

# 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 pojedynczego wniosku.

### 2.3.1 Data transfer objects

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

## 2.4 PESEL

Komunikację z systemem PESEL odpowiedzialnego za weryfikację 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

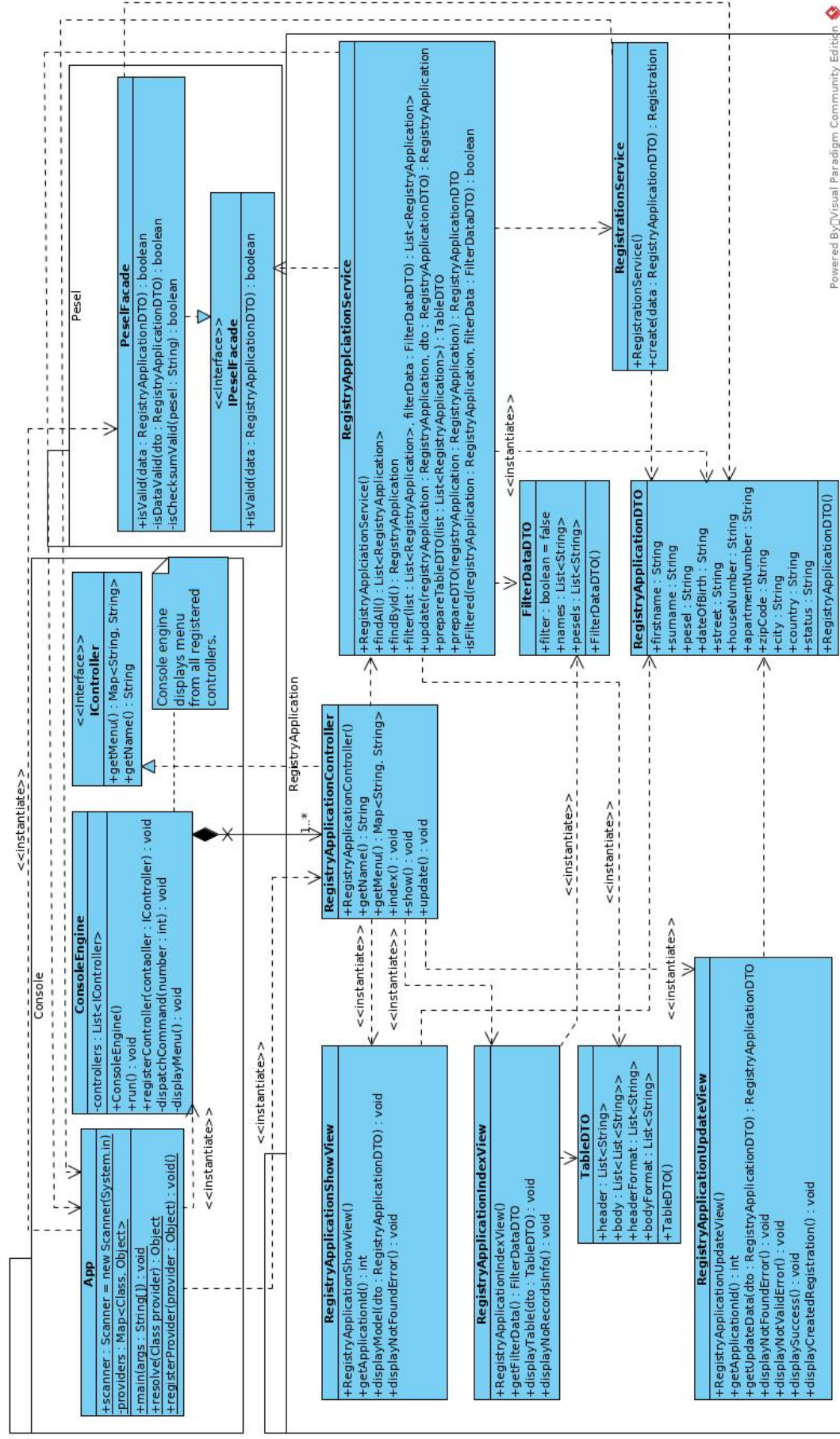
Rolę obiektów Flyweight pełnią klasy `RegistryAddressData` oraz `RegistryPersonalData`. Abstrakcyjnym klientem tych klas jest klasa `RegistrationBase`, z której dziedziczą klasy `RegistryApplication` oraz `Registration`.

## 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

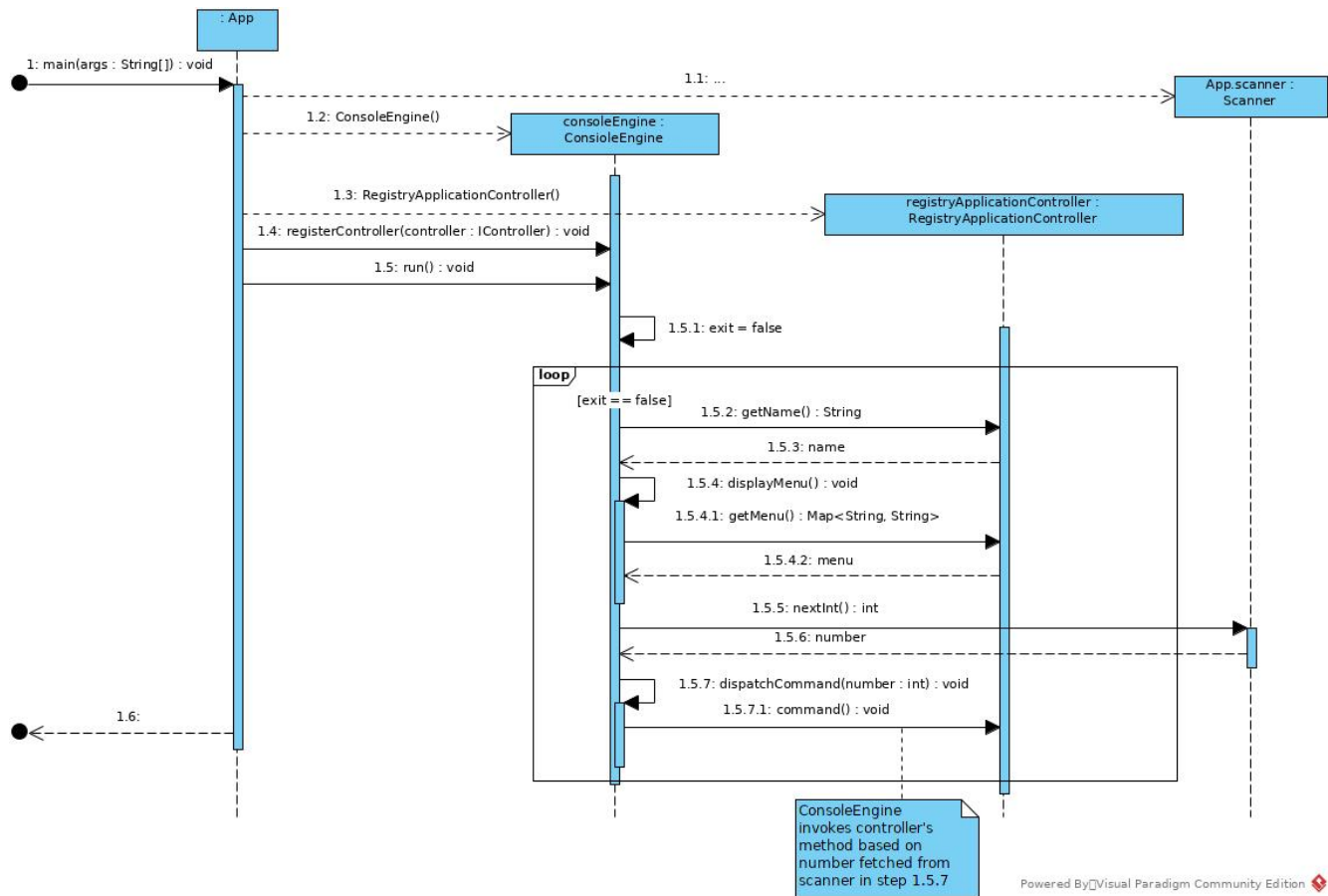
Wzorzec fasada użyty został przy zdefiniowaniu klasy `PeselFacade`, która udostępnia metody umożliwiające komunikację z zewnętrznym systemem. Późniejsza możliwość podmiany implementacji dzięki interfejsowi `IPeselFacade` zapewnia możliwość komunikacji z zewnętrznym systemem w dowolny sposób.



