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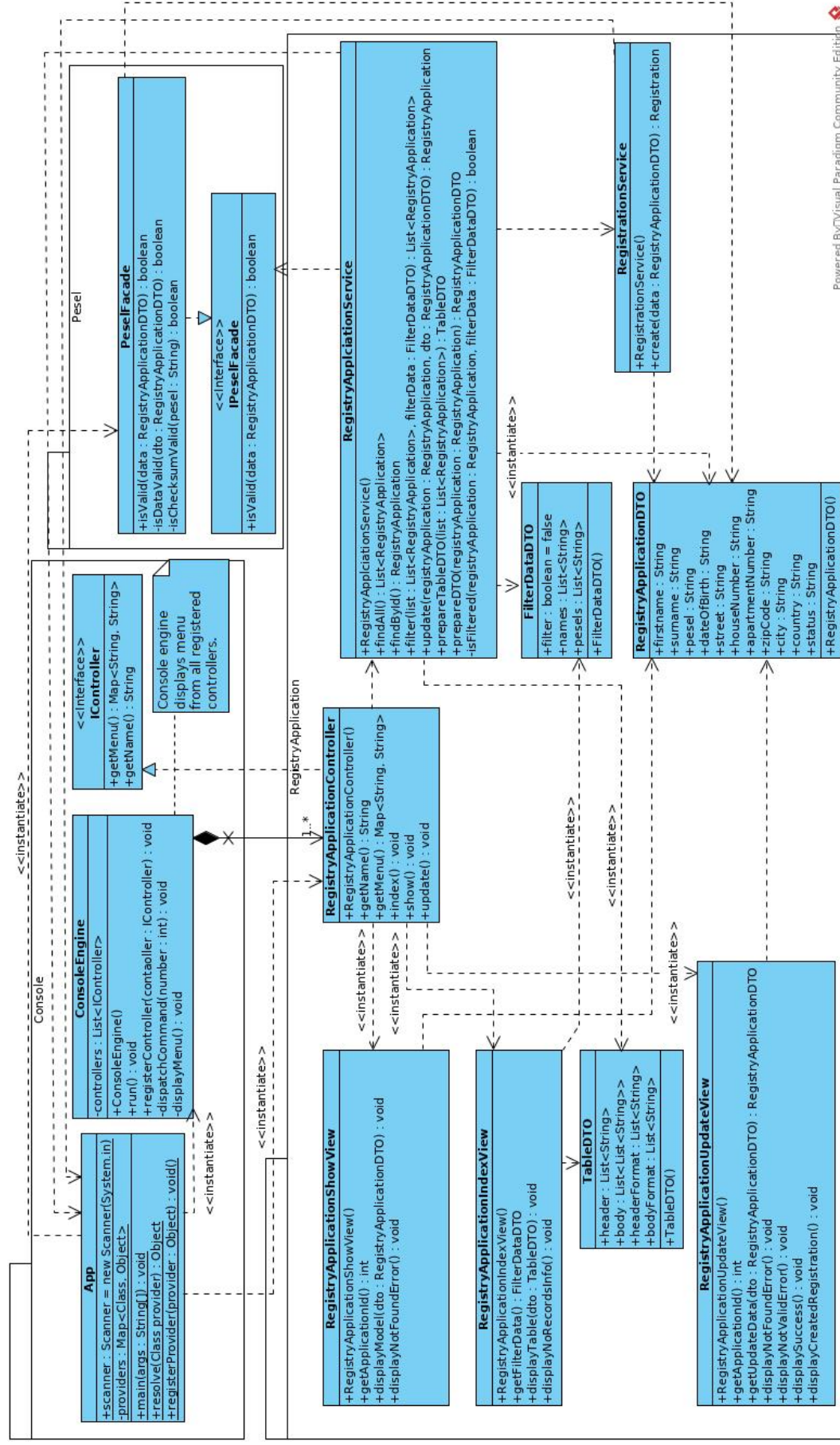
Rysunek 1: Diagram klas - widoki, kontrolery i serwisy.

