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

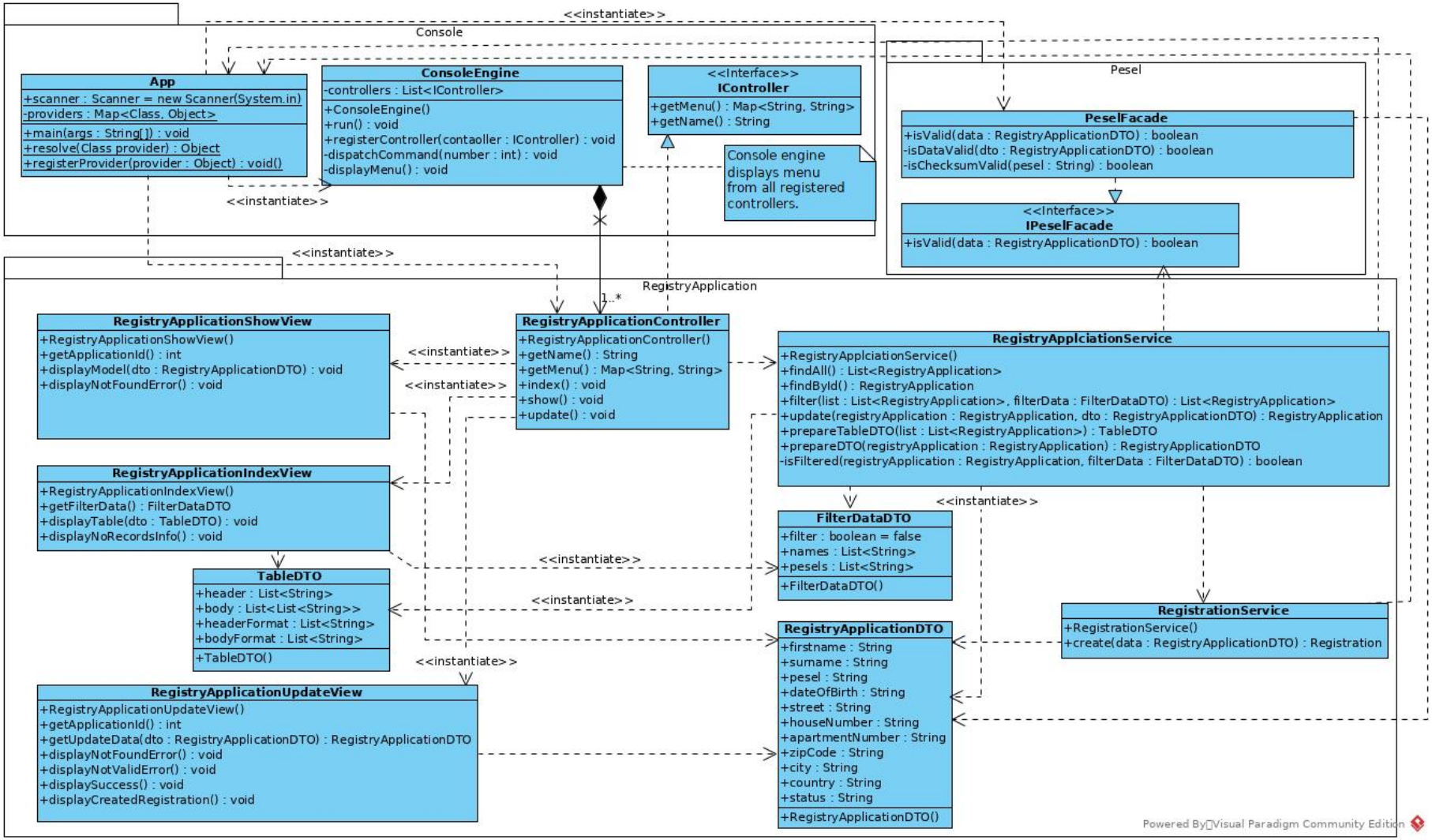
Rolę obiektów Flyweight pełnią klasy `RegistryAddressData` oraz `RegistryPersonalData`. Abstrakcyjnym klientem tych klas jest klasa `RegistrationBase`, z której dziedziczą klasy `RegistryApplication` oraz `Registration`.

4.2 Singleton

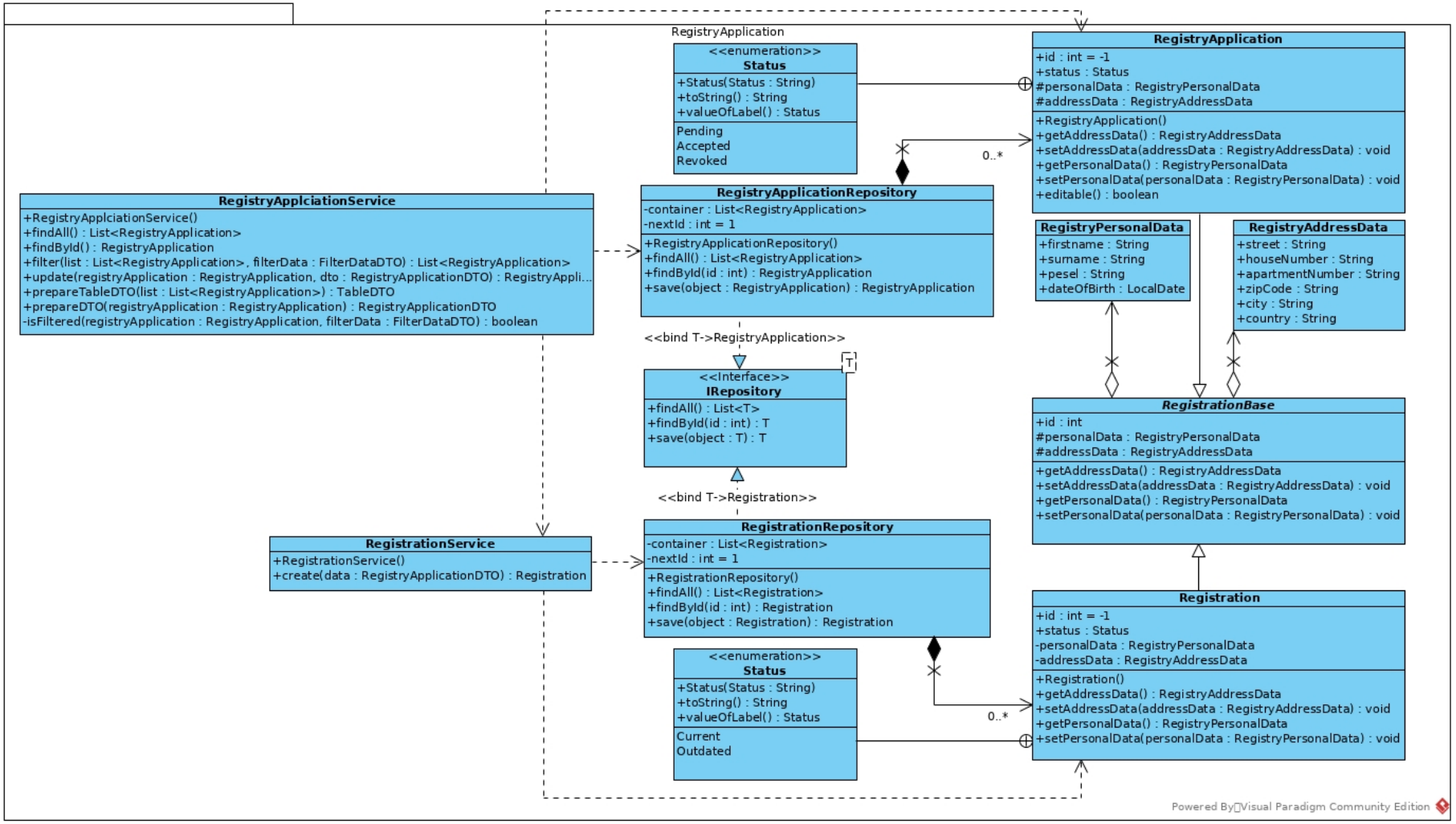
Serwisy są obiektami typu singleton posiadające tylko jedną instancję. Dostęp i zarządzanie nimi jest możliwy przez fasadę, którą implementuje klasa `App`. Zastosowanie tego wzorca ułatwi późniejsze testowanie i mockowanie implementacji serwisów.

