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

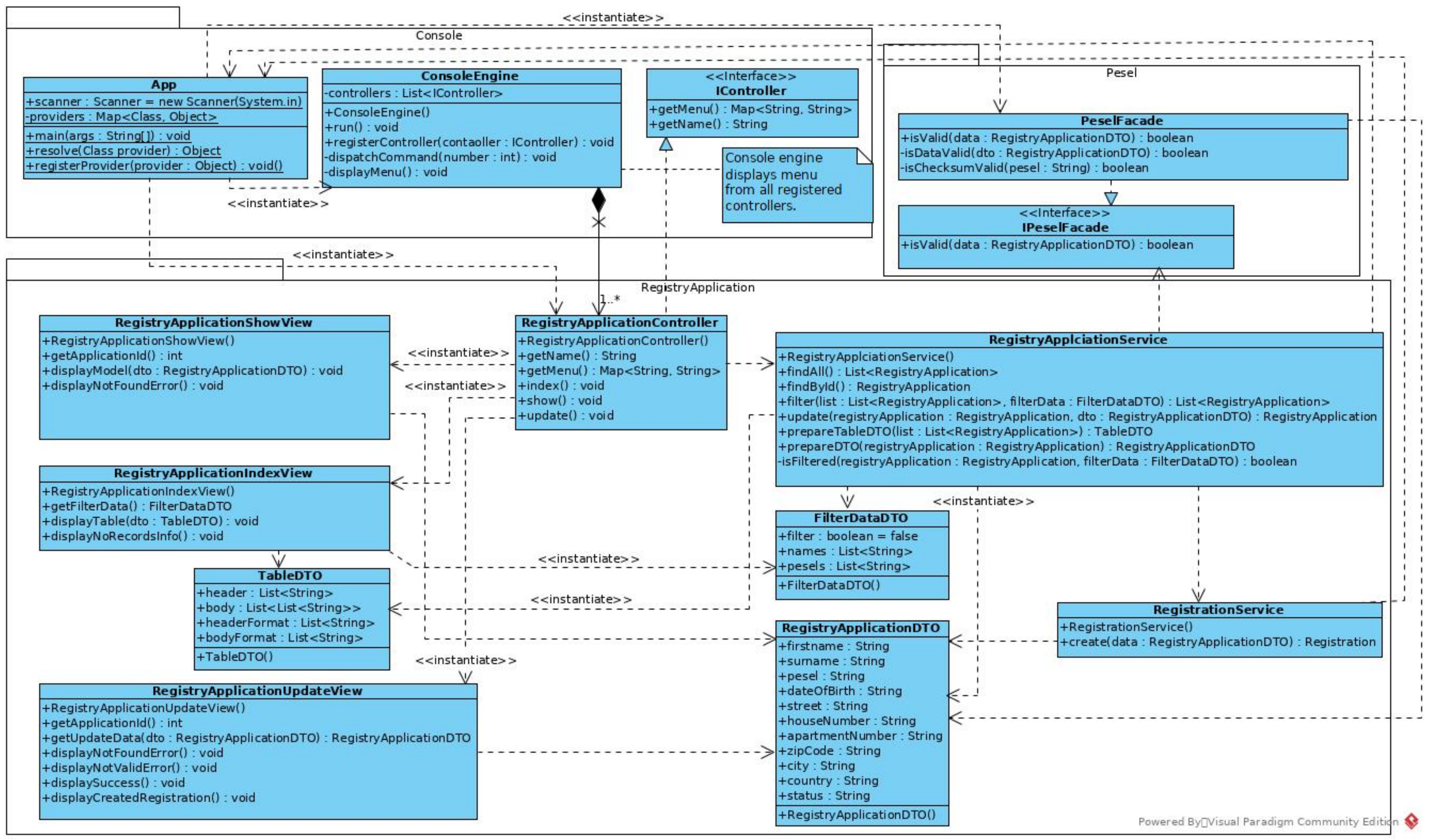
Rolę obiektów Flyweight pełnią klasy `RegistryAddressData` oraz `RegistryPersonalData`. Abstrakcyjnym klientem tych klas jest klasa `RegistrationBase`, z której dziedziczą klasy `RegistryApplication` oraz `Registration`.

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

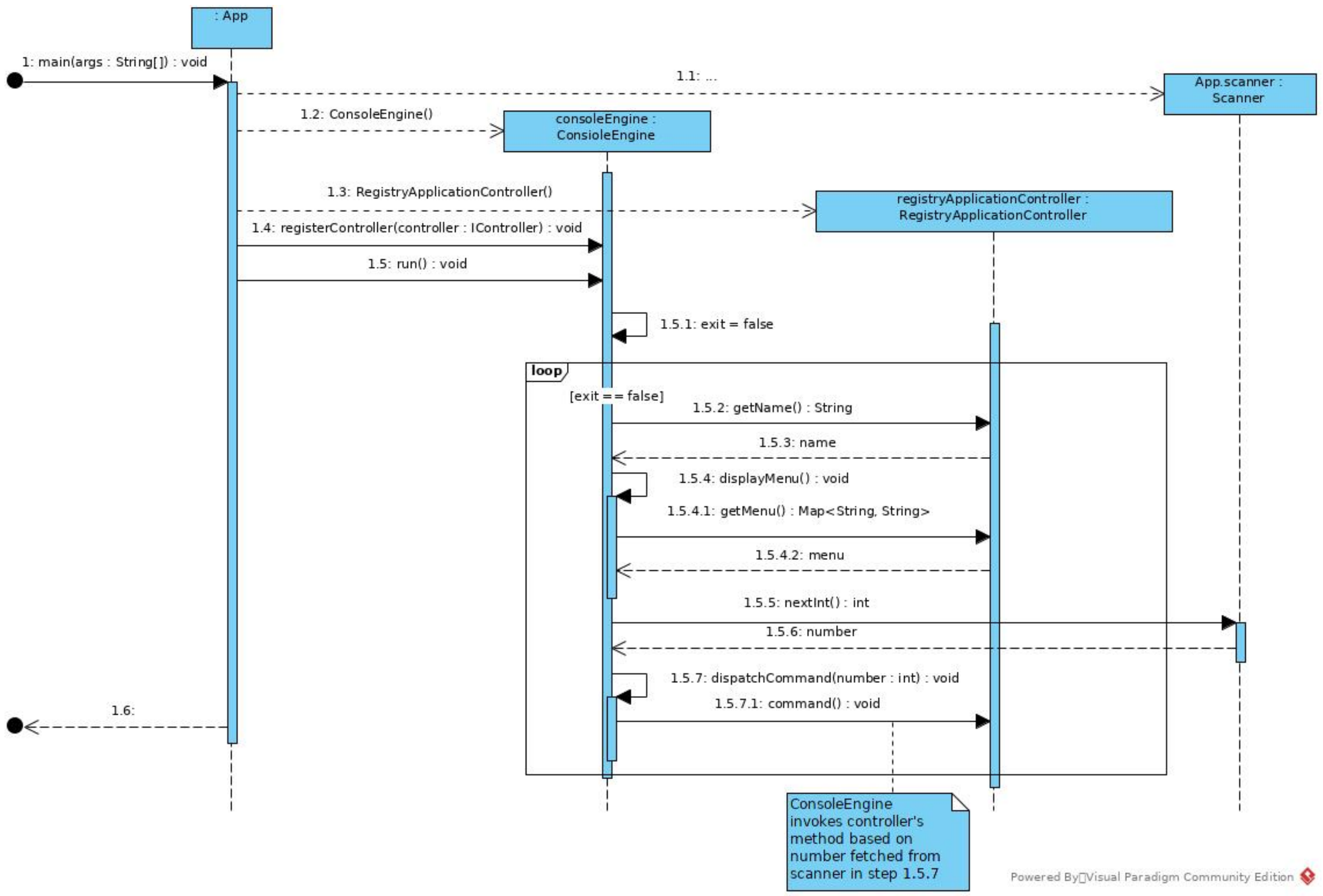
Wzorzec fasada użyty został przy zdefiniowaniu klasy `PeselFasade`, która udostępnia metody umożliwiające komunikację z zewnętrznym systemem. Późniejsza możliwość podmiany implementacji dzięki interfejsowi `IPeselFacade` zapewnia możliwość komunikacji z zewnętrznym systemem w dowolny sposób.