5 Diagramy sekwencji

5.1 Główna pętla sterowania
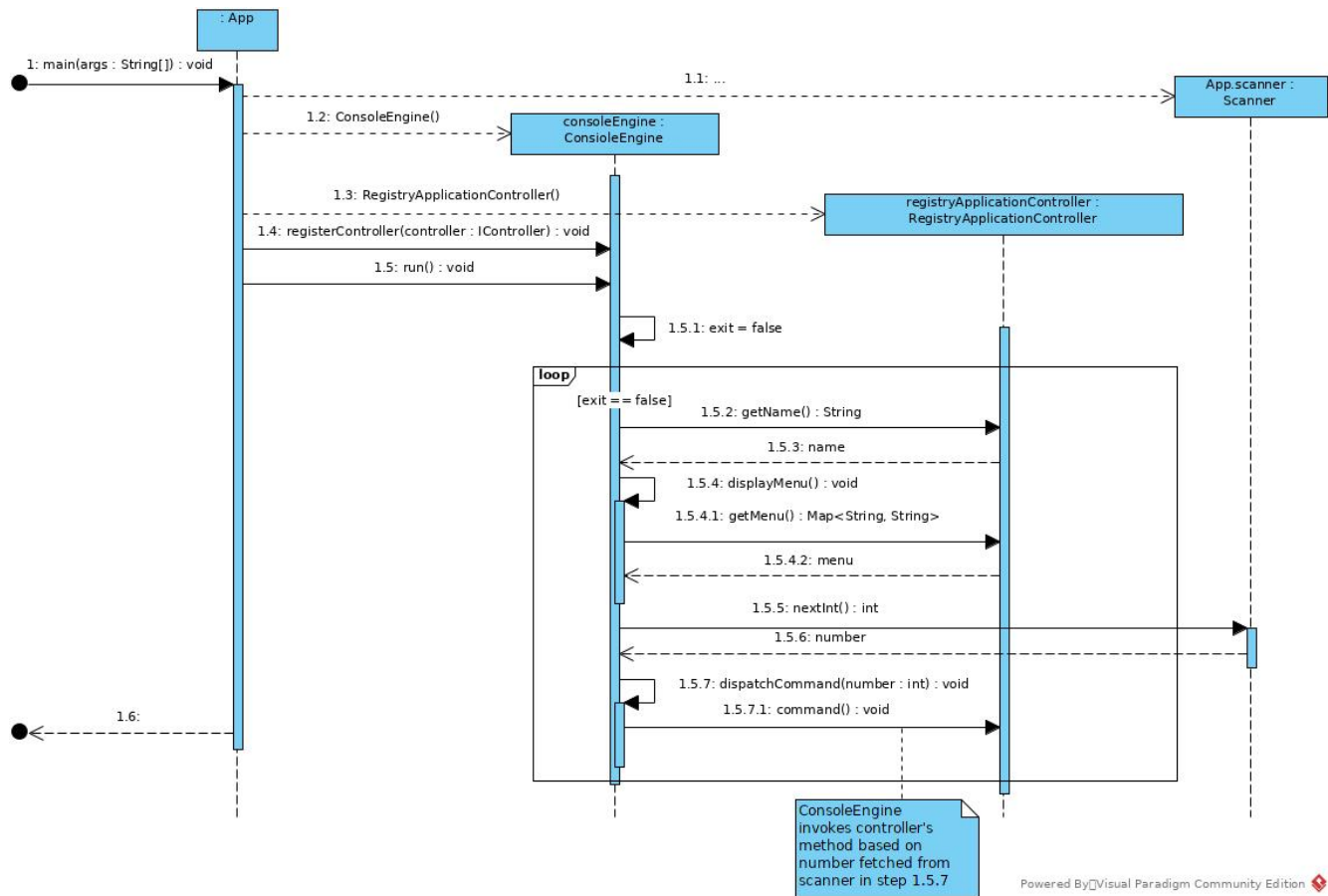
Rysunek 3: Diagram sekwencji - główna pętla przepływu sterowania.

```java
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();
```

```
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
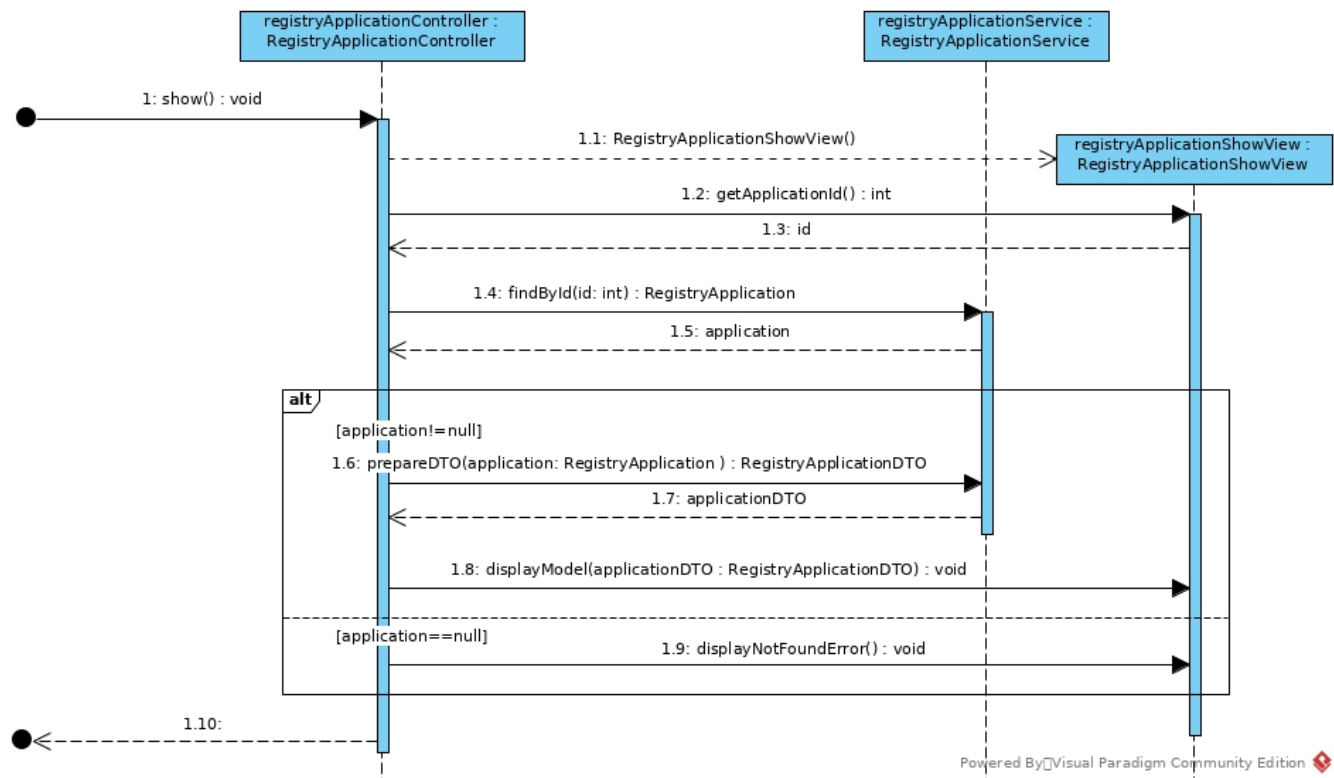
5.2 Wyświetlanie wniosków
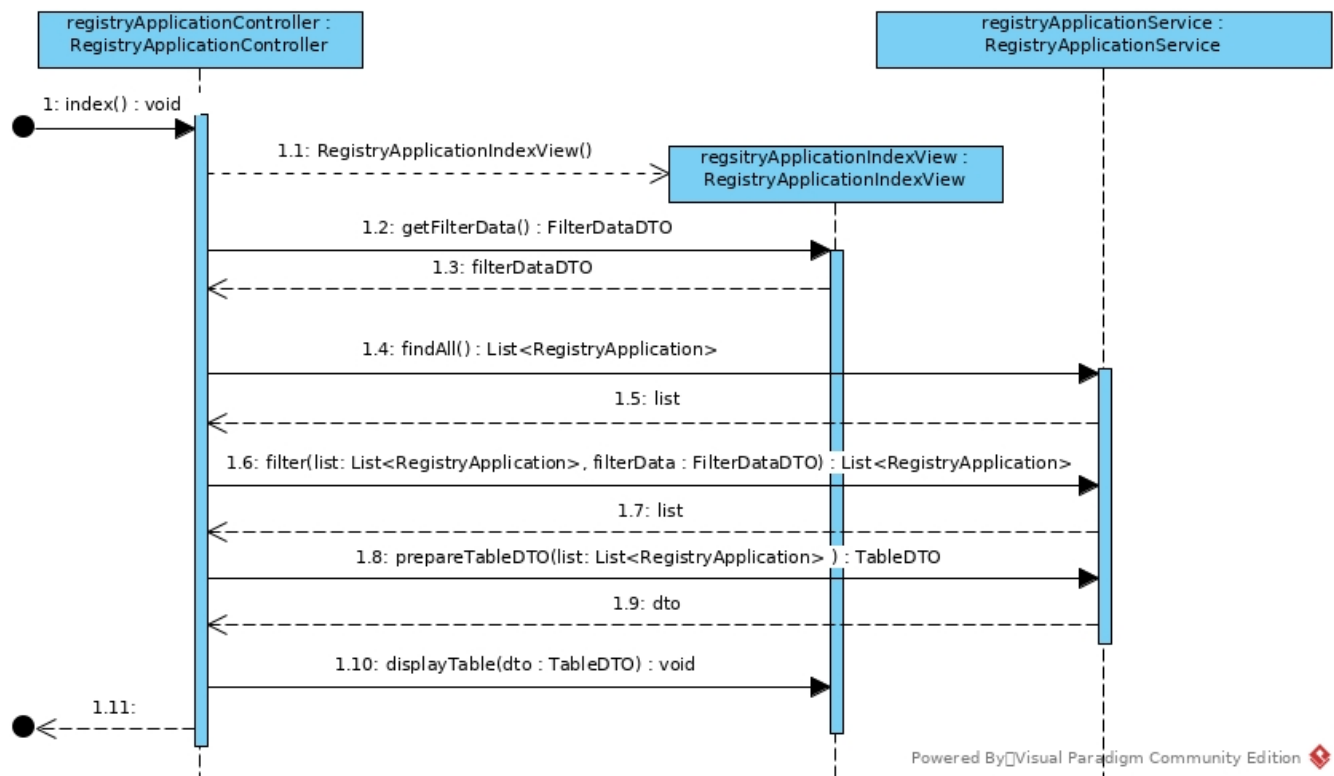
Rysunek 4: Diagram sekwencji - wyświetlanie wniosków.

