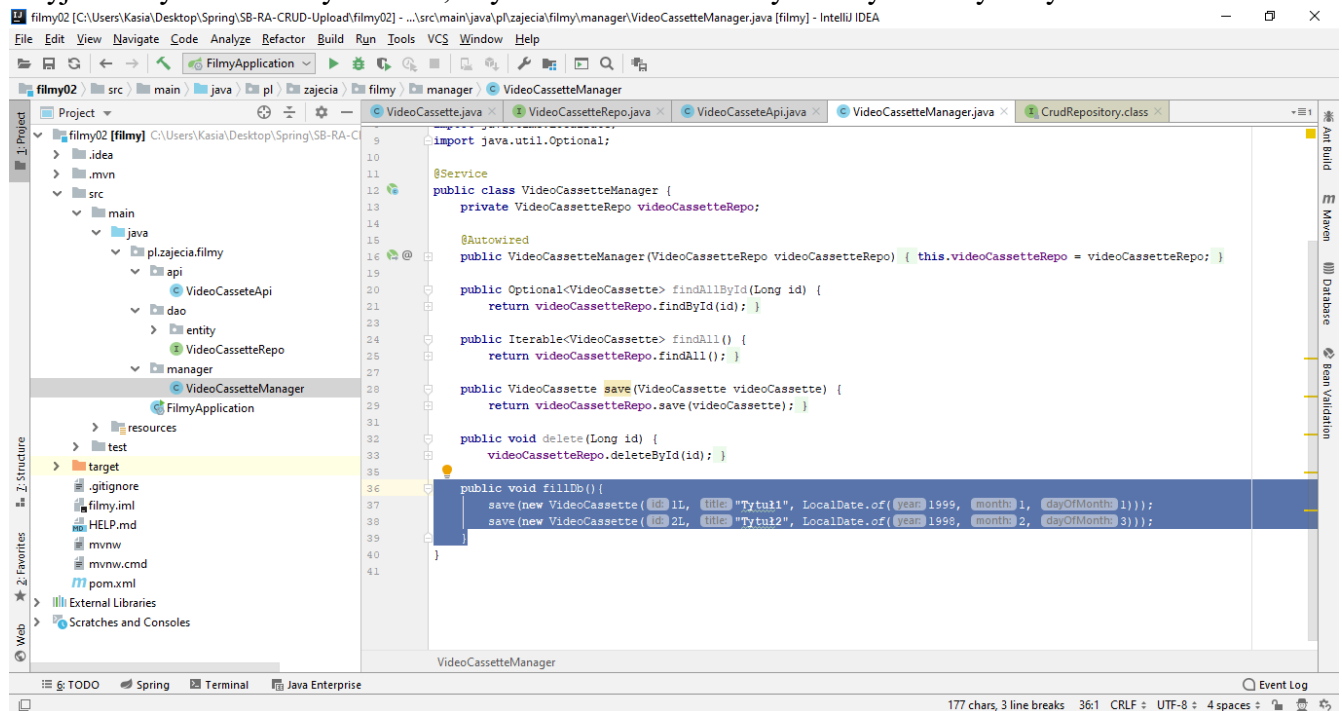
    @GetMapping("/{id}")
    public VideoCassette getById(@RequestParam int index) {
        Optional<VideoCassette> first = videoCassettes
            .stream()
            .filter(element -> element.getId() == index)
            .findFirst();
        return first.get();
    }

    @PostMapping
    public boolean addVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }

    @PutMapping
    public boolean updateVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }

    @DeleteMapping
    public boolean deleteVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.remove(videoCassette); }
}
```

Użyj metody save z klasy serwisu, aby dodać te dwa obiekty - filmy do bazy danych.



```
import java.util.Optional;

@Service
public class VideoCassetteManager {
    private VideoCassetteRepo videoCassetteRepo;

    @Autowired
    public VideoCassetteManager(VideoCassetteRepo videoCassetteRepo) { this.videoCassetteRepo = videoCassetteRepo; }

    public Optional<VideoCassette> findById(Long id) {
        return videoCassetteRepo.findById(id);
    }

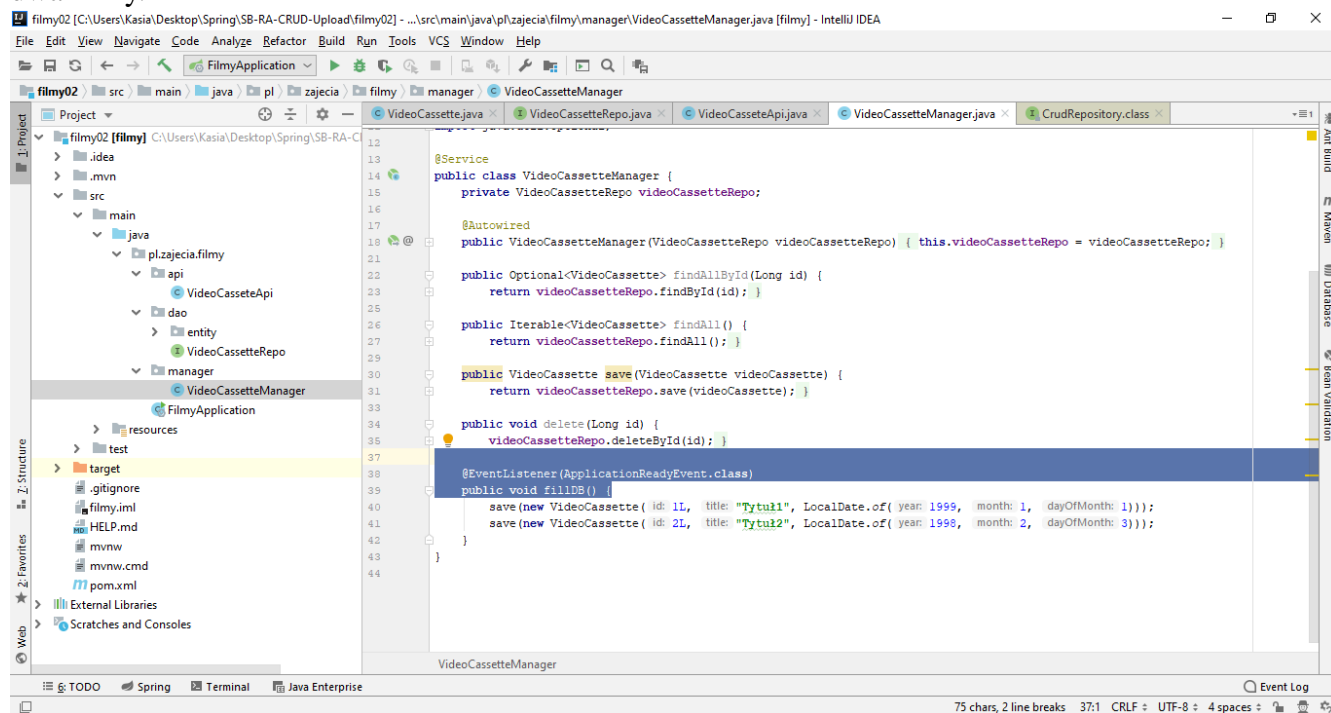
    public Iterable<VideoCassette> findAll() {
        return videoCassetteRepo.findAll();
    }

    public VideoCassette save(VideoCassette videoCassette) {
        return videoCassetteRepo.save(videoCassette);
    }

