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

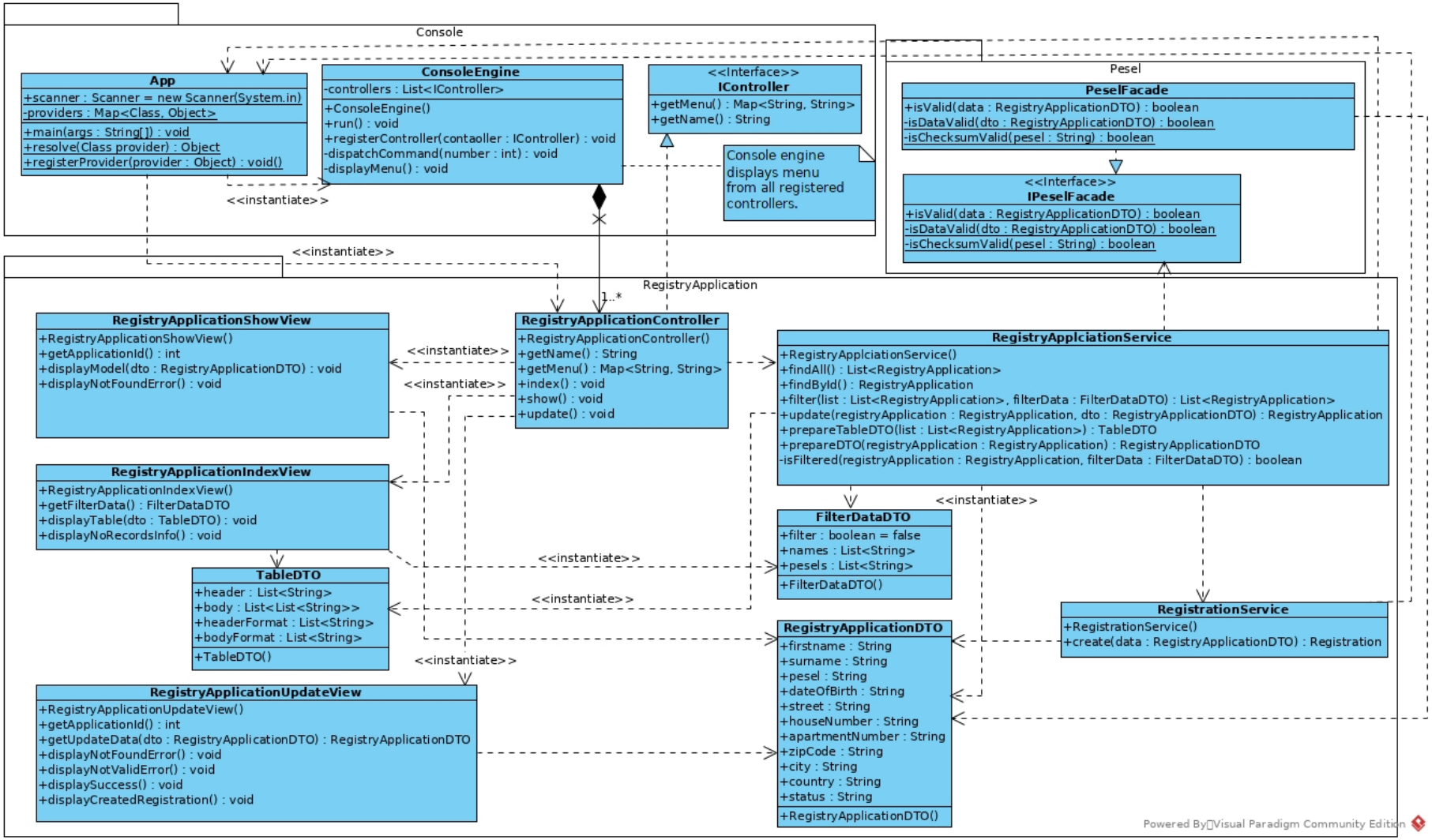
Rolę obiektów Flyweight pełnią klasy `RegistryAddressData` oraz `RegistryPersonalData`. Abstrakcyjnym klientem tych klas jest klasa `RegistrationBase`, z której dziedziczą klasy `RegistryApplication` oraz `Registration`.