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.



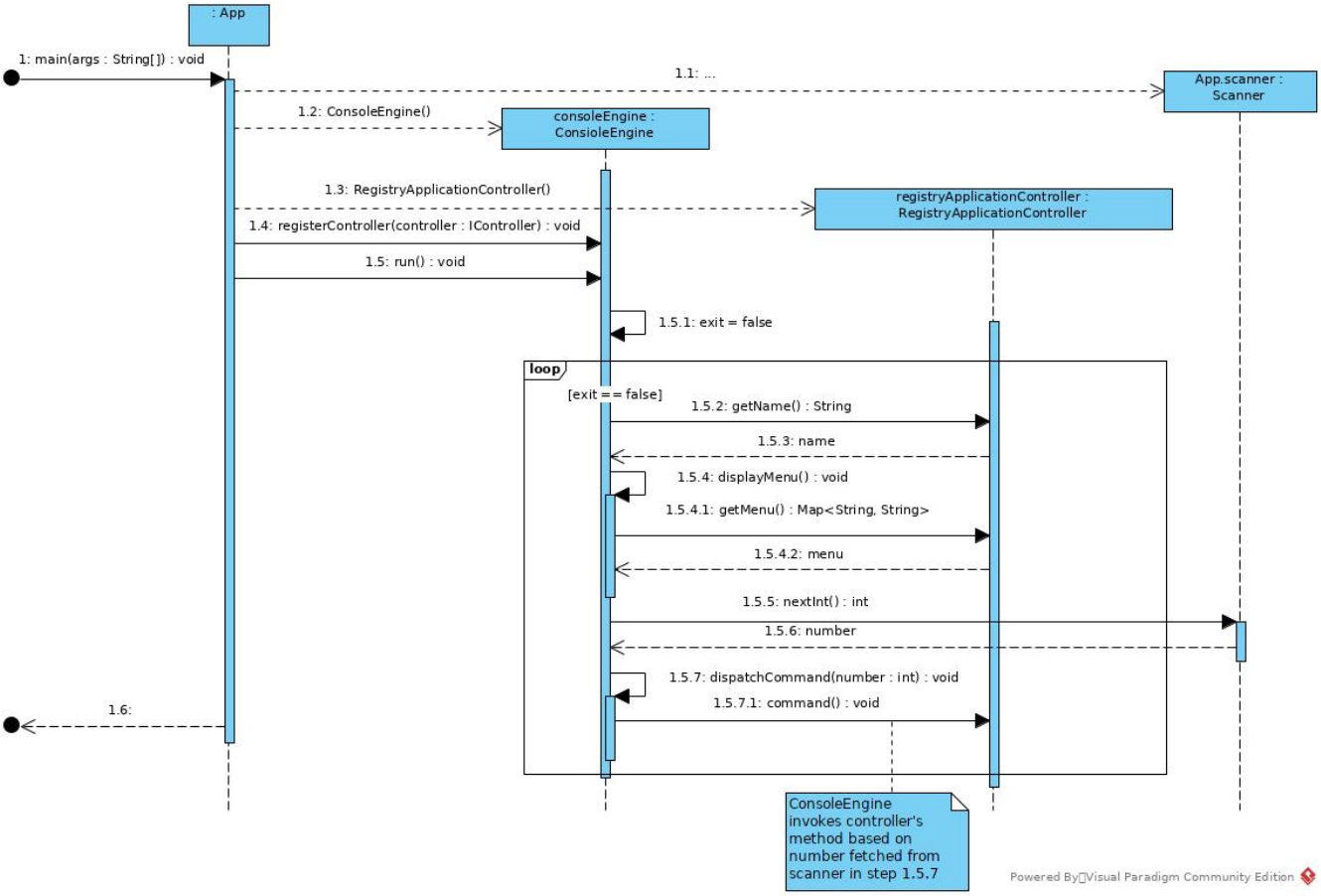
Rysunek 1: Diagram klas - widoki, kontrolery i serwisy.