    public void delete(Long id) {
        videoCassetteRepo.deleteById(id);
    }

    public void fillDb() {
        save(new VideoCassette(1L, "Tytuł1", LocalDate.of(1999, 1, 1)));
        save(new VideoCassette(2L, "Tytuł2", LocalDate.of(1998, 2, 3)));
    }
}
```

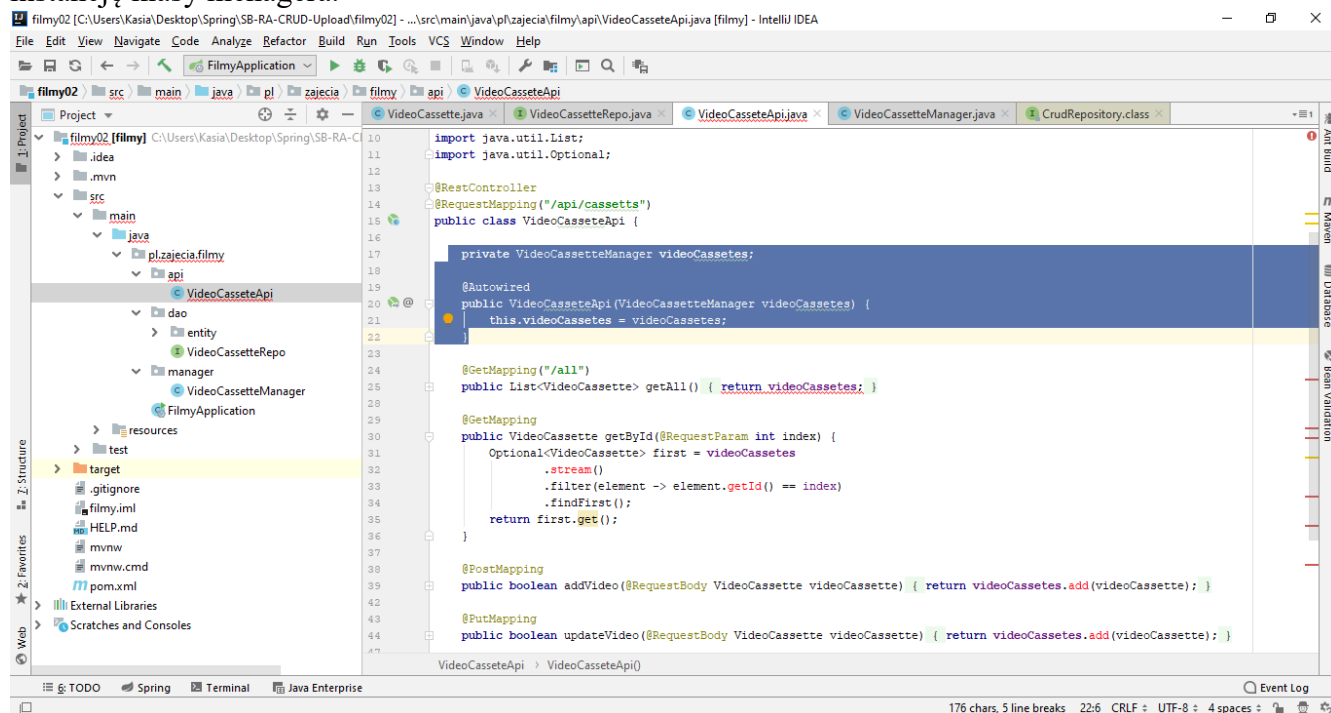
Do metody dodaj specjalną adnotację `EventListener` z atrybutem `ApplicationReadyEvent`, która oznacza nasłuchiwanie zdarzenia uruchomienia aplikacji. Ciało metody `fillDB` oznaczone tą adnotacją, zostanie wykonane po uruchomieniu aplikacji, co spowoduje, że na jej starcie w bazie danych zostaną zapisane dwa filmy.



W klasie `api` „zastąp” listę - bazą danych.

Nie będziemy już potrzebowali listy, bo informacje o kasetach będą przechowywane w bazie danych - usuń listę i dotychczasowy konstruktor w którym dodajemy kasety do listy.

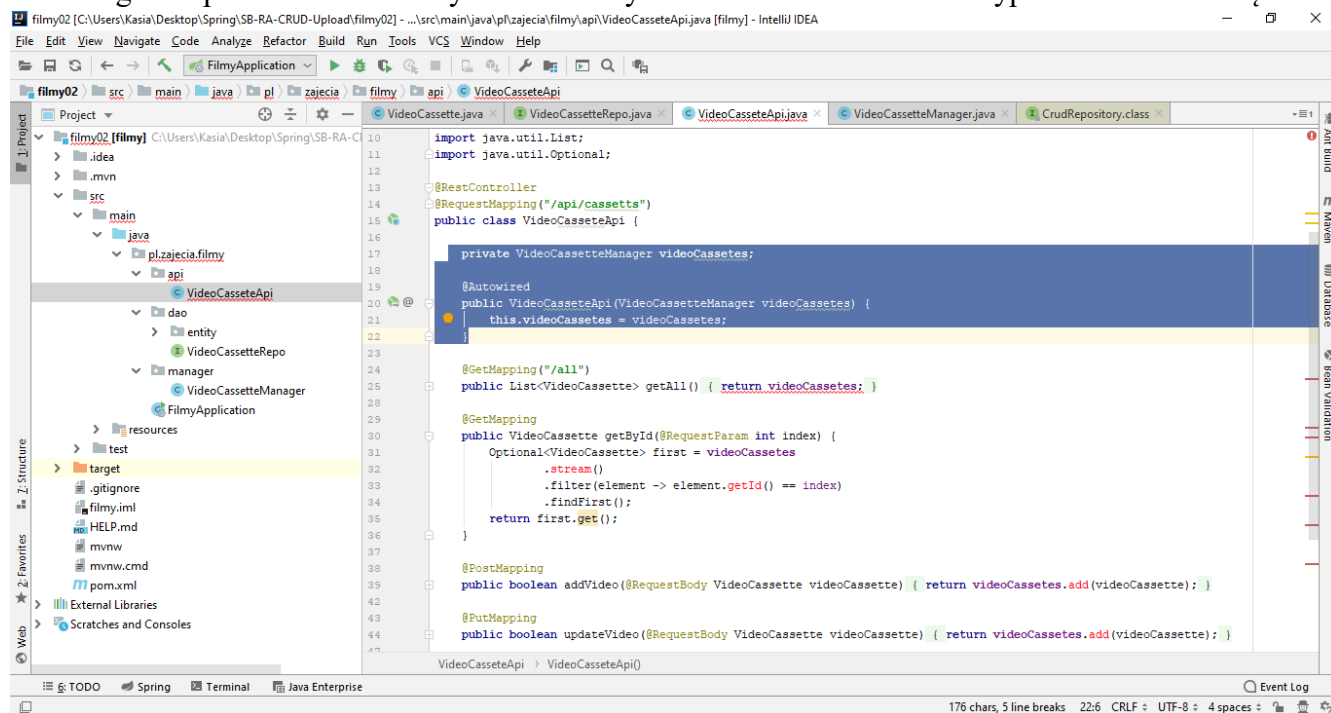
W to miejsce utwórz deklarację obiektu klasy menagera, dodaj import, i w konstruktorze wstrzyknij instancję klasy menagera.



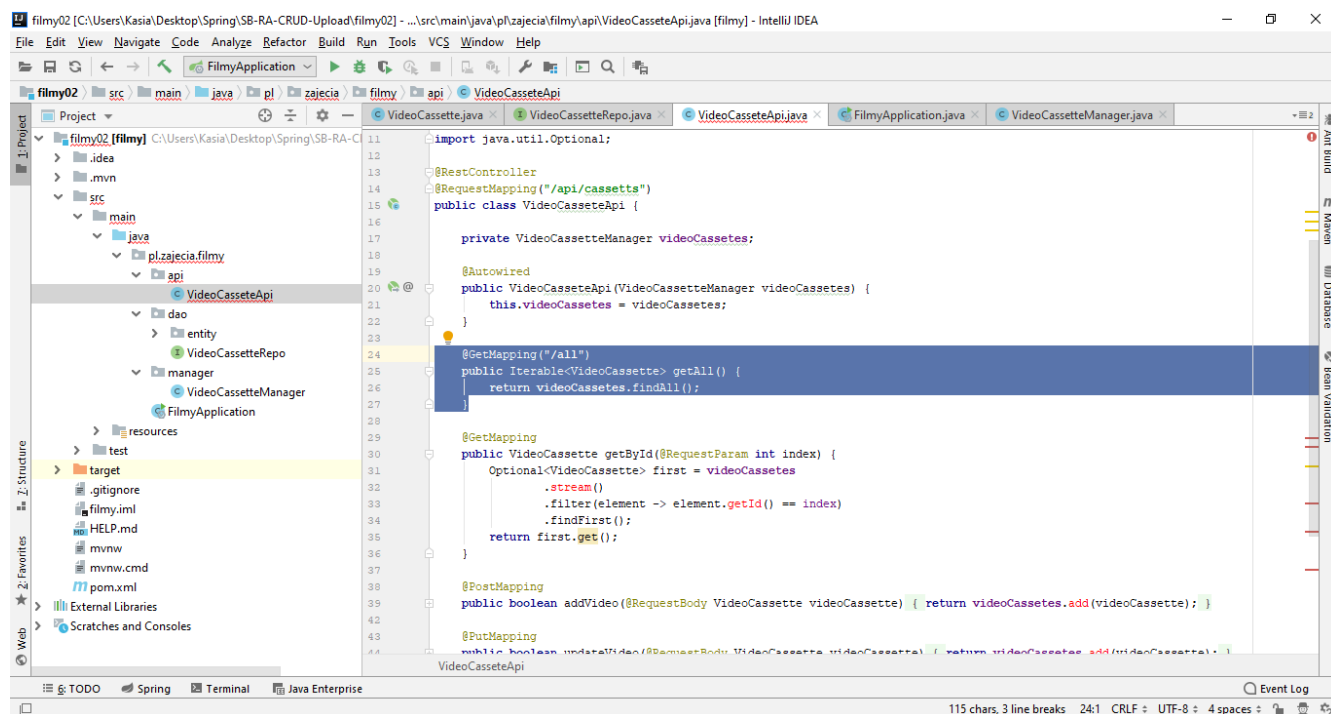
Posługując się instancją tej klasy odwołamy się do wszystkich elementów z bazy danych poprzez metody serwisu.

Obiekt videoCassettes reprezentuje teraz serwis a nie listę.

Metoda getAll powinna teraz korzystać z metody serwisu findAll i zwracać typ Iterable a nie listę.

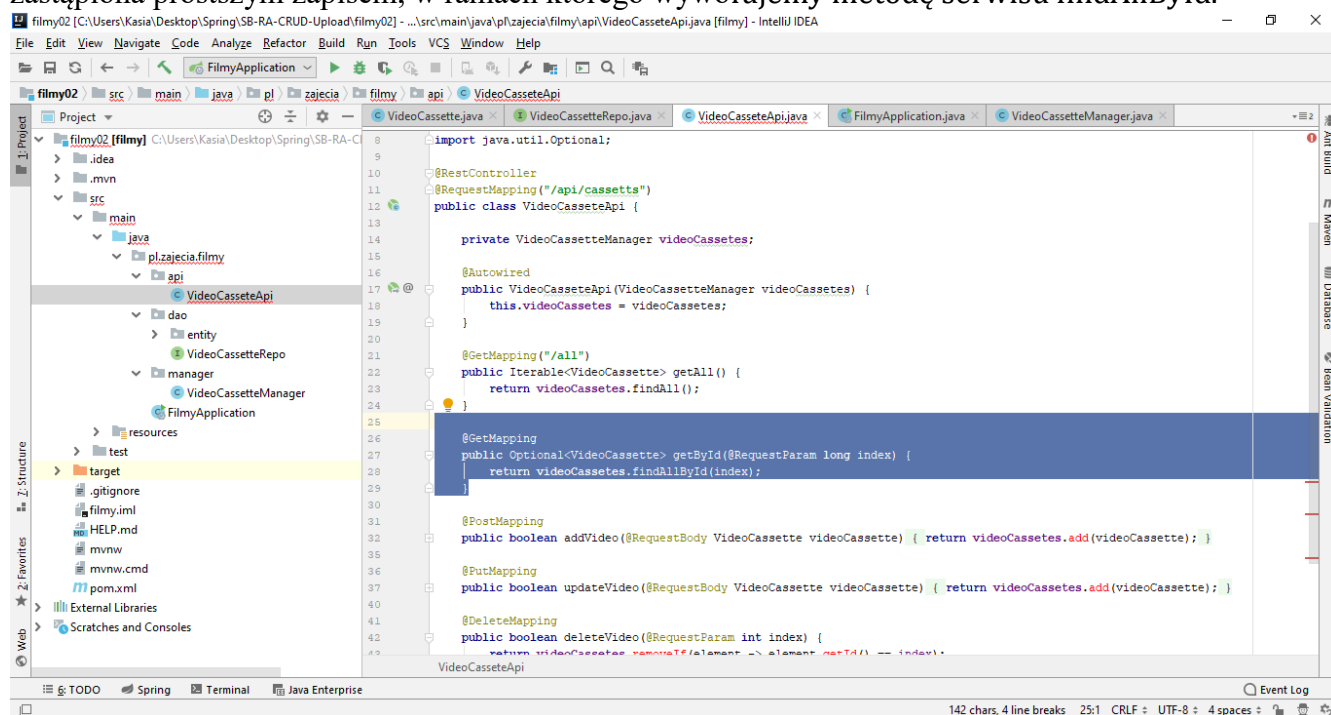


