Damian Koper, **241292**

Łukasz Handschuh, **241402**

Inżynieria oprogramowania - Etap 4

Dział ewidencji ludności

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

1 Przypadki użycia - zakres analizy

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

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

2 Analiza wspólności

2.1 Encje

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

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

2.2 Główny przepływ sterowania

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

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

Wszystkie klasy kontrolerów realizują interfejs `IController`.

2.3 Widoki

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

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

2.3.1 Data transfer objects

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

2.4 PESEL

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

3 Analiza zmienności

3.1 Encje

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

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

3.2 Przechowywanie danych

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

- `RegistryApplicationRepository`
- `RegistrationRepository`

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

3.3 Logika biznesowa

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

- `RegistryApplicationService`
- `RegistrationService`

4 Wzorce projektowe

4.1 Flyweight

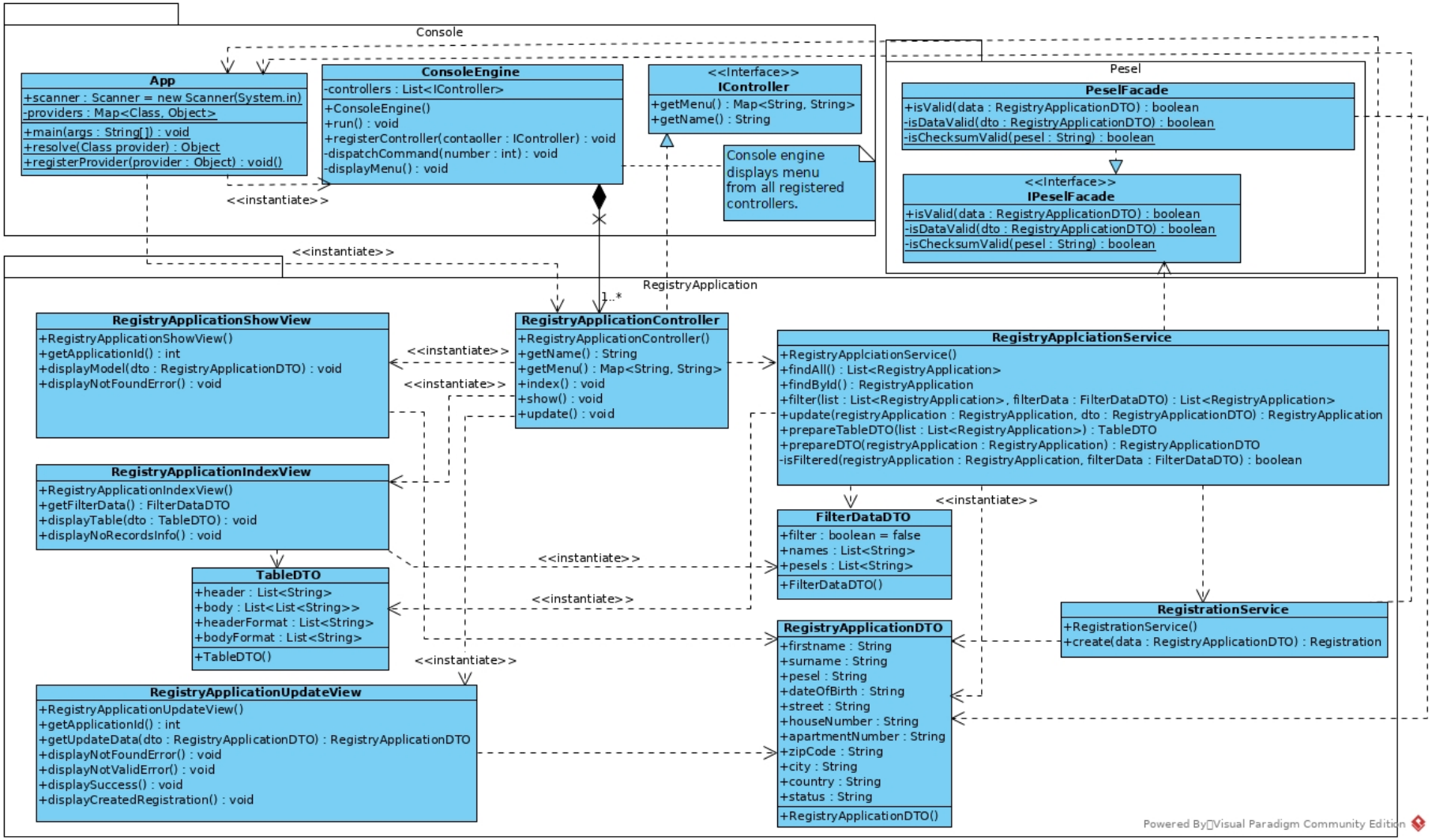
Rolę obiektów Flyweight pełnią klasy `RegistryAddressData` oraz `RegistryPersonalData`. Abstrakcyjnym klientem tych klas jest klasa `RegistrationBase`, z której dziedziczą klasy `RegistryApplication` oraz `Registration`.

