Damian Koper, **241292**

Łukasz Handschuh, **241402**

Inżynieria oprogramowania - Etap 4

Dział ewidencji ludności

Identyfikacja klas reprezentujących logikę biznesową projektowanego oprogramowania, definicja atrybutów i operacji klas oraz związków między klasami - na podstawie analizy scenariuszy przypadków użycia. Opracowanie diagramów klas i pakietów. Zastosowanie projektowych wzorców strukturalnych i wytwórczych.

1 Przypadki użycia - zakres analizy

W modelowaniu klas zastosowano wzorzec Model-View-Controller z separacją serwisów oraz wzorzec repozytorium. Analiza przeprowadzona została dla następujących przypadków użycia:

- Wyświetlanie wniosków,
- Zmiana kryterium wyświetlania wniosków,
- Edycja danych wniosku,
- Zmiana statusu wniosku,

2 Analiza wspólności

2.1 Encje

Analiza wykryła jedną abstrakcyjną klasę encji bazowej `RegistrationBase` - Dane meldunkowe. Zawiera ona dwa obiekty:

- `RegistryPersonalData` - dane osobowe, liczebność 1:1
- `RegistryAddressData` - dane adresowe, liczebność 1:1

2.2 Główny przepływ sterowania

Realizacja wszystkich przypadków użycia oparta jest o interfejs konsoli. Wykryto następujące klasy obsługujące przepływ sterowania w aplikacji:

- `ConsoleEngine` - klasa przechowuje instancje wszystkich kontrolerów i jest z nimi powiązana relacją kompozycji,
- `RegistryApplicationController`

Wszystkie klasy kontrolerów realizują interfejs `IController`.

2.3 Widoki

Wykryto następujące klasy widoków używane do wyświetlania i odpytywania użytkownika o dane:

- `RegistryApplicationIndexView` - Wyświetlanie i filtrowanie wszystkich wniosków,
- `RegistryApplicationShowView` - Wyświetlanie pojedynczego wniosku,
- `RegistryApplicationUpdateView` - Edytowanie pojedyncego wniosku.

2.3.1 Data transfer objects

- `TableDTO` - wyświetlanie tabel,
- `RegisterApplicationDTO` - dane wniosku,
- `FilterDataDTO` - dane filtracji wniosków.

2.4 PESEL

Komunikację z systemem PESEL odpowiedzialenego za weryfikacje danych osobowych będzie realizować będzie klasa `PecelFacade` realizująca interfejs `IPeselFacade`.

3 Analiza zmienności

3.1 Encje

Wykryto dwa podzbiory danych meldunkowych - wniosek i meldunek faktyczny. Zidentyfikowano następujące klasy pochodne klasy `RegistryApplicationBase`:

- `RegistryApplication` - Wniosek meldunkowy,
- `Registration` - Meldunek.

3.2 Przechowywanie danych

Dla każdej encji wykryto klasę repozytorium, która zapewnia odpowiedni poziom abstrakcji przy pobieraniu i zapisywaniu danych:

- `RegistryApplicationRepository`
- `RegistrationRepository`

Wszystkie klasy repozytoriów realizują interfejs `IRepository` i są powiązane z obiektami, które przechowują, relacją kompozycji.

3.3 Logika biznesowa

Dla każdej encji wykryto klasę serwisu, który realizuje operacje opisane w logice biznesowej przypadków użycia:

- `RegistryApplicationService`
- `RegistrationService`

4 Wzorce projektowe

4.1 Flyweight

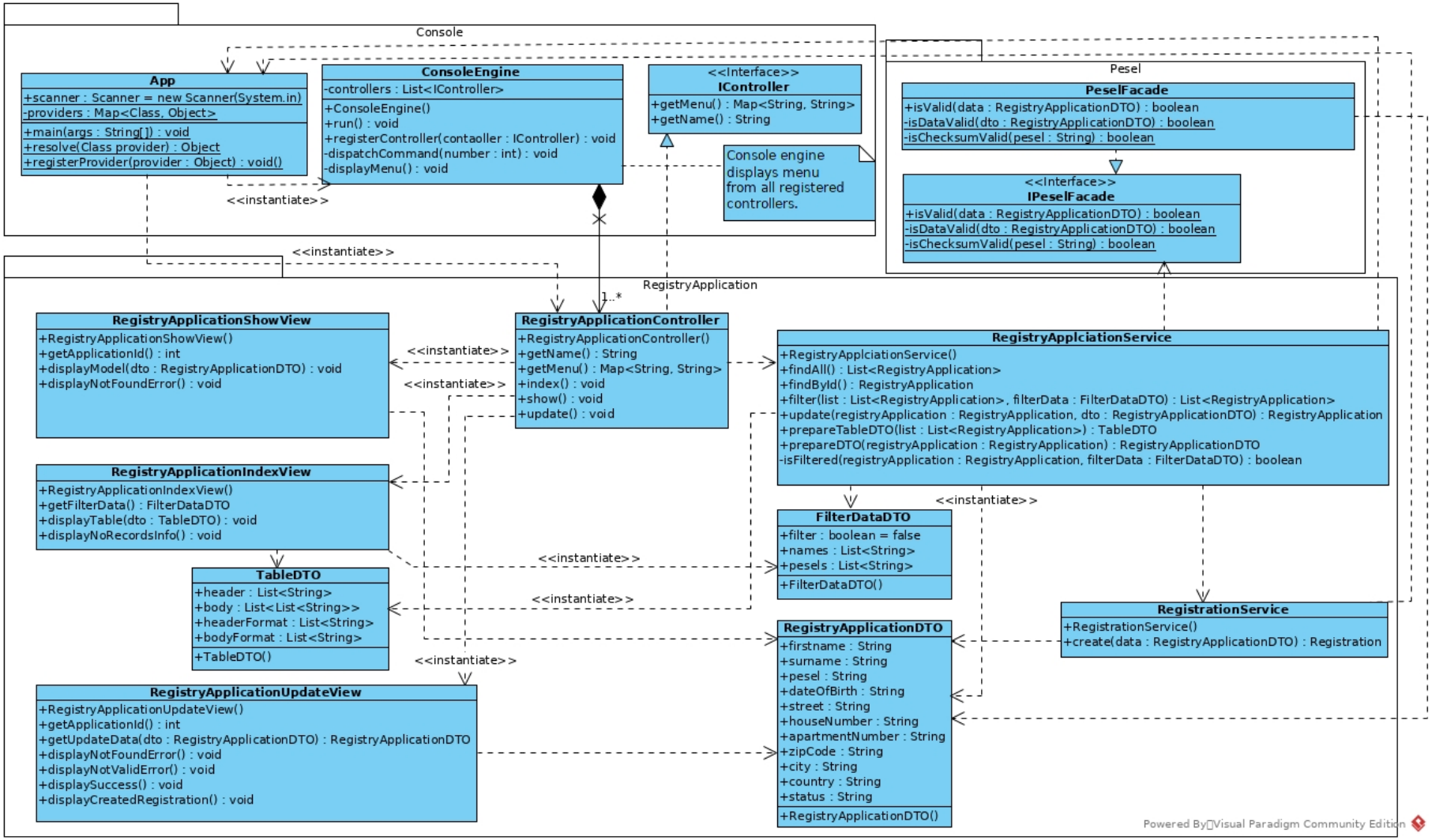
Rolę obiektów Flyweight pełnią klasy `RegistryAddressData` oraz `RegistryPersonalData`. Abstrakcyjnym klientem tych klas jest klasa `RegistrationBase`, z której dziedziczą klasy `RegistryApplication` oraz `Registration`.