**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 private static HashMap<Class<? extends Object>, Object> providers = new
2 HashMap<Class<? extends Object>, Object>();
3
4 public static void main(String[] args) {
5     RegistryApplicationRepository registryApplicationRepository = new
6     RegistryApplicationRepository();
7
8     /**
9      * Data seed
10     */
11     RegistryApplication registryApplication = new RegistryApplication();
```

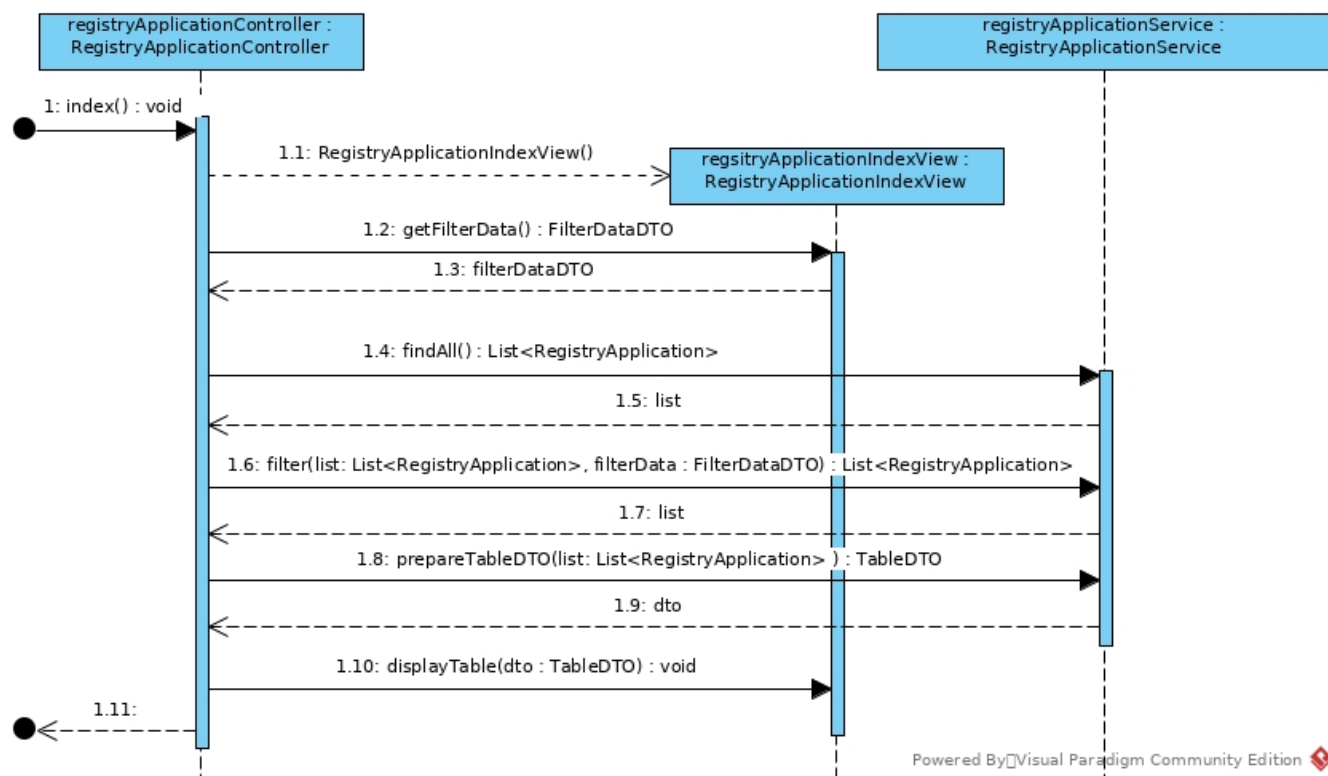
```