4.2 Singleton

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

4.3 Fasada

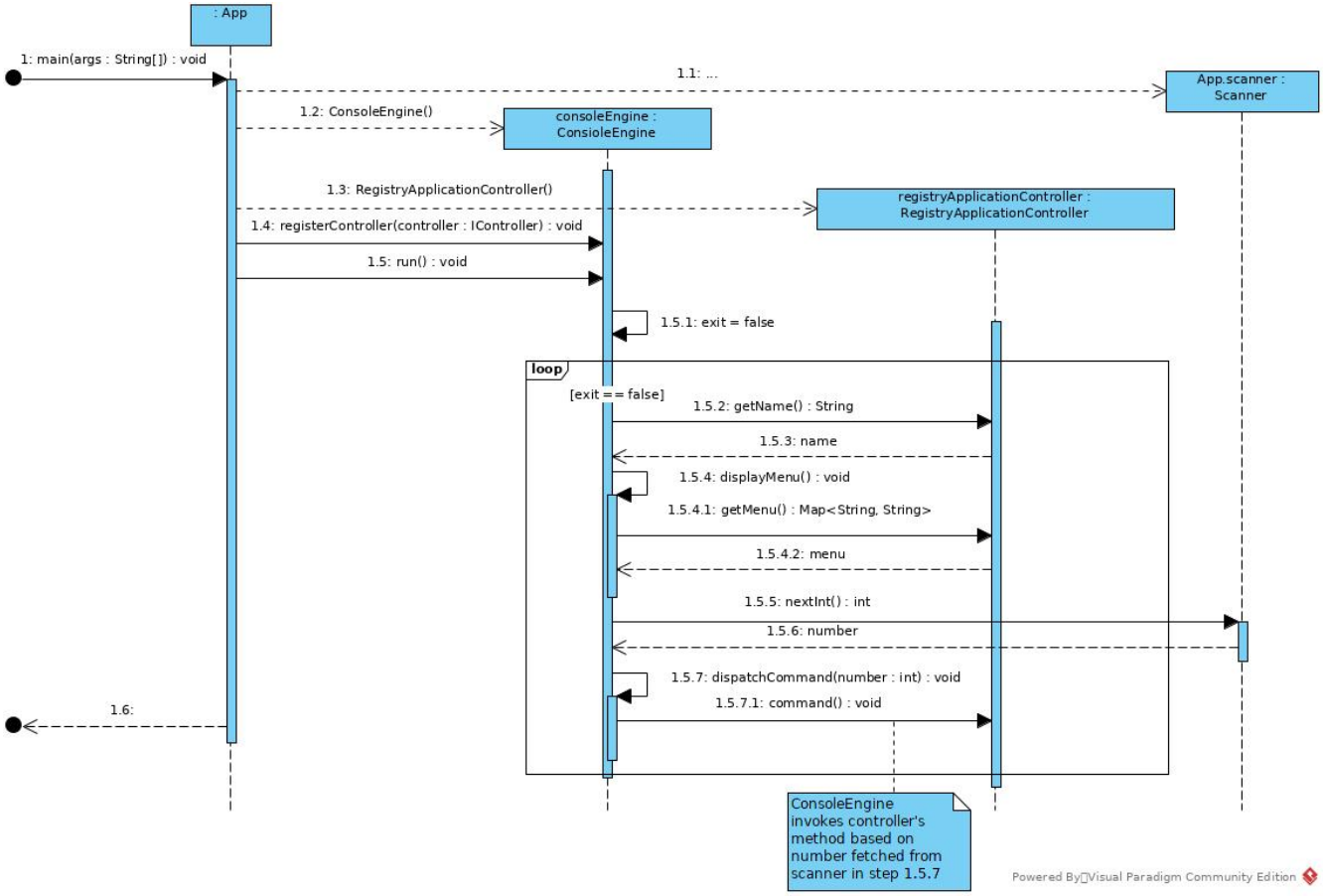
Wzorzec fasada użyty został przy zdefiniowaniu klasy `PeselFasade`, która udostępnia metody umożliwiające komunikację z zewnętrznym systemem. Późniejsza możliwość podmiany implementacji dzięki interfejsowi `IPeselFacade` zapewnia możliwość komunikacji z zewnętrznym systemem w dowolny sposób.