```java
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);
```

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
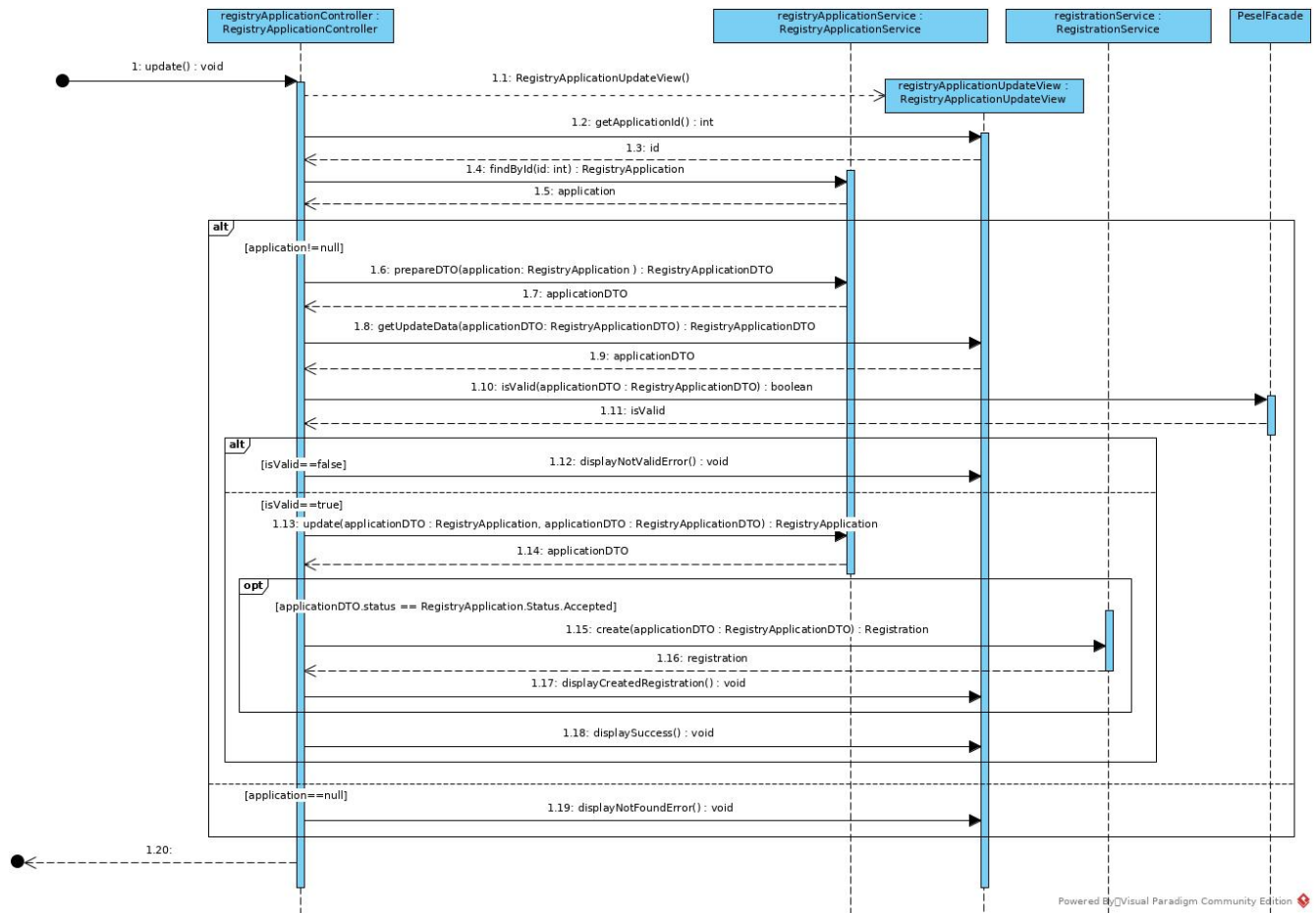
Rysunek 6: Diagram sekwencji - edycja danych wniosku.