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

5 Diagramy sekwencji

5.1 Główna pętla sterowania



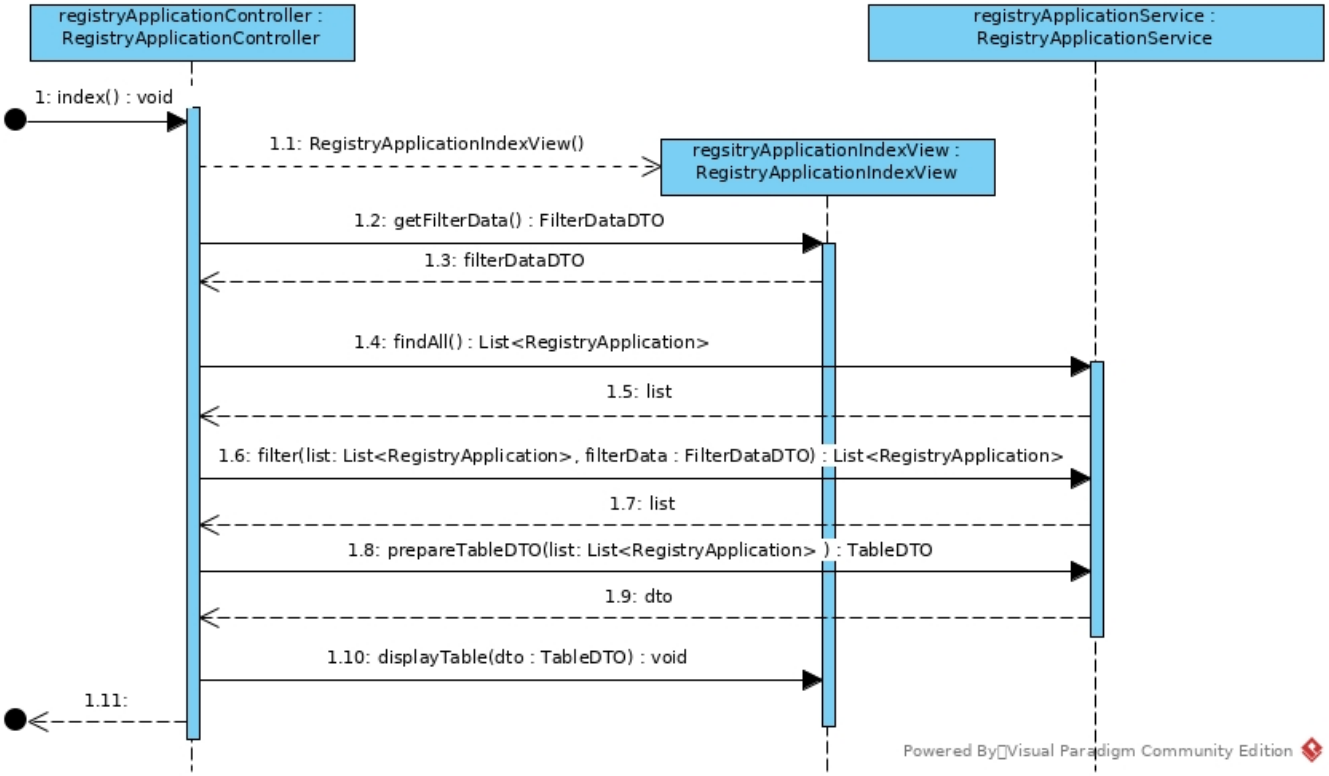
Rysunek 3: Diagram sekwencji - główna pętla przepływu sterowania.