```
10 import java.util.List;
11 import java.util.Optional;
12
13 @RestController
14 @RequestMapping("/api/cassettes")
15 public class VideoCassetteApi {
16
17     private VideoCassetteManager videoCassettes;
18
19     @Autowired
20     public VideoCassetteApi(VideoCassetteManager videoCassettes) {
21         this.videoCassettes = videoCassettes;
22     }
23
24     @GetMapping("/all")
25     public List<VideoCassette> getAll() { return videoCassettes; }
26
27     @GetMapping
28     public VideoCassette getById(@RequestParam int index) {
29         Optional<VideoCassette> first = videoCassettes
30             .stream()
31             .filter(element -> element.getId() == index)
32             .findFirst();
33         return first.get();
34     }
35
36     @PostMapping
37     public boolean addVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }
38
39     @PutMapping
40     public boolean updateVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }
41 }
```



```
11 import java.util.Optional;
12
13 @RestController
14 @RequestMapping("/api/cassettes")
15 public class VideoCassetteApi {
16
17     private VideoCassetteManager videoCassettes;
18
19     @Autowired
20     public VideoCassetteApi(VideoCassetteManager videoCassettes) {
21         this.videoCassettes = videoCassettes;
22     }
23
24     @GetMapping("/all")
25     public Iterable<VideoCassette> getAll() {
26         return videoCassettes.findAll();
27     }
28
29     @GetMapping
30     public VideoCassette getById(@RequestParam int index) {
31         Optional<VideoCassette> first = videoCassettes
32             .stream()
33             .filter(element -> element.getId() == index)
34             .findFirst();
35         return first.get();
36     }
37
38     @PostMapping
39     public boolean addVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }
40
41     @PutMapping
42     public boolean updateVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }
43 }
```


Kolejna metoda `getById` nie będzie już potrzebowała korzystać z wyrażenia `lambda` – zostanie zastąpiona prostszym zapisem, w ramach którego wywołujemy metodę serwisu `findAllById`.



```
import java.util.Optional;

@RestController
@RequestMapping("/api/cassetts")
public class VideoCassetteApi {

    private VideoCassetteManager videoCassettes;

    @Autowired
    public VideoCassetteApi(VideoCassetteManager videoCassettes) {
        this.videoCassettes = videoCassettes;
    }

    @GetMapping("/")
    public Iterable<VideoCassette> getAll() {
        return videoCassettes.findAll();
    }

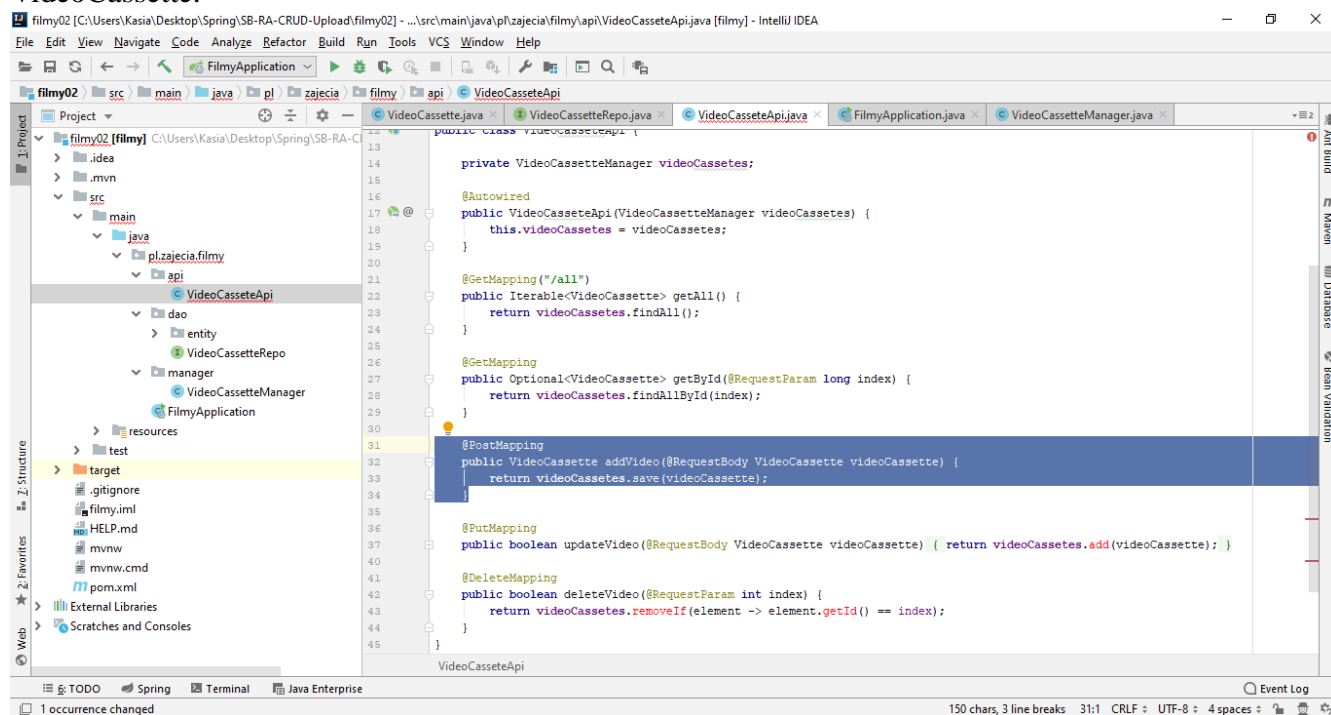
    @GetMapping
    public Optional<VideoCassette> getById(@RequestParam long index) {
        return videoCassettes.findAllById(index);
    }

    @PostMapping
    public boolean addVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }

    @PutMapping
    public boolean updateVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }

    @DeleteMapping
    public boolean deleteVideo(@RequestParam int index) {
        return videoCassettes.removeIf(element -> element.getId() == index);
    }
}
```

Aby zmodyfikować metodę dodającą obiekt/kasetę użyjemy metody z serwisu `save`, która zwraca dodany obiekt kasyety wideo, więc typ zwracany przez metodę `add` zamieniamy z `boolean` na `VideoCassette`.



```
public class VideoCassetteApi {

    private VideoCassetteManager videoCassettes;

    @Autowired
    public VideoCassetteApi(VideoCassetteManager videoCassettes) {
        this.videoCassettes = videoCassettes;
    }

    @GetMapping("/")
    public Iterable<VideoCassette> getAll() {
        return videoCassettes.findAll();
    }

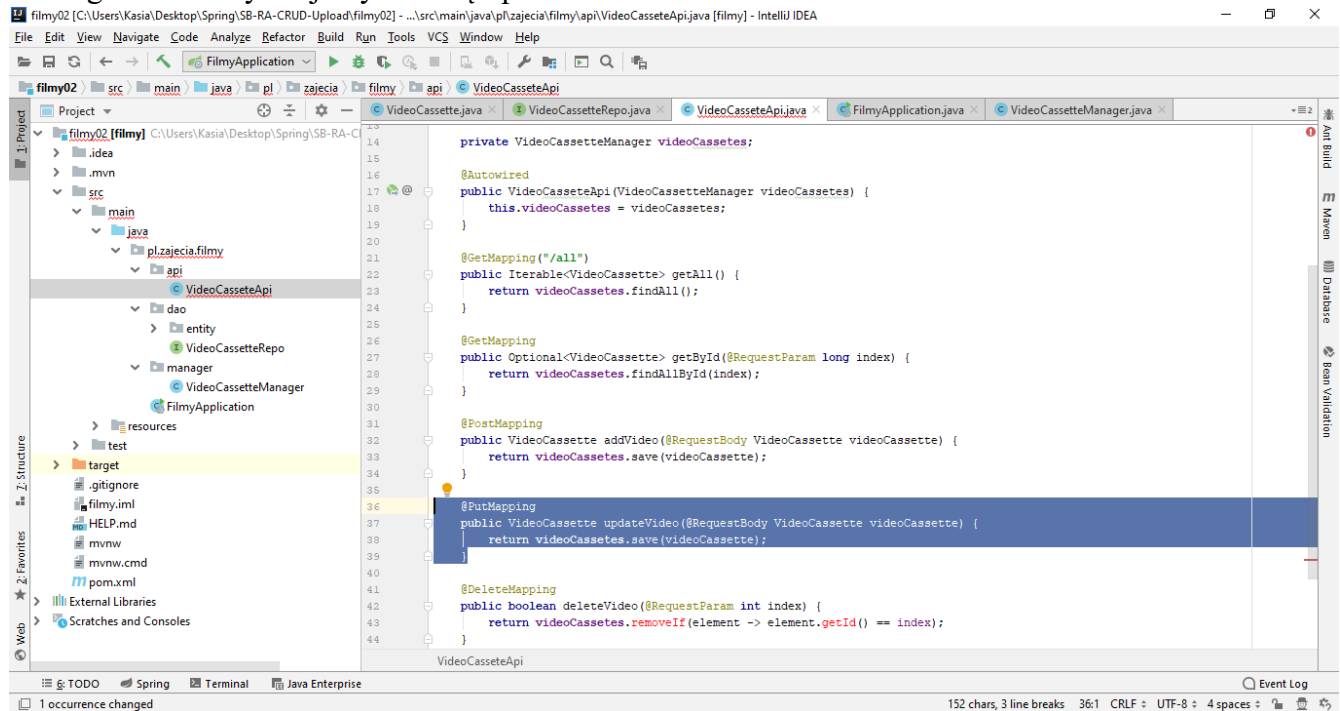
    @GetMapping
    public Optional<VideoCassette> getById(@RequestParam long index) {
        return videoCassettes.findAllById(index);
    }

