

Inżynieria Oprogramowania

Laboratorium

Michał Tłuczek 259050

Patryk Jurkiewicz 263896

Temat aplikacji: Program obsługujący Dział Ewidencji Ludności

1. Opis „świata rzeczywistego” Działu Ewidencji Ludności

1.1. Opis zasobów ludzkich

W Dziale Ewidencji Ludności pracują urzędnicy odpowiedzialni za rejestrowanie, aktualizowanie i zarządzanie danymi osobowymi mieszkańców danego kraju. Ich obowiązki obejmują przyjmowanie i przetwarzanie dokumentów tożsamości, rejestrację narodzin i zgonów, wydawanie dowodów osobistych, paszportów oraz innych dokumentów tożsamości. Dodatkowo, pracownicy mogą przeprowadzać spisy ludności i zbierać informacje statystyczne. Mieszkańcy mogą zgłaszać chęci wyrobienia dokumentów jeśli spełniają wymagania, odnowienia ważności czy zgłoszenia problemu np. zgubienia.

1.2. Przepisy, strategia działu

Dział Ewidencji Ludności podlega przepisom i regulacjom dotyczącym rejestrowania i przechowywania danych osobowych oraz przepisom dotyczącym bezpieczeństwa danych. W strategii działu priorytetem jest zapewnienie dokładności i integralności danych oraz skutecznego wsparcia obywateli w sprawach związanych z ich dokumentami tożsamości.

1.3. Dane techniczne

System obsługujący Dział Ewidencji Ludności musi być dostępny dla urzędników w biurach ewidencji ludności oraz dla obywateli, którzy mogą korzystać z usług online do spraw związanych z dokumentami tożsamości. System musi być w stanie obsługiwać duże ilości danych osobowych, w tym dane identyfikujące czy adresowe. Bezpieczeństwo i poufność danych są kluczowe.