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

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

Listing 1: Metoda main klasy App

5.2 Wyświetlanie wniosków

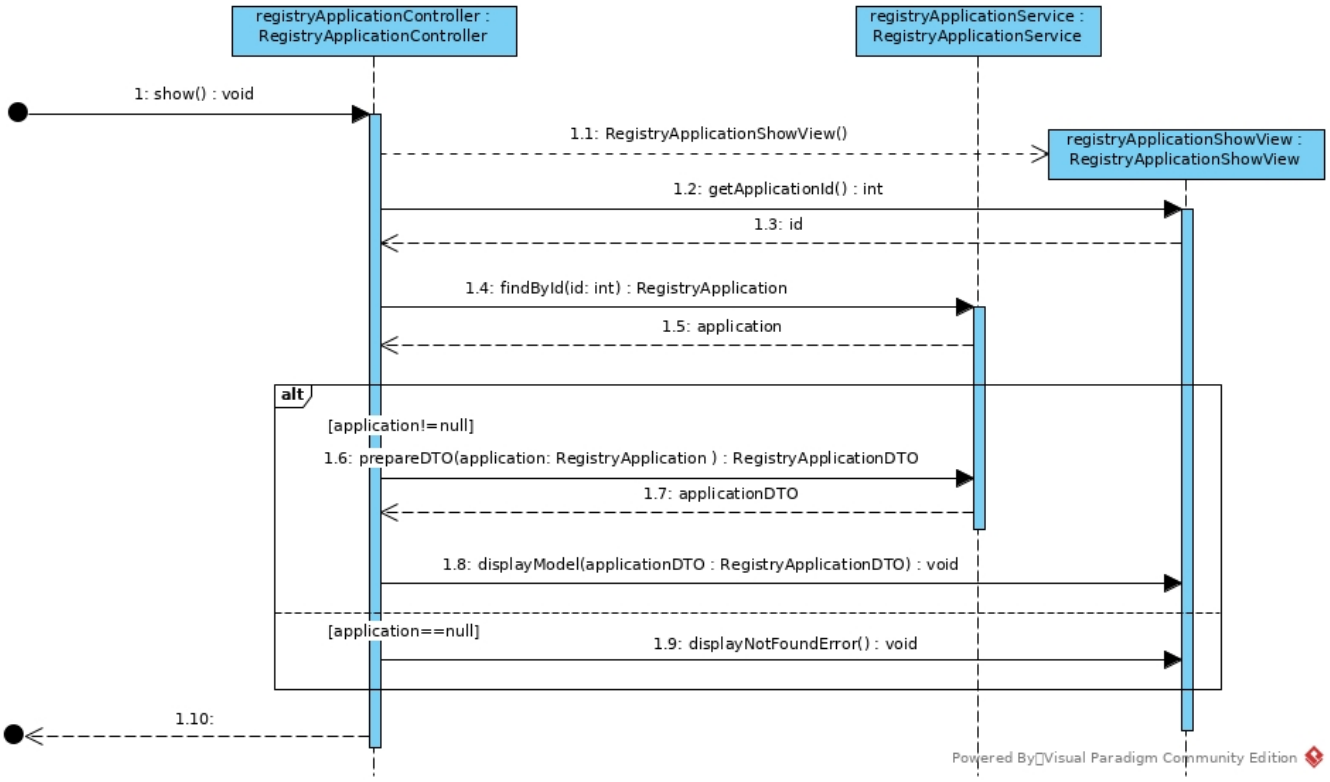


Rysunek 4: Diagram sekwencji - wyświetlanie wniosków.

```
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

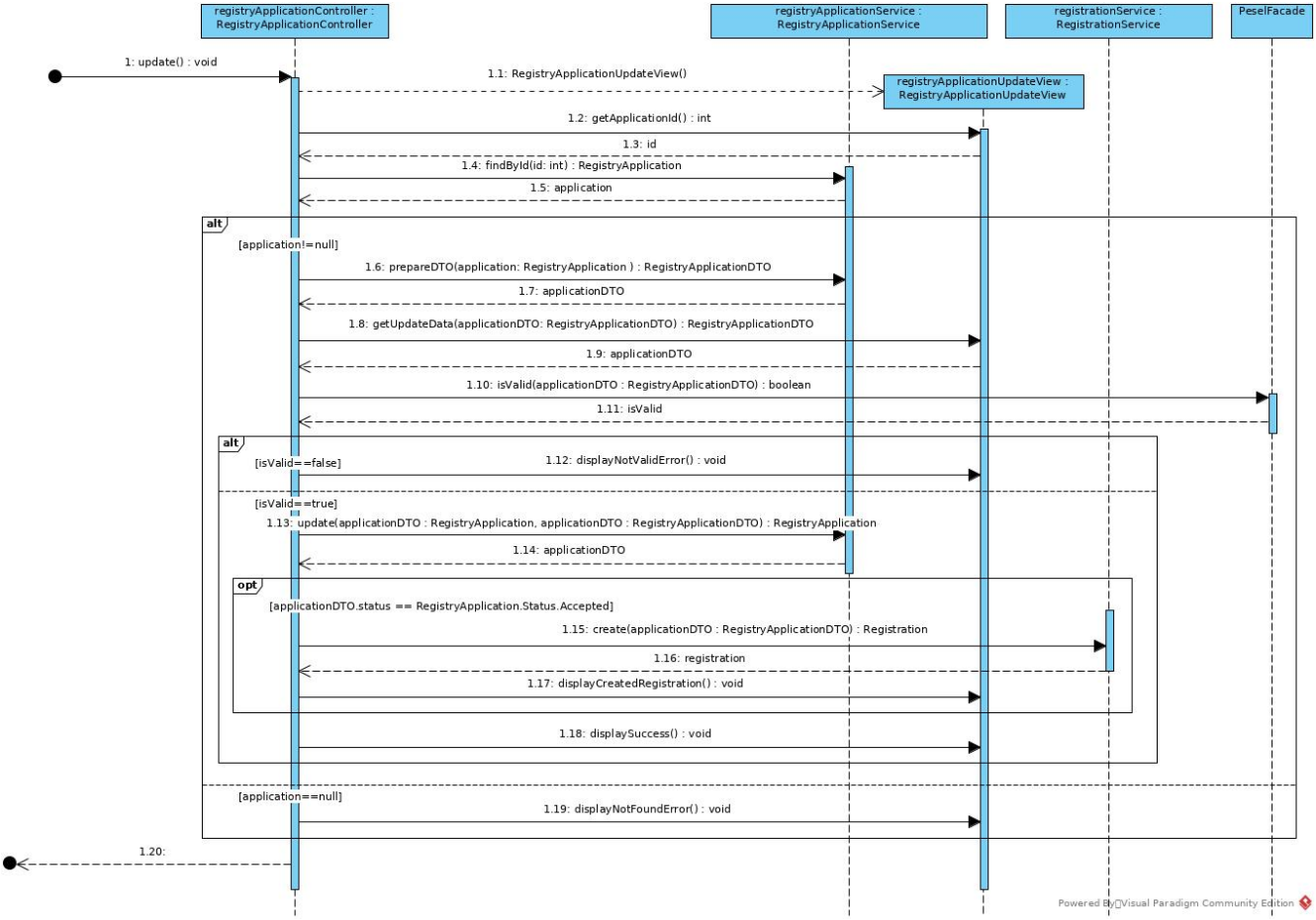


Rysunek 5: Diagram sekwencji - wyświetlanie pojedynczego wniosku.

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

Listing 3: Metoda show klasy RegistryApplicationController

5.4 Edycja danych wniosku



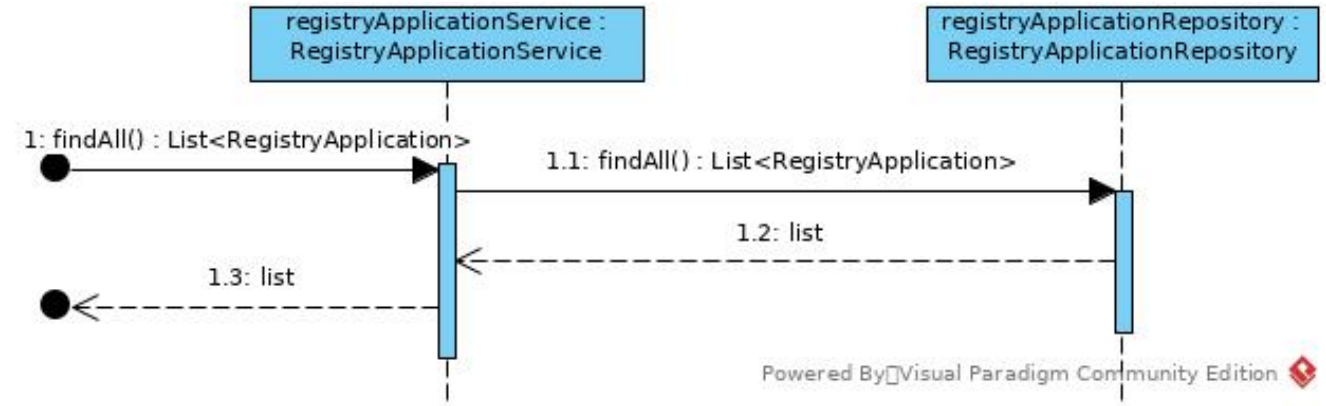
Rysunek 6: Diagram sekwencji - edycja danych wniosku.

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