```
public void update() {
  RegistryApplicationUpdateView view = new RegistryApplicationUpdateView();
  RegistryApplicationService registryApplicationService = (
  RegistryApplicationService) App
      .resolve(RegistryApplicationService.class);
  RegistrationService registrationService = (RegistrationService) App.resolve
  (RegistrationService.class);
  IPeselFacade peselFacade = (IPeselFacade) App.resolve(PeselFacade.class);
  int id = view.getApplicationId();
  RegistryApplication registryApplication = registryApplicationService.
  findById(id);
  if (registryApplication == null) {
    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
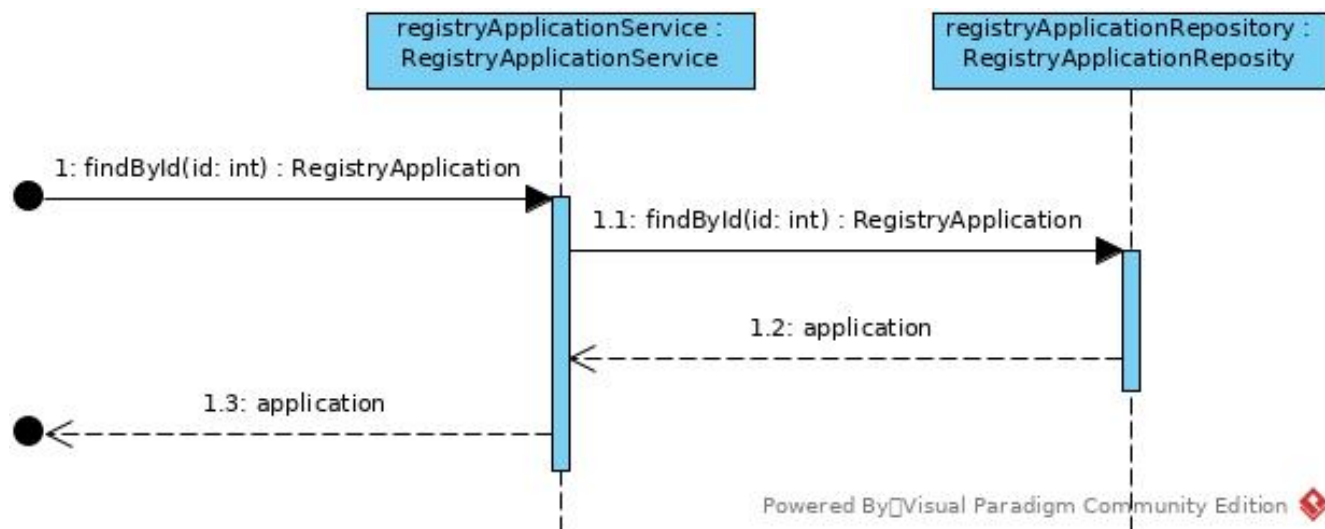
5.5 Metody klasy RegistryApplicationService
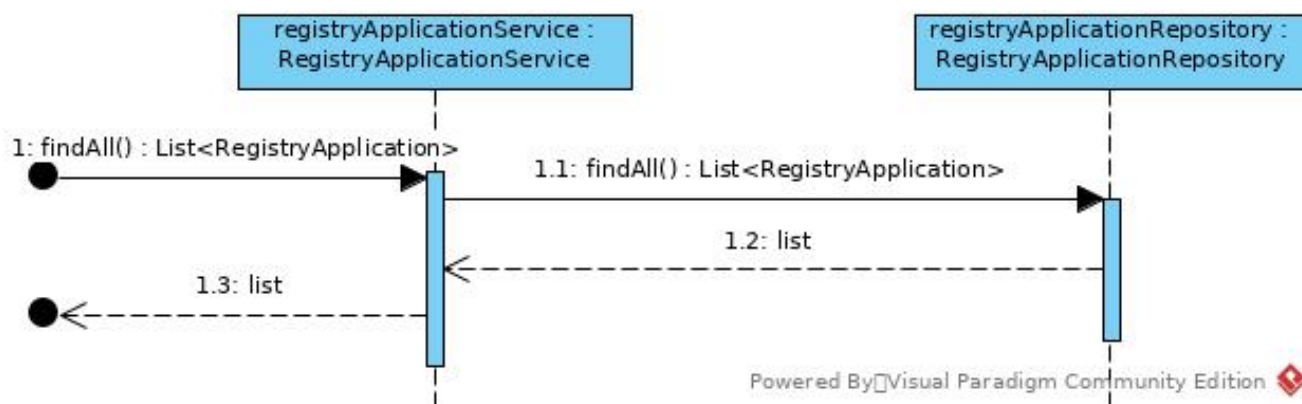
Rysunek 7: Diagram sekwencji - metoda findAll klasy RegistryApplicationService.