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

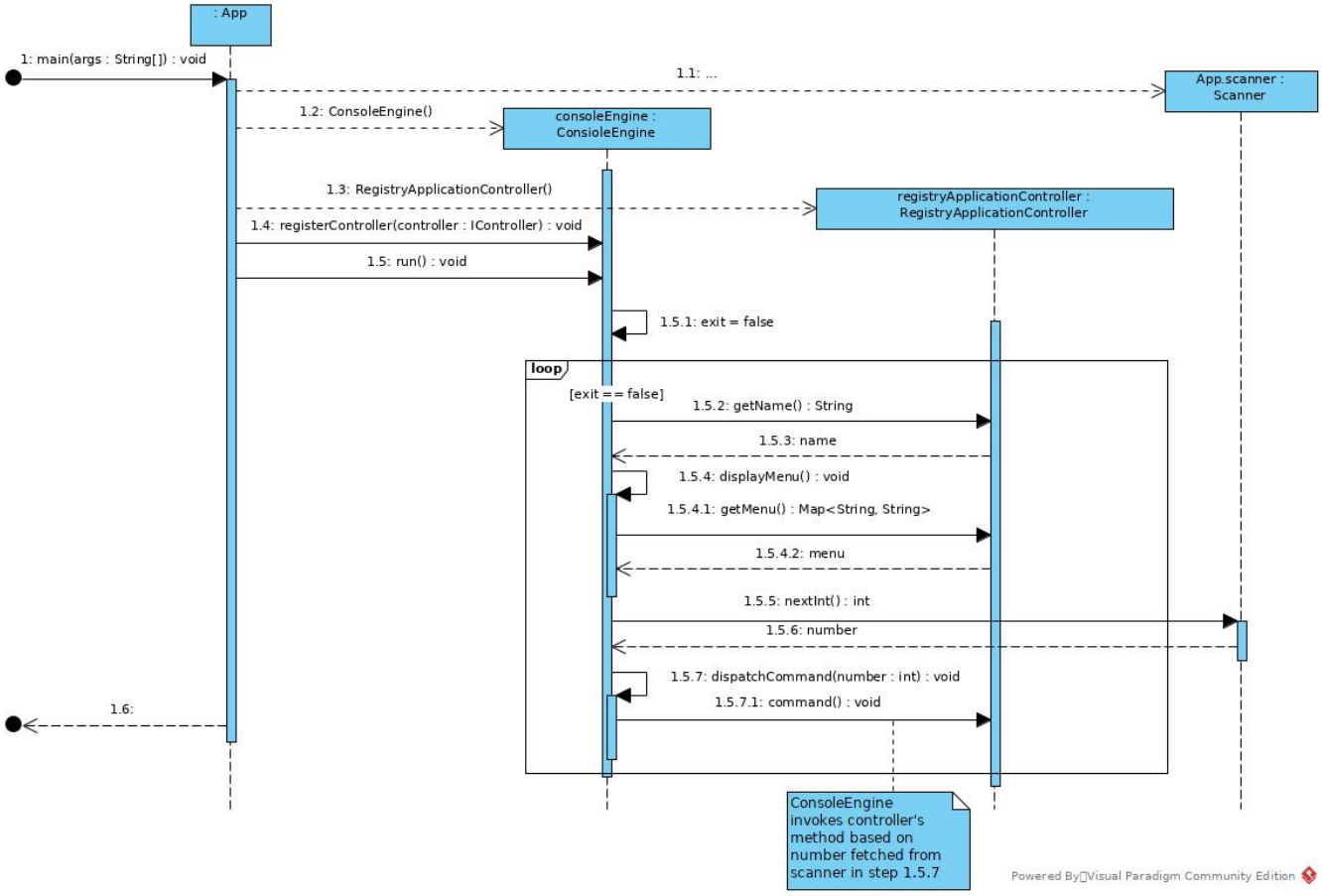
Wzorzec fasada użyty został przy zdefiniowaniu klasy `PeselFasade`, która udostępnia metody umożliwiające komunikację z zewnętrznym systemem. Późniejsza możliwość podmiany implementacji dzięki interfejsowi `IPeselFacade` zapewnia możliwość komunikacji z zewnętrznym systemem w dowolny sposób.