4.2 Singleton

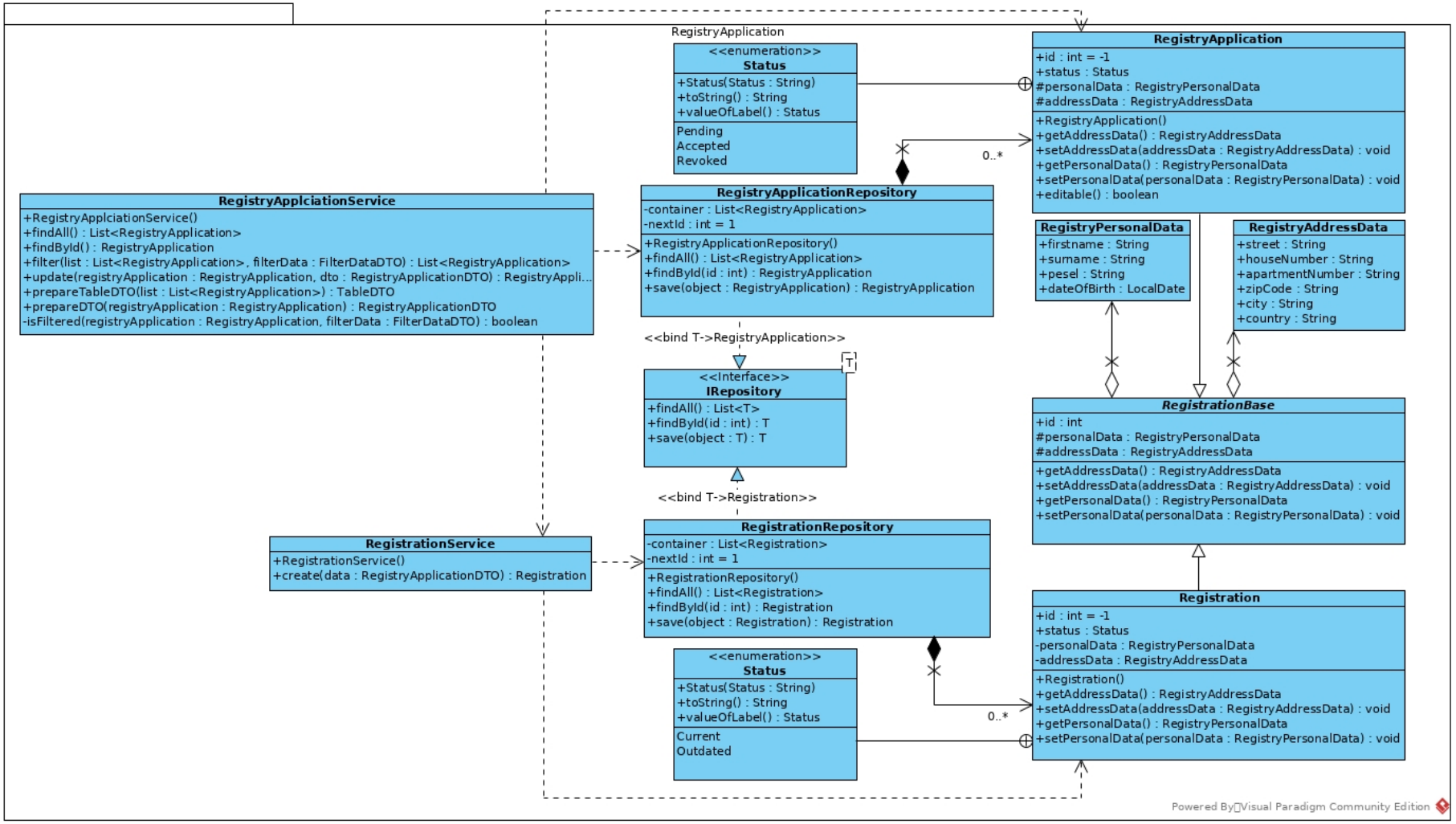
Serwisy są obiektami typu singleton posiadające tylko jedną instancję. Dostęp i zarządzanie nimi jest możliwy przez fasadę, którą implementuje klasa `App`. Zastosowanie tego wzorca ułatwi późniejsze testowanie i mockowanie implementacji serwisów.

4.3 Fasada

Wzorzec fasada użyty został przy zdefiniowaniu klasy `PeselFasade`, która udostępnia metody umożliwiające komunikację z zewnętrznym systemem. Późniejsza możliwość podmiany implementacji dzięki interfejsowi `IPeselFacade` zapewnia możliwość komunikacji z zewnętrznym systemem w dowolny sposób.