Console
<<instantiate>>
App
+scanner : Scanner = new Scanner(System.in)
-providers : Map<Class, Object>
+main(args : String[]) : void
+resolve(Class provider) : Object
+registerProvider(provider : Object) : void()
ConsoleEngine
-controllers : List<IController>
+ConsoleEngine()
+run() : void
+registerController(contaoller : IController) : void
-dispatchCommand(number : int) : void
-displayMenu() : void
<<Interface>>
IController
+getMenu() : Map<String, String>
+getName() : String
Console engine displays menu from all registered controllers.
Pesel
PeselFacade
+isValid(data : RegistryApplicationDTO) : boolean
-isDataValid(dto : RegistryApplicationDTO) : boolean
-isChecksumValid(pesel : String) : boolean
<<Interface>>
IPeselFacade
+isValid(data : RegistryApplicationDTO) : boolean
RegistryApplication
1..*
RegistryApplicationShowView
+RegistryApplicationShowView()
+getApplicationId() : int
+displayModel(dto : RegistryApplicationDTO) : void
+displayNotFoundError() : void
RegistryApplicationController
+RegistryApplicationController()
+getName() : String
+getMenu() : Map<String, String>
+index() : void
+show() : void
+update() : void
RegistryApplciationService
+RegistryApplciationService()
+findAll() : List<RegistryApplication>
+findById() : RegistryApplication
+filter(list : List<RegistryApplication>, filterData : FilterDataDTO) : List<RegistryApplication>
+update(registryApplication : RegistryApplication, dto : RegistryApplicationDTO) : RegistryApplication
+prepareTableDTO(list : List<RegistryApplication>) : TableDTO
+prepareDTO(registryApplication : RegistryApplication) : RegistryApplicationDTO
-isFiltered(registryApplication : RegistryApplication, filterData : FilterDataDTO) : boolean
RegistryApplicationIndexView
+RegistryApplicationIndexView()
+getFilterData() : FilterDataDTO
+displayTable(dto : TableDTO) : void
+displayNoRecordsInfo() : void
FilterDataDTO
+filter : boolean = false
+names : List<String>
+pesels : List<String>
+FilterDataDTO()
TableDTO
+header : List<String>
+body : List<List<String>>
+headerFormat : List<String>
+bodyFormat : List<String>
+TableDTO()
RegistrationService
+RegistrationService()
+create(data : RegistryApplicationDTO) : Registration
RegistryApplicationDTO
+firstname : String
+surname : String
+pesel : String
+dateOfBirth : String
+street : String
+houseNumber : String
+apartmentNumber : String
+zipCode : String
+city : String
+country : String
+status : String
+RegistryApplicationDTO()
RegistryApplicationUpdateView
+RegistryApplicationUpdateView()
+getApplicationId() : int
+getUpdateData(dto : RegistryApplicationDTO) : RegistryApplicationDTO
+displayNotFoundError() : void
+displayNotValidError() : void
+displaySuccess() : void
+displayCreatedRegistration() : void
Powered By Visual Paradigm Community Edition

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

5 Diagramy sekwencji

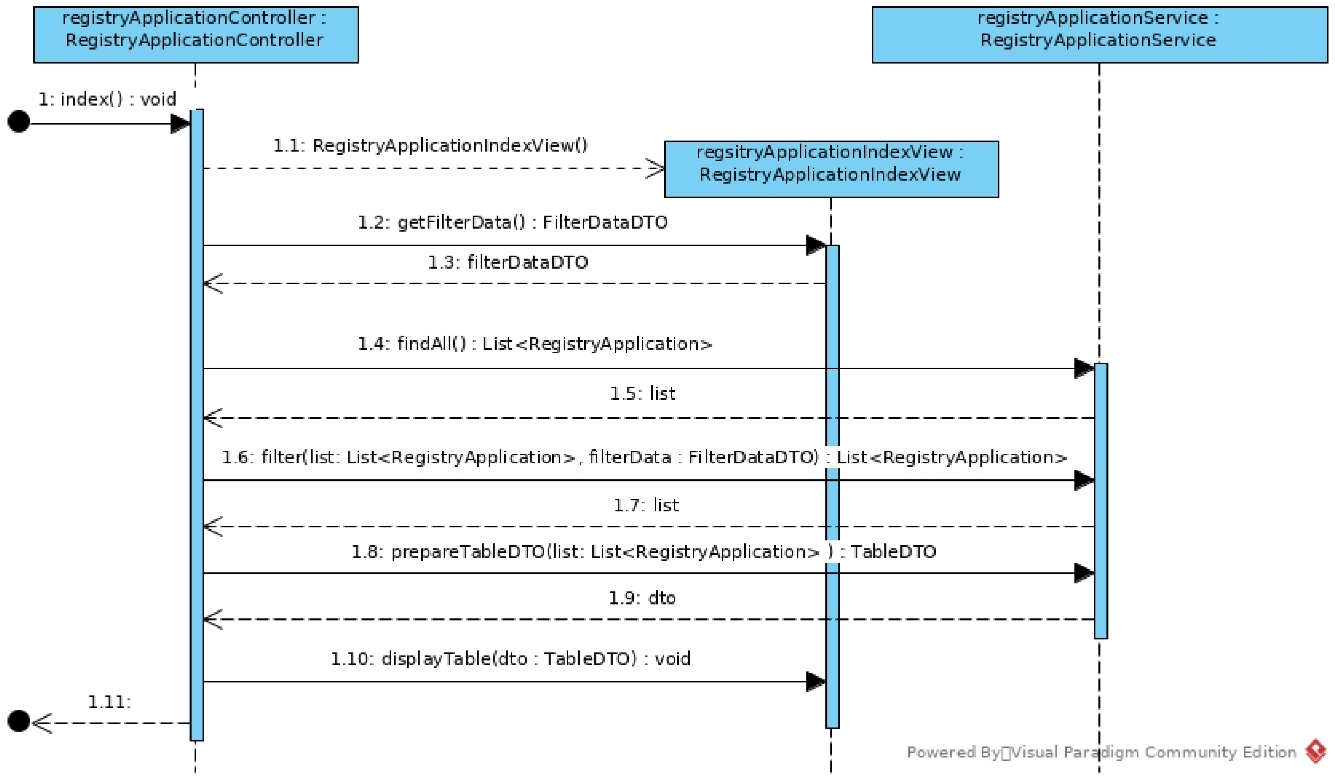
5.1 Główna pętla sterowania


: App
App.scanner : Scanner
consoleEngine : ConsioleEngine
registryApplicationController : RegistryApplicationController
1: main(args : String[]) : void
1.1: ...
1.2: ConsoleEngine()
1.3: RegistryApplicationController()
1.4: registerController(controller : IController) : void
1.5: run() : void
1.5.1: exit = false
loop
[exit == false]
1.5.2: getName() : String
1.5.3: name
1.5.4: displayMenu() : void
1.5.4.1: getMenu() : Map<String, String>
1.5.4.2: menu
1.5.5: nextInt() : int
1.5.6: number
1.5.7: dispatchCommand(number : int) : void
1.5.7.1: command() : void
1.6:
ConsoleEngine invokes controller's method based on number fetched from scanner in step 1.5.7
Powered By Visual Paradigm Community Edition