    @PostMapping
    public VideoCassette addVideo(@RequestBody VideoCassette videoCassette) {
        return videoCassettes.save(videoCassette);
    }

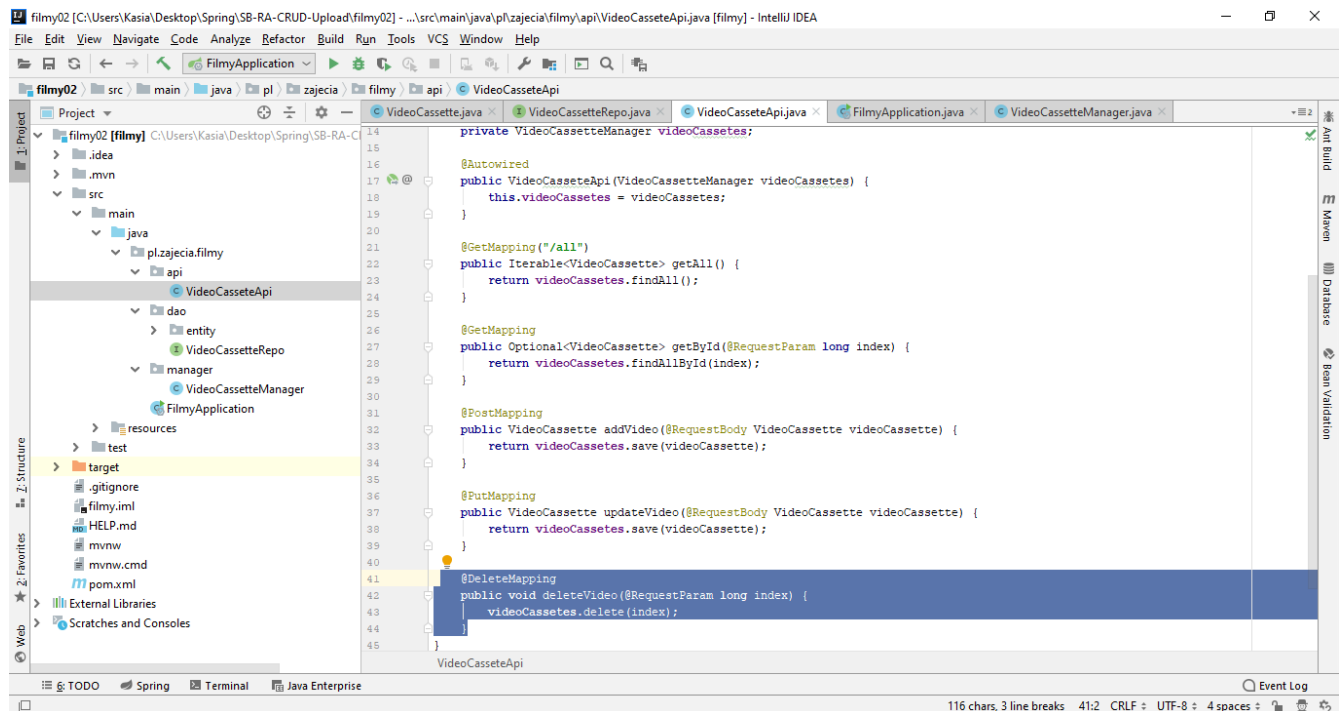
    @PutMapping
    public boolean updateVideo(@RequestBody VideoCassette videoCassette) { return videoCassettes.add(videoCassette); }