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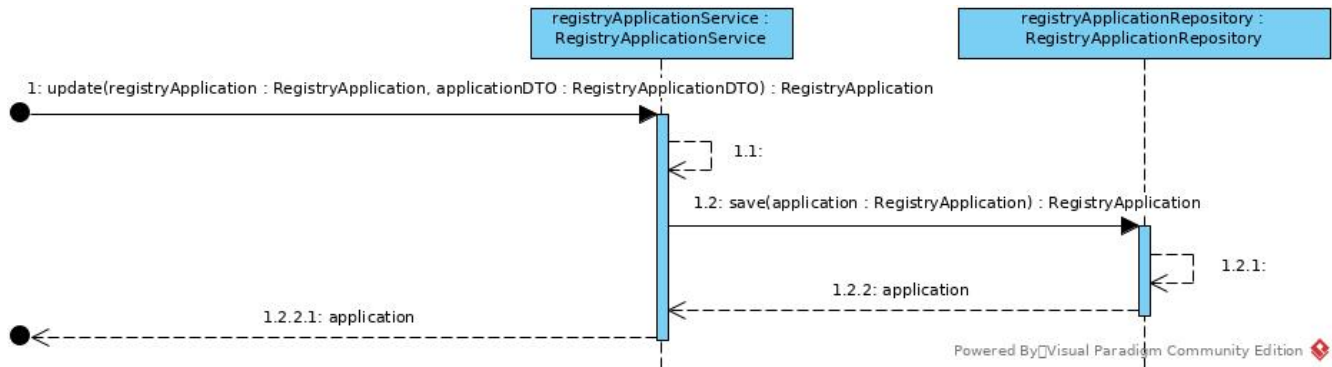
```
1    public RegistryApplication findById(int id) {
2        RegistryApplicationRepository repository = (
     RegistryApplicationRepository) App
```

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

```
20        return repository.save(registryApplication);
21    }
```