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

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";
```

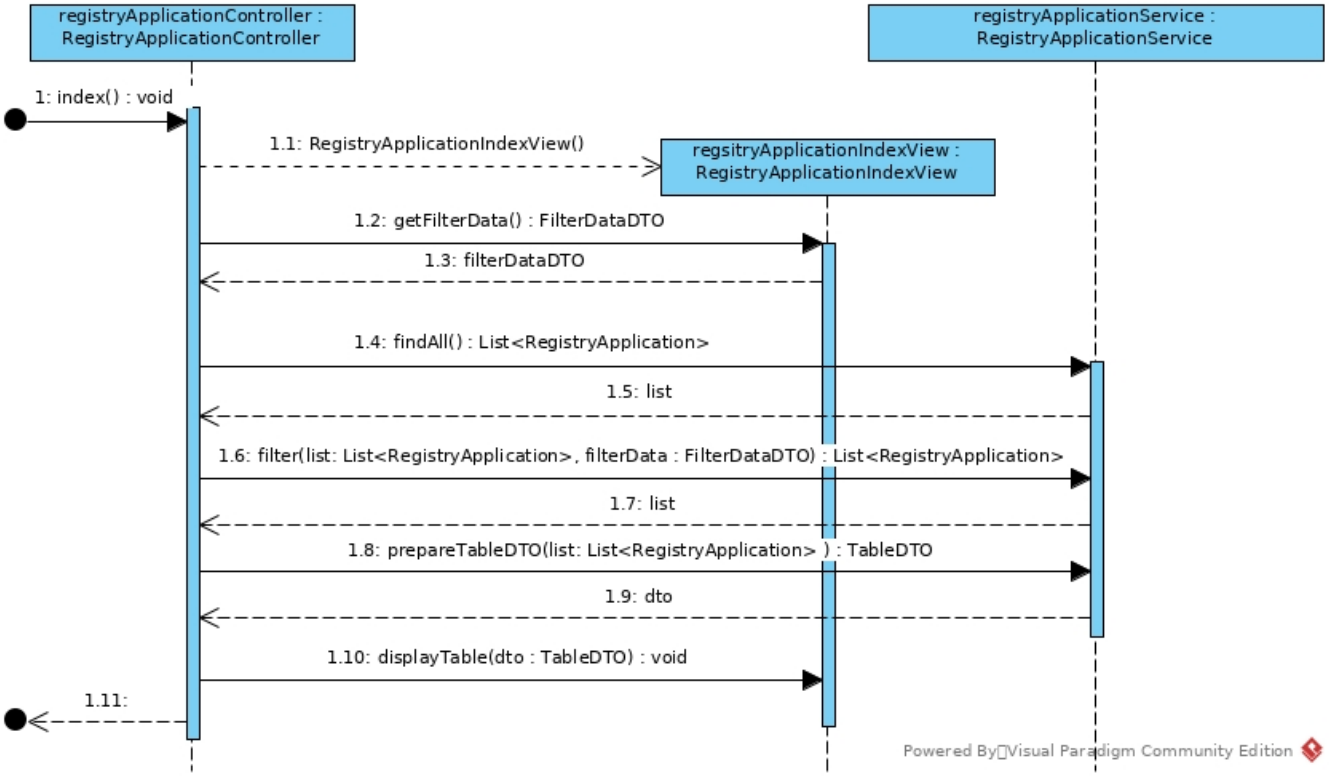
```
        registryApplication.getPersonalData().pesel = "72060319389";
        registryApplication.getAddressData().apartmentNumber = "20";
        registryApplication.getAddressData().houseNumber = "10";
        registryApplication.getAddressData().street = "Marszalkowska";
        registryApplication.getAddressData().zipCode = "00-043";
        registryApplication.getAddressData().country = "Polska";
        registryApplication.getAddressData().city = "Warszawa";
        registryApplicationRepository.save(registryApplication);

        App.registerProvider(new RegistryApplicationService());
        App.registerProvider(registryApplicationRepository);
        App.registerProvider(new RegistrationService());
        App.registerProvider(new RegistrationRepository());