    @DeleteMapping
    public boolean deleteVideo(@RequestParam int index) {
        return videoCassettes.removeIf(element -> element.getId() == index);
    }
}
```

Analogicznie modyfikujemy metodę update.

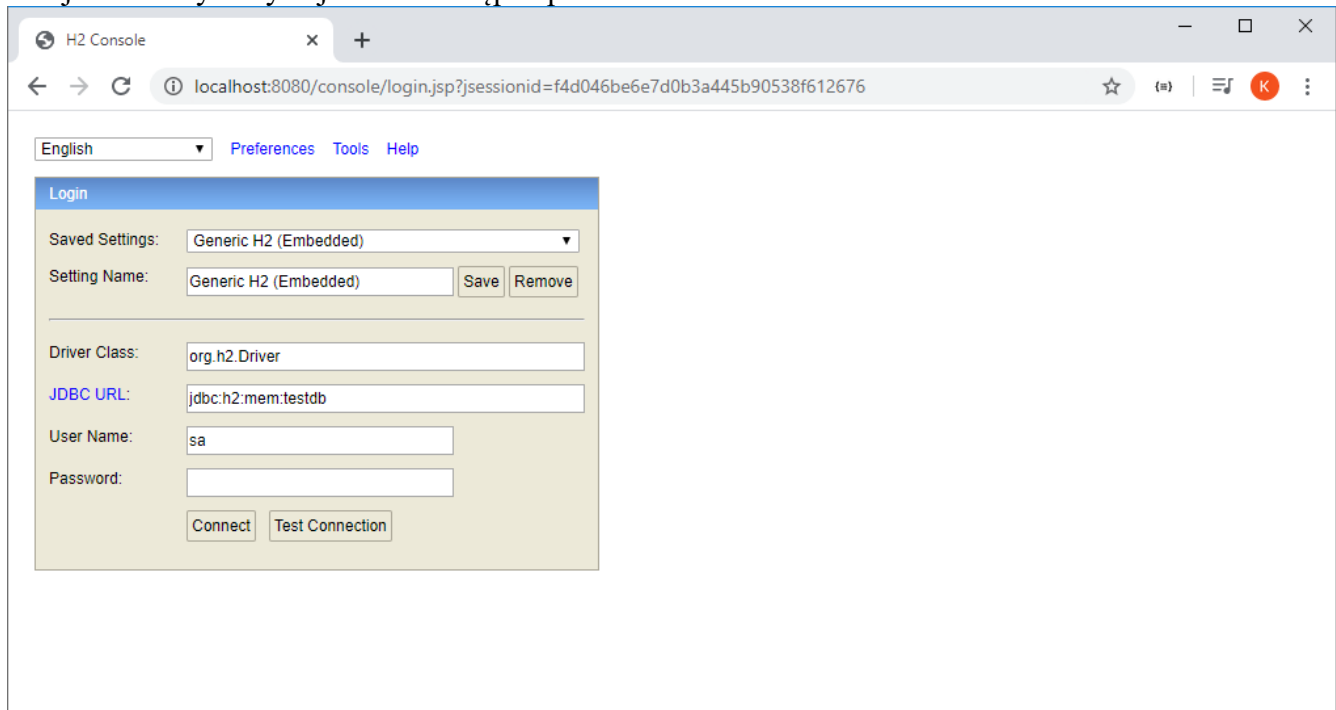


Przy modyfikacji ostatniej metody delete, korzystamy z metody z serwisu delete, która zwraca typ void.



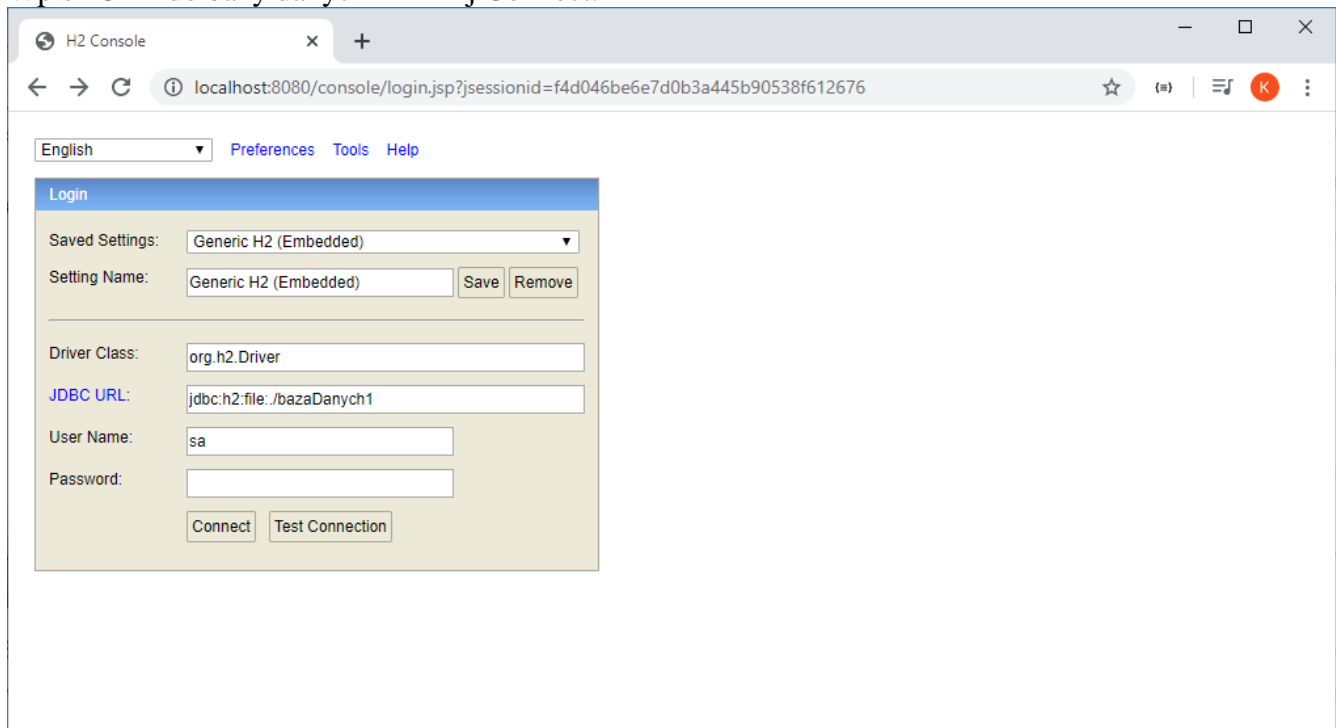
Implementacja aplikacji używającej bazy danych w miejsce listy została zakończona, uruchom i przetestuj jej działanie w Postmanie.

Przejdź do bazy danych jest ona dostępna przez: /console.



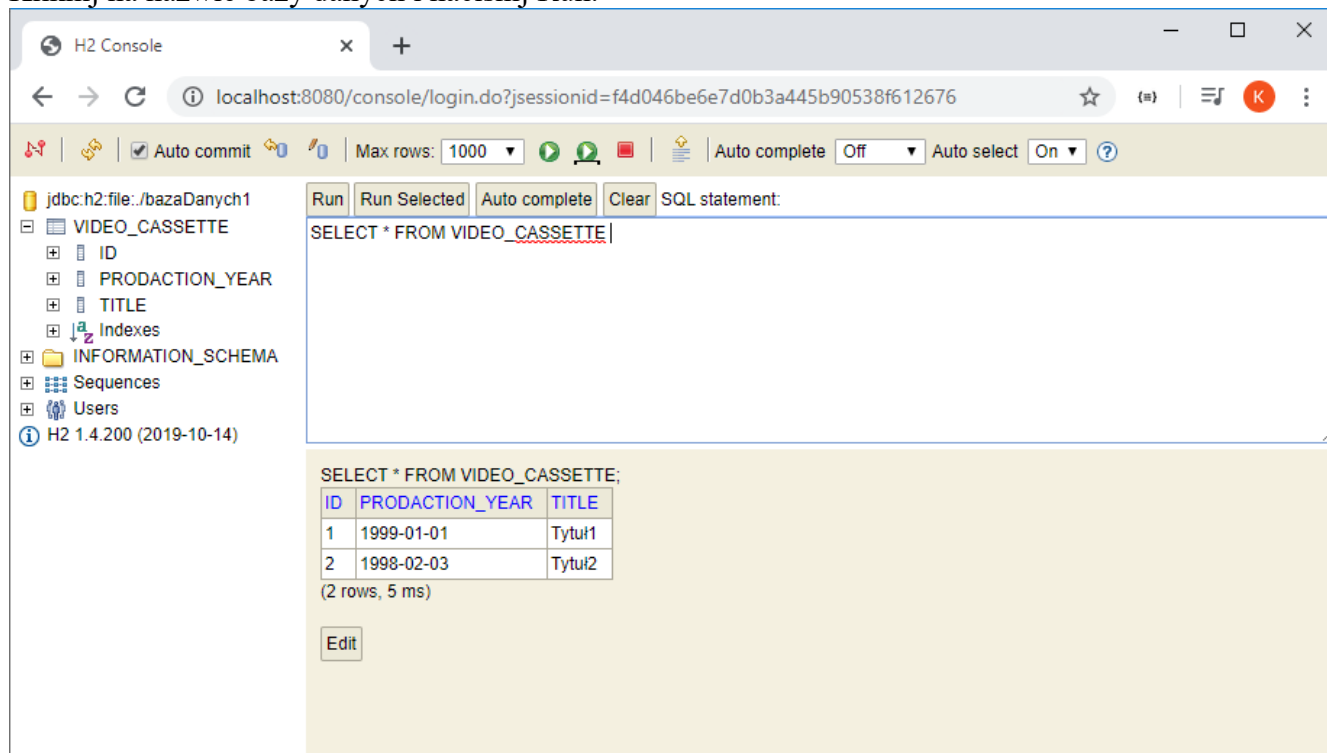
The screenshot shows the H2 Console web interface in a browser window. The address bar displays the URL `localhost:8080/console/login.jsp?jsessionid=f4d046be6e7d0b3a445b90538f612676`. The page has a language dropdown set to "English" and navigation links for "Preferences", "Tools", and "Help". The "Login" form is the central element, featuring a "Saved Settings" dropdown menu currently set to "Generic H2 (Embedded)". Below this, the "Setting Name" is also "Generic H2 (Embedded)", with "Save" and "Remove" buttons. The "Driver Class" is set to `org.h2.Driver`. The "JDBC URL" is `jdbc:h2:mem:testdb`. The "User Name" is `sa`, and the "Password" field is empty. At the bottom of the form are "Connect" and "Test Connection" buttons.

Wpisz URL do bazy danych i kliknij Connect.