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();
        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();
```
Listing 1: Metoda main klasy App

5.2 Wyświetlanie wniosków

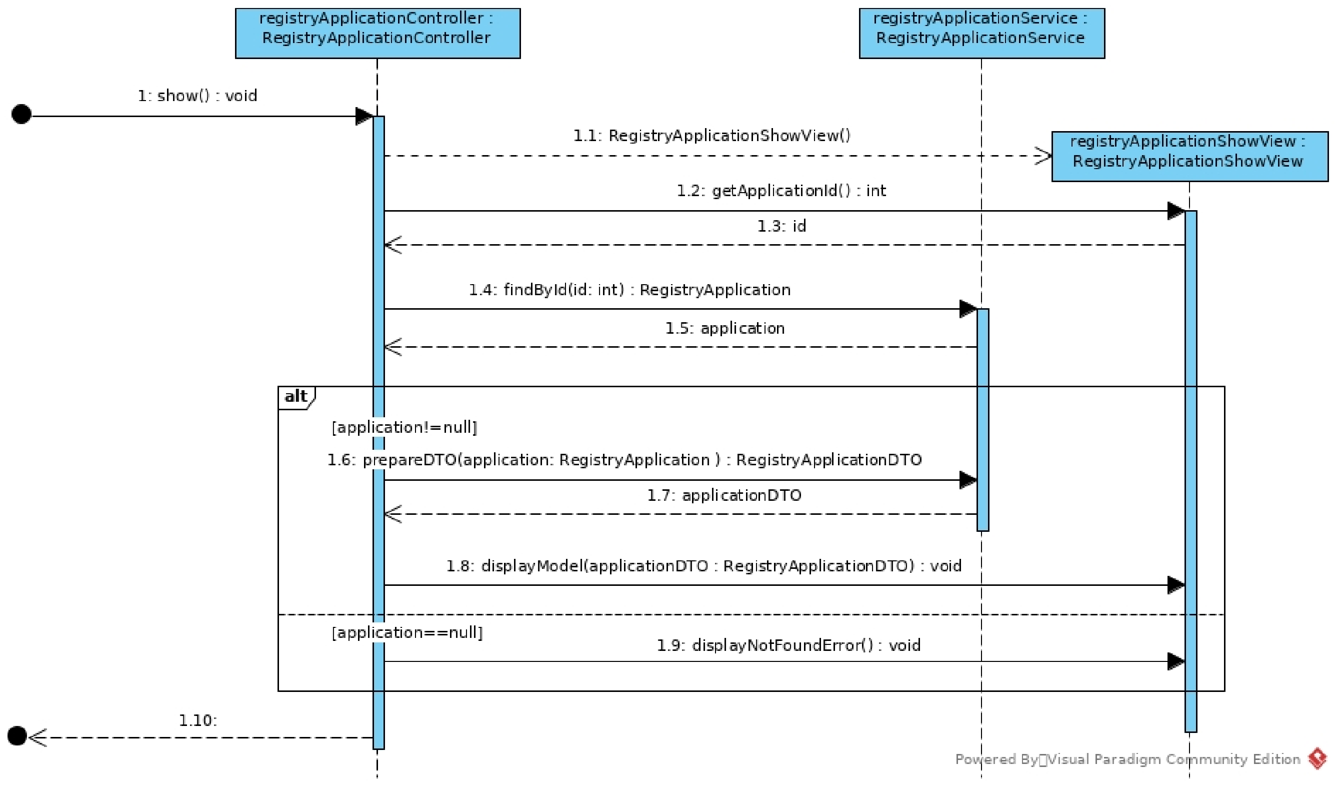

registryApplicationController : RegistryApplicationController
registryApplicationService : RegistryApplicationService
1: index() : void
1.1: RegistryApplicationIndexView()
regsitryApplicationIndexView : RegistryApplicationIndexView
1.2: getFilterData() : FilterDataDTO
1.3: filterDataDTO
1.4: findAll() : List<RegistryApplication>
1.5: list
1.6: filter(list: List<RegistryApplication>, filterData : FilterDataDTO) : List<RegistryApplication>
1.7: list
1.8: prepareTableDTO(list: List<RegistryApplication>) : TableDTO
1.9: dto
1.10: displayTable(dto : TableDTO) : void
1.11:
Powered By Visual Paradigm Community Edition

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

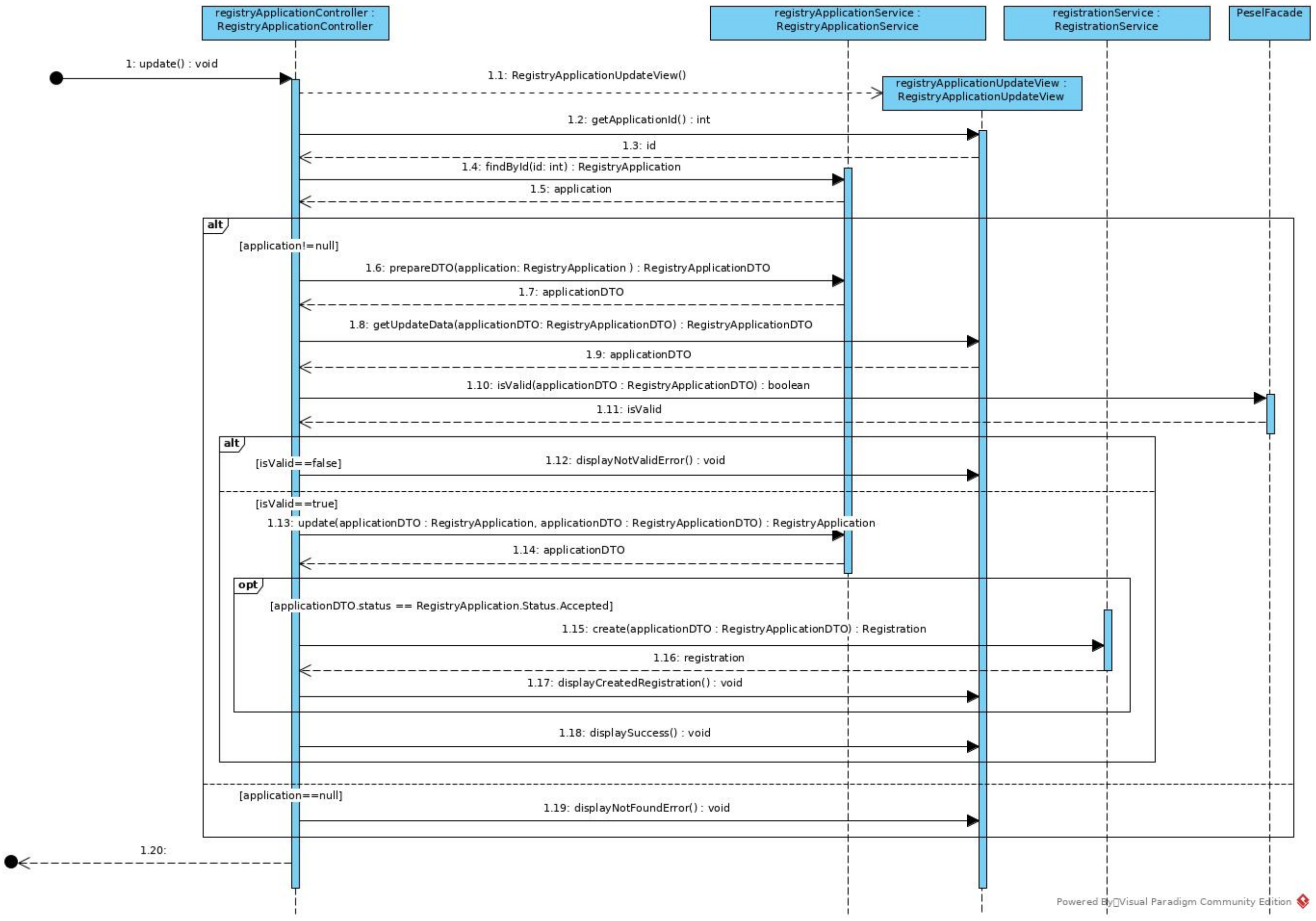

registryApplicationController : RegistryApplicationController
registryApplicationService : RegistryApplicationService
registryApplicationShowView : RegistryApplicationShowView
1: show() : void
1.1: RegistryApplicationShowView()
1.2: getApplicationId() : int
1.3: id
1.4: findById(id: int) : RegistryApplication
1.5: application
alt
[application!=null]
1.6: prepareDTO(application: RegistryApplication) : RegistryApplicationDTO
1.7: applicationDTO
1.8: displayModel(applicationDTO : RegistryApplicationDTO) : void
[application==null]
1.9: displayNotFoundError() : void
1.10:
Powered By Visual Paradigm Community Edition

Rysunek 5: Diagram sekwencji - wyświetlanie pojedynczego wniosku.