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

5 Diagramy sekwencji

5.1 Główna pętla sterowania



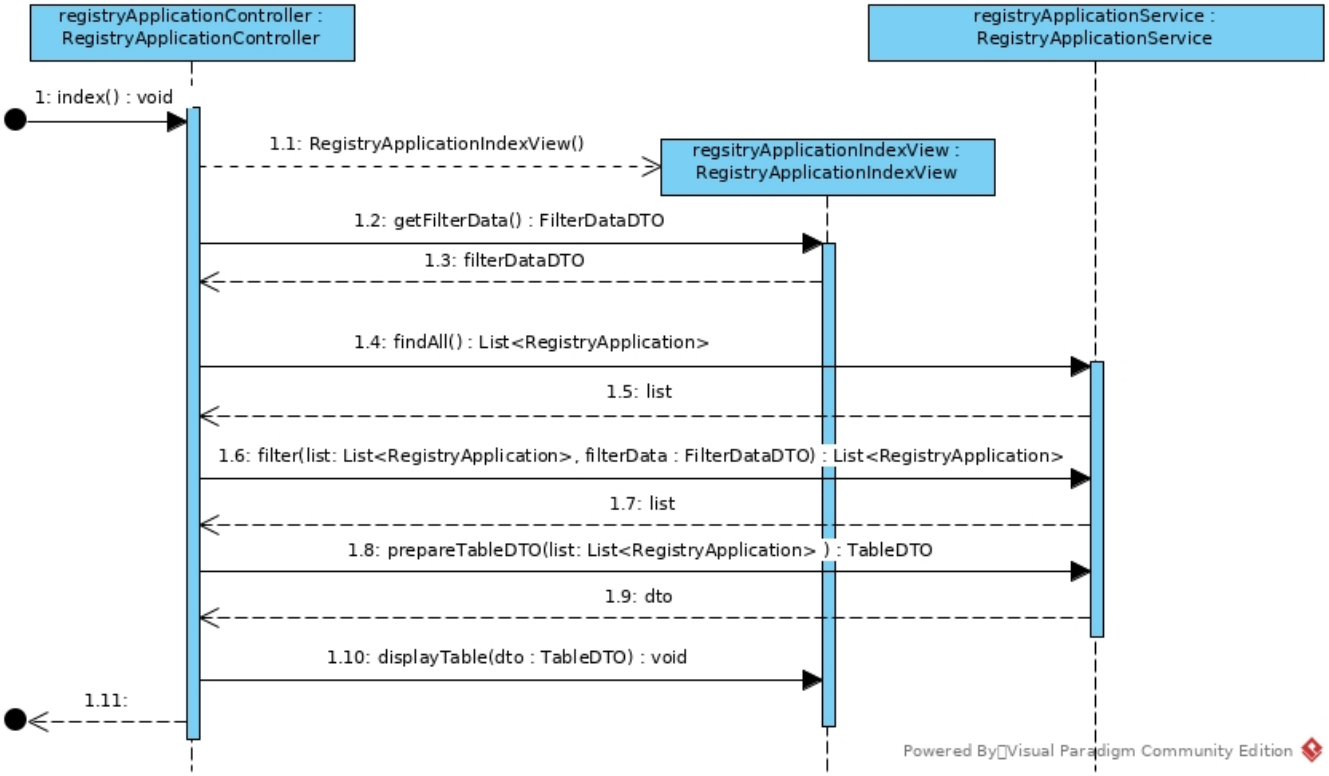
Rysunek 3: Diagram sekwencji - główna pętla przepływu sterowania.