2. Zdefiniowanie wymagań funkcjonalnych i нефункциональных

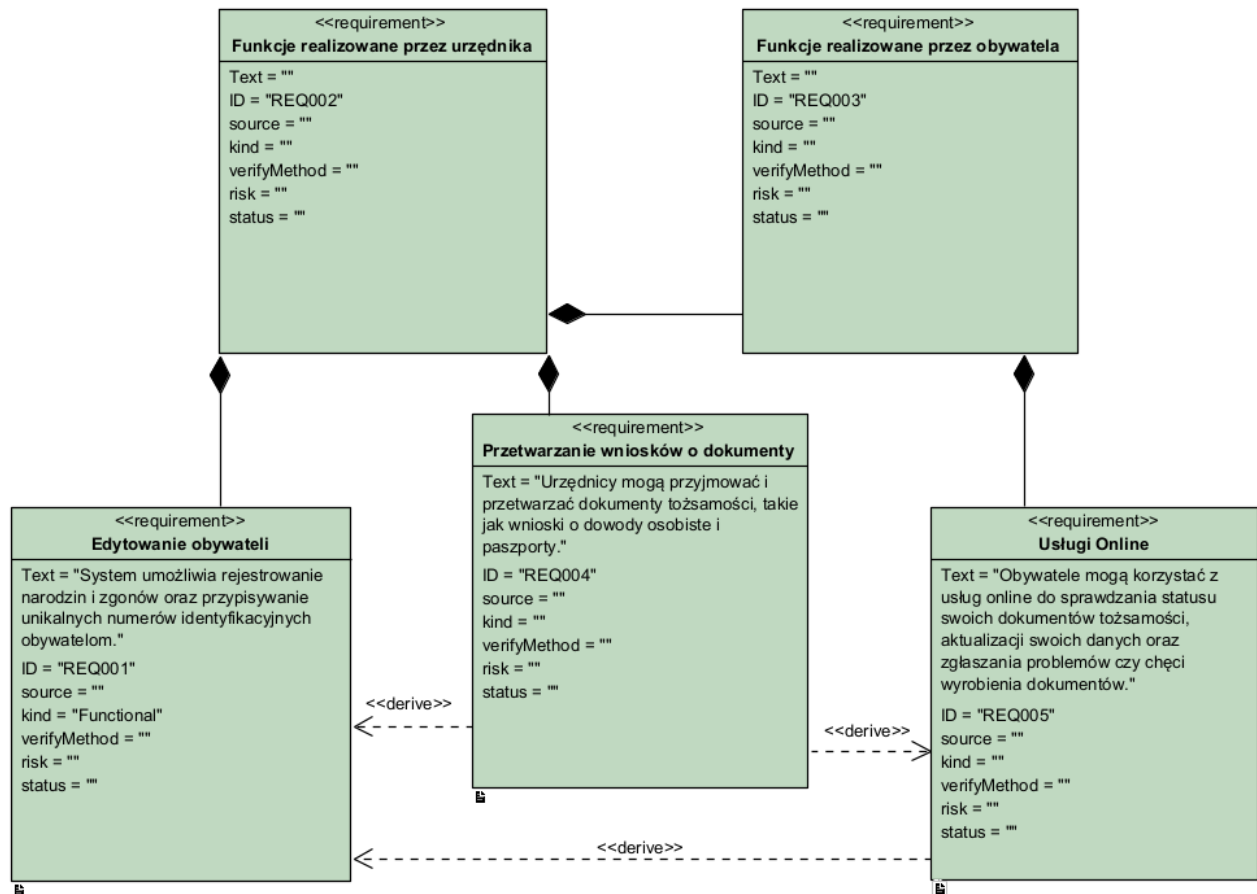
2.1. Wymagania funkcjonalne

1. System umożliwia rejestrowanie narodzin i zgonów oraz przypisywanie unikalnych numerów identyfikacyjnych obywatelom.
2. Urzędnicy mogą przyjmować i przetwarzać dokumenty tożsamości, takie jak wnioski i dowody osobiste i paszporty.
3. Obywatele mogą korzystać z usług online do sprawdzenia statusu swoich dokumentów tożsamości, aktualizacji swoich danych oraz zgłaszania problemów czy chęci wyrobienia dokumentów.

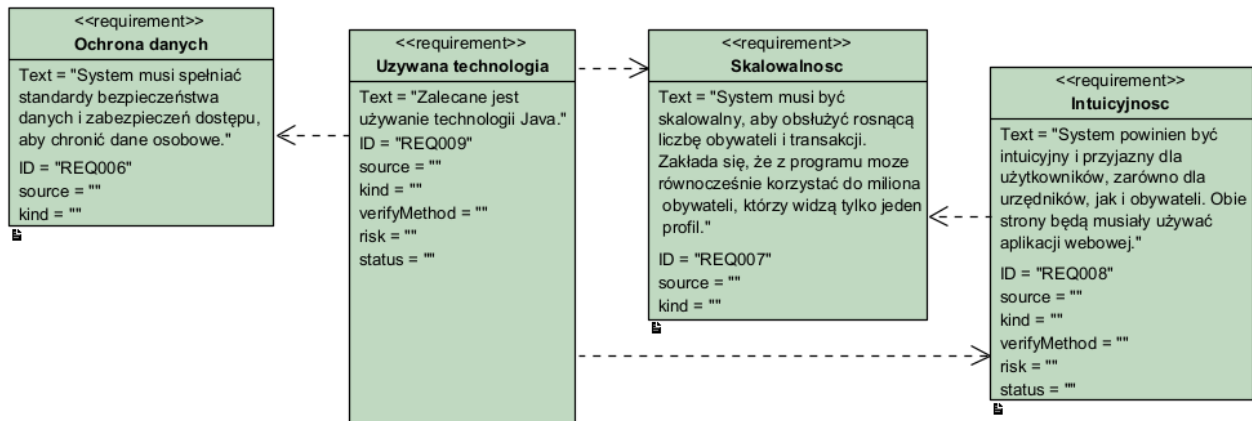
2.2. Wymagania нефункциональные

1. System musi spełniać standardy bezpieczeństwa danych i zabezpieczeń dostępu, aby chronić dane osobowe.
2. System musi być skalowalny, aby obsłużyć rosnącą liczbę obywateli i transakcji. Zakłada się, że z programu może równocześnie korzystać do miliona obywateli, którzy widzą tylko jeden profil.
3. System powinien być intuicyjny i przyjazny dla użytkowników, zarówno dla urzędników, jak i obywateli. Obie strony będą musiały używać aplikacji webowej.
4. Zaleca się stosowanie technologii Java.