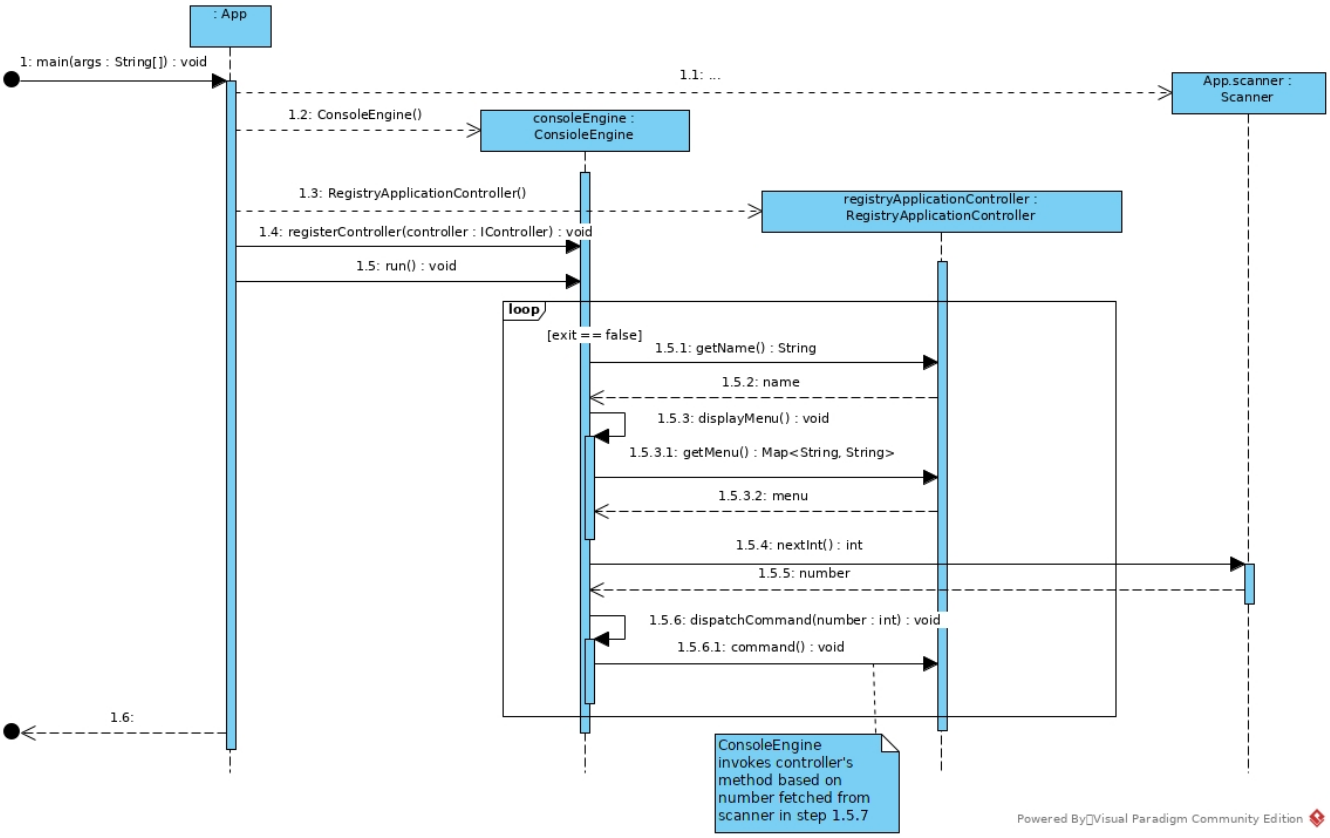
Rysunek 1: Diagram klas - widoki, kontrolery i serwisy.



Rysunek 2: Diagram klas - serwisy, repozytoria i encje.