```
1    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

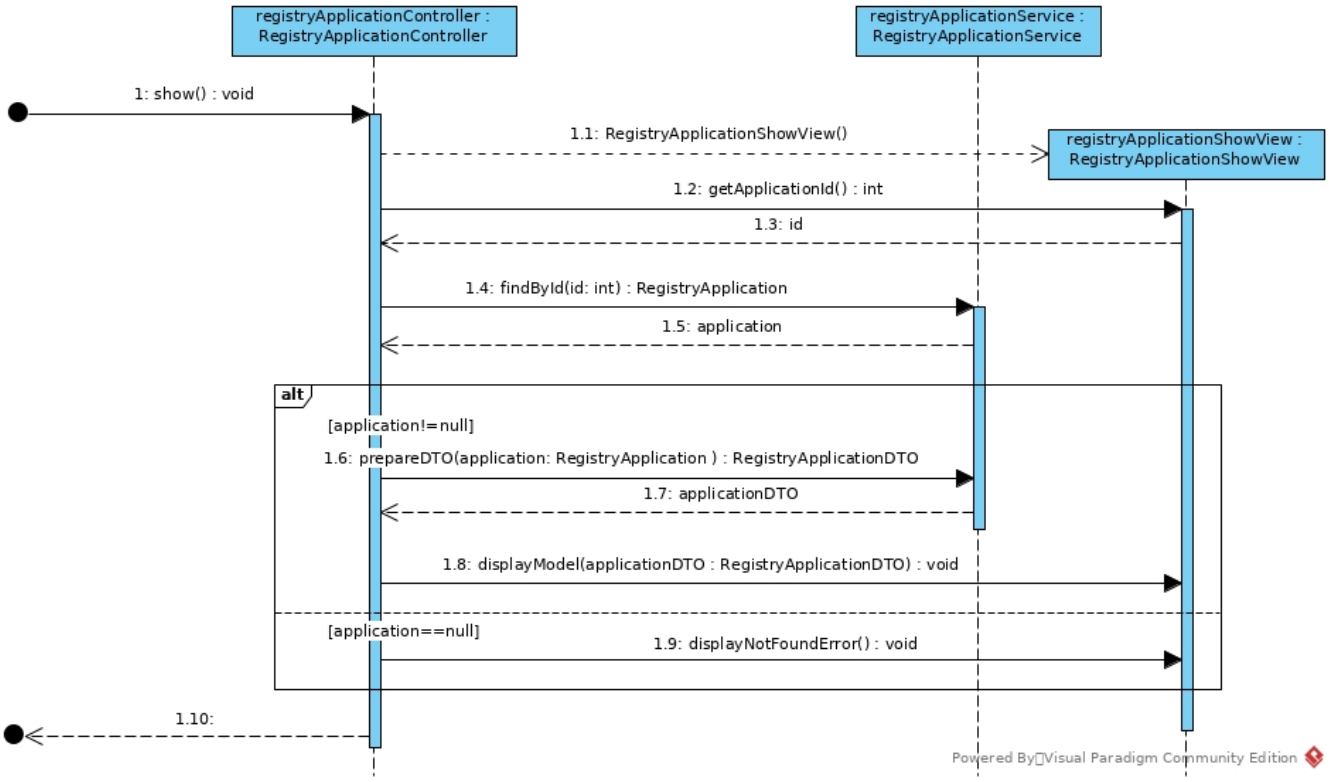


Rysunek 4: Diagram sekwencji - wyświetlanie wniosków.