10     registryApplication.getPersonalData().dateOfBirth = LocalDate.of(1990,
11     01, 01);
12     registryApplication.getPersonalData().firstname = "Damian";
13     registryApplication.getPersonalData().surname = "Koper";
14     registryApplication.getPersonalData().pesel = "72060319389";
15     registryApplication.getAddressData().apartmentNumber = "20";
16     registryApplication.getAddressData().houseNumber = "10";
17     registryApplication.getAddressData().street = "Marszalkowska";
18     registryApplication.getAddressData().zipCode = "00-043";
19     registryApplication.getAddressData().country = "Polska";
20     registryApplication.getAddressData().city = "Warszawa";
21     registryApplicationRepository.save(registryApplication);
22
23     App.registerProvider(new RegistryApplicationService());
24     App.registerProvider(registryApplicationRepository);
25     App.registerProvider(new RegistrationService());
26     App.registerProvider(new RegistrationRepository());
27     App.registerProvider(new PeselFacade());
28
29     ConsoleEngine engine = new ConsoleEngine();

```

**Listing 1:** Metoda main klasy App

## 5.2 Wyświetlanie wniosków



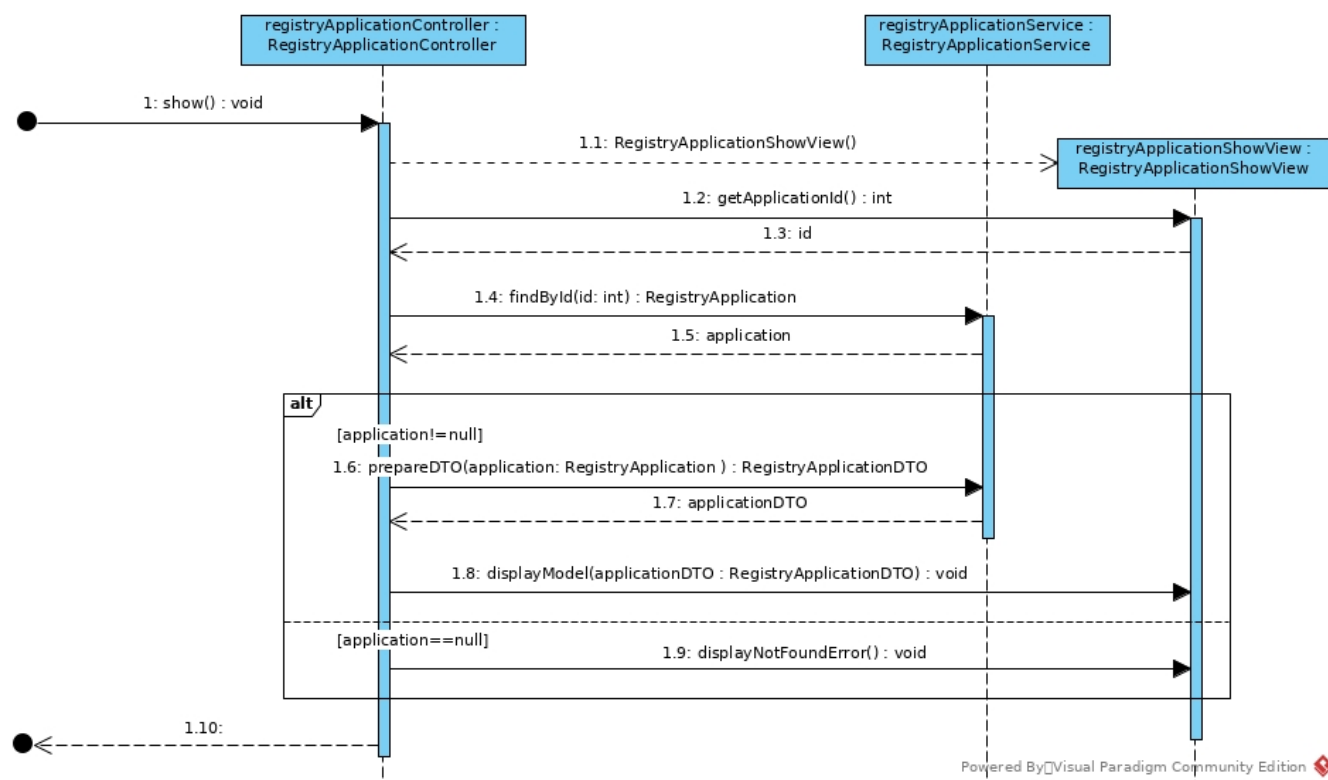
Rysunek 4: Diagram sekwencji - wyświetlanie wniosków.

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

Listing 2: Metoda index klasy RegistryApplicationController



## 5.3 Wyświetlanie pojedynczego wniosku

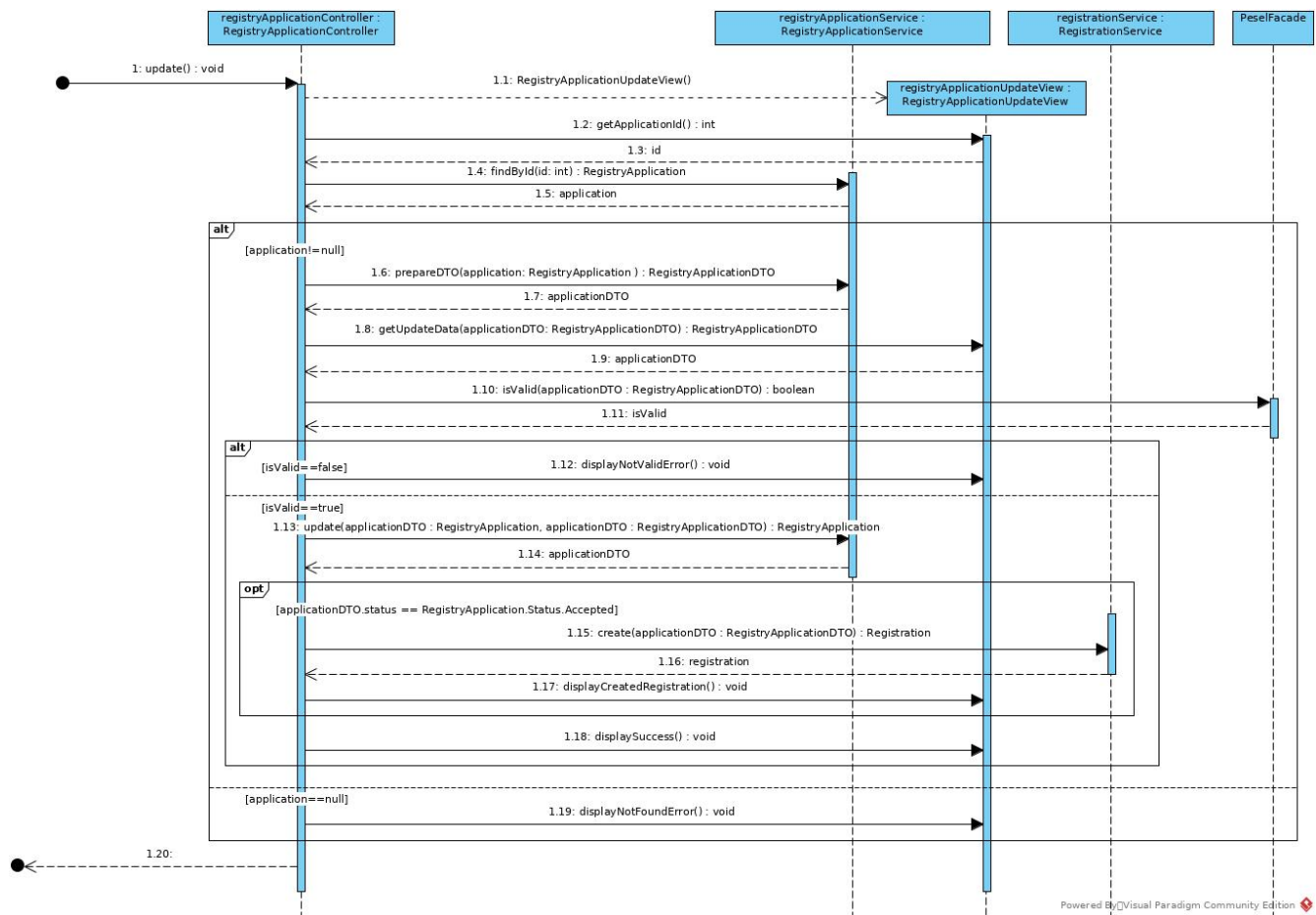


Rysunek 5: Diagram sekwencji - wyświetlanie pojedynczego wniosku.

```
1
2 public void show() {
3     RegistryApplicationShowView view = new RegistryApplicationShowView();
4     RegistryApplicationService service = (RegistryApplicationService) App.
5     resolve(RegistryApplicationService.class);
6     int id = view.getApplicationId();
7     RegistryApplication registryApplication = service.findById(id);
8
9     if (registryApplication == null) {
10         view.displayNotFoundError();
11     } else {
12         RegistryApplicationDTO dto = service.prepareDTO(registryApplication);
13         view.displayModel(dto);
14     }
15 }
```

Listing 3: Metoda show klasy RegistryApplicationController

## 5.4 Edycja danych wniosku



Rysunek 6: Diagram sekwencji - edycja danych wniosku.

```

1
2 public void update() {
3     RegistryApplicationUpdateView view = new RegistryApplicationUpdateView();
4     RegistryApplicationService registryApplicationService = (
5     RegistryApplicationService) App
6         .resolve(RegistryApplicationService.class);
7     RegistrationService registrationService = (RegistrationService) App.resolve
8     (RegistrationService.class);
9     IPeselFacade peselFacade = (IPeselFacade) App.resolve(PeselFacade.class);
10    int id = view.getApplicationId();
11    RegistryApplication registryApplication = registryApplicationService.
12    findById(id);
13    if (registryApplication == null) {
14        view.displayNotFoundError();
15    }
16 }

```

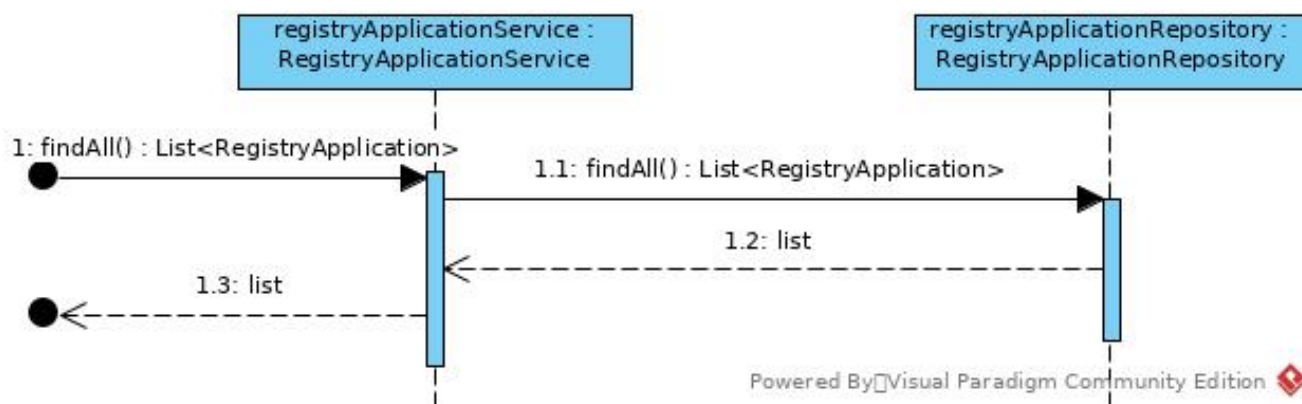
```