        ConsoleEngine engine = new ConsoleEngine();
        engine.registerController(new RegistryApplicationController());
        engine.run();
    }
```

Listing 1: Metoda main klasy App

5.2 Wyświetlanie wniosków

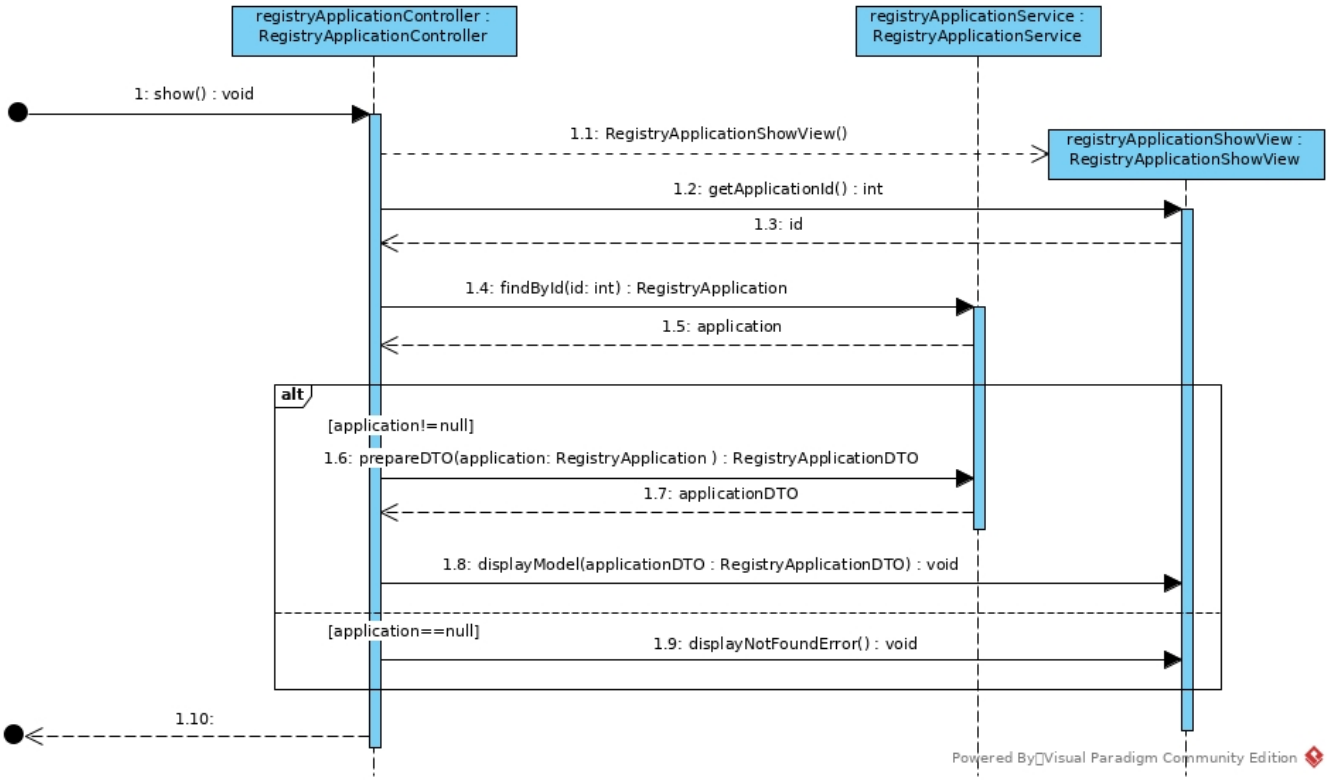


Rysunek 4: Diagram sekwencji - wyświetlanie wniosków.