3. Diagram wymagań funkcjonalnych



4. Diagram wymagań niefunkcyjnych



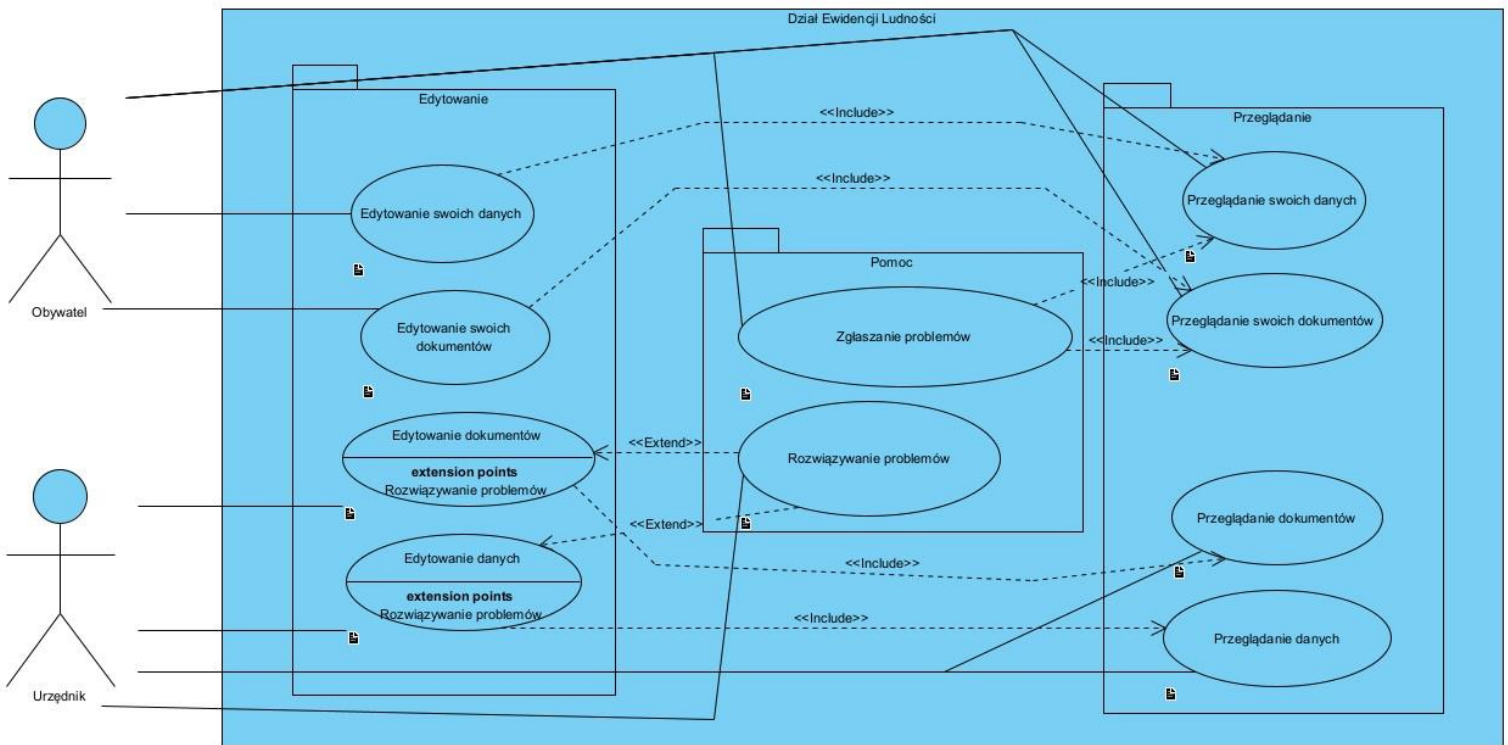
5. Przykład specyfikacji przypadków użycia za pomocą diagramu przypadków użycia (DPU)

Specyfikacja przypadków użycia jest nieodłącznym elementem procesu analizy wymagań w inżynierii oprogramowania. Wykorzystanie narzędzi takich jak Diagramy Przypadków Użycia (DPU) ułatwia precyzyjne i zrozumiałe przedstawienie oczekiwań użytkowników oraz zachowań systemu. Omówimy identyfikację aktorów, definiowanie przypadków użycia oraz ich atrybuty i związki.