This screenshot shows the same H2 Console login page, but with the "JDBC URL" field updated to `jdbc:h2:file:./bazaDanych1`. All other fields remain the same: "Saved Settings" is "Generic H2 (Embedded)", "Setting Name" is "Generic H2 (Embedded)", "Driver Class" is `org.h2.Driver`, "User Name" is `sa`, and the "Password" field is empty. The "Connect" and "Test Connection" buttons are still present at the bottom of the form.

Kliknij na nazwie bazy danych i naciśnij Run.

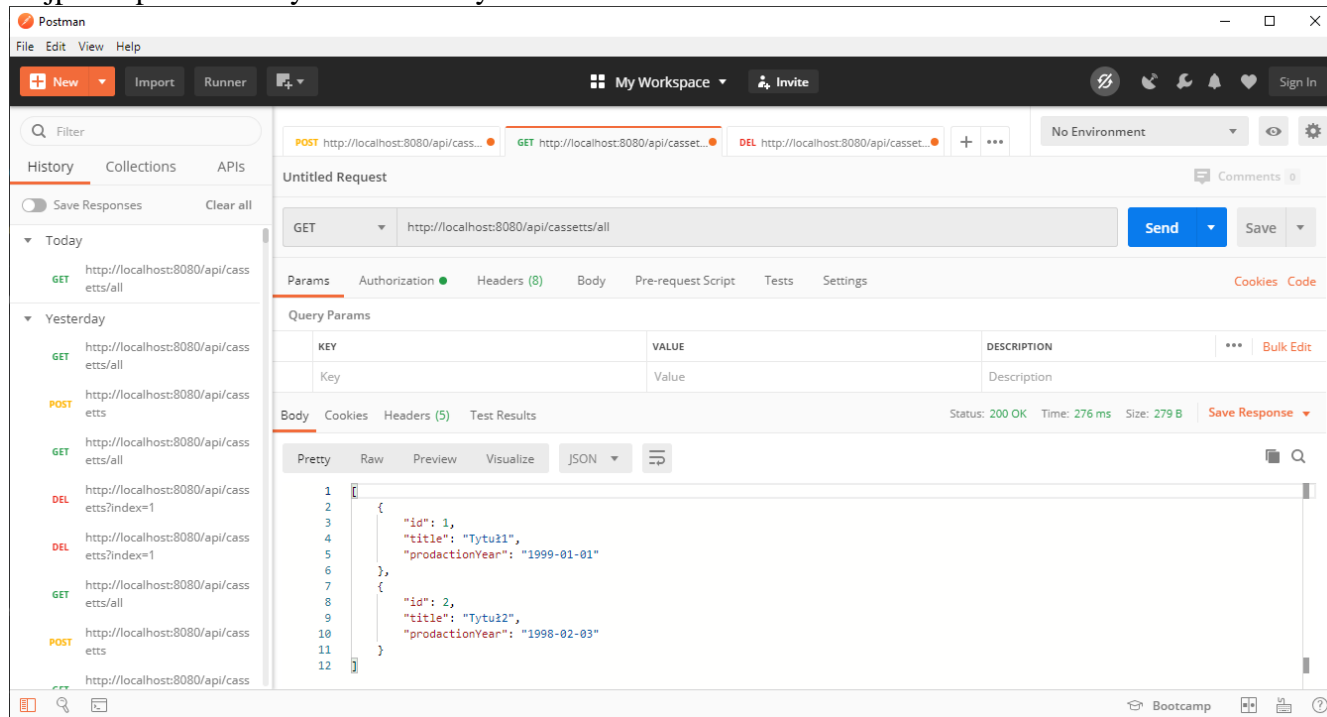


The screenshot shows the H2 Console interface. On the left, a tree view displays the database structure: jdbc:h2:file:./bazaDanych1, VIDEO_CASSETTE, ID, PRODUCTION_YEAR, TITLE, Indexes, INFORMATION_SCHEMA, Sequences, Users, and H2 1.4.200 (2019-10-14). The main area shows the SQL statement: `SELECT * FROM VIDEO_CASSETTE;`. Below the statement, the results are displayed in a table with 2 rows and 3 columns: ID, PRODUCTION_YEAR, and TITLE. The first row contains 1, 1999-01-01, and Tytuł1. The second row contains 2, 1998-02-03, and Tytuł2. The status bar indicates (2 rows, 5 ms).

ID	PRODUCTION_YEAR	TITLE
1	1999-01-01	Tytuł1
2	1998-02-03	Tytuł2

Po upewnieniu się, że w bazie danych są dwa elementy, wykonaj test metod w kliencie http – Postmanie.

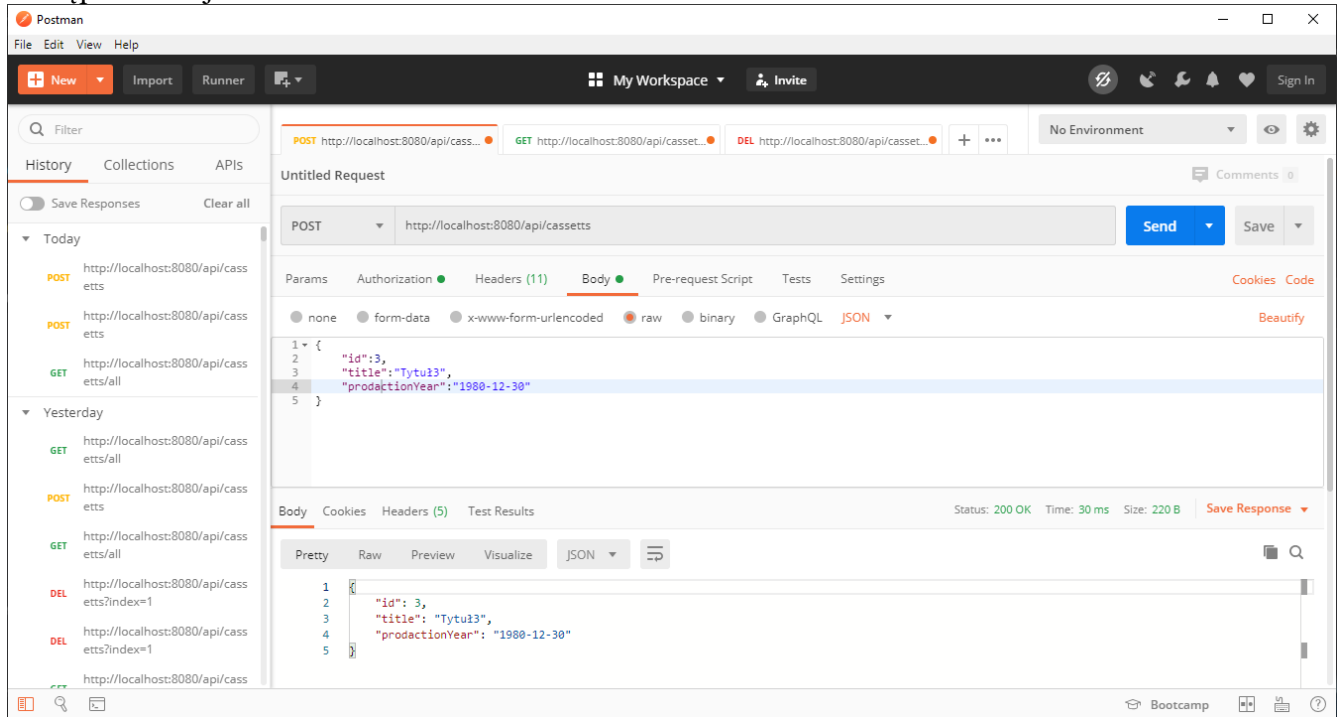
Najpierw pobierz wszystkie elementy.



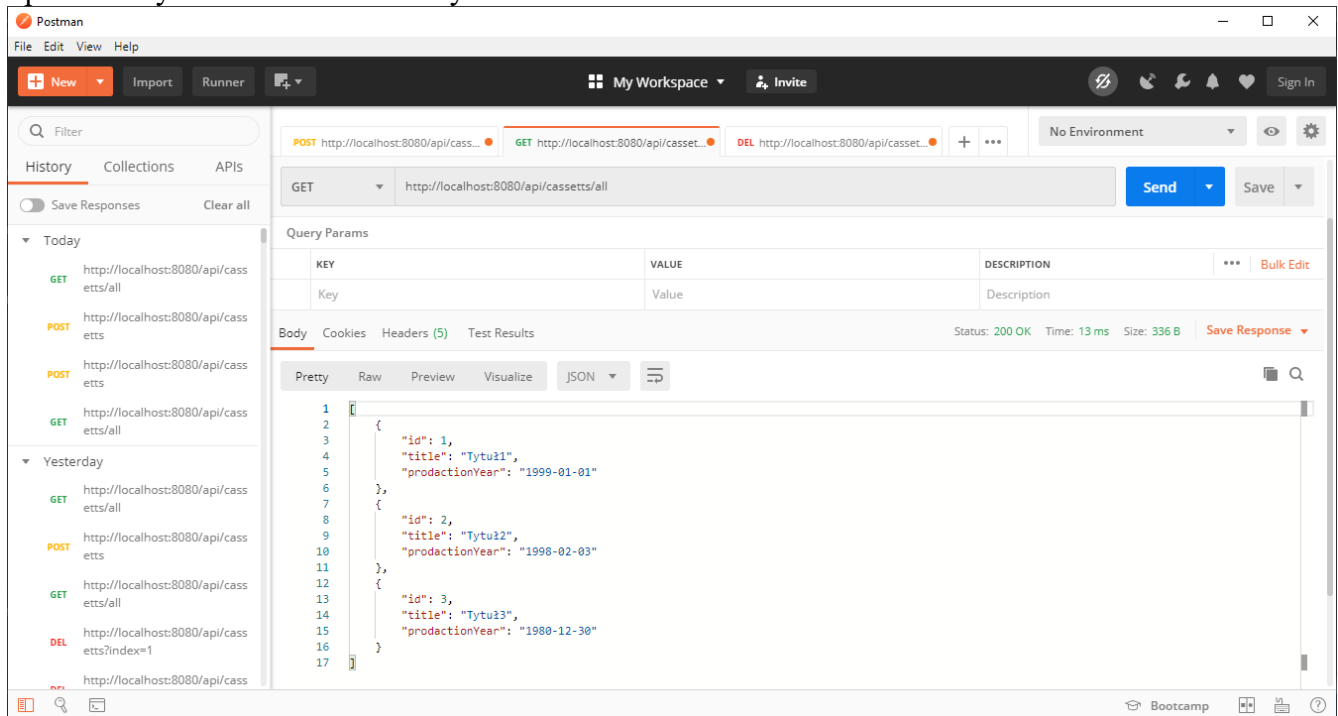
The screenshot shows the Postman interface. The left sidebar displays a list of requests, including a GET request to `http://localhost:8080/api/cassets/all`. The main area shows the details of the selected request, which is a GET request to `http://localhost:8080/api/cassets/all`. The response is displayed in the 'Body' tab, showing a JSON array of two objects. The status bar indicates Status: 200 OK, Time: 276 ms, Size: 279 B.