```
1  public void index() {
2    RegistryApplicationIndexView view = new RegistryApplicationIndexView();
3    FilterDataDTO filterDataDTO = view.getFilterData();
4    RegistryApplicationService service = (RegistryApplicationService) App.
   resolve(RegistryApplicationService.class);
5    List<RegistryApplication> list = service.findAll();
6    list = service.filter(list, filterDataDTO);
7    TableDTO tableDTO = service.prepareTableDTO(list);
8    view.displayTable(tableDTO);
9  }
```

Listing 2: Metoda index klasy RegistryApplicationController

5.3 Wyświetlanie pojedynczego wniosku



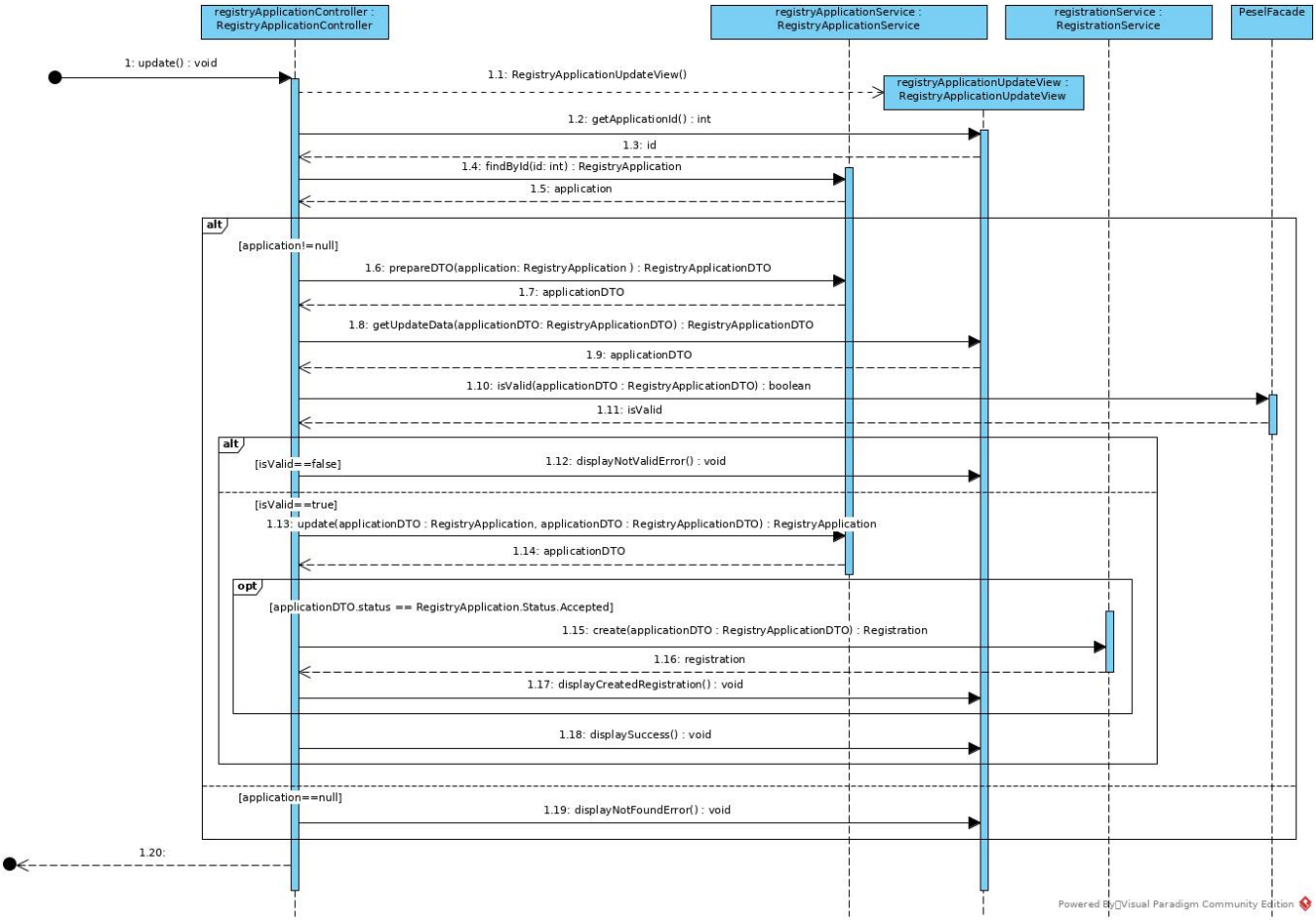
Rysunek 5: Diagram sekwencji - wyświetlanie pojedynczego wniosku.