```java

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



registryApplicationController : RegistryApplicationController
registryApplicationService : RegistryApplicationService
registrationService : RegistrationService
PeselFacade
1: update() : void
1.1: RegistryApplicationUpdateView()
registryApplicationUpdateView : RegistryApplicationUpdateView
1.2: getApplicationId() : int
1.3: id
1.4: findById(id: int) : RegistryApplication
1.5: application
alt
[application!=null]
1.6: prepareDTO(application: RegistryApplication) : RegistryApplicationDTO
1.7: applicationDTO
1.8: getUpdateData(applicationDTO: RegistryApplicationDTO) : RegistryApplicationDTO
1.9: applicationDTO
1.10: isValid(applicationDTO : RegistryApplicationDTO) : boolean
1.11: isValid
alt
[isValid==false]
1.12: displayNotValidError() : void
[isValid==true]
1.13: update(applicationDTO : RegistryApplication, applicationDTO : RegistryApplicationDTO) : RegistryApplication
1.14: applicationDTO
opt
[applicationDTO.status == RegistryApplication.Status.Accepted]
1.15: create(applicationDTO : RegistryApplicationDTO) : Registration
1.16: registration
1.17: displayCreatedRegistration() : void
1.18: displaySuccess() : void
[application==null]
1.19: displayNotFoundError() : void
1.20:
Powered By Visual Paradigm Community Edition

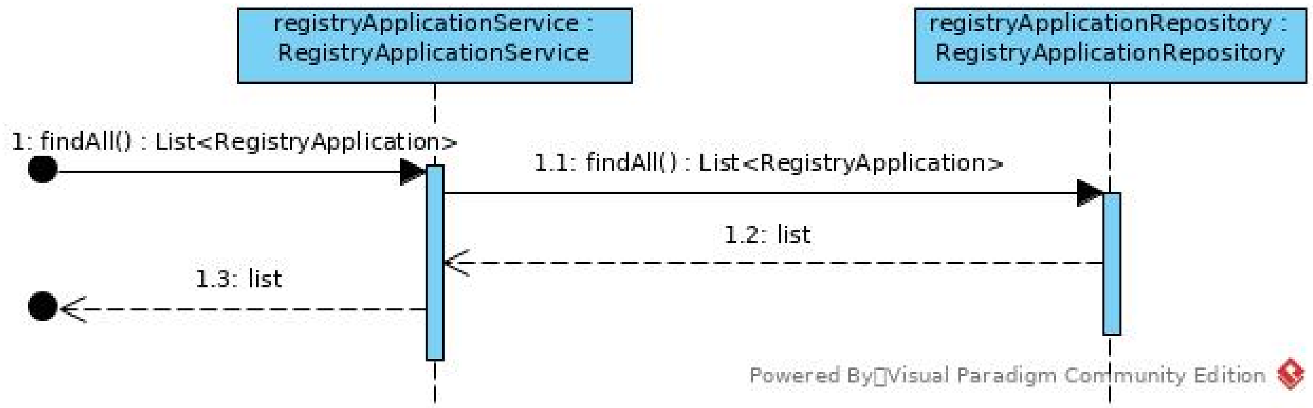
Rysunek 6: Diagram sekwencji - edycja danych wniosku.

```

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();
```

```java
    } else {
      RegistryApplicationDTO dto = registryApplicationService.prepareDTO(
registryApplication);
      dto = view.getUpdateData(dto);
      boolean isValid = peselFacade.isValid(dto);
      if (!isValid) {
        view.displayNotValidError();
      } else {
        registryApplicationService.update(registryApplication, dto);
        if (registryApplication.status.equals(RegistryApplication.Status.
Accepted)) {
          registrationService.create(dto);
          view.displayCreatedRegistration();
        }
        view.displaySuccess();
      }
```