Listing 7: Metoda update klasy RegistryApplicationService

tutaj uzupełnić o brakujące

Łukaszu Rękawiczko zrób to ładnie i pięknie

5.6 Metody klasy RegistrationService

tutaj uzupełnić

Łukaszu Rękawiczko zrób to ładnie i pięknie

6 Kod źródłowy aplikacji

```
1 package populationRegistry;
2
3 import java.time.LocalDate;
4 import java.util.HashMap;
5 import java.util.Scanner;
6
7 import populationRegistry.console.ConsoleEngine;
8 import populationRegistry.pesel.PeselFacade;
9 import populationRegistry.registryApplication.controllers.
     RegistryApplicationController;
10 import populationRegistry.registryApplication.models.RegistryApplication;
11 import populationRegistry.registryApplication.repositories.
     RegistrationRepository;
12 import populationRegistry.registryApplication.repositories.
     RegistryApplicationRepository;
13 import populationRegistry.registryApplication.services.IPeselFacade;
14 import populationRegistry.registryApplication.services.RegistrationService;
15 import populationRegistry.registryApplication.services.
     RegistryApplicationService;
16
17 public class App {
18
19    public static Scanner scanner = new Scanner(System.in);
```

```java
20    private static HashMap<Class<? extends Object>, Object> providers = new
      HashMap<Class<? extends Object>, Object>();
21
22    public static void main(String[] args) {
23        RegistryApplicationRepository registryApplicationRepository = new
      RegistryApplicationRepository();
24
25        /**
26         * Data seed
27         */
28        RegistryApplication registryApplication = new RegistryApplication();
29        registryApplication.getPersonalData().dateOfBirth = LocalDate.of(1990,
      01, 01);
30        registryApplication.getPersonalData().firstname = "Damian";
31        registryApplication.getPersonalData().surname = "Koper";
32        registryApplication.getPersonalData().pesel = "72060319389";
33        registryApplication.getAddressData().apartmentNumber = "20";
34        registryApplication.getAddressData().houseNumber = "10";
35        registryApplication.getAddressData().street = "Marszalkowska";
36        registryApplication.getAddressData().zipCode = "00-043";
37        registryApplication.getAddressData().country = "Polska";
38        registryApplication.getAddressData().city = "Warszawa";
39        registryApplicationRepository.save(registryApplication);
40
41        App.registerProvider(new RegistryApplicationService());
42        App.registerProvider(registryApplicationRepository);
43        App.registerProvider(new RegistrationService());
44        App.registerProvider(new RegistrationRepository());
45        App.registerProvider(new PeselFacade());
46
47        ConsoleEngine engine = new ConsoleEngine();
48        engine.registerController(new RegistryApplicationController());
49        engine.run();
50    }
51
52    public static Object resolve(Class<? extends Object> provider) {
53        return App.providers.get(provider);
54    }
```

```
55
56     public static void registerProvider(Object provider) {
57         App.providers.put(provider.getClass(), provider);
58     }
59 }
```

Listing 8: Klasa App

```
1 package populationRegistry.console;
2
3 import java.util.Map;
4
5 /**
6  * IController
7  */
8 public interface IController {
9
10   public Map<String, String> getMenu();
11
12   public String getName();
13 }
```

Listing 9: Interface IController

```
1 package populationRegistry.console;
2
3 import java.lang.reflect.InvocationTargetException;
4 import java.lang.reflect.Method;
5 import java.util.LinkedList;
6 import populationRegistry.App;
7
8 /**
9  * ConsoleEngine
10  */
11 public class ConsoleEngine {
12
13   private LinkedList<IController> controllers = new LinkedList<>();
14
15   public void run() {
```

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

  public void registerController(IController controller) {
    controllers.add(controller);
  }

  private void dispatchCommand(int number) {
    int commands = 1;
    for (IController iController : controllers) {
      int commandCount = iController.getMenu().size();
      if (number <= commandCount - commands + 1) {
        try {
          Method method = iController.getClass()
              .getMethod(iController.getMenu().keySet().toArray()[number -
  commands].toString());
          method.invoke(iController);
        } catch (IllegalAccessException | IllegalArgumentException |
  InvocationTargetException
            | NoSuchMethodException e) {
          e.printStackTrace();
        }
        return;
      }
      commands += commandCount;
    }
  }
```

17

```
52    private void displayMenu() {
53      int option = 1;
54      System.out.println("\n### Menu:");
55      System.out.println("[0] Wyjscie");
56      for (IController iController : controllers) {
57        System.out.println("--- " + iController.getName());
58        for (String name : iController.getMenu().keySet()) {
59          System.out.println("[" + String.valueOf(option) + "] " + iController.
      getMenu().get(name));
60          option = option + 1;
61        }
62      }
63    }
64  }
```

Listing 10: Klasa ConsoleEngine

```
1  package populationRegistry.registryApplication.services;
2
3  import populationRegistry.registryApplication.services.dto.
      RegistryApplicationDTO;
4
5  /**
6   * PeselFacade
7   */
8  public interface IPeselFacade {
9
10     public boolean isValid(RegistryApplicationDTO dto);
11
12 }
```

Listing 11: Interface IPeselFacade

```
1  package populationRegistry.pesel;
2
3  import java.util.ArrayList;
4
5  import populationRegistry.registryApplication.services.IPeselFacade;
6  import populationRegistry.registryApplication.services.dto.
      RegistryApplicationDTO;
```

```java
7
/**
 * PeselFacade
 */
public class PeselFacade implements IPeselFacade {
    private boolean isChecksumValid(String pesel) {
        String integers[] = pesel.split("");
        if (integers.length != 11) {
            return false;
        }
        ArrayList<Integer> values = new ArrayList<>();
        for (String string : integers) {
            values.add(Integer.parseInt(string));
        }
        int[] m = { 1, 3, 7, 9 };
        int sum = 0;
        for (int i = 0; i < values.size() - 1; i++) {
            sum += m[i % 4] * values.get(i);
        }
        sum += values.get(values.size() - 1);
        sum %= 10;

        return sum == 0;
    }

    private boolean isDataValid(RegistryApplicationDTO dto) {
        /**
         * Validation hidden behind facade. Connection to PESEL system.
         */
        return true;
    }