```
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

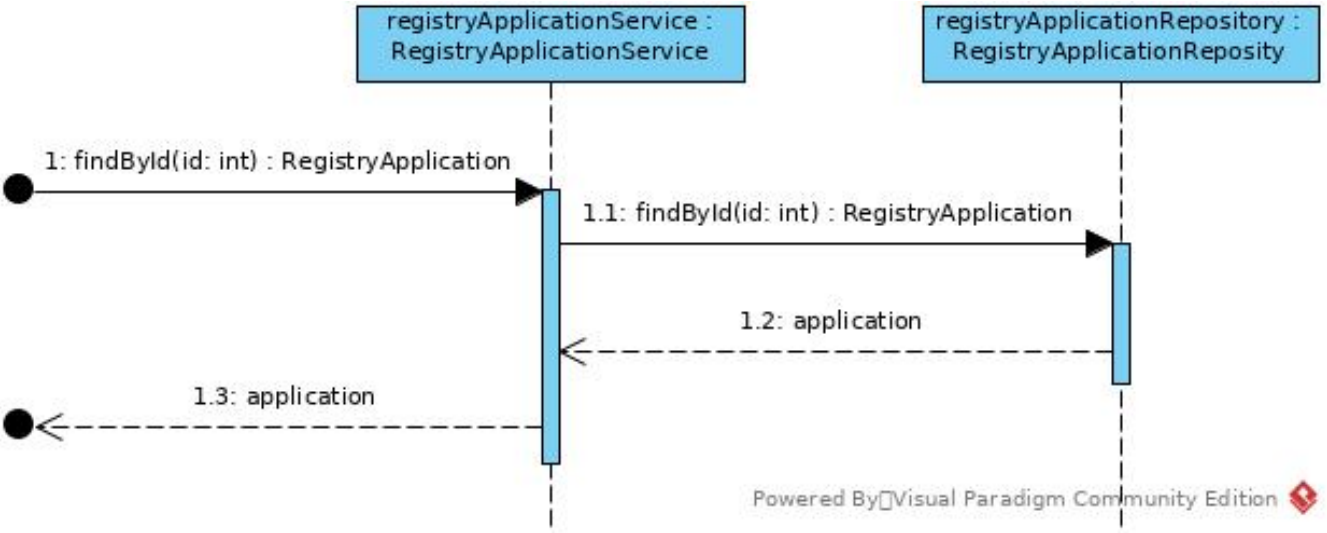
5.5 Metody klasy RegistryApplicationService



Rysunek 7: Diagram sekwencji - metoda findAll klasy RegistryApplicationService.

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

Listing 5: Metoda findAll klasy RegistryApplicationService

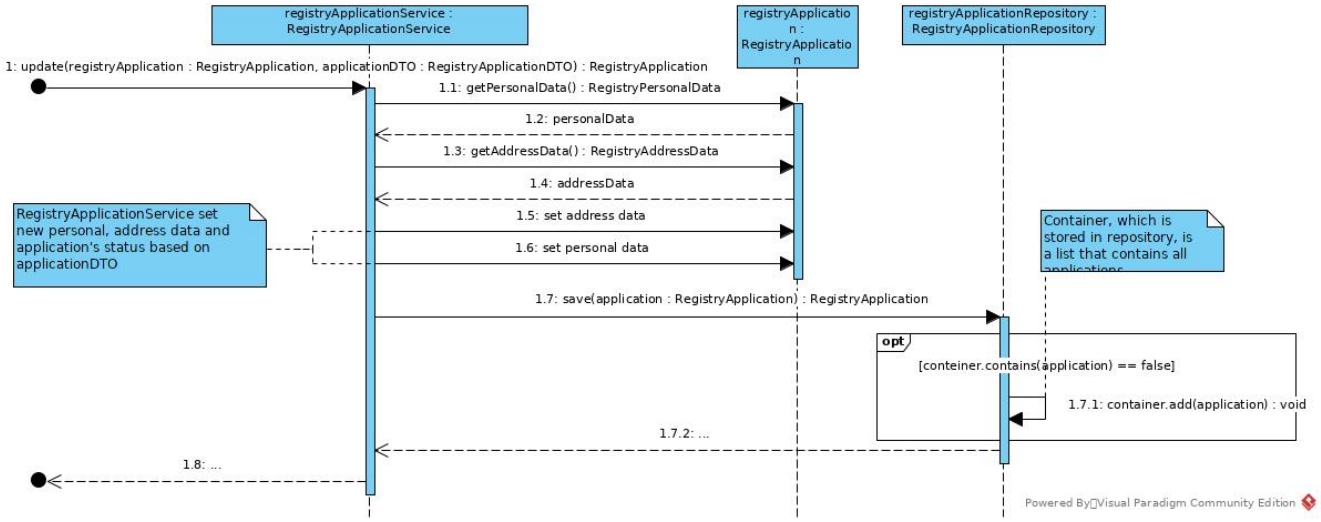


Rysunek 8: Diagram sekwencji - metoda findById klasy RegistryApplicationService.

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

```
                .resolve(RegistryApplicationRepository.class);
        return repository.findById(id);
    }
```