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

```

**Listing 4:** Metoda update klasy RegistryApplicationController

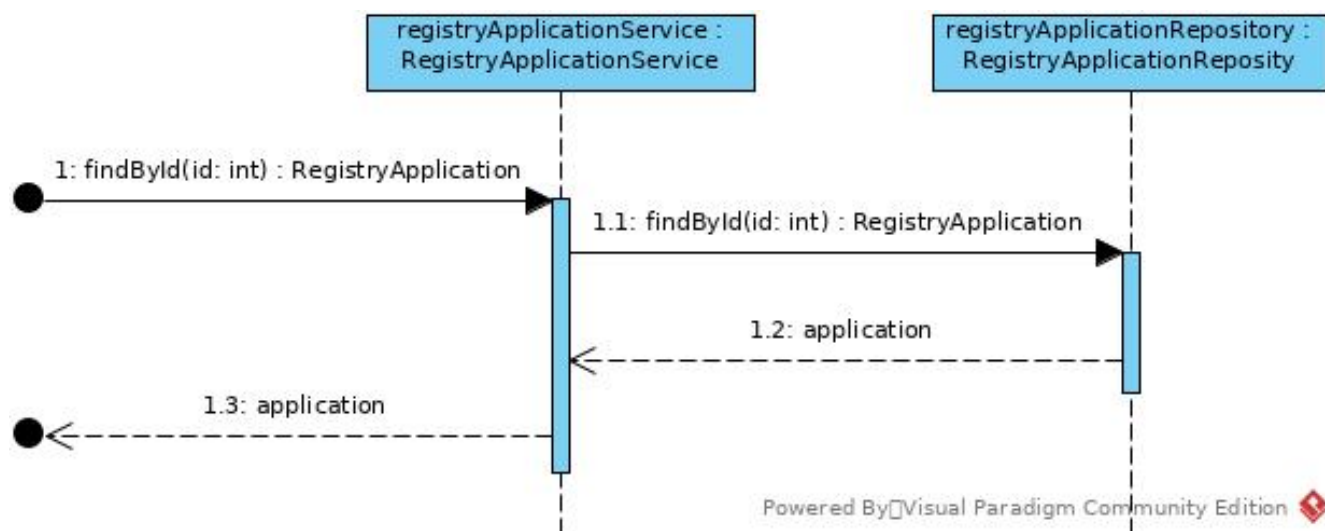
## 5.5 Metody klasy RegistryApplicationService



**Rysunek 7:** Diagram sekwencji - metoda findAll klasy RegistryApplicationService.

```
1 public List<RegistryApplication> findAll() {
2     RegistryApplicationRepository repository = (
3     RegistryApplicationRepository) App
4         .resolve(RegistryApplicationRepository.class);
5     return repository.findAll();
6 }
```

**Listing 5:** Metoda findAll klasy RegistryApplicationService



**Rysunek 8:** Diagram sekwencji - metoda findById klasy RegistryApplicationService.

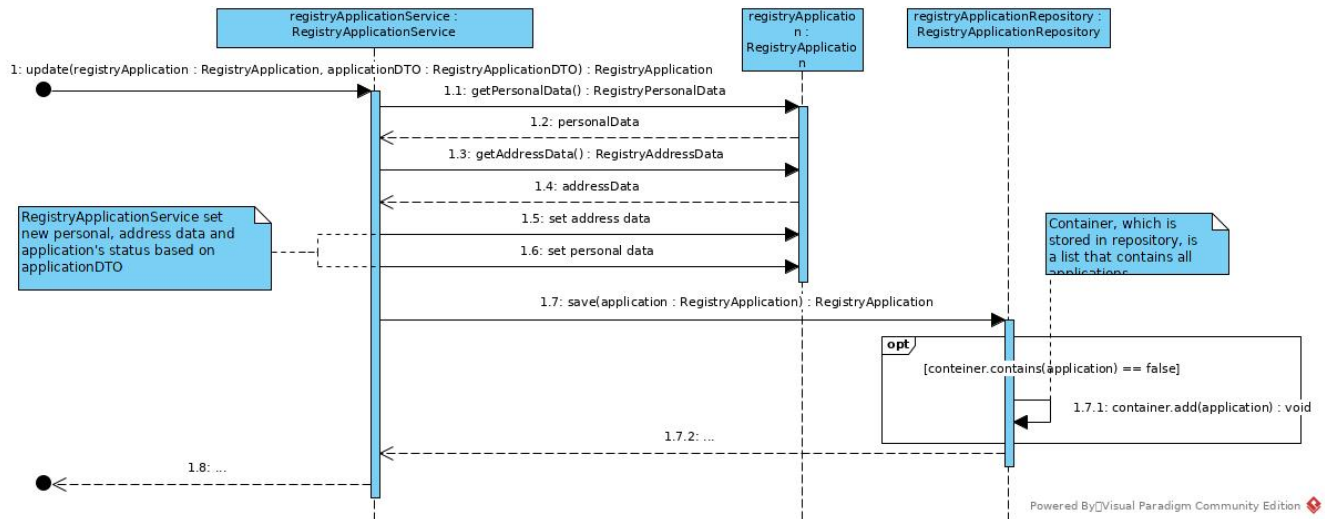
```
1 public RegistryApplication findById(int id) {
2     RegistryApplicationRepository repository = (
3     RegistryApplicationRepository) App
```

```

3         .resolve(RegistryApplicationRepository.class);
4         return repository.findById(id);
5     }

```

**Listing 6:** Metoda findById klasy RegistryApplicationService



**Rysunek 9:** Diagram sekwencji - metoda filter klasy RegistryApplicationService.

```

1
2     public RegistryApplication update(RegistryApplication registryApplication,
3     RegistryApplicationDTO dto) {
4         RegistryPersonalData personal = registryApplication.getPersonalData();
5         RegistryAddressData address = registryApplication.getAddressData();
6
7         personal.firstname = dto.firstname;
8         personal.surname = dto.surname;
9         personal.pesel = dto.pesel;
10        address.apartmentNumber = dto.apartmentNumber;
11        address.city = dto.city;
12        address.country = dto.country;
13        address.houseNumber = dto.houseNumber;
14        address.zipCode = dto.zipCode;
15        address.street = dto.street;
16        personal.dateOfBirth = LocalDate.parse(dto.dateOfBirth);
17
18        registryApplication.status = RegistryApplication.Status.valueOfLabel(
19        dto.status);