Listing 4: Metoda update klasy RegistryApplicationController

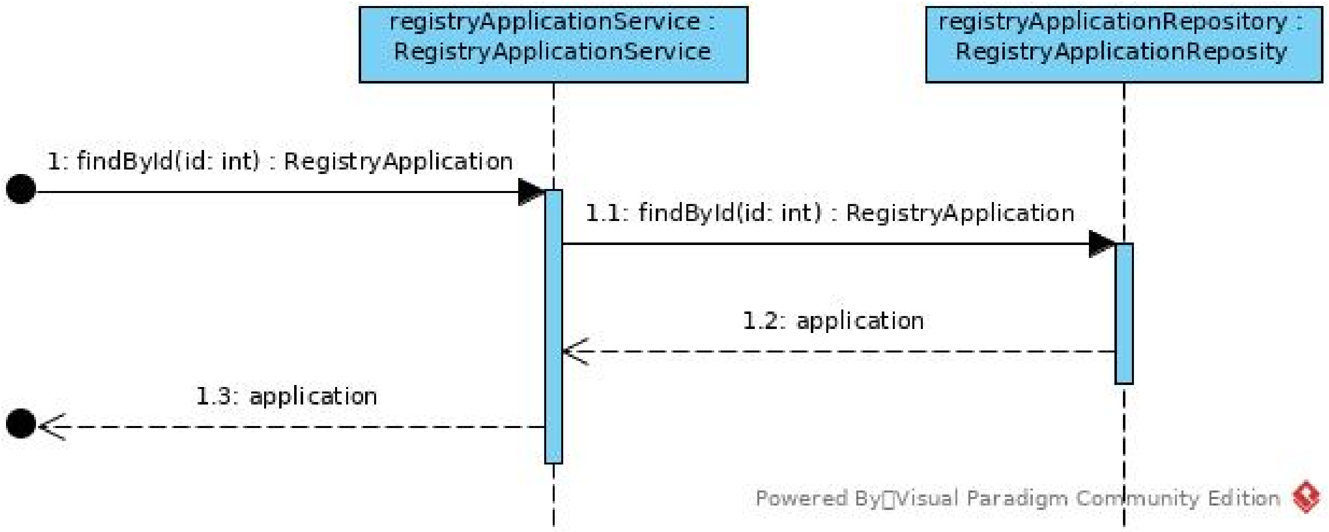
5.5 Metody klasy RegistryApplicationService


registryApplicationService : RegistryApplicationService
registryApplicationRepository : RegistryApplicationRepository
1: findAll() : List<RegistryApplication>
1.1: findAll() : List<RegistryApplication>
1.2: list
1.3: list
Powered By Visual Paradigm Community Edition

Rysunek 7: Diagram sekwencji - metoda findAll klasy RegistryApplicationService.

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

Listing 5: Metoda findAll klasy RegistryApplicationService

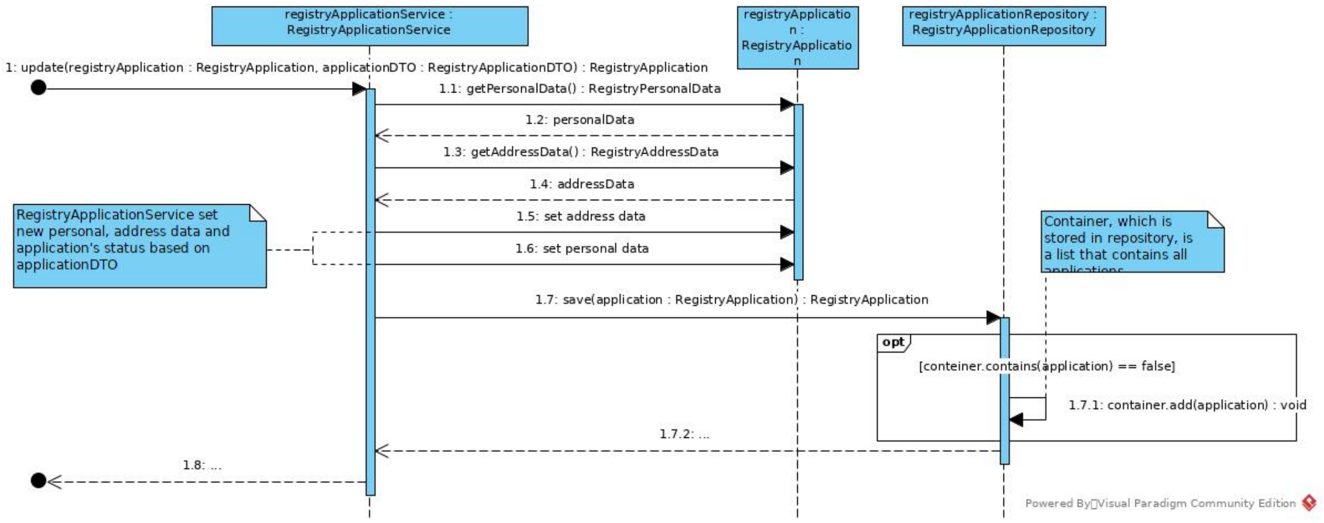

registryApplicationService : RegistryApplicationService
registryApplicationRepository : RegistryApplicationReposity
1: findById(id: int) : RegistryApplication
1.1: findById(id: int) : RegistryApplication
1.2: application
1.3: application
Powered By Visual Paradigm Community Edition

Rysunek 8: Diagram sekwencji - metoda findById klasy RegistryApplicationService.