Listing 6: Metoda findById klasy RegistryApplicationService

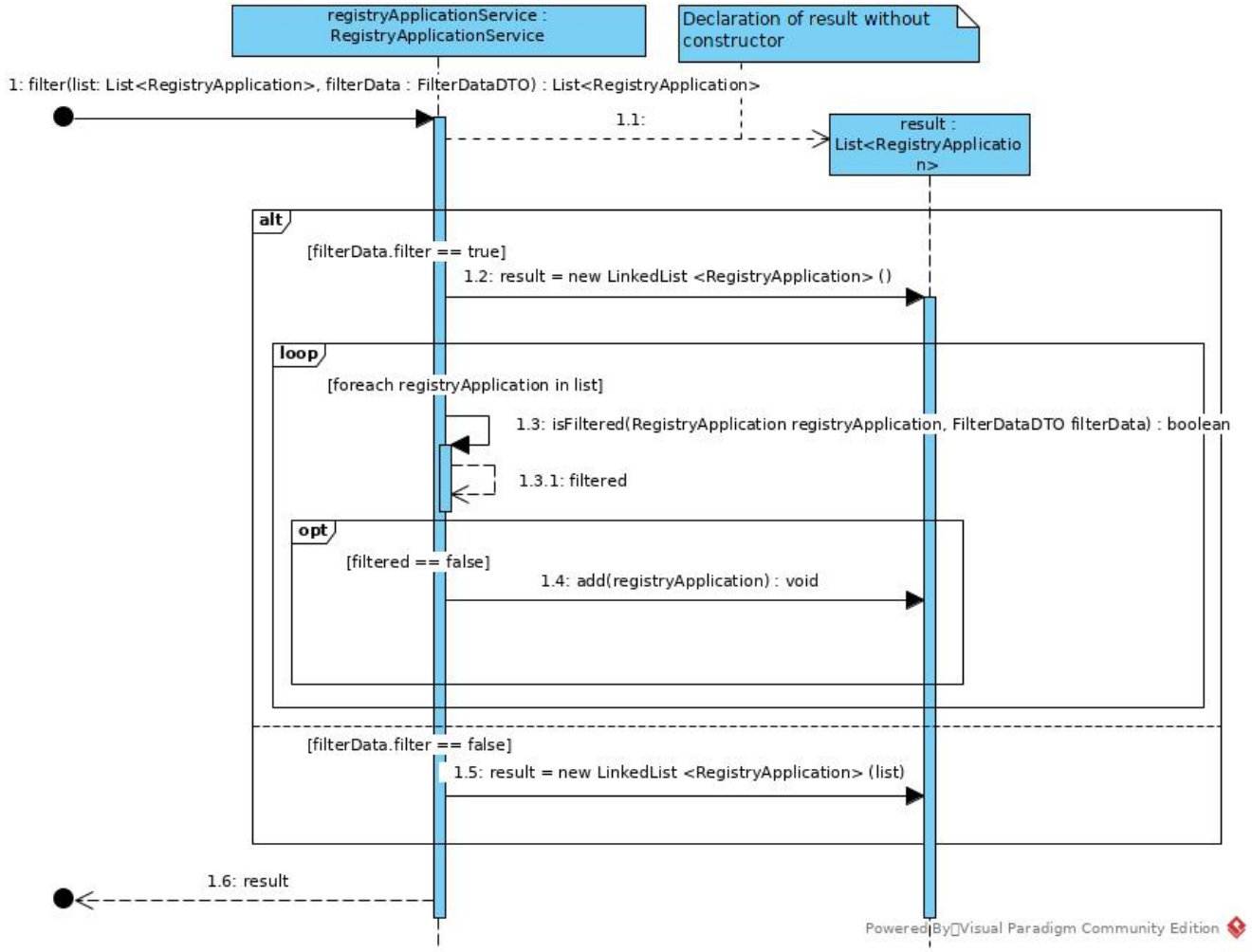


Rysunek 9: Diagram sekwencji - metoda filter klasy RegistryApplicationService.

```

    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);
```

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

Listing 7: Metoda update klasy RegistryApplicationService

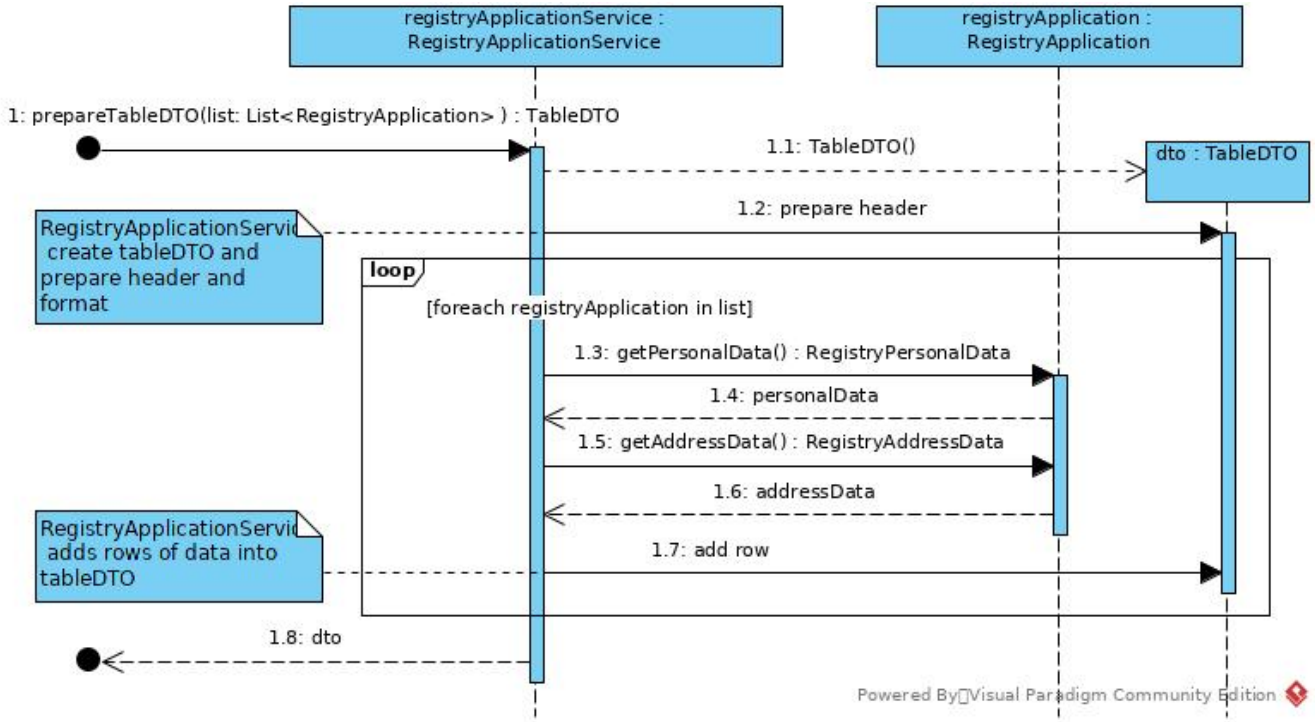


Rysunek 10: Diagram sekwencji - metoda filter klasy RegistryApplicationService.

```
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);
```

```
                if (!filtered) {
                    result.add(registryApplication);
                }
            }
        } else {
            result = new LinkedList<RegistryApplication>(list);
        }
        return result;
    }

    private boolean isFiltered(RegistryApplication registryApplication,
FilterDataDTO filterData) {
        if (filterData.names.stream().anyMatch(name -> {
            return registryApplication.getPersonalData().firstname.toLowerCase
() == name.toLowerCase()
                    || registryApplication.getPersonalData().surname.
toLowerCase() == name.toLowerCase();
        })) {
            return false;
        }
        if (filterData.pesels.stream().anyMatch(pesel -> {
            return registryApplication.getPersonalData().pesel.toLowerCase() ==
 pesel.toLowerCase();
        })) {
            return false;
        }
        return true;
    }
```