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

```
45 }
```

Listing 12: Klasa PeselFacade

```java
1  package populationRegistry.registryApplication.repositories;
2
3  import java.util.List;
4
5  /**
6   * IRepository
7   */
8  public interface IRepository<T> {
9
10     public List<T> findAll();
11
12     public T findById(int id);
13
14     public T save(T object);
15 }
```

Listing 13: Interface IRepository

```java
1  package populationRegistry.registryApplication.repositories;
2
3  import java.util.LinkedList;
4  import java.util.List;
5
6  import populationRegistry.registryApplication.models.Registration;
7
8  /**
9   * RegistryApplicationRepository
10  */
11 public class RegistrationRepository implements IRepository<Registration> {
12
13     private int nextId = 1;
14     private LinkedList<Registration> container = new LinkedList<>();
15
16     @Override
17     public List<Registration> findAll() {
```

```
18          return container;
19      }
20
21      @Override
22      public Registration findById(int id) {
23          return container.stream().filter(o -> o.id == id).findAny().orElse(null
    );
24      }
25
26      @Override
27      public Registration save(Registration object) {
28          if (!container.contains(object)) {
29              container.add(object);
30              object.id = nextId++;
31          }
32          return object;
33      }
34
35 }
```

Listing 14: Klasa RegistrationRepository

```
1 package populationRegistry.registryApplication.repositories;
2
3 import java.util.LinkedList;
4 import java.util.List;
5
6 import populationRegistry.registryApplication.models.RegistryApplication;
7
8 /**
9  * RegistryApplicationRepository
10  */
11 public class RegistryApplicationRepository implements IRepository<
    RegistryApplication> {
12
13      private int nextId = 1;
14      private LinkedList<RegistryApplication> container = new LinkedList<>();
15
16      @Override
```

```java
17    public List<RegistryApplication> findAll() {
18        return container;
19    }
20
21    @Override
22    public RegistryApplication findById(int id) {
23        return container.stream().filter(o -> o.id == id).findAny().orElse(null
   );
24    }
25
26    @Override
27    public RegistryApplication save(RegistryApplication object) {
28        if (!container.contains(object)) {
29            container.add(object);
30            object.id = nextId++;
31        }
32        return object;
33    }
34
35 }
```

Listing 15: Klasa RegistryApplicationRepository

```java
1  package populationRegistry.registryApplication.services;
2
3  import java.time.LocalDate;
4
5  import populationRegistry.App;
6  import populationRegistry.registryApplication.models.Registration;
7  import populationRegistry.registryApplication.models.RegistryAddressData;
8  import populationRegistry.registryApplication.models.RegistryPersonalData;
9  import populationRegistry.registryApplication.repositories.
     RegistrationRepository;
10 import populationRegistry.registryApplication.services.dto.
     RegistryApplicationDTO;
11
12 /**
13  * RegistrationService
14  */
```

```java
public class RegistrationService {

    public Registration create(RegistryApplicationDTO dto) {
        Registration registration = new Registration();
        RegistryPersonalData personal = registration.getPersonalData();
        RegistryAddressData address = registration.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);
        registration.status = Registration.Status.Current;

        RegistrationRepository repository = (RegistrationRepository) App.
    resolve(RegistrationRepository.class);

        return repository.save(registration);
    }

}
```