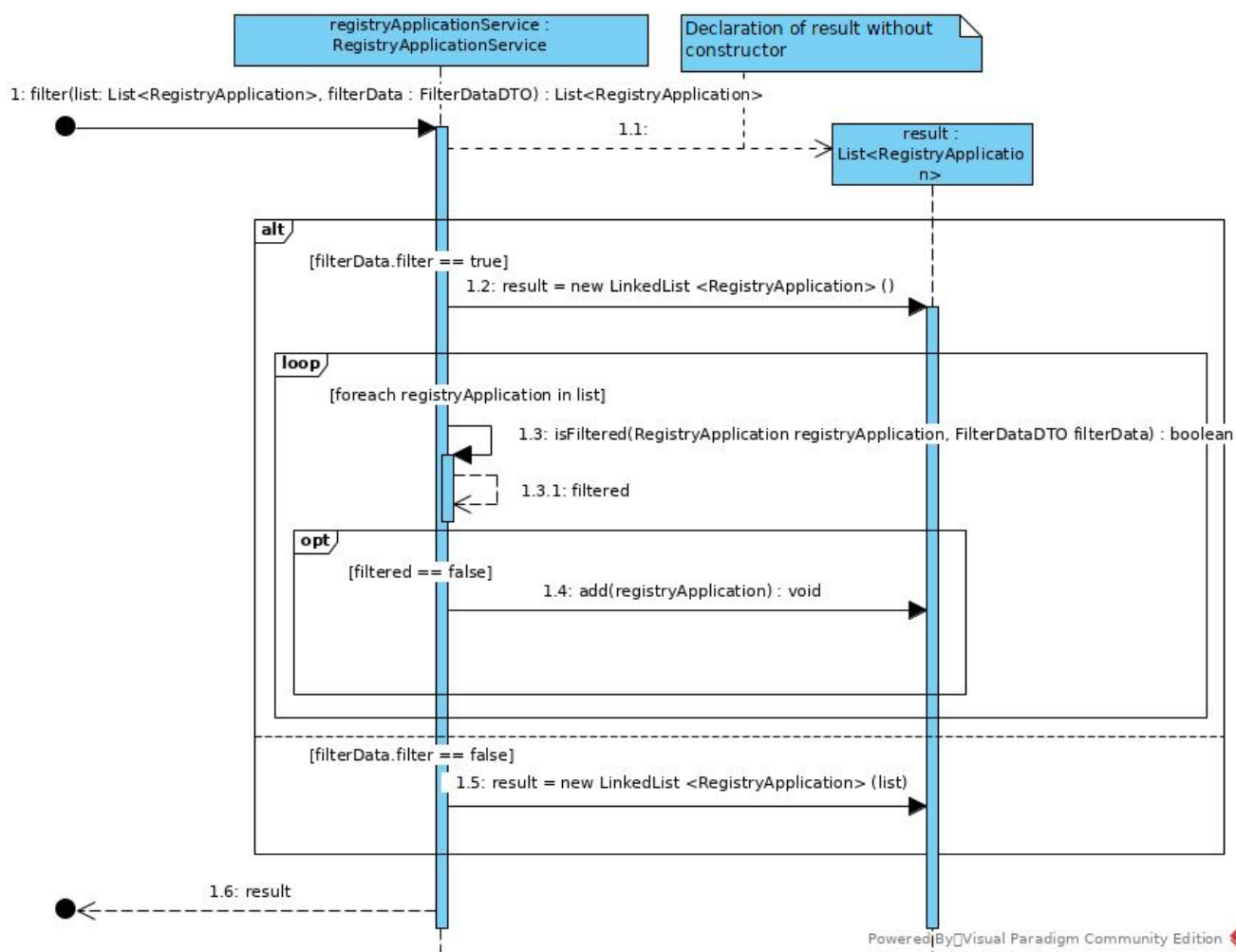
```

```

18
19 RegistryApplicationRepository repository = (
RegistryApplicationRepository) App
20     .resolve(RegistryApplicationRepository.class);
21     return repository.save(registryApplication);

```

**Listing 7:** Metoda update klasy RegistryApplicationService



**Rysunek 10:** Diagram sekwencji - metoda filter klasy RegistryApplicationService.

```

1 public List<RegistryApplication> filter(List<RegistryApplication> list,
FilterDataDTO filterData) {
2     List<RegistryApplication> result;
3     if (filterData.filter) {
4         result = new LinkedList<RegistryApplication>();
5         for (RegistryApplication registryApplication : list) {
6             boolean filtered = isFiltered(registryApplication, filterData);

```

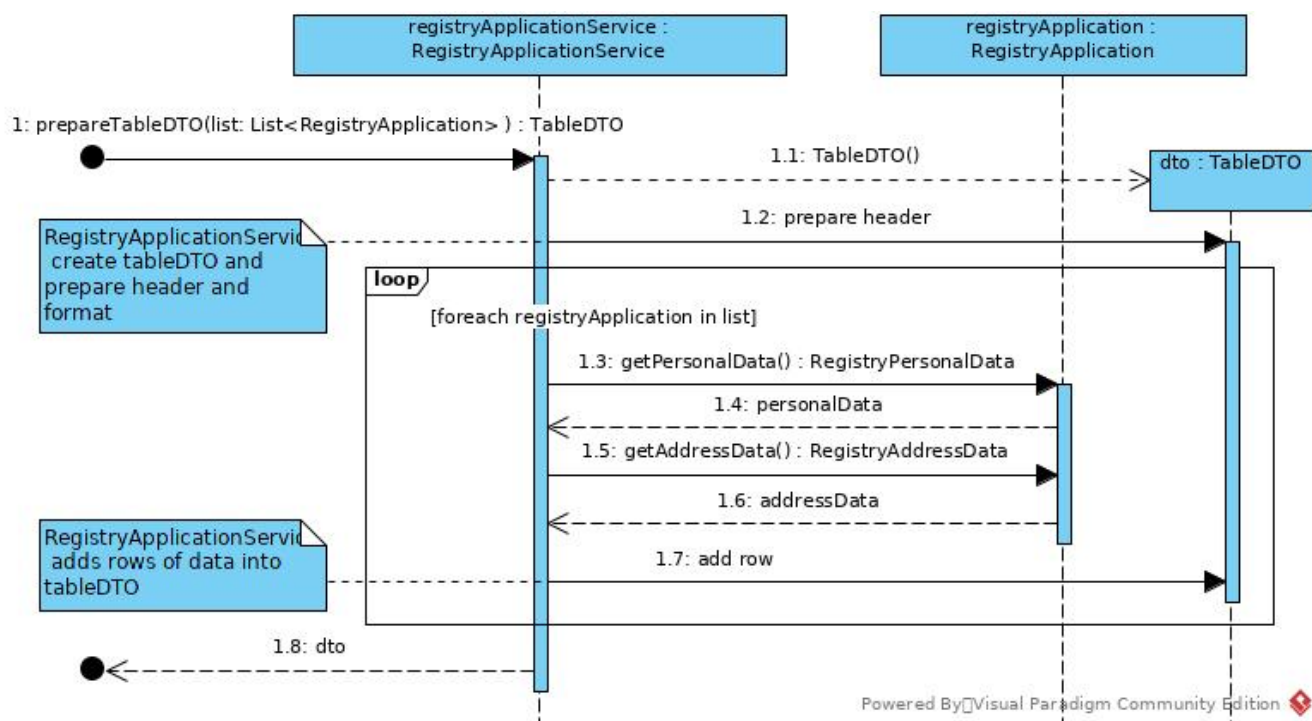
```

7         if (!filtered) {
8             result.add(registryApplication);
9         }
10    }
11    } else {
12        result = new LinkedList<RegistryApplication>(list);
13    }
14    return result;
15 }

16
17 private boolean isFiltered(RegistryApplication registryApplication,
18 FilterDataDTO filterData) {
19     if (filterData.names.stream().anyMatch(name -> {
20         return registryApplication.getPersonalData().firstname.toLowerCase()
21         () == name.toLowerCase()
22         || registryApplication.getPersonalData().surname.
23         toLowerCase() == name.toLowerCase();
24     }))) {
25         return false;
26     }
27     if (filterData.pesels.stream().anyMatch(pesel -> {
28         return registryApplication.getPersonalData().pesel.toLowerCase() ==
29         pesel.toLowerCase();
30     }))) {
31         return false;
32     }
33     return true;
34 }

```

**Listing 8:** Metoda filter klasy RegistryApplicationService



Rysunek 11: Diagram sekwencji - metoda prepareTableDTO klasy RegistryApplicationService.

```

1  public TableDTO prepareTableDTO(List<RegistryApplication> list) {
2      TableDTO dto = new TableDTO();
3      dto.header = new LinkedList<String>(
4          Arrays.asList("Id", "Imie", "Nazwisko", "PESEL", "Wnioskowany
adres zameldowania"));
5      LinkedList<String> format = new LinkedList<String>(Arrays.asList("%5s",
"%15s", "%15s", "%11s", "%60s"));
6      dto.headerFormat = format;
7      dto.bodyFormat = format;
8      for (RegistryApplication registryApplication : list) {
9          LinkedList<String> row = new LinkedList<>();
10         row.add(String.valueOf(registryApplication.id));
11
12         RegistryPersonalData personal = registryApplication.getPersonalData
();
13         row.add(personal.firstname);
14         row.add(personal.surname);
15         row.add(personal.pesel);
16
17         RegistryAddressData address = registryApplication.getAddressData();
  
```

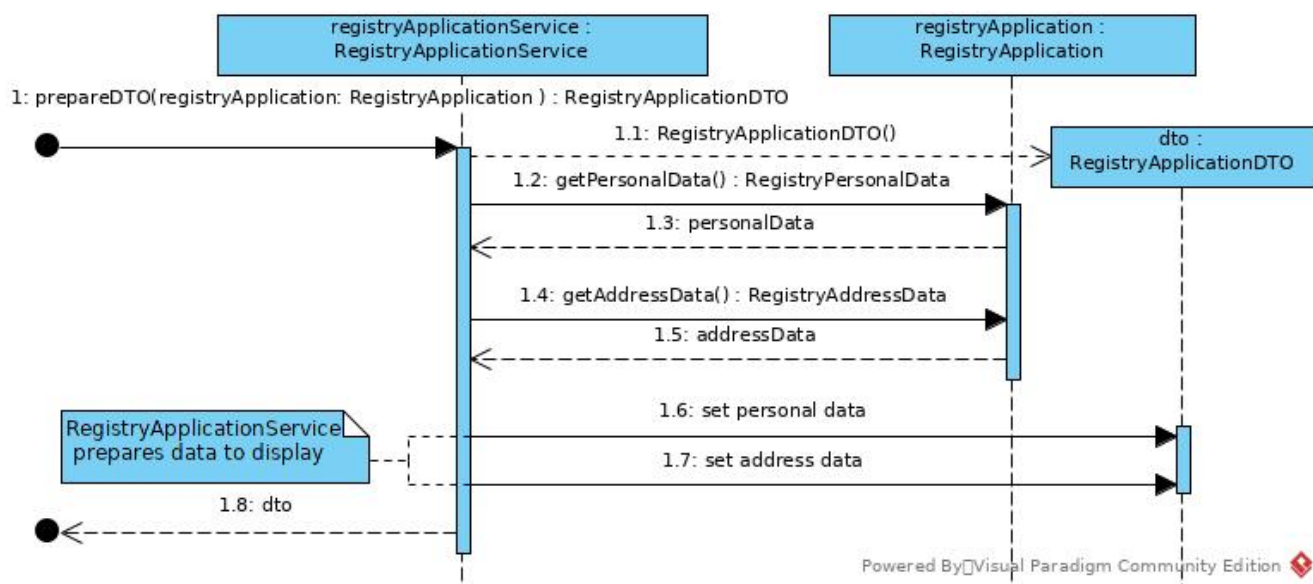