```
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

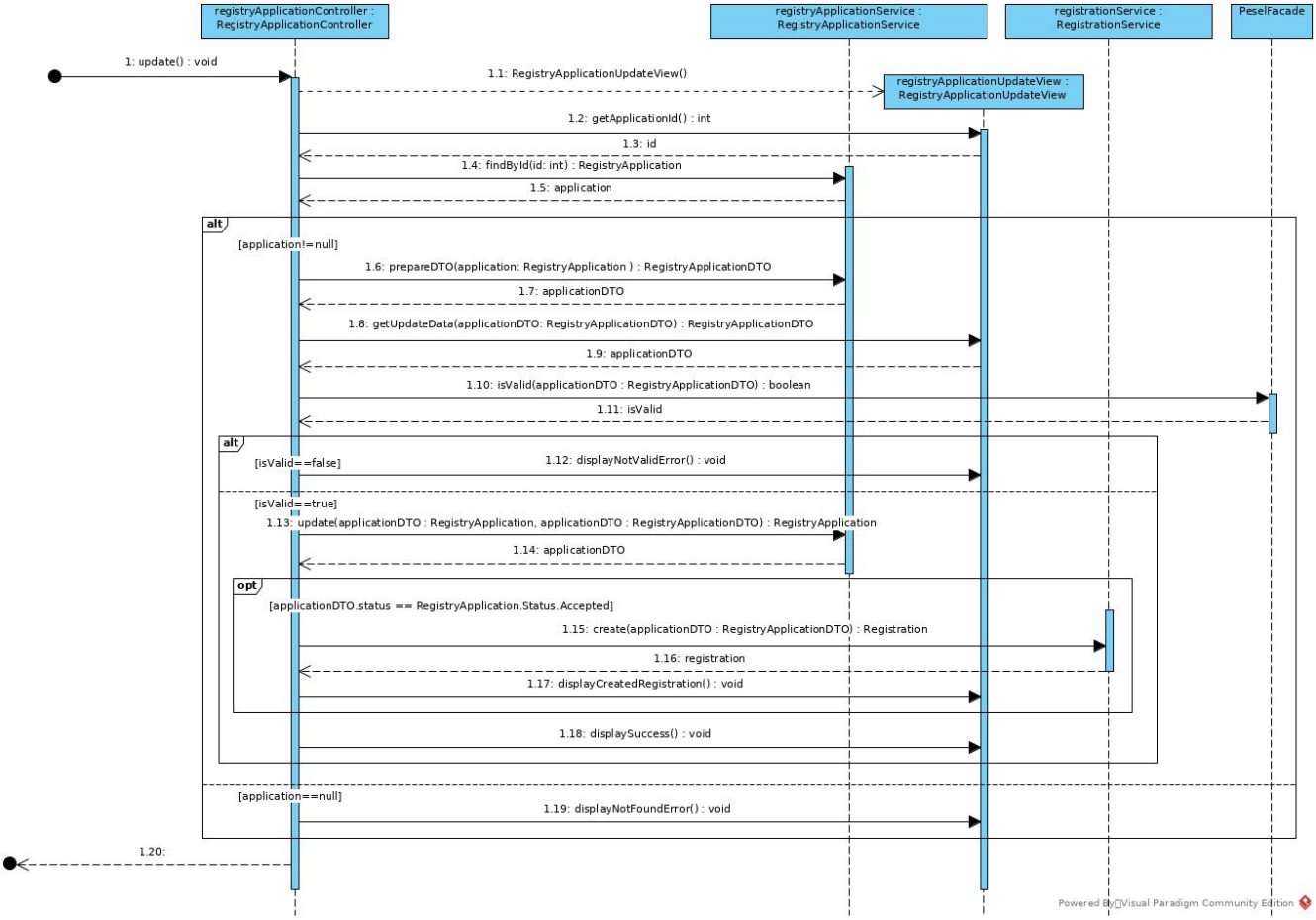


Rysunek 5: Diagram sekwencji - wyświetlanie pojedynczego wniosku.

```
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



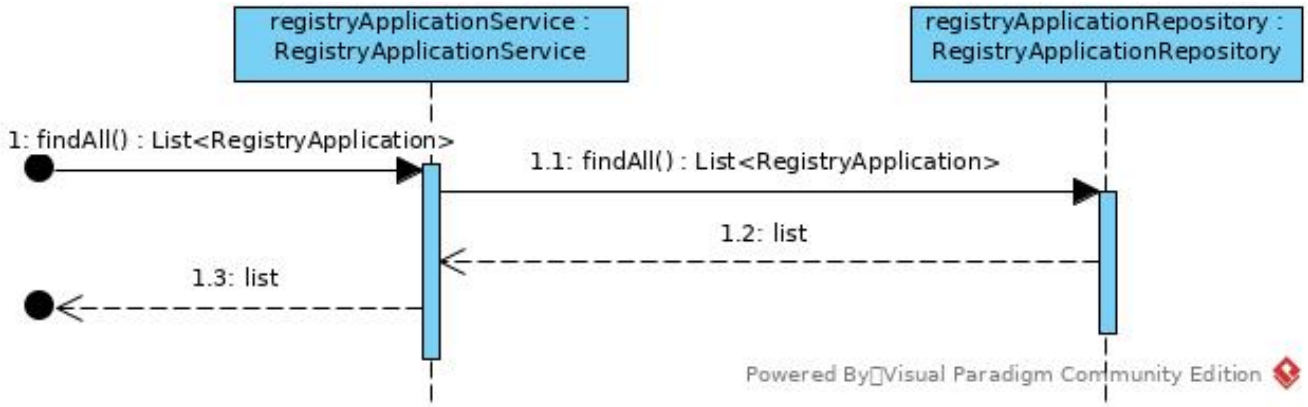