Listing 16: Klasa RegistrationService

```java
package populationRegistry.registryApplication.services;

import java.time.LocalDate;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

import populationRegistry.App;
import populationRegistry.registryApplication.models.RegistryAddressData;
import populationRegistry.registryApplication.models.RegistryApplication;
```

```java
import populationRegistry.registryApplication.models.RegistryPersonalData;
import populationRegistry.registryApplication.repositories.
    RegistryApplicationRepository;
import populationRegistry.registryApplication.services.dto.FilterDataDTO;
import populationRegistry.registryApplication.services.dto.
    RegistryApplicationDTO;
import populationRegistry.registryApplication.views.dto.TableDTO;

/**
 * RegistryApplicationService
 */
public class RegistryApplicationService {

    public List<RegistryApplication> findAll() {
        RegistryApplicationRepository repository = (
    RegistryApplicationRepository) App
                .resolve(RegistryApplicationRepository.class);
        return repository.findAll();
    }

    public RegistryApplication findById(int id) {
        RegistryApplicationRepository repository = (
    RegistryApplicationRepository) App
                .resolve(RegistryApplicationRepository.class);
        return repository.findById(id);
    }

    public List<RegistryApplication> filter(List<RegistryApplication> list,
    FilterDataDTO filterData) {
        List<RegistryApplication> result;
        if (filterData.filter) {
            result = new LinkedList<RegistryApplication>();
            for (RegistryApplication registryApplication : list) {
                if (!isFiltered(registryApplication, filterData)) {
                    result.add(registryApplication);
                }
            }
        } else {
```

```java
                    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;
    }

    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;
```

```
77          personal.dateOfBirth = LocalDate.parse(dto.dateOfBirth);

78

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

80

81          RegistryApplicationRepository repository = (
    RegistryApplicationRepository) App
82                  .resolve(RegistryApplicationRepository.class);
83          return repository.save(registryApplication);
84      }

85

86   public RegistryApplicationDTO prepareDTO(RegistryApplication
    registryApplication) {
87          RegistryPersonalData personal = registryApplication.getPersonalData();
88          RegistryAddressData address = registryApplication.getAddressData();

89

90          RegistryApplicationDTO dto = new RegistryApplicationDTO();
91          dto.id = registryApplication.id;
92          dto.firstname = personal.firstname;
93          dto.surname = personal.surname;
94          dto.pesel = personal.pesel;
95          dto.dateOfBirth = personal.dateOfBirth.toString();
96          dto.apartmentNumber = address.apartmentNumber;
97          dto.houseNumber = address.houseNumber;
98          dto.city = address.city;
99          dto.country = address.country;
100         dto.street = address.street;
101         dto.status = registryApplication.status.toString();
102         dto.zipCode = address.zipCode;

103

104         return dto;
105     }

106

107  public TableDTO prepareTableDTO(List<RegistryApplication> list) {
108         TableDTO dto = new TableDTO();
109         dto.header = new LinkedList<String>(
110                 Arrays.asList("Id", "Imie", "Nazwisko", "PESEL", "Wnioskowany
    adres zameldowania"));
```

```
111         LinkedList<String> format = new LinkedList<String>(Arrays.asList("%5s",
    "%15s", "%15s", "%11s", "%60s"));
112         dto.headerFormat = format;
113         dto.bodyFormat = format;
114         for (RegistryApplication registryApplication : list) {
115             LinkedList<String> row = new LinkedList<>();
116             row.add(String.valueOf(registryApplication.id));
117
118             RegistryPersonalData personal = registryApplication.getPersonalData
    ();
119             row.add(personal.firstname);
120             row.add(personal.surname);
121             row.add(personal.pesel);
122
123             RegistryAddressData address = registryApplication.getAddressData();
124             row.add(address.city + " ul." + address.street + " " + address.
    houseNumber + "/" + address.apartmentNumber
125                     + " " + address.zipCode + ", " + address.country);
126             dto.body.add(row);
127         }
128         return dto;
129     }
130 }
```

Listing 17: Klasa RegistryApplicationService

```
1 package populationRegistry.registryApplication.services.dto;
2
3 import java.util.LinkedList;
4
5 public class FilterDataDTO {
6
7   public boolean filter = false;
8   public LinkedList<String> names = new LinkedList<>();
9   public LinkedList<String> pesels = new LinkedList<>();
10 }
```

Listing 18: Klasa FilterDataDTO

```
1 package populationRegistry.registryApplication.services.dto;
```

```
2
3  public class RegistryApplicationDTO {
4
5    public int id;
6    public String firstname;
7    public String surname;
8    public String pesel;
9    public String dateOfBirth;
10   public String street;
11   public String houseNumber;
12   public String apartmentNumber;
13   public String zipCode;
14   public String city;
15   public String country;
16   public String status;
17 }
```

Listing 19: Klasa RegistryApplicationDTO

```
1  package populationRegistry.registryApplication.models;
2
3  /**
4   * RegistryApplicationData
5   */
6  public abstract class RegistrationBase {
7    public int id = -1;
8    protected RegistryAddressData addressData = new RegistryAddressData();
9    protected RegistryPersonalData personalData = new RegistryPersonalData();
10
11   /**
12    * @return the addressData
13    */
14   public RegistryAddressData getAddressData() {
15     return addressData;
16   }
17
18   /**
19    * @return the personalData
20    */
```

```java
21   public RegistryPersonalData getPersonalData() {
22     return personalData;
23   }
24 }
```

Listing 20: Klasa RegistrationBase

```java
1 package populationRegistry.registryApplication.models;
2
3 /**
4  * Registration
5  */
6 public class Registration extends RegistrationBase {
7   public Status status = Status.Current;
8
9   public enum Status {
10     Current("Obecny"), Outdated("Przeszly");
11
12     private String status;
13
14     Status(String status) {
15       this.status = status;
16     }
17
18     @Override
19     public String toString() {
20       return status;
21     }
22   }
23 }
```

Listing 21: Klasa Registration

```java
1 package populationRegistry.registryApplication.models;
2
3 /**
4  * RegistryApplication
5  */
6 public class RegistryApplication extends RegistrationBase {
```

```java
   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 22: Klasa RegistryApplication

```java
package populationRegistry.registryApplication.models;

/**
 * RegistryPersonalData
 */
public class RegistryAddressData {

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

```
11    public String zipCode = "";
12    public String city = "";
13    public String country = "";
14 }
```

Listing 23: Klasa RegistryAddressData

```
1 package populationRegistry.registryApplication.models;
2
3 import java.time.LocalDate;
4
5 /**
6  * RegistryPersonalData
7  */
8 public class RegistryPersonalData {
9
10    public String firstname = "";
11    public String surname = "";
12    public String pesel = "";
13    public LocalDate dateOfBirth = LocalDate.of(1970, 01, 01);
14 }
```

Listing 24: Klasa RegistryPersonalData