```

18         row.add(address.city + " ul." + address.street + " " + address.
19         houseNumber + "/" + address.apartmentNumber
20             + " " + address.zipCode + ", " + address.country);
21         dto.body.add(row);
22     }
23     return dto;
24 }

```

**Listing 9:** Metoda prepareTableDTO klasy RegistryApplicationService



**Rysunek 12:** Diagram sekwencji - metoda prepareDTO klasy RegistryApplicationService.

```

1     public RegistryApplicationDTO prepareDTO(RegistryApplication
2     registryApplication) {
3         RegistryPersonalData personal = registryApplication.getPersonalData();
4         RegistryAddressData address = registryApplication.getAddressData();
5
6         RegistryApplicationDTO dto = new RegistryApplicationDTO();
7         dto.id = registryApplication.id;
8         dto.firstname = personal.firstname;
9         dto.surname = personal.surname;
10        dto.pesel = personal.pesel;
11        dto.dateOfBirth = personal.dateOfBirth.toString();
12        dto.apartmentNumber = address.apartmentNumber;
13        dto.houseNumber = address.houseNumber;

```

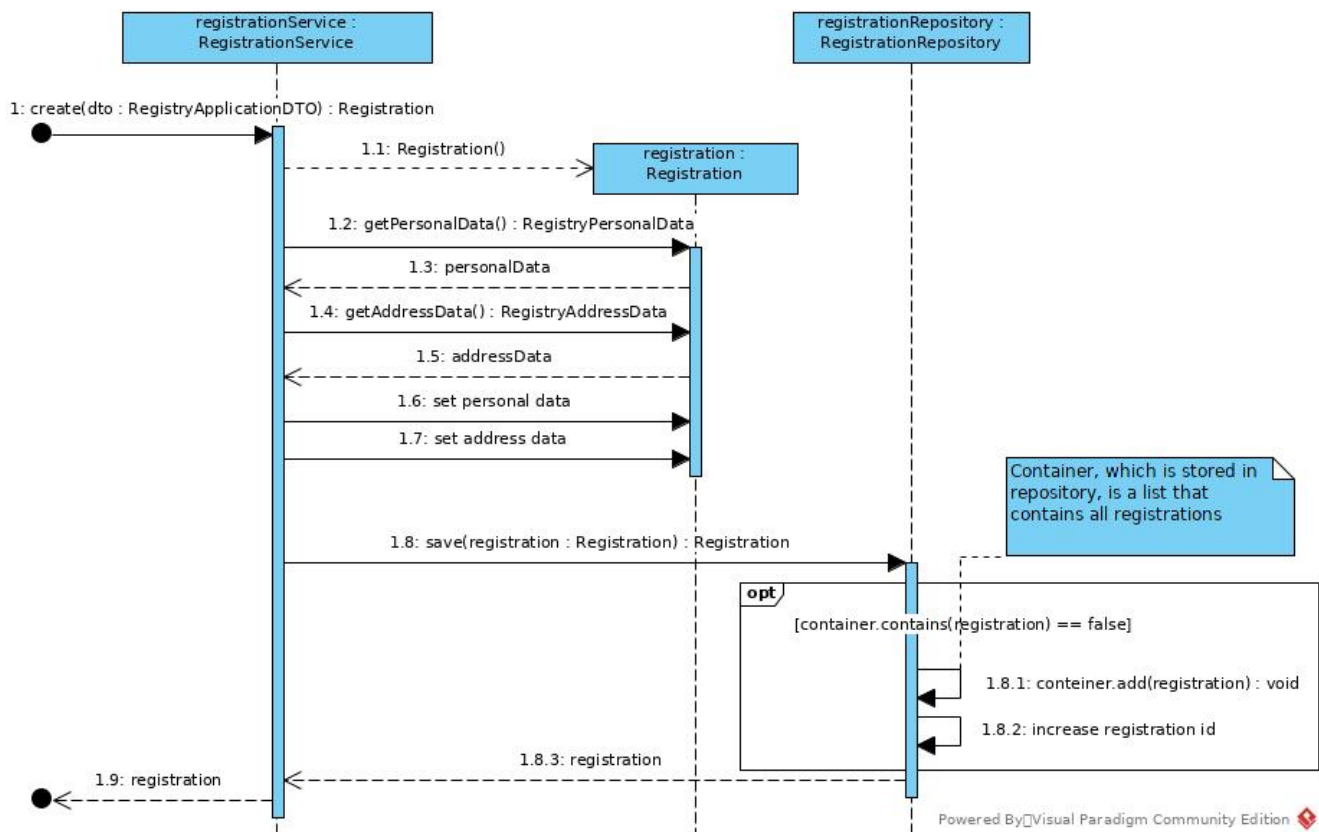
```

13     dto.city = address.city;
14     dto.country = address.country;
15     dto.street = address.street;
16     dto.status = registryApplication.status.toString();
17     dto.zipCode = address.zipCode;
18
19     return dto;
20 }

```

Listing 10: Metoda prepareDTO klasy RegistryApplicationService

## 5.6 Metody klasy RegistrationService



Rysunek 13: Diagram sekwencji - metoda create klasy RegistrationService.

```

1     public Registration create(RegistryApplicationDTO dto) {
2         Registration registration = new Registration();
3         RegistryPersonalData personal = registration.getPersonalData();
4         RegistryAddressData address = registration.getAddressData();
5

```

```

6      personal.firstname = dto.firstname;
7      personal.surname = dto.surname;
8      personal.pesel = dto.pesel;
9      address.apartmentNumber = dto.apartmentNumber;
10     address.city = dto.city;
11     address.country = dto.country;
12     address.houseNumber = dto.houseNumber;
13     address.zipCode = dto.zipCode;
14     address.street = dto.street;
15     personal.dateOfBirth = LocalDate.parse(dto.dateOfBirth);
16     registration.status = Registration.Status.Current;
17
18     RegistrationRepository repository = (RegistrationRepository) App.
19     resolve(RegistrationRepository.class);
20
21     return repository.save(registration);
22 }

```

**Listing 11:** Metoda create klasy RegistrationService

## 6 Prezentacja działania aplikacji

```
### Menu:
[0] Wyjście
--- Wnioski meldunkowe
[1] Wyświetl wnioski
[2] Wyświetl dane wniosku
[3] Edytuj wniosek
1
Włączyc filtrowanie? [y|n]: n

--- Znalezione wnioski:
  Id |      Imię |      Nazwisko |      PESEL |      Wnioskowany adres zameldowania
-----
  1 |    Damian |      Koper | 72060319389 | Warszawa ul.Marszałkowska 10/20 00-043, Polska
```

Rysunek 14: Wyświetlanie wniosków bez filtrowania.