```
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);
  }
}
```

Listing 3: Metoda show klasy RegistryApplicationController

5.4 Edycja danych wniosku



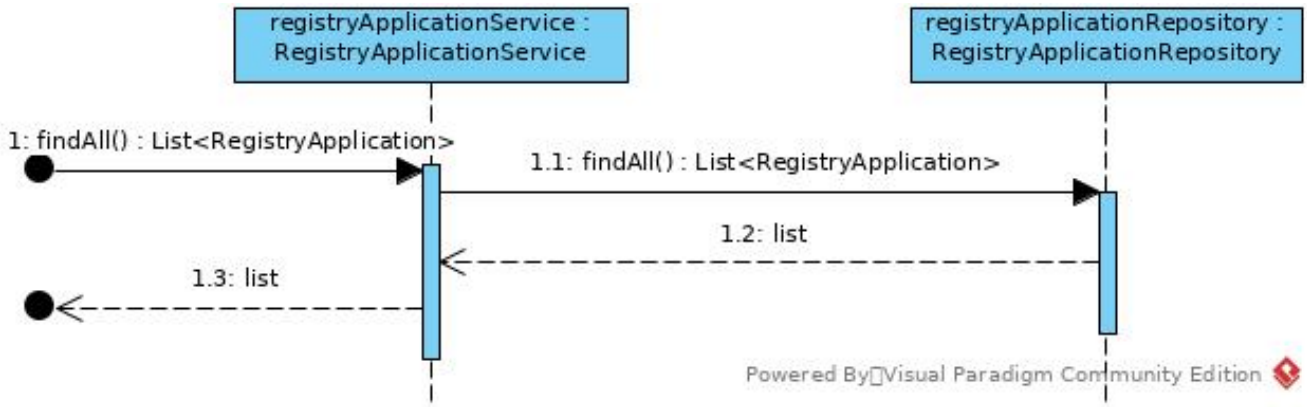
Rysunek 6: Diagram sekwencji - edycja danych wniosku.

```
1  public void update() {
2    RegistryApplicationUpdateView view = new RegistryApplicationUpdateView();
3    RegistryApplicationService registryApplicationService = (
   RegistryApplicationService) App
4        .resolve(RegistryApplicationService.class);
5    RegistrationService registrationService = (RegistrationService) App.resolve
   (RegistrationService.class);
6    int id = view.getApplicationId();
7    RegistryApplication registryApplication = registryApplicationService.
   findById(id);
8    if (registryApplication == null) {
9      view.displayNotFoundError();
10   } else {
```

```
11      RegistryApplicationDTO dto = registryApplicationService.prepareDTO(
    registryApplication);
12      dto = view.getUpdateData(dto);
13      boolean isValid = PeselFacade.isValid(dto);
14      if (!isValid) {
15        view.displayNotValidError();
16      } else {
17        registryApplicationService.update(registryApplication, dto);
18        if (registryApplication.status.equals(RegistryApplication.Status.
    Accepted)) {
19          registrationService.create(dto);
20          view.displayCreatedRegistration();
21        }
22        view.displaySuccess();
23      }
24    }
25  }
```

Listing 4: Metoda update klasy RegistryApplicationController

5.5 Metody klasy RegistryApplicationService

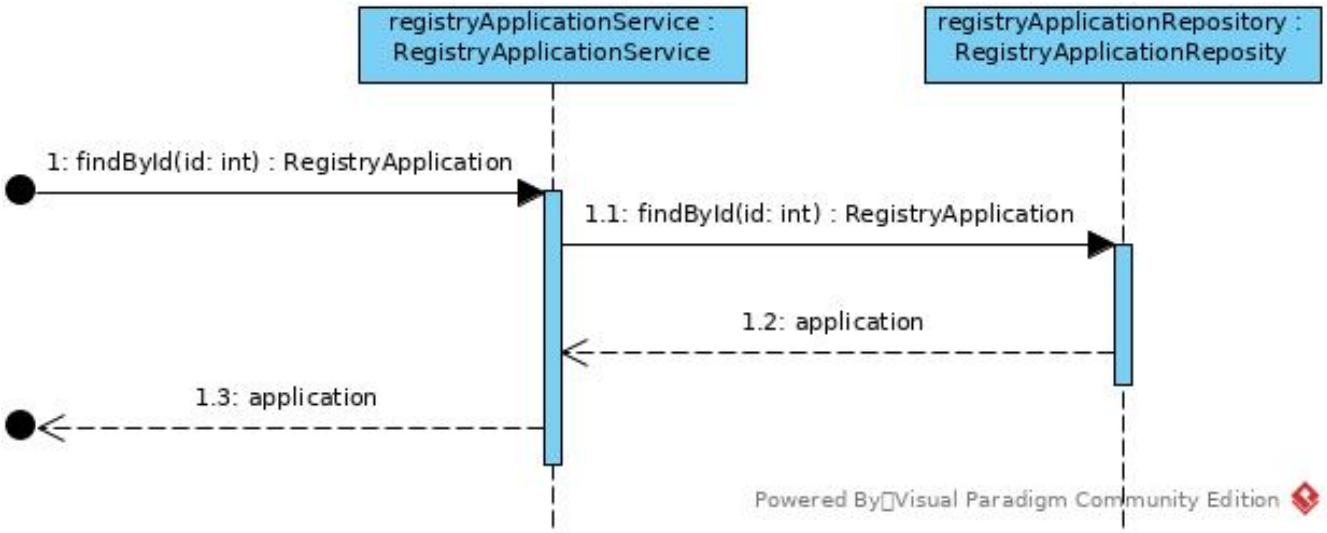


Rysunek 7: Diagram sekwencji - metoda findAll klasy RegistryApplicationService.