5 Diagramy sekwencji

5.1 Główna pętla sterowania

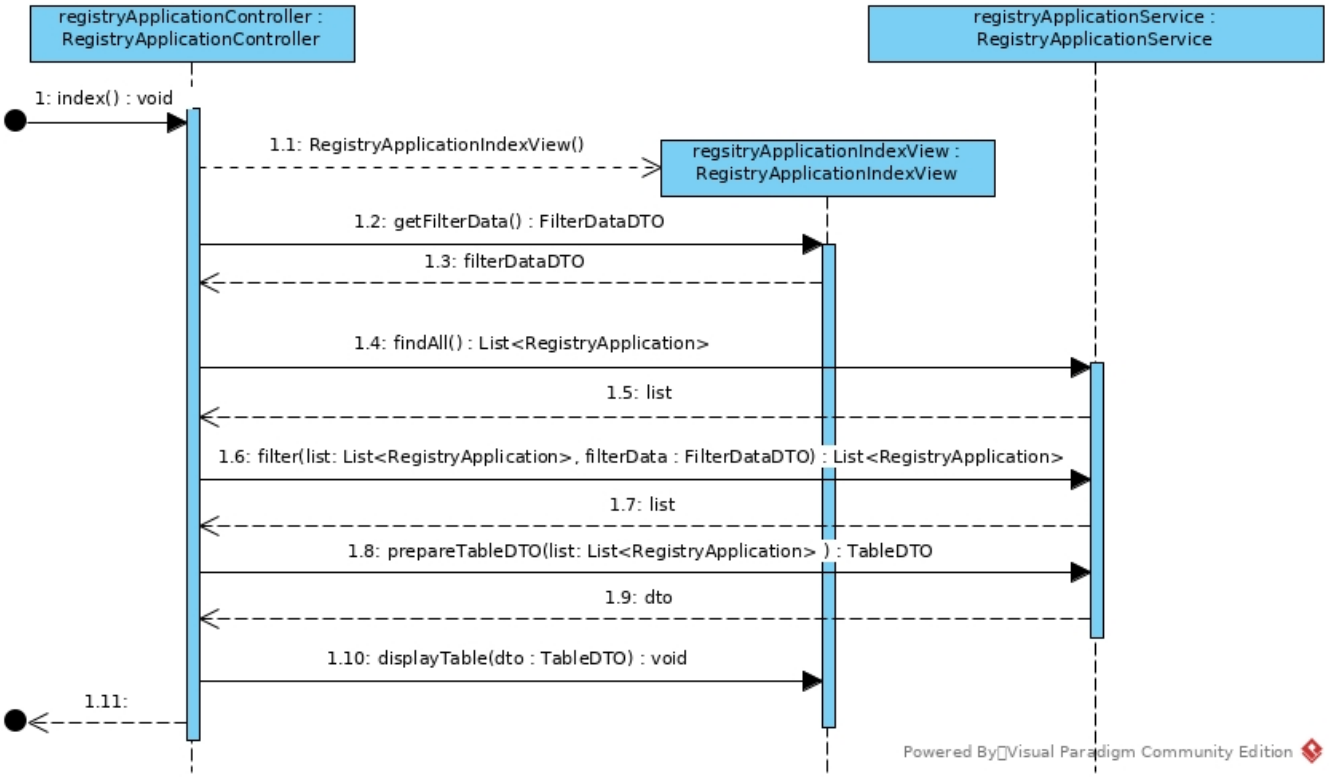


Rysunek 3: Diagram sekwencji - główna pętla przepływu sterowania.

```
1    public static void main(String[] args) {
2        RegistryApplicationRepository registryApplicationRepository = new
     RegistryApplicationRepository();
3
4        /**
5         * Data seed
6         */
7        RegistryApplication registryApplication = new RegistryApplication();
8        registryApplication.getPersonalData().dateOfBirth = LocalDate.of(1990,
     01, 01);
9        registryApplication.getPersonalData().firstname = "Damian";
10       registryApplication.getPersonalData().surname = "Koper";
11       registryApplication.getPersonalData().pesel = "72060319389";
12       registryApplication.getAddressData().apartmentNumber = "20";
```

```
13          registryApplication.getAddressData().houseNumber = "10";
14          registryApplication.getAddressData().street = "Marszalkowska";
15          registryApplication.getAddressData().zipCode = "00-043";
16          registryApplication.getAddressData().country = "Polska";
17          registryApplication.getAddressData().city = "Warszawa";
18          registryApplicationRepository.save(registryApplication);
19
20          App.registerProvider(new RegistryApplicationService());
21          App.registerProvider(registryApplicationRepository);
22          App.registerProvider(new RegistrationService());
23          App.registerProvider(new RegistrationRepository());
24
25          ConsoleEngine engine = new ConsoleEngine();
26          engine.registerController(new RegistryApplicationController());
27          engine.run();
28      }
```