1. Przykład definicji aktorów

Aktor	Opis	Przypadek użycia
Obywatel	Obywatel korzystając z przeglądarki może sprawdzać i edytować swoje dane oraz dokumenty. W razie problemów lub braku uprawnień może zgłosić problem do urzędnika.	PU Edytowanie swoich Danych <include> Przeglądanie swoich danych, PU Edytowanie swoich dokumentów <include> Przeglądanie swoich dokumentów, PU Zgłaszanie problemów <include> Przeglądanie swoich danych, PU Zgłaszanie problemów <include> Przeglądanie swoich dokumentów
Urzędnik	Urzędnik rozwiązuje problemy zgłoszone przez obywateli.	Rozwiązywanie problemów <extend> Edytowanie dokumentów <include> Przeglądanie dokumentów, Rozwiązywanie problemów <extend> Edytowanie danych <include> Przeglądanie danych

2. Diagram DPU



6. Przykłady opisu przypadków użycia (wraz z definicją scenariuszy) (PU – przypadek użycia)

Przypadek Użycia (PU): Edytowanie swoich Danych

Opis:

Cel: Użytkownik chce zmienić informacje w swoim profilu.

Warunek Wstępny: Użytkownik rozpoczął przeglądanie swoich danych.

Warunek Końcowy: Zaktualizowane informacje w profilu użytkownika.

Przebieg:

Użytkownik uruchamia tryb edycji danych osobowych z PU "Przeglądanie swoich danych"

Dokonuje żądanych zmian w wybranych atrybutach, a następnie potwierdza wprowadzone modyfikacje.

Baza danych z danymi użytkownika zostaje zaktualizowana, odzwierciedlając wprowadzone zmiany.

Przypadek Użycia (PU): Edytowanie swoich Dokumentów

Opis:

Cel: Użytkownik pragnie zmienić atrybuty swoich dokumentów lub dodać/usunąć dokumenty.

Warunek Wstępny: Użytkownik otworzył swój panel dokumentów.

Warunek Końcowy: Baza danych dokumentów użytkownika jest zaktualizowana.

Przebieg:

Użytkownik otwiera swój panel dokumentów.

Wchodzi w tryb edycji dokumentów z PU "Przeglądanie swoich dokumentów"

Może usuwać lub dodawać dokumenty.

Użytkownik wybiera konkretny dokument.

Wchodzi w tryb edycji wybranego dokumentu.

Dokonuje zmian w wybranych atrybutach, jeśli to możliwe, a następnie potwierdza wprowadzone zmiany.

Baza danych z dokumentami użytkownika zostaje zaktualizowana, odzwierciedlając wprowadzone zmiany.

Przypadek Użycia (PU): Edytowanie Dokumentów

Opis:

Cel: Urzędnik modyfikuje atrybuty dokumentów dowolnego użytkownika.

Warunek Wstępny: Urzędnik znajduje użytkownika lub wybiera go z listy problemów do rozwiązania.

Warunek Końcowy: Baza danych z dokumentami użytkownika jest zaktualizowana.

Przebieg:

Urzędnik włącza edytowanie dokumentów z PU "Rozwiązywanie problemów"

Wybiera opcję edytowania dokumentów.

2.1. Może usuwać lub dodawać nowe dokumenty.

2.2. Wchodzi w wybrany dokument.

2.2.1. Dokonuje zmian w wybranych atrybutach dokumentu, a następnie potwierdza wprowadzone zmiany.

Baza danych z dokumentami użytkownika zostaje zaktualizowana, odzwierciedlając wprowadzone modyfikacje.

Przypadek Użycia (PU): Edytowanie Danych

Opis:

Cel: Urzędnik zmienia atrybuty profilu dowolnego użytkownika.

Warunek Wstępny: Urzędnik wyszukuje użytkownika lub wybiera go z listy do rozwiązania problemu.

Warunek Końcowy: Baza danych z profilami użytkowników jest zaktualizowana.

Przebieg:

Urzędnik wchodzi w edytowanie danych z PU "Rozwiązywanie problemów"

Modyfikuje wybrane atrybuty profilu użytkownika zgodnie z potrzebami.

Baza danych z profilami użytkowników zostaje zaktualizowana, aby odzwierciedlić wprowadzone zmiany.

Przypadek Użycia (PU): Zgłaszanie Problemów

Opis:

Cel: Użytkownik potrzebuje zmienić swoje dane lub dokumenty, do których nie ma uprawnień edycji.