```
[{"id": 1, "title": "Tytuł1", "productionYear": "1999-01-01"}, {"id": 2, "title": "Tytuł2", "productionYear": "1998-02-03"}]
```

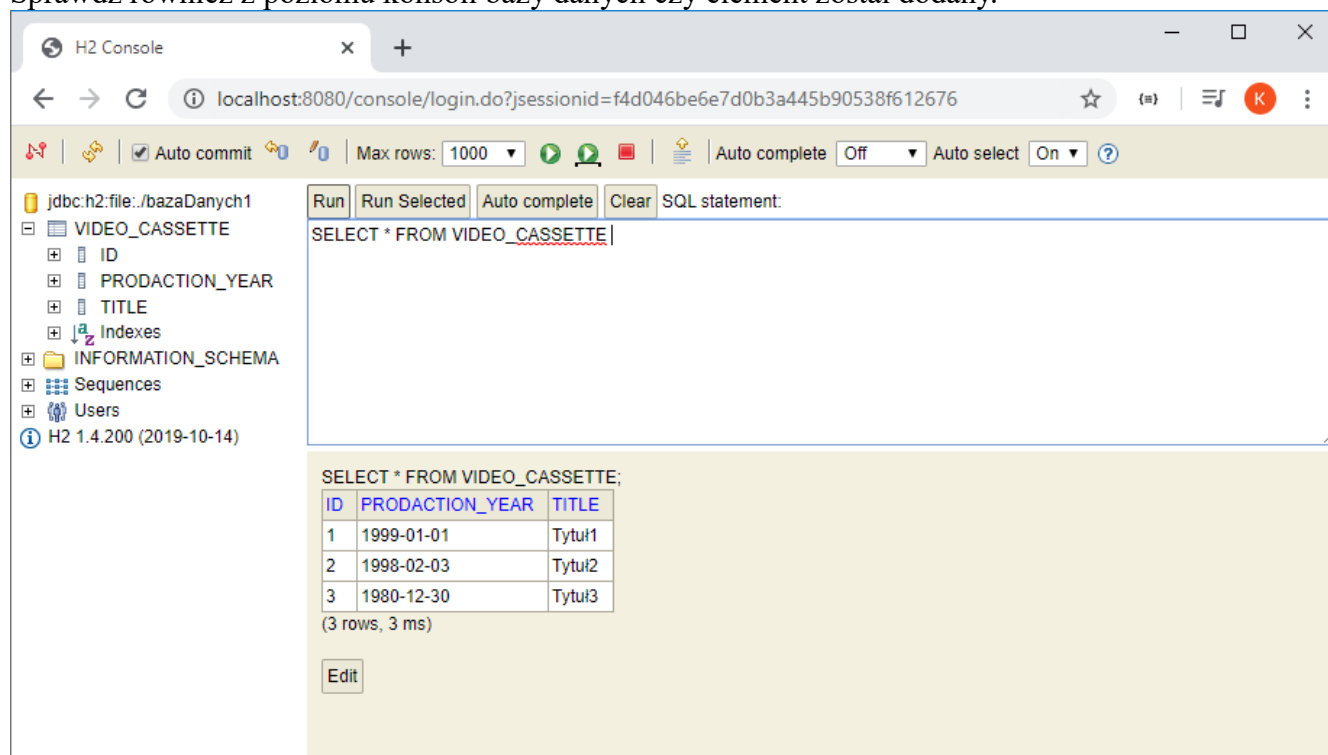
Następnie dodaj element.



Sprawdź czy element został dodany.



Sprawdź również z poziomu konsoli bazy danych czy element został dodany.

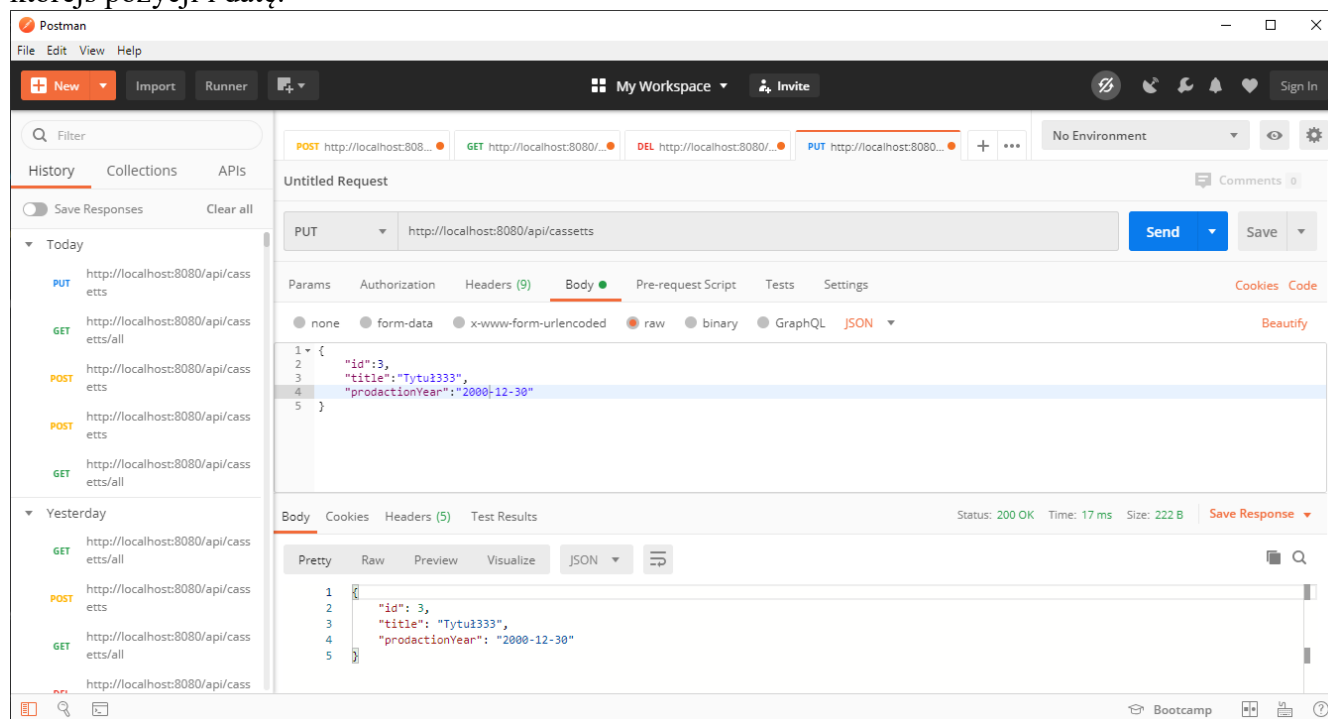


The screenshot shows the H2 Console web interface. On the left, a tree view displays the database structure, including a table named VIDEO_CASSETTE. The main area shows a SQL statement: `SELECT * FROM VIDEO_CASSETTE;`. Below the statement, the results are displayed in a table format, showing 3 rows and 3 columns (ID, PRODUCTION_YEAR, TITLE).

ID	PRODUCTION_YEAR	TITLE
1	1999-01-01	Tytuł1
2	1998-02-03	Tytuł2
3	1980-12-30	Tytuł3

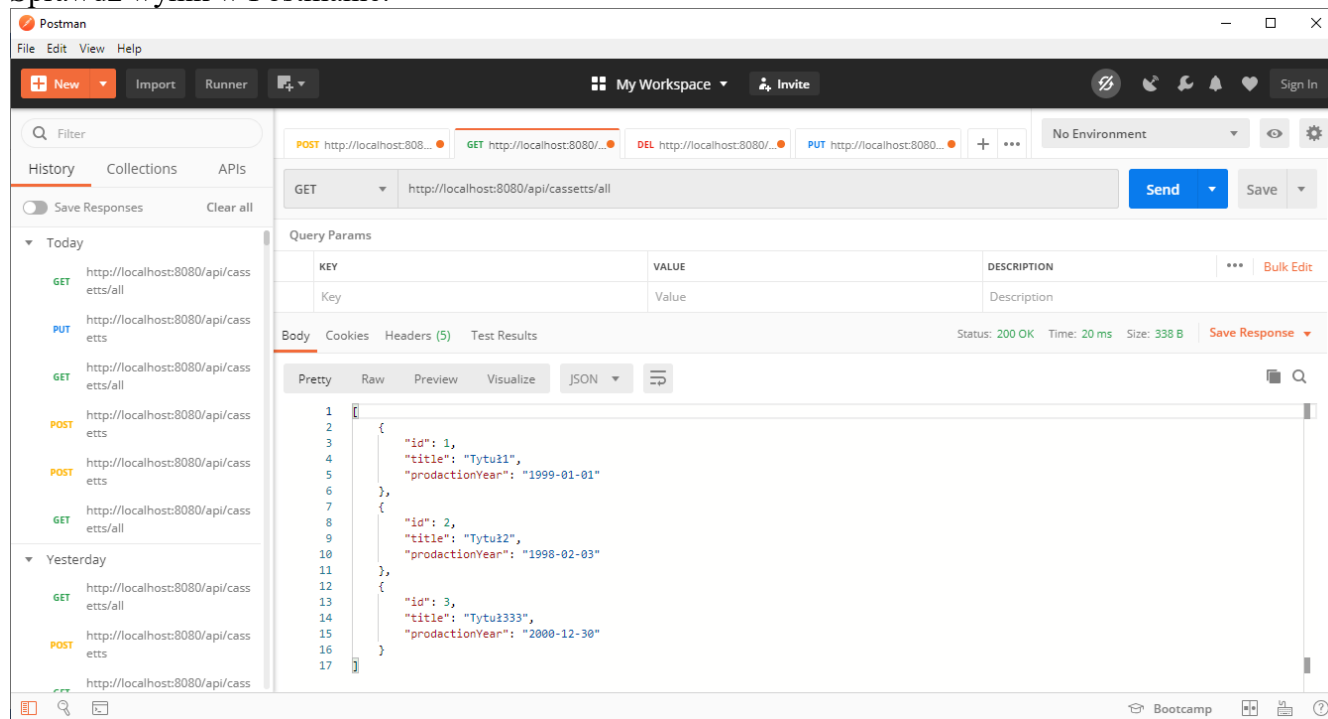
(3 rows, 3 ms)

Przetestuj działanie metody PUT, która służy do modyfikacji pól danego elementu np. zmień tytuł którejś pozycji i datę.

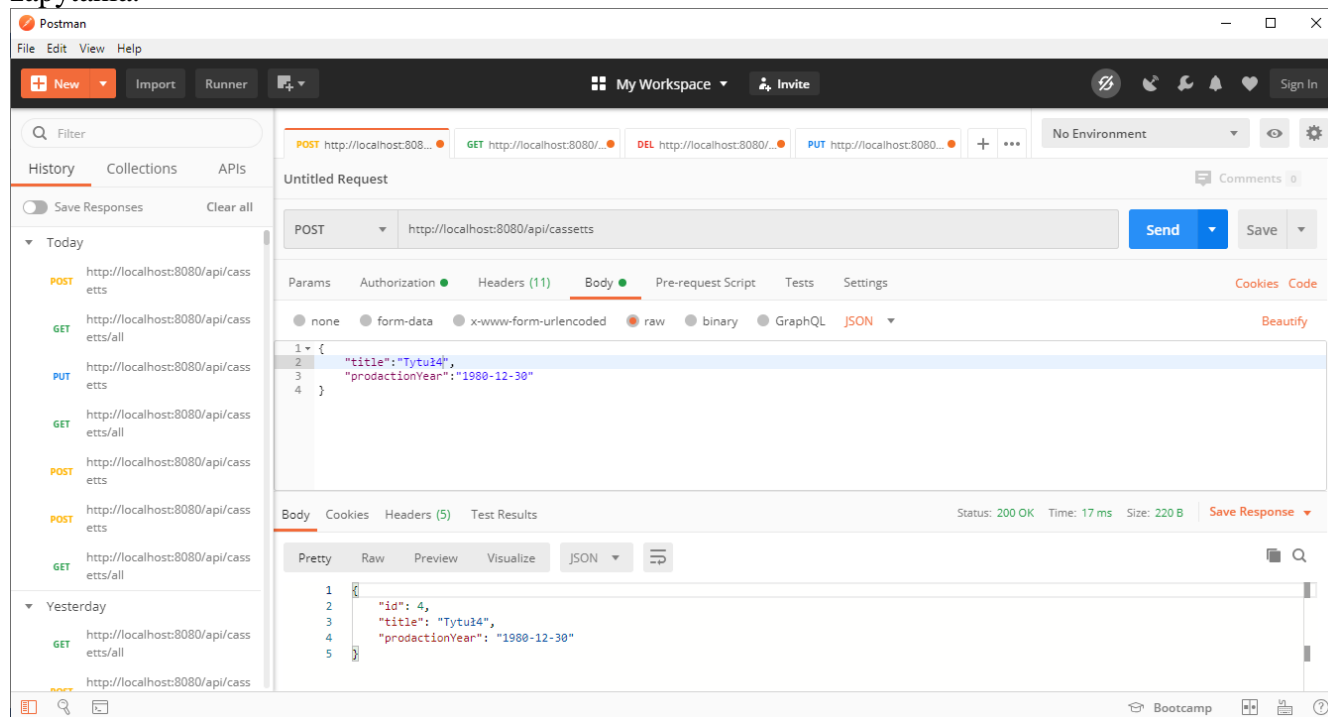


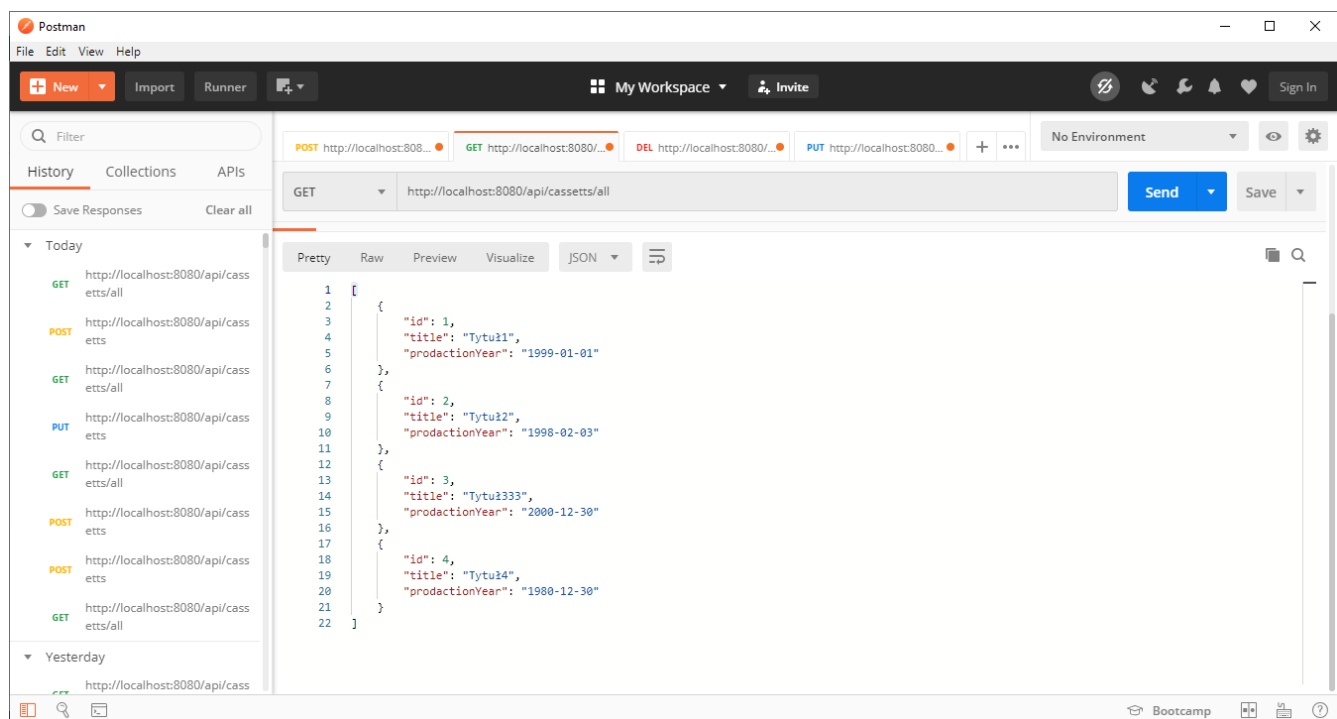
The screenshot shows the Postman API client interface. A PUT request is configured to `http://localhost:8080/api/cassetts`. The request body is a JSON object: `{ "id": 3, "title": "Tytuł333", "productionYear": "2000-12-30" }`. The response status is 200 OK, with a time of 17 ms and a size of 222 B. The response body is displayed in the Pretty view, showing the same JSON object.

Sprawdź wynik w Postmanie.

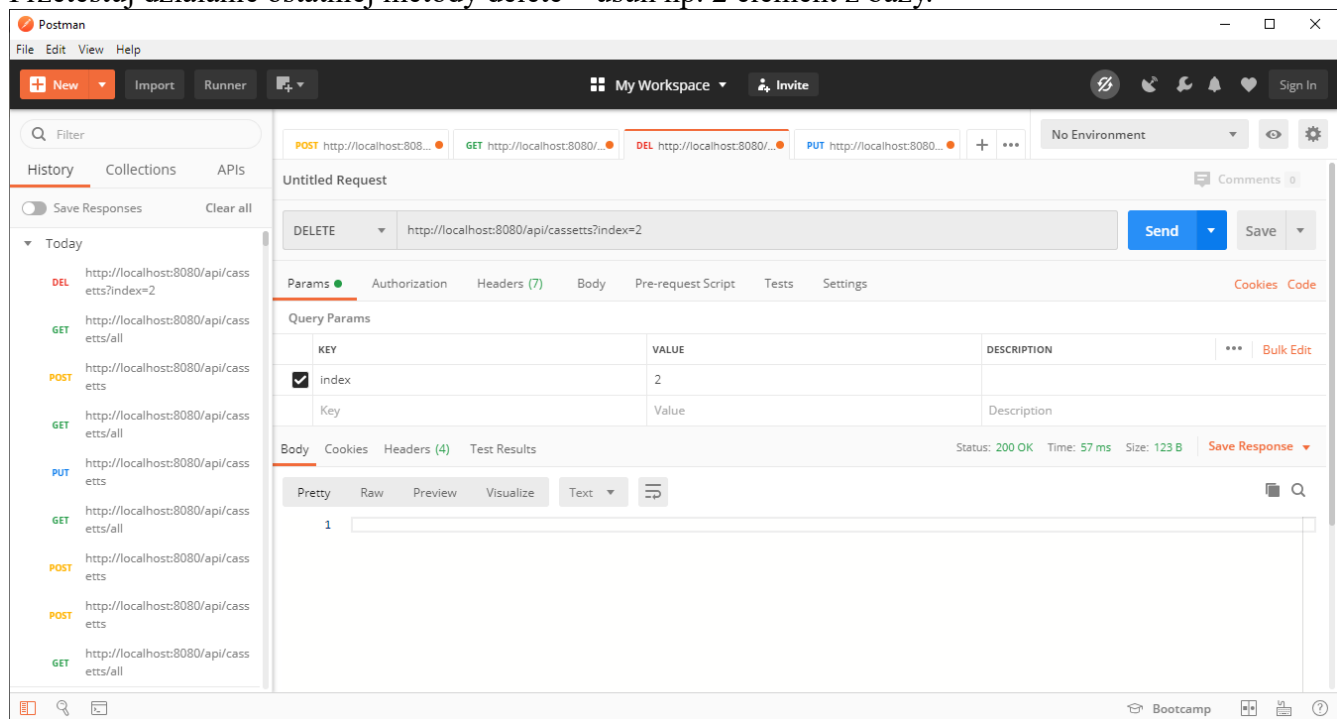


W metodzie POST nie powinno się dodawać id bo on jest automatycznie generowany. Wykonaj zapytania.