```
### Menu:
[0] Wyjście
--- Wnioski meldunkowe
[1] Wyświetl wnioski
[2] Wyświetl dane wniosku
[3] Edytuj wniosek
2
Numer wniosku: 1

--- Dane wniosku 1
Imię: Damian
Nazwisko: Koper
PESEL: 72060319389
Data urodzenia: 1990-01-01
Ulica: Marszałkowska
Nr domu: 10
Nr mieszkania: 20
Kod pocztowy: 00-043
Miasto: Warszawa
Kraj: Polska
Status: Oczekujący
```

Rysunek 15: Wyświetlanie wniosku.

```
### Menu:
[0] Wyjście
--- Wnioski meldunkowe
[1] Wyświetl wnioski
[2] Wyświetl dane wniosku
[3] Edytuj wniosek
3
Numer wniosku: 1

--- Aktualizacja danych wniosku 1
--- Pusta linia nie zmienia danych
Imię[Damian]:
Nazwisko[Koper]: Pietruszka
PESEL[72060319389]:
Data urodzenia[1990-01-01]: 1990-01-02
Ulica[Marszałkowska]:
Nr domu[10]: 11
Nr mieszkania[20]:
Kod pocztowy[00-043]:
Miasto[Warszawa]:
Kraj[Polska]:
Status[Oczekujący][Oczekujący|Zaakceptowany|Odrzucony]:

Operacja przeprowadzona pomyślnie!
```

Rysunek 16: Edycja danych wniosku.

## 7 Kod źródłowy aplikacji

W celu skrócenia sprawozdania wszystkie sekcje importu na listingach zostały usunięte.

```
1 public class App {
2
3     public static Scanner scanner = new Scanner(System.in);
4     private static HashMap<Class<? extends Object>, Object> providers = new
    HashMap<Class<? extends Object>, Object>();
5
6     public static void main(String[] args) {
7         RegistryApplicationRepository registryApplicationRepository = new
    RegistryApplicationRepository();
8
9         /**
10          * Data seed
11          */
12         RegistryApplication registryApplication = new RegistryApplication();
13         registryApplication.getPersonalData().dateOfBirth = LocalDate.of(1990,
    01, 01);
14         registryApplication.getPersonalData().firstname = "Damian";
15         registryApplication.getPersonalData().surname = "Koper";
16         registryApplication.getPersonalData().pesel = "72060319389";
17         registryApplication.getAddressData().apartmentNumber = "20";
18         registryApplication.getAddressData().houseNumber = "10";
19         registryApplication.getAddressData().street = "Marszalkowska";
20         registryApplication.getAddressData().zipCode = "00-043";
21         registryApplication.getAddressData().country = "Polska";
22         registryApplication.getAddressData().city = "Warszawa";
23         registryApplicationRepository.save(registryApplication);
24
25         App.registerProvider(new RegistryApplicationService());
26         App.registerProvider(registryApplicationRepository);
27         App.registerProvider(new RegistrationService());
28         App.registerProvider(new RegistrationRepository());
29         App.registerProvider(new PeselFacade());
30
31         ConsoleEngine engine = new ConsoleEngine();
32         engine.registerController(new RegistryApplicationController());
```

```

33     engine.run();
34 }
35
36 public static Object resolve(Class<? extends Object> provider) {
37     return App.providers.get(provider);
38 }
39
40 public static void registerProvider(Object provider) {
41     App.providers.put(provider.getClass(), provider);
42 }
43 }

```

**Listing 12:** Klasa App

```

1 public interface IController {
2
3     public Map<String, String> getMenu();
4
5     public String getName();
6 }

```

**Listing 13:** Interface IController

```

1 public class ConsoleEngine {
2
3     private LinkedList<IController> controllers = new LinkedList<>();
4
5     public void run() {
6         boolean exit = false;
7         int input = 0;
8         while (!exit) {
9             displayMenu();
10            input = App.scanner.nextInt();
11            if (input == 0) {
12                exit = true;
13            } else {
14                dispatchCommand(input);
15            }
16        }
17    }
18 }

```

```

17 }
18
19 public void registerController(IController controller) {
20     controllers.add(controller);
21 }
22
23 private void dispatchCommand(int number) {
24     int commands = 1;
25     for (IController iController : controllers) {
26         int commandCount = iController.getMenu().size();
27         if (number <= commandCount - commands + 1) {
28             try {
29                 Method method = iController.getClass()
30                     .getMethod(iController.getMenu().keySet().toArray()[number -
31 commands].toString());
32                 method.invoke(iController);
33             } catch (IllegalAccessException | IllegalArgumentException |
34 InvocationTargetException
35 | NoSuchMethodException e) {
36                 e.printStackTrace();
37             }
38             return;
39         }
40         commands += commandCount;
41     }
42 }
43
44 private void displayMenu() {
45     int option = 1;
46     System.out.println("\n### Menu:");
47     System.out.println("[0] Wyjście");
48     for (IController iController : controllers) {
49         System.out.println("--- " + iController.getName());
50         for (String name : iController.getMenu().keySet()) {
51             System.out.println "[" + String.valueOf(option) + "]" + iController.
52 getMenu().get(name));
53             option = option + 1;
54         }
55     }
56 }

```

```

52     }
53 }
54 }

```

**Listing 14:** Klasa ConsoleEngine

```

1 public interface IPeselFacade {
2
3     public boolean isValid(RegistryApplicationDTO dto);
4
5 }

```

**Listing 15:** Interface IPeselFacade

```

1 public class PeselFacade implements IPeselFacade {
2     private boolean isChecksumValid(String pesel) {
3         String integers[] = pesel.split("");
4         if (integers.length != 11) {
5             return false;
6         }
7         ArrayList<Integer> values = new ArrayList<>();
8         for (String string : integers) {
9             values.add(Integer.parseInt(string));
10        }
11        int[] m = { 1, 3, 7, 9 };
12        int sum = 0;
13        for (int i = 0; i < values.size() - 1; i++) {
14            sum += m[i % 4] * values.get(i);
15        }
16        sum += values.get(values.size() - 1);
17        sum %= 10;
18
19        return sum == 0;
20    }
21
22    private boolean isValid(RegistryApplicationDTO dto) {
23        /**
24         * Validation hidden behind facade. Connection to PESEL system.
25         */

```



```

26         return true;
27     }
28
29     public boolean isValid(RegistryApplicationDTO dto) {
30         if (!isCheckedValid(dto.pesel))
31             return false;
32         return isValid(dto);
33     }
34
35 }

```

**Listing 16:** Klasa PeselFacade

```

1 public interface IRepository<T> {
2
3     public List<T> findAll();
4
5     public T findById(int id);
6
7     public T save(T object);
8 }

```

**Listing 17:** Interface IRepository

```

1 public class RegistrationRepository implements IRepository<Registration> {
2
3     private int nextId = 1;
4     private LinkedList<Registration> container = new LinkedList<>();
5
6     @Override
7     public List<Registration> findAll() {
8         return container;
9     }
10
11     @Override
12     public Registration findById(int id) {
13         return container.stream().filter(o -> o.id == id).findAny().orElse(null);
14     }
15 }

```

```

15
16     @Override
17     public Registration save(Registration object) {
18         if (!container.contains(object)) {
19             container.add(object);
20             object.id = nextId++;
21         }
22         return object;
23     }
24
25 }

```

**Listing 18:** Klasa RegistrationRepository

```

1 public class RegistryApplicationRepository implements IRepository<
   RegistryApplication> {
2
3     private int nextId = 1;
4     private LinkedList<RegistryApplication> container = new LinkedList<>();
5
6     @Override
7     public List<RegistryApplication> findAll() {
8         return container;
9     }
10
11     @Override
12     public RegistryApplication findById(int id) {
13         return container.stream().filter(o -> o.id == id).findAny().orElse(null);
14     }
15
16     @Override
17     public RegistryApplication save(RegistryApplication object) {
18         if (!container.contains(object)) {
19             container.add(object);
20             object.id = nextId++;
21         }
22         return object;
23     }

```

24  
25 }

### Listing 19: Klasa RegistryApplicationRepository