Rysunek 6: Diagram sekwencji - edycja danych wniosku.

```
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

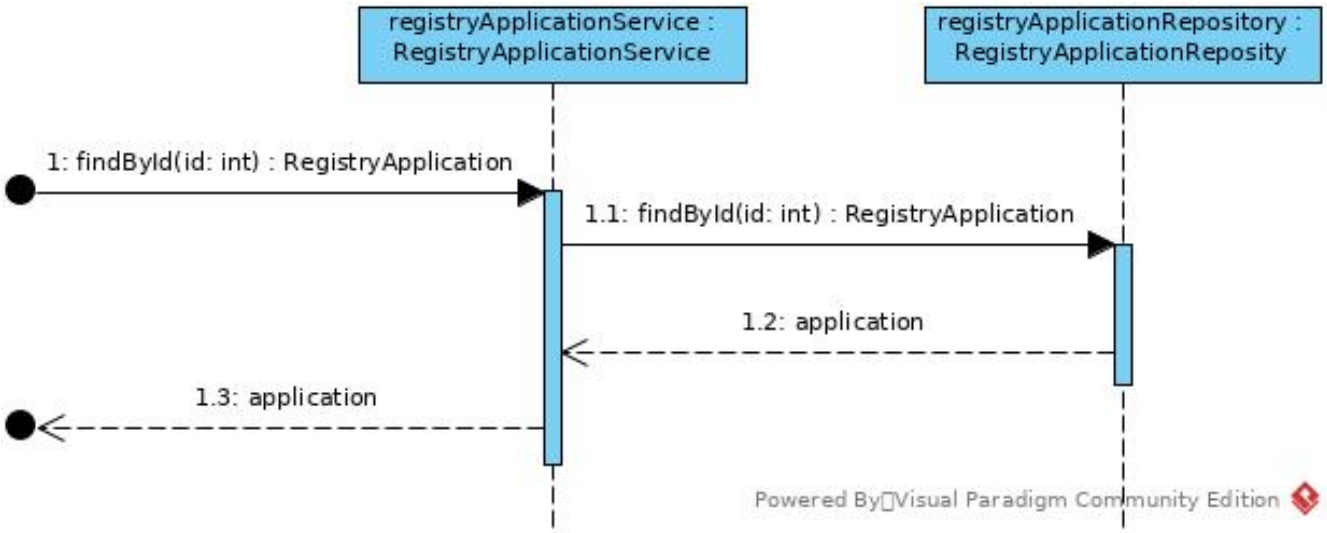


Rysunek 7: Diagram sekwencji - metoda findAll klasy RegistryApplicationService.