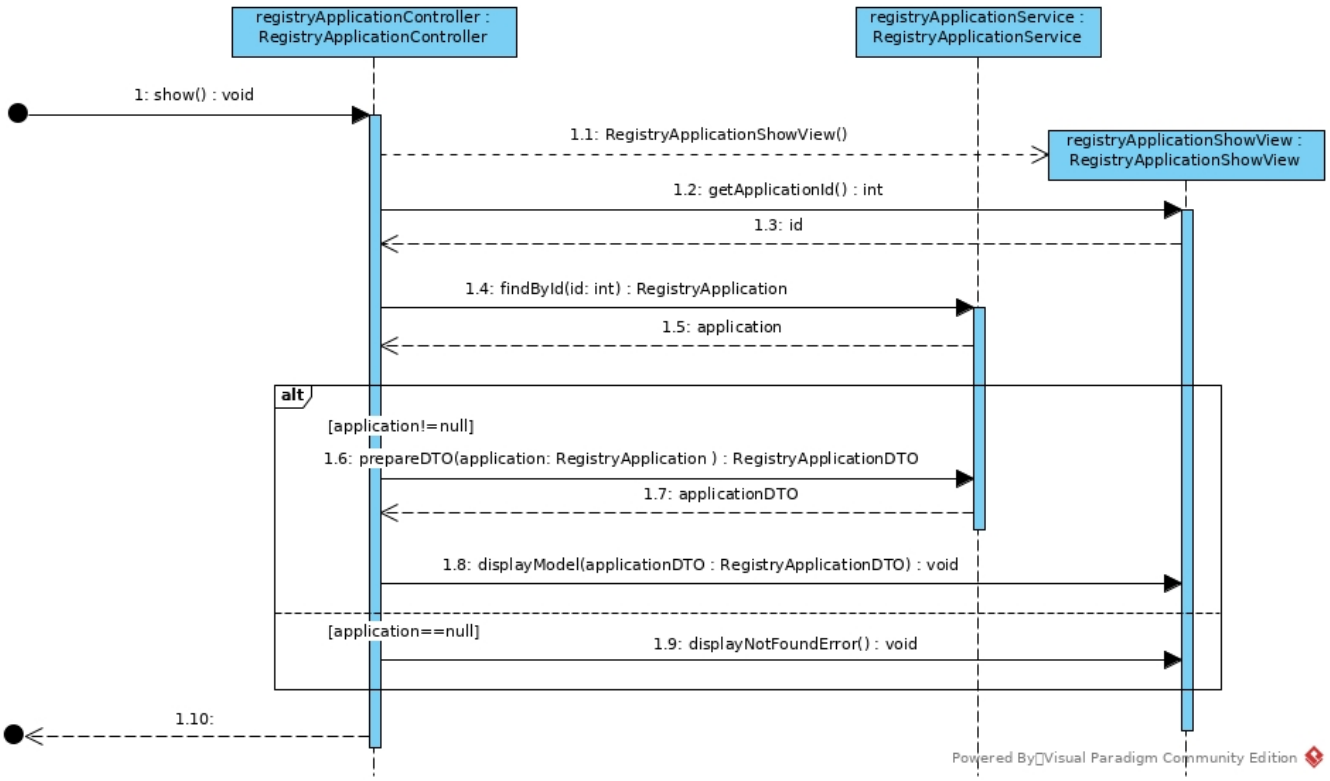
5.2 Wyświetlanie wniosków



Rysunek 4: Diagram sekwencji - wyświetlanie wniosków.

```
śś
  public void index() {
    RegistryApplicationIndexView view = new RegistryApplicationIndexView();
    FilterDataDTO filterDataDTO = view.getFilterData();
    RegistryApplicationService service = (RegistryApplicationService) App.
   resolve(RegistryApplicationService.class);
    List<RegistryApplication> list = service.findAll();
    list = service.filter(list, filterDataDTO);
    TableDTO tableDTO = service.prepareTableDTO(list);
    view.displayTable(tableDTO);
  }
```

5.3 Wyświetlanie pojedynczego wniosku



Rysunek 5: Diagram sekwencji - wyświetlanie pojedynczego wniosku.

```
śś
  public void show() {
    RegistryApplicationShowView view = new RegistryApplicationShowView();
    RegistryApplicationService service = (RegistryApplicationService) App.
   resolve(RegistryApplicationService.class);
```

```
    int id = view.getApplicationId();
    RegistryApplication registryApplication = service.findById(id);

    if (registryApplication == null) {
      view.displayNotFoundError();
    } else {
      RegistryApplicationDTO dto = service.prepareDTO(registryApplication);
      view.displayModel(dto);
    }
  }
```