Przetestuj działanie ostatniej metody delete – usuń np. 2 element z bazy.



Postman

File Edit View Help

My Workspace Invite

Filter

History Collections APIs

Save Responses Clear all

Today

- GET http://localhost:8080/api/cassets/all
- DEL http://localhost:8080/api/cassets?index=2
- GET http://localhost:8080/api/cassets/all
- POST http://localhost:8080/api/cassets
- GET http://localhost:8080/api/cassets/all
- PUT http://localhost:8080/api/cassets
- GET http://localhost:8080/api/cassets/all
- POST http://localhost:8080/api/cassets
- POST http://localhost:8080/api/cassets

POST http://localhost:8080/... GET http://localhost:8080/... DEL http://localhost:8080/... PUT http://localhost:8080/...

GET http://localhost:8080/api/cassets/all

Send Save

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 1,
4     "title": "Tytuł1",
5     "productionYear": "1999-01-01"
6   },
7   {
8     "id": 3,
9     "title": "Tytuł333",
10    "productionYear": "2000-12-30"
11  },
12  {
13    "id": 4,
14    "title": "Tytuł4",
15    "productionYear": "1980-12-30"
16  }
17 }
```

H2 Console

localhost:8080/console/login.do?sessionId=f4d046be6e7d0b3a445b90538f612676

Auto commit Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:file:./bazaDanych1

- VIDEO_CASSETTE
 - ID
 - PRODUCTION_YEAR
 - TITLE
 - Indexes
- INFORMATION_SCHEMA
- Sequences
- Users
- H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM VIDEO_CASSETTE

SELECT * FROM VIDEO_CASSETTE;

ID	PRODUCTION_YEAR	TITLE
1	1999-01-01	Tytuł1
3	2000-12-30	Tytuł333
4	1980-12-30	Tytuł4

(3 rows, 2 ms)

Edit