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

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

Listing 6: Metoda findById klasy RegistryApplicationService

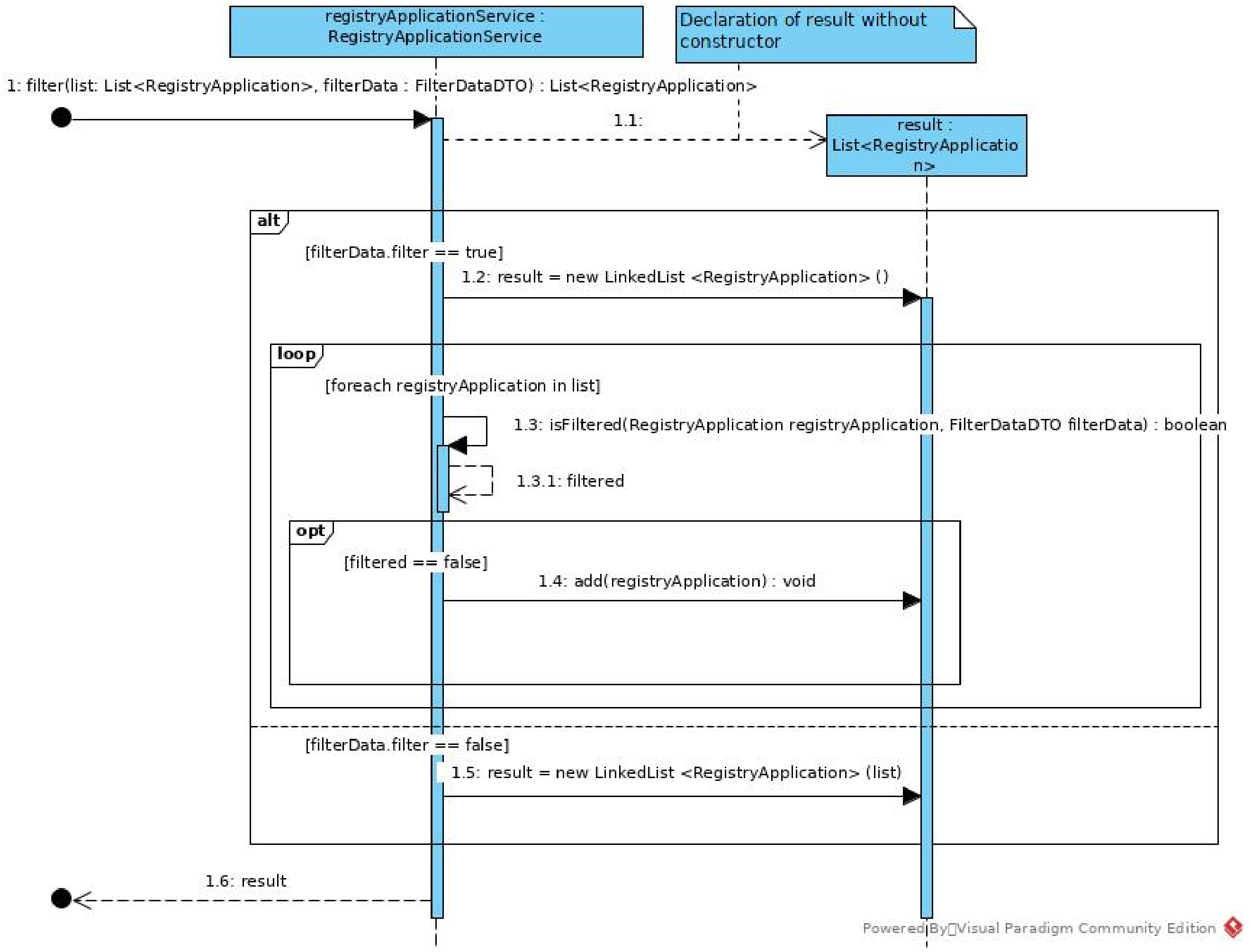

registryApplicationService : RegistryApplicationService
registryApplicatio n : RegistryApplicatio n
registryApplicationRepository : RegistryApplicationRepository
1: update(registryApplication : RegistryApplication, applicationDTO : RegistryApplicationDTO) : RegistryApplication
1.1: getPersonalData() : RegistryPersonalData
1.2: personalData
1.3: getAddressData() : RegistryAddressData
1.4: addressData
1.5: set address data
1.6: set personal data
RegistryApplicationService set new personal, address data and application's status based on applicationDTO
Container, which is stored in repository, is a list that contains all applications
1.7: save(application : RegistryApplication) : RegistryApplication
opt
[conteiner.contains(application) == false]
1.7.1: container.add(application) : void
1.7.2: ...
1.8: ...
Powered By Visual Paradigm Community Edition

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);
```

```

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

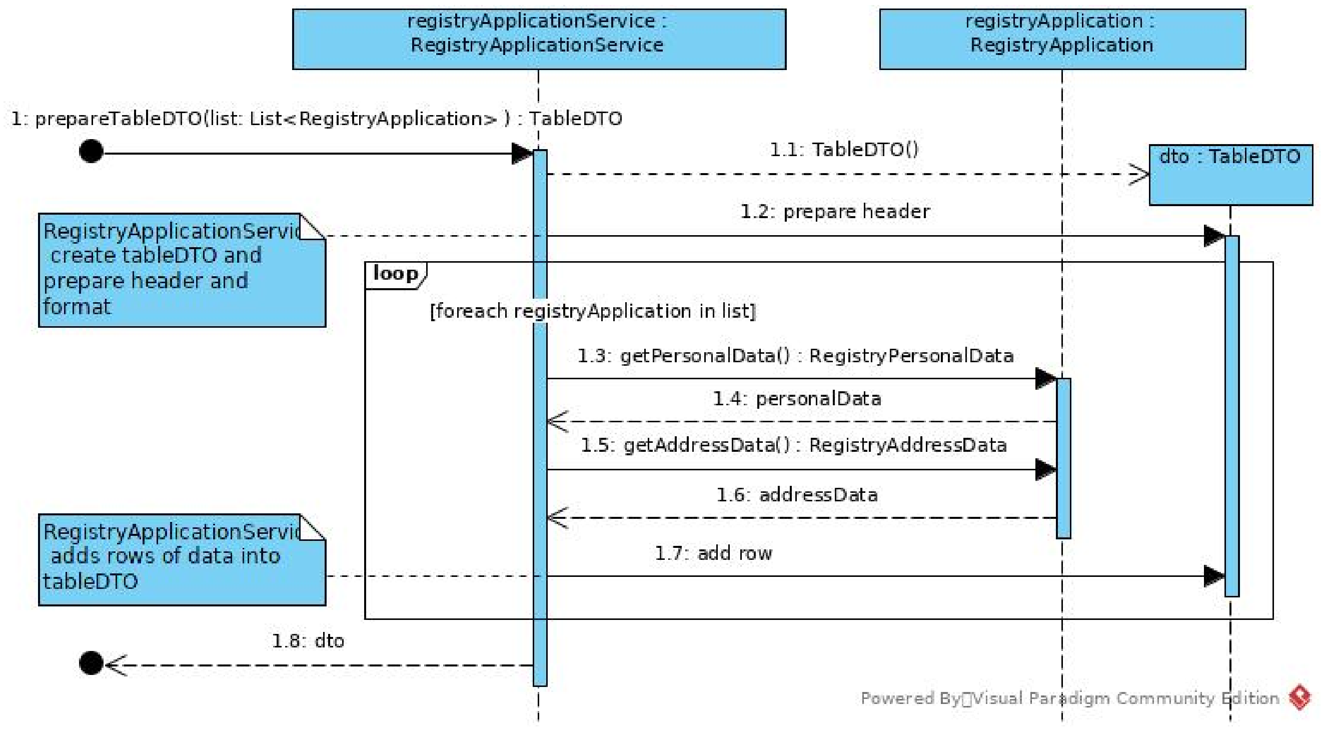
Listing 7: Metoda update klasy RegistryApplicationService


registryApplicationService : RegistryApplicationService
Declaration of result without constructor
1: filter(list: List<RegistryApplication>, filterData : FilterDataDTO) : List<RegistryApplication>
1.1:
result : List<RegistryApplication>
alt
[filterData.filter == true]
1.2: result = new LinkedList <RegistryApplication> ()
loop
[foreach registryApplication in list]
1.3: isFiltered(RegistryApplication registryApplication, FilterDataDTO filterData) : boolean
1.3.1: filtered
opt
[filtered == false]
1.4: add(registryApplication) : void
[filterData.filter == false]
1.5: result = new LinkedList <RegistryApplication> (list)
1.6: result
Powered By Visual Paradigm Community Edition

Rysunek 10: Diagram sekwencji - metoda filter klasy RegistryApplicationService.