```
1 public class RegistrationService {
2
3     public Registration create(RegistryApplicationDTO dto) {
4         Registration registration = new Registration();
5         RegistryPersonalData personal = registration.getPersonalData();
6         RegistryAddressData address = registration.getAddressData();
7
8         personal.firstname = dto.firstname;
9         personal.surname = dto.surname;
10        personal.pesel = dto.pesel;
11        address.apartmentNumber = dto.apartmentNumber;
12        address.city = dto.city;
13        address.country = dto.country;
14        address.houseNumber = dto.houseNumber;
15        address.zipCode = dto.zipCode;
16        address.street = dto.street;
17        personal.dateOfBirth = LocalDate.parse(dto.dateOfBirth);
18        registration.status = Registration.Status.Current;
19
20        RegistrationRepository repository = (RegistrationRepository) App.
        resolve(RegistrationRepository.class);
21
22        return repository.save(registration);
23    }
24
25 }
```

### Listing 20: Klasa RegistrationService

```
1 public class RegistryApplicationService {
2
3     public List<RegistryApplication> findAll() {
4         RegistryApplicationRepository repository = (
        RegistryApplicationRepository) App
```

```

5         .resolve(RegistryApplicationRepository.class);
6         return repository.findAll();
7     }
8
9     public RegistryApplication findById(int id) {
10         RegistryApplicationRepository repository = (
11             RegistryApplicationRepository) App
12                 .resolve(RegistryApplicationRepository.class);
13         return repository.findById(id);
14     }
15
16     public List<RegistryApplication> filter(List<RegistryApplication> list,
17         FilterDataDTO filterData) {
18         List<RegistryApplication> result;
19         if (filterData.filter) {
20             result = new LinkedList<RegistryApplication>();
21             for (RegistryApplication registryApplication : list) {
22                 boolean filtered = isFiltered(registryApplication, filterData);
23                 if (!filtered) {
24                     result.add(registryApplication);
25                 }
26             }
27         } else {
28             result = new LinkedList<RegistryApplication>(list);
29         }
30         return result;
31     }
32
33     private boolean isFiltered(RegistryApplication registryApplication,
34         FilterDataDTO filterData) {
35         if (filterData.names.stream().anyMatch(name -> {
36             return registryApplication.getPersonalData().firstname.toLowerCase()
37                 == name.toLowerCase()
38                 || registryApplication.getPersonalData().surname.
39                 toLowerCase() == name.toLowerCase();
40         })) {
41             return false;
42         }
43     }

```

```

38         if (filterData.pesels.stream().anyMatch(pesel -> {
39             return registryApplication.getPersonalData().pesel.toLowerCase() ==
pesel.toLowerCase();
40         }))) {
41             return false;
42         }
43         return true;
44     }
45
46     public RegistryApplication update(RegistryApplication registryApplication,
RegistryApplicationDTO dto) {
47         RegistryPersonalData personal = registryApplication.getPersonalData();
48         RegistryAddressData address = registryApplication.getAddressData();
49
50         personal.firstname = dto.firstname;
51         personal.surname = dto.surname;
52         personal.pesel = dto.pesel;
53         address.apartmentNumber = dto.apartmentNumber;
54         address.city = dto.city;
55         address.country = dto.country;
56         address.houseNumber = dto.houseNumber;
57         address.zipCode = dto.zipCode;
58         address.street = dto.street;
59         personal.dateOfBirth = LocalDate.parse(dto.dateOfBirth);
60
61         registryApplication.status = RegistryApplication.Status.valueOfLabel(
dto.status);
62
63         RegistryApplicationRepository repository = (
RegistryApplicationRepository) App
64             .resolve(RegistryApplicationRepository.class);
65         return repository.save(registryApplication);
66     }
67
68     public RegistryApplicationDTO prepareDTO(RegistryApplication
registryApplication) {
69         RegistryPersonalData personal = registryApplication.getPersonalData();
70         RegistryAddressData address = registryApplication.getAddressData();

```

```

71
72     RegistryApplicationDTO dto = new RegistryApplicationDTO();
73     dto.id = registryApplication.id;
74     dto.firstname = personal.firstname;
75     dto.surname = personal.surname;
76     dto.pesel = personal.pesel;
77     dto.dateOfBirth = personal.dateOfBirth.toString();
78     dto.apartmentNumber = address.apartmentNumber;
79     dto.houseNumber = address.houseNumber;
80     dto.city = address.city;
81     dto.country = address.country;
82     dto.street = address.street;
83     dto.status = registryApplication.status.toString();
84     dto.zipCode = address.zipCode;
85
86     return dto;
87 }
88
89 public TableDTO prepareTableDTO(List<RegistryApplication> list) {
90     TableDTO dto = new TableDTO();
91     dto.header = new LinkedList<String>(
92         Arrays.asList("Id", "Imie", "Nazwisko", "PESEL", "Wnioskowany
adres zameldowania"));
93     LinkedList<String> format = new LinkedList<String>(Arrays.asList("%5s",
"%15s", "%15s", "%11s", "%60s"));
94     dto.headerFormat = format;
95     dto.bodyFormat = format;
96     for (RegistryApplication registryApplication : list) {
97         LinkedList<String> row = new LinkedList<>();
98         row.add(String.valueOf(registryApplication.id));
99
100         RegistryPersonalData personal = registryApplication.getPersonalData
();
101         row.add(personal.firstname);
102         row.add(personal.surname);
103         row.add(personal.pesel);
104
105         RegistryAddressData address = registryApplication.getAddressData();

```

```

106         row.add(address.city + " ul." + address.street + " " + address.
houseNumber + "/" + address.apartmentNumber
107             + " " + address.zipCode + ", " + address.country);
108         dto.body.add(row);
109     }
110     return dto;
111 }
112 }

```

**Listing 21:** Klasa RegistryApplicationService

```

1 public class FilterDataDTO {
2
3     public boolean filter = false;
4     public LinkedList<String> names = new LinkedList<>();
5     public LinkedList<String> pesels = new LinkedList<>();
6 }

```

**Listing 22:** Klasa FilterDataDTO

```

1 public class RegistryApplicationDTO {
2
3     public int id;
4     public String firstname;
5     public String surname;
6     public String pesel;
7     public String dateOfBirth;
8     public String street;
9     public String houseNumber;
10    public String apartmentNumber;
11    public String zipCode;
12    public String city;
13    public String country;
14    public String status;
15 }

```

**Listing 23:** Klasa RegistryApplicationDTO

```

1 public abstract class RegistrationBase {
2     public int id = -1;

```

```

3  protected RegistryAddressData addressData = new RegistryAddressData();
4  protected RegistryPersonalData personalData = new RegistryPersonalData();
5
6  /**
7   * @return the addressData
8   */
9  public RegistryAddressData getAddressData() {
10     return addressData;
11 }
12
13 /**
14  * @return the personalData
15  */
16 public RegistryPersonalData getPersonalData() {
17     return personalData;
18 }
19 }

```

**Listing 24:** Klasa RegistrationBase

```

1  public class Registration extends RegistrationBase {
2      public Status status = Status.Current;
3
4      public enum Status {
5          Current("Obecny"), Outdated("Przeszly");
6
7          private String status;
8
9          Status(String status) {
10              this.status = status;
11          }
12
13          @Override
14          public String toString() {
15              return status;
16          }
17      }
18  }

```



18 }

### Listing 25: Klasa Registration

```
1 public class RegistryApplication extends RegistrationBase {
2     public Status status = Status.Pending;
3
4     public enum Status {
5         Pending("Oczekujacy"), Accepted("Zaakceptowany"), Revoked("Odrzucony");
6
7         private String status;
8
9         Status(String status) {
10             this.status = status;
11         }
12
13         @Override
14         public String toString() {
15             return status;
16         }
17
18         public static Status valueOfLabel(String label) {
19             for (Status e : values()) {
20                 if (e.status.equals(label)) {
21                     return e;
22                 }
23             }
24             return null;
25         }
26     }
27 }
```

### Listing 26: Klasa RegistryApplication

```
1 public class RegistryAddressData {
2
3     public String street = "";
4     public String houseNumber = "";
5     public String apartmentNumber = "";
```

```
6     public String zipCode = "";
7     public String city = "";
8     public String country = "";
9 }
```

**Listing 27:** Klasa RegistryAddressData

```
1 public class RegistryPersonalData {
2
3     public String firstname = "";
4     public String surname = "";
5     public String pesel = "";
6     public LocalDate dateOfBirth = LocalDate.of(1970, 01, 01);
7 }
```

**Listing 28:** Klasa RegistryPersonalData