Listing 8: Metoda filter klasy RegistryApplicationService

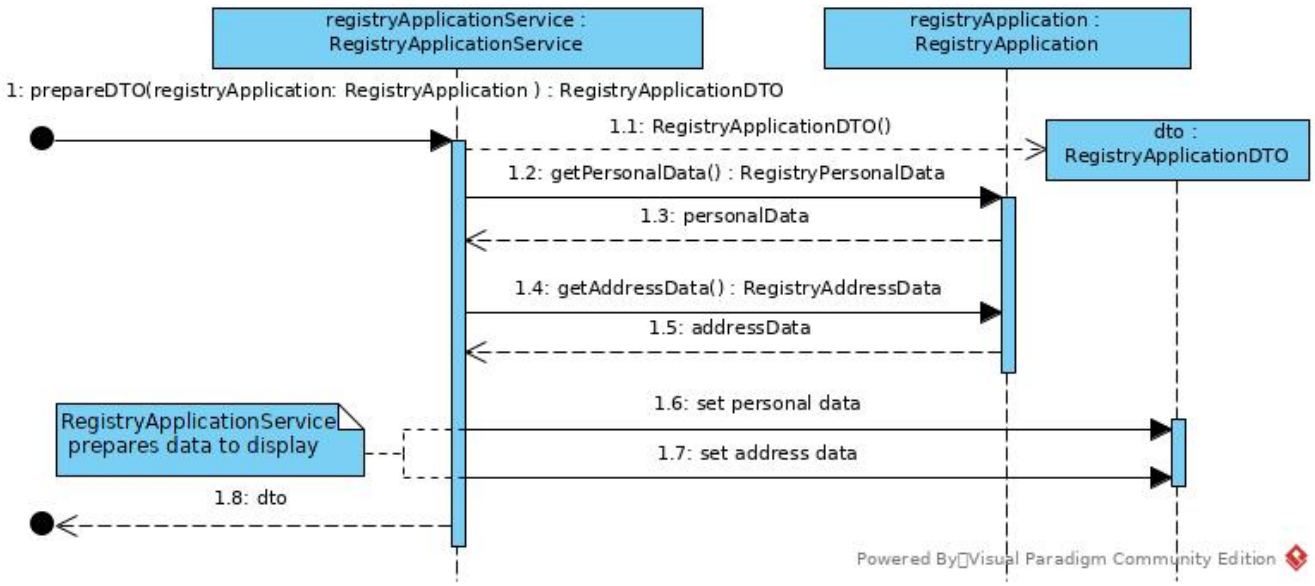


Rysunek 11: Diagram sekwencji - metoda prepareTableDTO klasy RegistryApplicationService.

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

Listing 9: Metoda prepareTableDTO klasy RegistryApplicationService



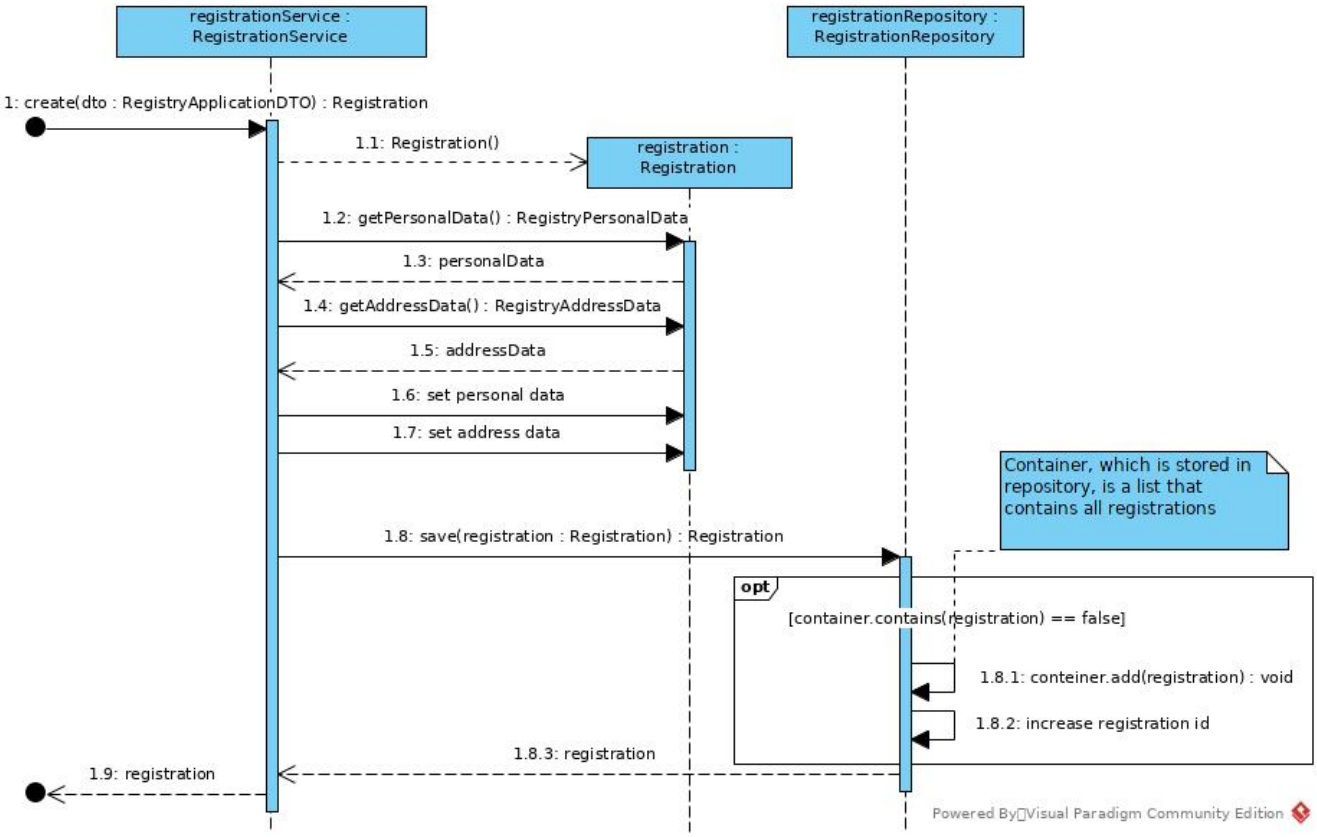
Rysunek 12: Diagram sekwencji - metoda prepareDTO klasy RegistryApplicationService.

```
1     public RegistryApplicationDTO prepareDTO(RegistryApplication
    registryApplication) {
2         RegistryPersonalData personal = registryApplication.getPersonalData();
3         RegistryAddressData address = registryApplication.getAddressData();
4
5         RegistryApplicationDTO dto = new RegistryApplicationDTO();
6         dto.id = registryApplication.id;
7         dto.firstname = personal.firstname;
8         dto.surname = personal.surname;
9         dto.pesel = personal.pesel;
10        dto.dateOfBirth = personal.dateOfBirth.toString();
11        dto.apartmentNumber = address.apartmentNumber;
12        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.
    resolve(RegistrationRepository.class);
19
20        return repository.save(registration);
21    }
```