Warunek Wstępny: Użytkownik wchodzi na stronę "Pomoc".

Warunek Końcowy: Prośba użytkownika jest przesłana do urzędnika.

Przebieg:

Użytkownik wchodzi w "Pomoc" z PU "Przeglądanie swoich danych" lub PU "Przeglądanie swoich dokumentów"

Wybiera rodzaj problemu: dane lub dokumenty.

Wybiera konkretny atrybut, który chce zmienić.

Wypełnia formularz zgłoszeniowy i przesyła prośbę do urzędnika.

Przypadek Użycia (PU): Rozwiązywanie Problemów

Opis:

Cel: Urzędnik otrzymuje i rozpatruje zgłoszenie od użytkownika.

Warunek Wstępny: Urzędnik wchodzi na stronę "Pomoc" w systemie.

Warunek Końcowy: Prośba użytkownika jest rozwiązana lub zamknięta bez dokonywania zmian.

Przebieg:

Urzędnik otwiera stronę "Pomoc" w systemie.

Przegląda listę zgłoszeń od użytkowników.

Wybiera zgłoszenie, które chce rozwiązać.

Rozpatruje prośbę i dokonuje zmian w danych użytkownika lub zamyka zgłoszenie, jeśli nie ma potrzeby dokonywania zmian.

Przypadek Użycia (PU): Przeglądanie swoich Danych Opis:

Cel: Użytkownik otwiera stronę i przegląda swoje dane.

Warunek Wstępny: Użytkownik wchodzi na stronę systemu.

Warunek Końcowy: Dane użytkownika są wyświetlane na stronie startowej.

Przebieg:

Użytkownik otwiera stronę systemu.

Dane użytkownika są wyświetlane na stronie startowej.

Przypadek Użycia (PU): Przeglądanie Własnych Dokumentów

Opis:

Cel: Użytkownik otwiera stronę i przegląda swoje dokumenty.

Warunek Wstępny: Użytkownik wchodzi na stronę systemu.

Warunek Końcowy: Dokumenty użytkownika są wyświetlane na stronie startowej.

Przebieg:

Użytkownik otwiera stronę systemu.

Dokumenty użytkownika są wyświetlane na stronie startowej.

Przypadek Użycia (PU): Przeglądanie Dokumentów

Opis:

Cel: Urzędnik potrzebuje dostępu do dokumentów użytkownika.

Warunek Wstępny: Urzędnik otwiera wyszukiwarkę użytkowników i wyszukuje użytkownika po numerze PESEL.

Warunek Końcowy: Urzędnik znajduje poszukiwane dokumenty użytkownika lub otrzymuje komunikat, jeśli użytkownik nie istnieje.

Przebieg:

Urzędnik otwiera wyszukiwarkę użytkowników i wprowadza numer PESEL użytkownika.

System wyświetla wyniki wyszukiwania.

Jeśli użytkownik istnieje, urzędnik wybiera go, a jego dokumenty zostają wyświetlone. Jeśli użytkownik nie istnieje, system zwraca odpowiedni komunikat.

Przypadek Użycia (PU): Przeglądanie Danych

Opis:

Cel: Urzędnik potrzebuje dostępu do danych użytkownika.

Warunek Wstępny: Urzędnik otwiera wyszukiwarkę użytkowników i wyszukuje użytkownika po numerze PESEL.

Warunek Końcowy: Urzędnik znajduje poszukiwane dane lub otrzymuje komunikat, jeśli użytkownik nie istnieje.

Przebieg:

Urzędnik otwiera wyszukiwarkę użytkowników i wprowadza numer PESEL użytkownika.

System wyświetla wyniki wyszukiwania.