```
1     public List<RegistryApplication> findAll() {
2         RegistryApplicationRepository repository = (
    RegistryApplicationRepository) App
3                 .resolve(RegistryApplicationRepository.class);
4         return repository.findAll();
```

```
5     }
```

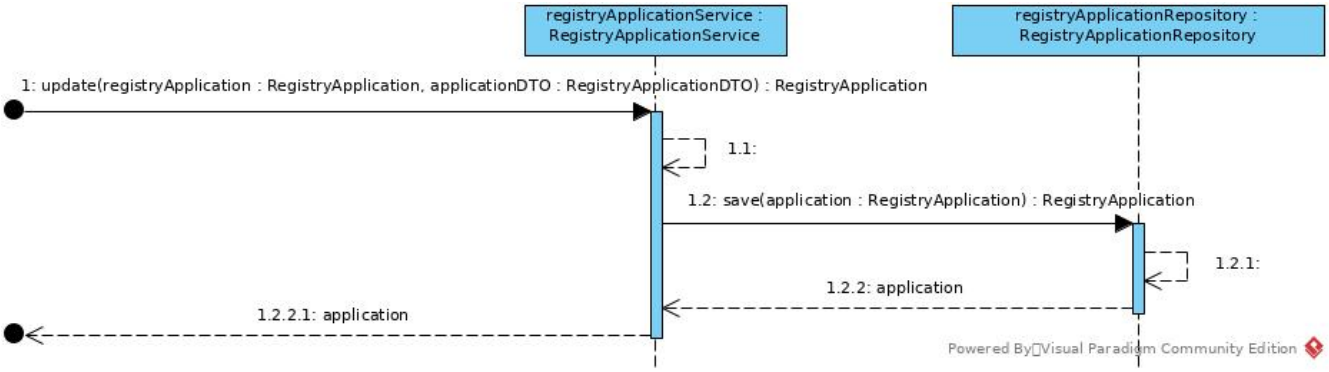
Listing 5: Metoda findAll klasy RegistryApplicationService



Rysunek 8: Diagram sekwencji - metoda findById klasy RegistryApplicationService.

```
1     public RegistryApplication findById(int id) {
2         RegistryApplicationRepository repository = (
    RegistryApplicationRepository) App
3                 .resolve(RegistryApplicationRepository.class);
4         return repository.findById(id);
5     }
```

Listing 6: Metoda findById klasy RegistryApplicationService



Rysunek 9: Diagram sekwencji - metoda update klasy RegistryApplicationService.

```
1     public RegistryApplication update(RegistryApplication registryApplication,
    RegistryApplicationDTO dto) {
```

```
        RegistryPersonalData personal = registryApplication.getPersonalData();
        RegistryAddressData address = registryApplication.getAddressData();

        personal.firstname = dto.firstname;
        personal.surname = dto.surname;
        personal.pesel = dto.pesel;
        address.apartmentNumber = dto.apartmentNumber;
        address.city = dto.city;
        address.country = dto.country;
        address.houseNumber = dto.houseNumber;
        address.zipCode = dto.zipCode;
        address.street = dto.street;
        personal.dateOfBirth = LocalDate.parse(dto.dateOfBirth);

        registryApplication.status = RegistryApplication.Status.valueOfLabel(
    dto.status);

        RegistryApplicationRepository repository = (
    RegistryApplicationRepository) App
                .resolve(RegistryApplicationRepository.class);
        return repository.save(registryApplication);
     }
```

Listing 7: Metoda update klasy RegistryApplicationService