Listing 11: Metoda create klasy RegistrationService

6 Prezentacja działania aplikacji

```
### Menu:
[0] Wyjscie
--- Wnioski meldunkowe
[1] Wyswietl wnioski
[2] Wyswietl dane wniosku
[3] Edytuj wniosek
1
Wlaczyc filtrowanie? [y|n]: n

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

Rysunek 14: Wyświetlanie wniosków bez filtrowania.

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

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

Rysunek 15: Wyświetlanie wniosku.

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

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

Operacja przeprowadzona pomyslnie!
```

Rysunek 16: Edycja danych wniosku.

7 Kod źródłowy aplikacji

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

```
public class App {

    public static Scanner scanner = new Scanner(System.in);
    private static HashMap<Class<? extends Object>, Object> providers = new
   HashMap<Class<? extends Object>, Object>();

    public static void main(String[] args) {
        RegistryApplicationRepository registryApplicationRepository = new
   RegistryApplicationRepository();

        /**
         * Data seed
         */
        RegistryApplication registryApplication = new RegistryApplication();
        registryApplication.getPersonalData().dateOfBirth = LocalDate.of(1990,
   01, 01);
        registryApplication.getPersonalData().firstname = "Damian";
        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());
        App.registerProvider(new PeselFacade());

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

```
        engine.run();
    }

    public static Object resolve(Class<? extends Object> provider) {
        return App.providers.get(provider);
    }

    public static void registerProvider(Object provider) {
        App.providers.put(provider.getClass(), provider);
    }
}
```

Listing 12: Klasa App

```
public interface IController {

  public Map<String, String> getMenu();

  public String getName();
}
```

Listing 13: Interface IController

```
public class ConsoleEngine {

  private LinkedList<IController> controllers = new LinkedList<>();

  public void run() {
    boolean exit = false;
    int input = 0;
    while (!exit) {
      displayMenu();
      input = App.scanner.nextInt();
      if (input == 0) {
        exit = true;
      } else {
        dispatchCommand(input);
      }
    }
```

```
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 -
     commands].toString());
31          method.invoke(iController);
32        } catch (IllegalAccessException | IllegalArgumentException |
     InvocationTargetException
33            | NoSuchMethodException e) {
34          e.printStackTrace();
35        }
36        return;
37      }
38      commands += commandCount;
39    }
40  }
41
42  private void displayMenu() {
43    int option = 1;
44    System.out.println("\n### Menu:");
45    System.out.println("[0] Wyjscie");
46    for (IController iController : controllers) {
47      System.out.println("--- " + iController.getName());
48      for (String name : iController.getMenu().keySet()) {
49        System.out.println("[" + String.valueOf(option) + "] " + iController.
     getMenu().get(name));
50        option = option + 1;
51      }
```

```
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 isDataValid(RegistryApplicationDTO dto) {
23         /**
24          * Validation hidden behind facade. Connection to PESEL system.
25          */
```

```
        return true;
    }

    public boolean isValid(RegistryApplicationDTO dto) {
        if (!isChecksumValid(dto.pesel))
            return false;
        return isDataValid(dto);
    }

}
```

Listing 16: Klasa PeselFacade

```
public interface IRepository<T> {

    public List<T> findAll();

    public T findById(int id);

    public T save(T object);
}
```

Listing 17: Interface IRepository

```
public class RegistrationRepository implements IRepository<Registration> {

    private int nextId = 1;
    private LinkedList<Registration> container = new LinkedList<>();

    @Override
    public List<Registration> findAll() {
        return container;
    }

    @Override
    public Registration findById(int id) {
        return container.stream().filter(o -> o.id == id).findAny().orElse(null
    );
    }
```

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

```
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;
```

```
  protected RegistryAddressData addressData = new RegistryAddressData();
  protected RegistryPersonalData personalData = new RegistryPersonalData();

  /**
   * @return the addressData
   */
  public RegistryAddressData getAddressData() {
    return addressData;
  }

  /**
   * @return the personalData
   */
  public RegistryPersonalData getPersonalData() {
    return personalData;
  }
}
```

Listing 24: Klasa RegistrationBase

```
public class Registration extends RegistrationBase {
  public Status status = Status.Current;

  public enum Status {
    Current("Obecny"), Outdated("Przeszly");

    private String status;

    Status(String status) {
      this.status = status;
    }

    @Override
    public String toString() {
      return status;
    }
  }
```

```
}
```

Listing 25: Klasa Registration

```
public class RegistryApplication extends RegistrationBase {
  public Status status = Status.Pending;

  public enum Status {
    Pending("Oczekujacy"), Accepted("Zaakceptowany"), Revoked("Odrzucony");

    private String status;

    Status(String status) {
      this.status = status;
    }

    @Override
    public String toString() {
      return status;
    }

    public static Status valueOfLabel(String label) {
      for (Status e : values()) {
        if (e.status.equals(label)) {
          return e;
        }
      }
      return null;
    }
  }
}
```

Listing 26: Klasa RegistryApplication

```
public class RegistryAddressData {

    public String street = "";
    public String houseNumber = "";
    public String apartmentNumber = "";
```

```
    public String zipCode = "";
    public String city = "";
    public String country = "";
}
```

Listing 27: Klasa RegistryAddressData

```
public class RegistryPersonalData {

    public String firstname = "";
    public String surname = "";
    public String pesel = "";
    public LocalDate dateOfBirth = LocalDate.of(1970, 01, 01);
}
```

Listing 28: Klasa RegistryPersonalData