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

```
5       }
```

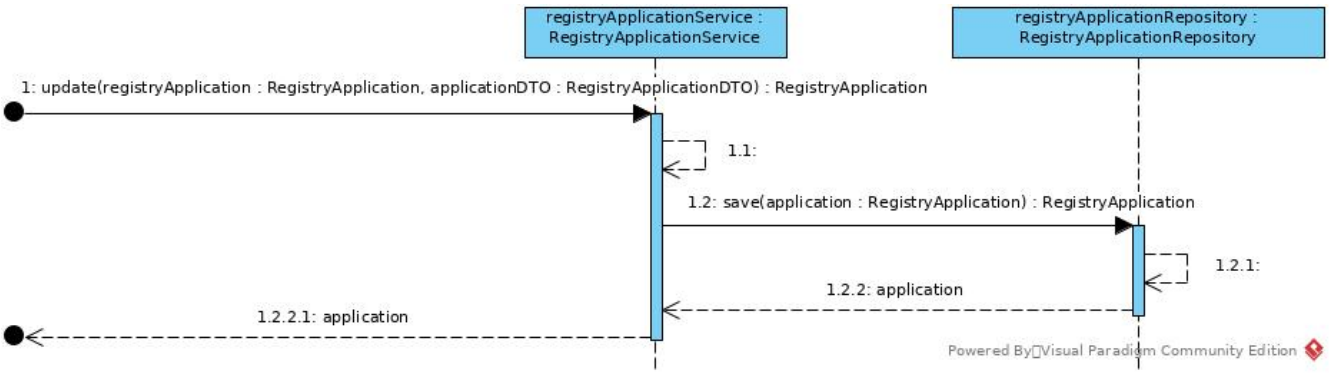
Listing 5: Metoda findAll klasy RegistryApplicationService



Rysunek 8: Diagram sekwencji - metoda findById klasy RegistryApplicationService.

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

Listing 6: Metoda findById klasy RegistryApplicationService



Rysunek 9: Diagram sekwencji - metoda update klasy RegistryApplicationService.

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

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

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

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

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

Listing 7: Metoda update klasy RegistryApplicationService

5.6 Metody klasy RegistrationService

tutaj uzupełnić

6 Kod źródłowy aplikacji

```
package populationRegistry;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Scanner;

import populationRegistry.console.ConsoleEngine;
```

```
import populationRegistry.pesel.PeselFacade;
import populationRegistry.registryApplication.controllers.
   RegistryApplicationController;
import populationRegistry.registryApplication.models.RegistryApplication;
import populationRegistry.registryApplication.repositories.
   RegistrationRepository;
import populationRegistry.registryApplication.repositories.
   RegistryApplicationRepository;
import populationRegistry.registryApplication.services.IPeselFacade;
import populationRegistry.registryApplication.services.RegistrationService;
import populationRegistry.registryApplication.services.
   RegistryApplicationService;

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";
```

```
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

```java
package populationRegistry.console;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.LinkedList;
import populationRegistry.App;

/**
 * ConsoleEngine
 */
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);
      }
    }
  }

  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 {
```

```
39            Method method = iController.getClass()
40                .getMethod(iController.getMenu().keySet().toArray()[number -
    commands].toString());
41            method.invoke(iController);
42          } catch (IllegalAccessException | IllegalArgumentException |
    InvocationTargetException
43              | NoSuchMethodException e) {
44            e.printStackTrace();
45          }
46          return;
47        }
48        commands += commandCount;
49      }
50    }
51
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
```

```
 */
public interface IPeselFacade {

    public boolean isValid(RegistryApplicationDTO dto);

}
```

Listing 11: Interface IPeselFacade

```
package populationRegistry.pesel;

import java.util.ArrayList;

import populationRegistry.registryApplication.services.IPeselFacade;
import populationRegistry.registryApplication.services.dto.
   RegistryApplicationDTO;

/**
 * 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;
```

```
30      }
31
32      private boolean isDataValid(RegistryApplicationDTO dto) {
33          /**
34           * Validation hidden behind facade. Connection to PESEL system.
35           */
36          return true;
37      }
38
39      public boolean isValid(RegistryApplicationDTO dto) {
40          if (!isChecksumValid(dto.pesel))
41              return false;
42          return isDataValid(dto);
43      }
44
45  }
```

Listing 12: Klasa PeselFacade

```
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

```
1   package populationRegistry.registryApplication.repositories;
2
```

```
import java.util.LinkedList;
import java.util.List;

import populationRegistry.registryApplication.models.Registration;

/**
 * RegistryApplicationRepository
 */
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
    );
    }

    @Override
    public Registration save(Registration object) {
        if (!container.contains(object)) {
            container.add(object);
            object.id = nextId++;
        }
        return object;
    }

}
```

Listing 14: Klasa RegistrationRepository

```
package populationRegistry.registryApplication.repositories;

```

```
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
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