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

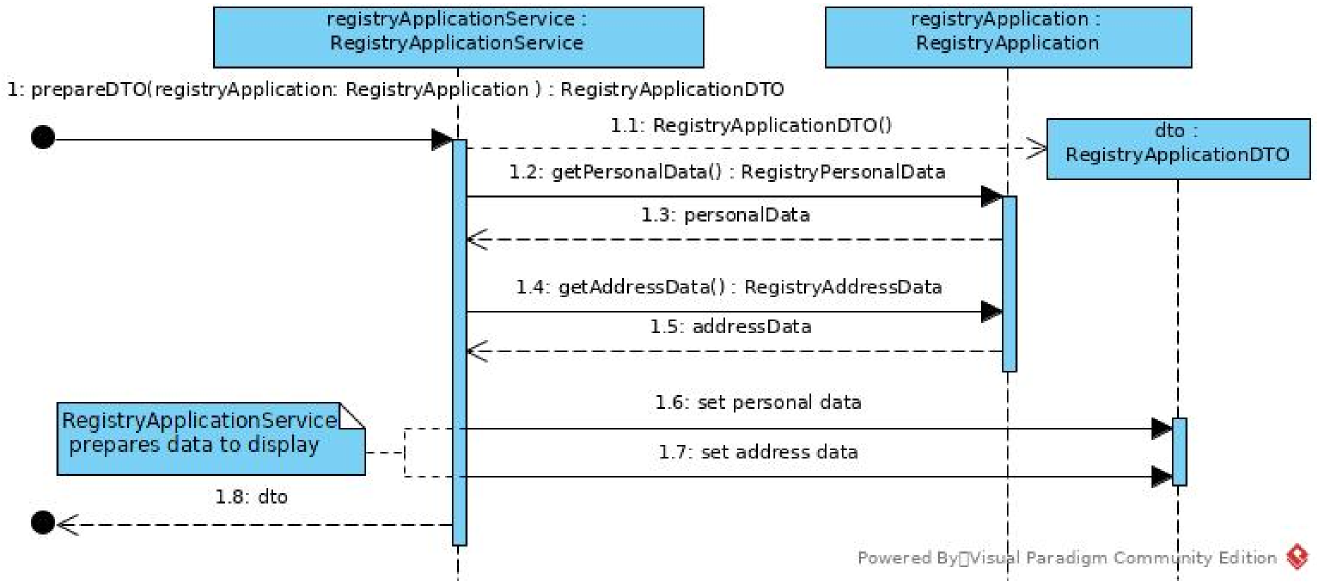

registryApplicationService : RegistryApplicationService
registryApplication : RegistryApplication
1: prepareTableDTO(list: List<RegistryApplication>) : TableDTO
1.1: TableDTO()
dto : TableDTO
1.2: prepare header
RegistryApplicationServic create tableDTO and prepare header and format
loop
[foreach registryApplication in list]
1.3: getPersonalData() : RegistryPersonalData
1.4: personalData
1.5: getAddressData() : RegistryAddressData
1.6: addressData
RegistryApplicationServic adds rows of data into tableDTO
1.7: add row
1.8: dto
Powered By Visual Paradigm Community Edition

Rysunek 11: Diagram sekwencji - metoda prepareTableDTO klasy RegistryApplicationService.

```
 public TableDTO prepareTableDTO(List<RegistryApplication> list) {
     TableDTO dto = new TableDTO();
     dto.header = new LinkedList<String>(
             Arrays.asList("Id", "Imie", "Nazwisko", "PESEL", "Wnioskowany
adres zameldowania"));
     LinkedList<String> format = new LinkedList<String>(Arrays.asList("%5s",
 "%15s", "%15s", "%11s", "%60s"));
     dto.headerFormat = format;
     dto.bodyFormat = format;
     for (RegistryApplication registryApplication : list) {
         LinkedList<String> row = new LinkedList<>();
         row.add(String.valueOf(registryApplication.id));

         RegistryPersonalData personal = registryApplication.getPersonalData
();
         row.add(personal.firstname);
         row.add(personal.surname);
         row.add(personal.pesel);

         RegistryAddressData address = registryApplication.getAddressData();
```

```
            row.add(address.city + " ul." + address.street + " " + address.
    houseNumber + "/" + address.apartmentNumber
                    + " " + address.zipCode + ", " + address.country);
            dto.body.add(row);
        }
        return dto;
    }
```

Listing 9: Metoda prepareTableDTO klasy RegistryApplicationService

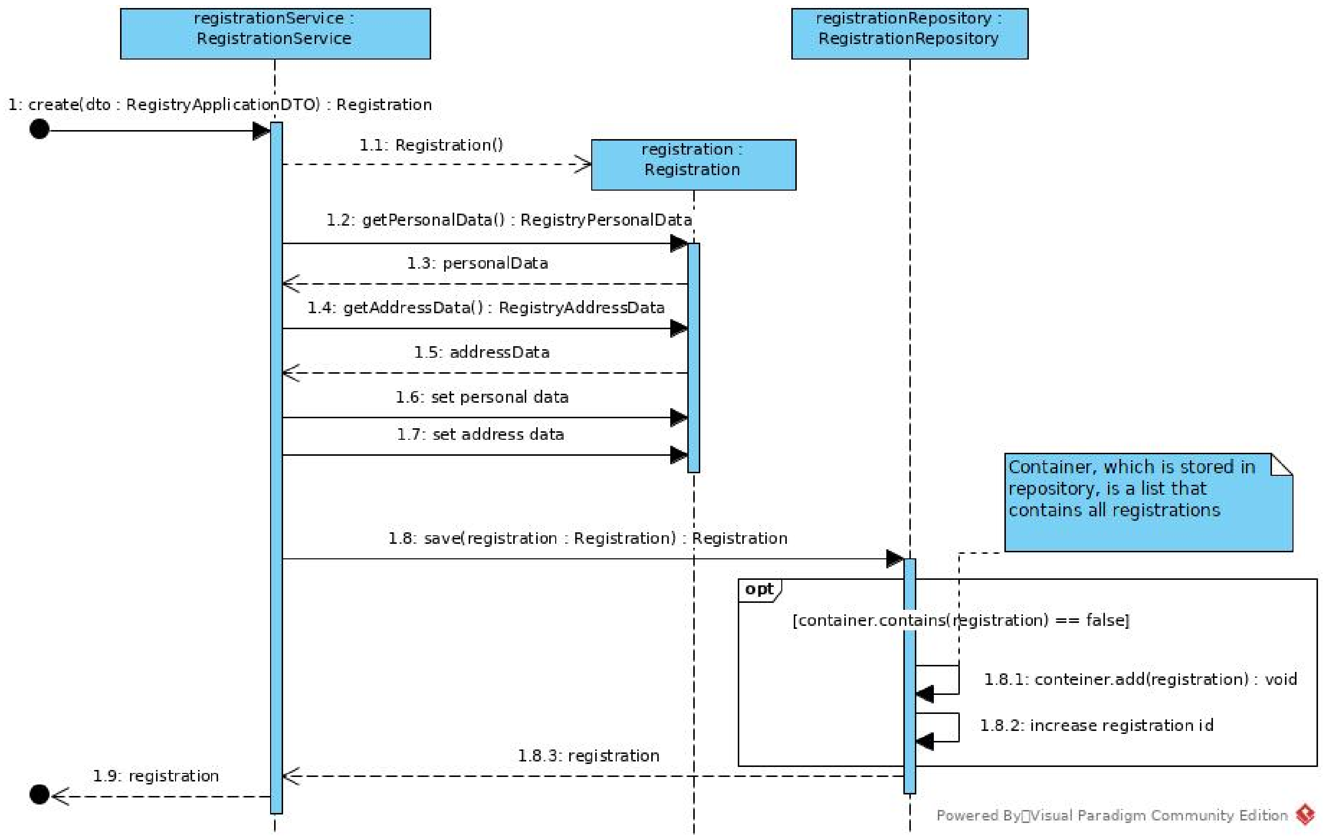

registryApplicationService : RegistryApplicationService
registryApplication : RegistryApplication
1: prepareDTO(registryApplication: RegistryApplication) : RegistryApplicationDTO
1.1: RegistryApplicationDTO()
dto : RegistryApplicationDTO
1.2: getPersonalData() : RegistryPersonalData
1.3: personalData
1.4: getAddressData() : RegistryAddressData
1.5: addressData
1.6: set personal data
1.7: set address data
RegistryApplicationService prepares data to display
1.8: dto
Powered By Visual Paradigm Community Edition

Rysunek 12: Diagram sekwencji - metoda prepareDTO klasy RegistryApplicationService.

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

        RegistryApplicationDTO dto = new RegistryApplicationDTO();
        dto.id = registryApplication.id;
        dto.firstname = personal.firstname;
        dto.surname = personal.surname;
        dto.pesel = personal.pesel;
        dto.dateOfBirth = personal.dateOfBirth.toString();
        dto.apartmentNumber = address.apartmentNumber;
        dto.houseNumber = address.houseNumber;
        dto.city = address.city;
        dto.country = address.country;
        dto.street = address.street;
        dto.status = registryApplication.status.toString();
        dto.zipCode = address.zipCode;

        return dto;
    }
```

Listing 10: Metoda prepareDTO klasy RegistryApplicationService

5.6 Metody klasy RegistrationService


registrationService : RegistrationService
registrationRepository : RegistrationRepository
1: create(dto : RegistryApplicationDTO) : Registration
1.1: Registration()
registration : Registration
1.2: getPersonalData() : RegistryPersonalData
1.3: personalData
1.4: getAddressData() : RegistryAddressData
1.5: addressData
1.6: set personal data
1.7: set address data
Container, which is stored in repository, is a list that contains all registrations
1.8: save(registration : Registration) : Registration
opt
[container.contains(registration) == false]
1.8.1: conteiner.add(registration) : void
1.8.2: increase registration id
1.8.3: registration
1.9: registration
Powered By Visual Paradigm Community Edition

Rysunek 13: Diagram sekwencji - metoda create klasy 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 11: Metoda create klasy RegistrationService

6 Kod źródłowy aplikacji

```java
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";
        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

```
package populationRegistry.console;

import java.util.Map;

/**
 * IController
```

```java
 */
public interface IController {

  public Map<String, String> getMenu();

  public String getName();
}
```

Listing 13: 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 {
          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;
    }
  }

  private void displayMenu() {
    int option = 1;
    System.out.println("\n### Menu:");
    System.out.println("[0] Wyjscie");
    for (IController iController : controllers) {
      System.out.println("--- " + iController.getName());
      for (String name : iController.getMenu().keySet()) {
        System.out.println("[" + String.valueOf(option) + "] " + iController.getMenu().get(name));
        option = option + 1;
      }
    }
  }
```

```java
}
```

Listing 14: Klasa ConsoleEngine

```java
package populationRegistry.registryApplication.services;

import populationRegistry.registryApplication.services.dto.
    RegistryApplicationDTO;

/**
 * PeselFacade
 */
public interface IPeselFacade {

    public boolean isValid(RegistryApplicationDTO dto);

}
```

Listing 15: Interface IPeselFacade

```java
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;
    }

    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);
    }

}
```

Listing 16: Klasa PeselFacade

```java
package populationRegistry.registryApplication.repositories;

import java.util.List;

/**
 * IRepository
 */
public interface IRepository<T> {

    public List<T> findAll();

    public T findById(int id);

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

Listing 17: Interface IRepository

```java
package populationRegistry.registryApplication.repositories;

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 18: Klasa RegistrationRepository

```java
package populationRegistry.registryApplication.repositories;

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

import populationRegistry.registryApplication.models.RegistryApplication;

/**
 * RegistryApplicationRepository
 */
public class RegistryApplicationRepository implements IRepository<
    RegistryApplication> {

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

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

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

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

}
```

Listing 19: Klasa RegistryApplicationRepository

```java
package populationRegistry.registryApplication.services;

import java.time.LocalDate;

import populationRegistry.App;
import populationRegistry.registryApplication.models.Registration;
import populationRegistry.registryApplication.models.RegistryAddressData;
import populationRegistry.registryApplication.models.RegistryPersonalData;
import populationRegistry.registryApplication.repositories.
   RegistrationRepository;
import populationRegistry.registryApplication.services.dto.
   RegistryApplicationDTO;

/**
 * RegistrationService
 */
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 20: Klasa RegistrationService

```
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;
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 {

```

```java
    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) {
                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();
```

```java
        })) {
            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;
        personal.dateOfBirth = LocalDate.parse(dto.dateOfBirth);

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

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

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

        RegistryApplicationDTO dto = new RegistryApplicationDTO();
        dto.id = registryApplication.id;
        dto.firstname = personal.firstname;
        dto.surname = personal.surname;
        dto.pesel = personal.pesel;
        dto.dateOfBirth = personal.dateOfBirth.toString();
        dto.apartmentNumber = address.apartmentNumber;
        dto.houseNumber = address.houseNumber;
        dto.city = address.city;
        dto.country = address.country;
        dto.street = address.street;
        dto.status = registryApplication.status.toString();
        dto.zipCode = address.zipCode;

        return dto;
    }

    public TableDTO prepareTableDTO(List<RegistryApplication> list) {
        TableDTO dto = new TableDTO();
        dto.header = new LinkedList<String>(
                Arrays.asList("Id", "Imie", "Nazwisko", "PESEL", "Wnioskowany
adres zameldowania"));
        LinkedList<String> format = new LinkedList<String>(Arrays.asList("%5s",
 "%15s", "%15s", "%11s", "%60s"));
        dto.headerFormat = format;
        dto.bodyFormat = format;
        for (RegistryApplication registryApplication : list) {
            LinkedList<String> row = new LinkedList<>();
            row.add(String.valueOf(registryApplication.id));

            RegistryPersonalData personal = registryApplication.getPersonalData
();
            row.add(personal.firstname);
            row.add(personal.surname);
```

```java
            row.add(personal.pesel);

            RegistryAddressData address = registryApplication.getAddressData();
            row.add(address.city + " ul." + address.street + " " + address.
    houseNumber + "/" + address.apartmentNumber
                    + " " + address.zipCode + ", " + address.country);
            dto.body.add(row);
        }
        return dto;
    }
}
```

Listing 21: Klasa RegistryApplicationService

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

import java.util.LinkedList;

public class FilterDataDTO {

  public boolean filter = false;
  public LinkedList<String> names = new LinkedList<>();
  public LinkedList<String> pesels = new LinkedList<>();
}
```

Listing 22: Klasa FilterDataDTO

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

public class RegistryApplicationDTO {

  public int id;
  public String firstname;
  public String surname;
  public String pesel;
  public String dateOfBirth;
  public String street;
  public String houseNumber;
  public String apartmentNumber;
  public String zipCode;
  public String city;
  public String country;
  public String status;
}
```

Listing 23: Klasa RegistryApplicationDTO

```
package populationRegistry.registryApplication.models;

/**
 * RegistryApplicationData
 */
public abstract class RegistrationBase {
  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

```
package populationRegistry.registryApplication.models;

/**
 * Registration
```

```java
 */
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

```java
package populationRegistry.registryApplication.models;

/**
 * RegistryApplication
 */
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

```java
package populationRegistry.registryApplication.models;

/**
 * RegistryPersonalData
 */
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

```java
package populationRegistry.registryApplication.models;

import java.time.LocalDate;

```

```java
/**
 * RegistryPersonalData
 */
public class RegistryPersonalData {

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

Listing 28: Klasa RegistryPersonalData