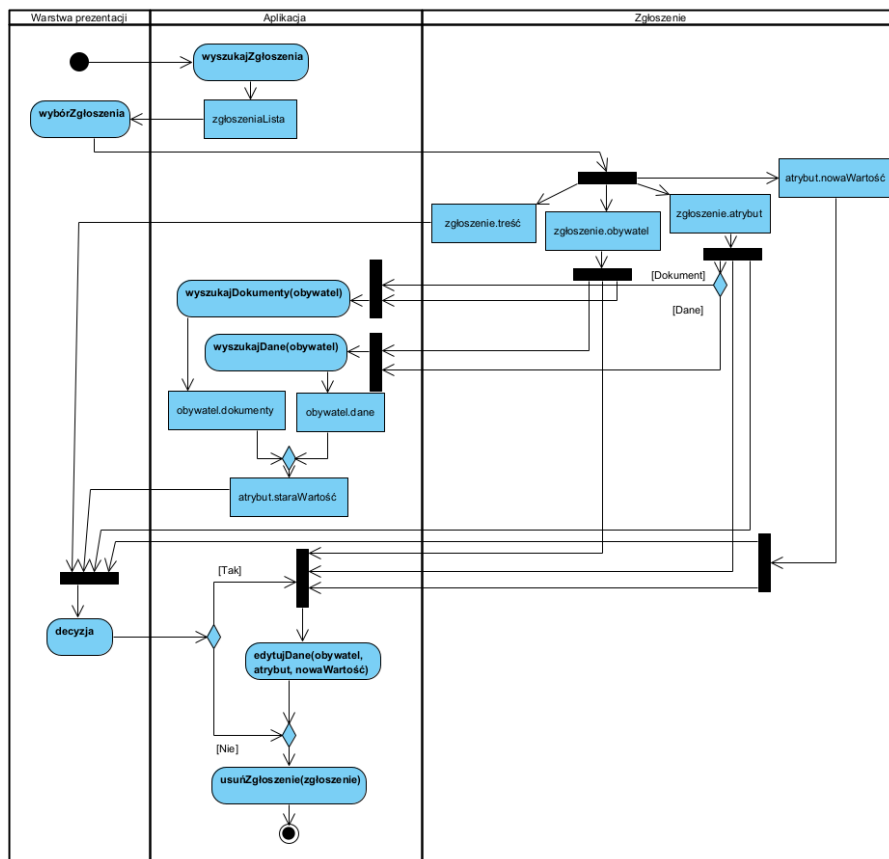
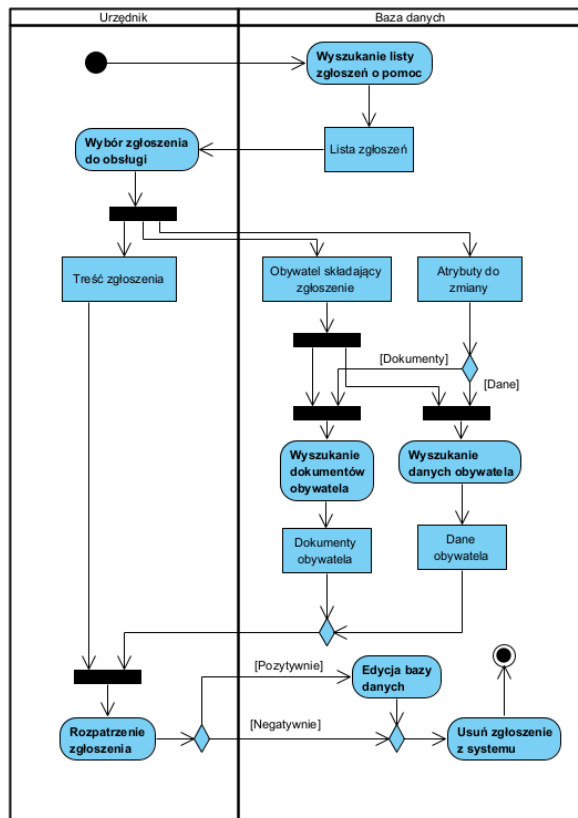
Jeśli użytkownik istnieje, urzędnik wybiera go, a jego dane zostają wyświetlone. Jeśli użytkownik nie istnieje, system zwraca odpowiedni komunikat.

7. Diagramy czynności

Kolejną częścią naszego projektu będą diagramy czynności. Diagramy czynności są skutecznym narzędziem wizualizacji i analizy procesów biznesowych oraz operacyjnych. Stanowią one graficzną reprezentację kroków wykonywanych w ramach danego zadania lub procesu, umożliwiając zrozumienie interakcji między różnymi elementami. Poprzez przyjrzenie się strukturze, elementom składowym oraz zasadom tworzenia diagramów czynności, czytelnik będzie mógł uzyskać pełniejsze zrozumienie tej skutecznej metody reprezentacji procesów, co stanowi klucz do efektywnego projektowania, analizy i optymalizacji działań w różnych obszarach działalności.

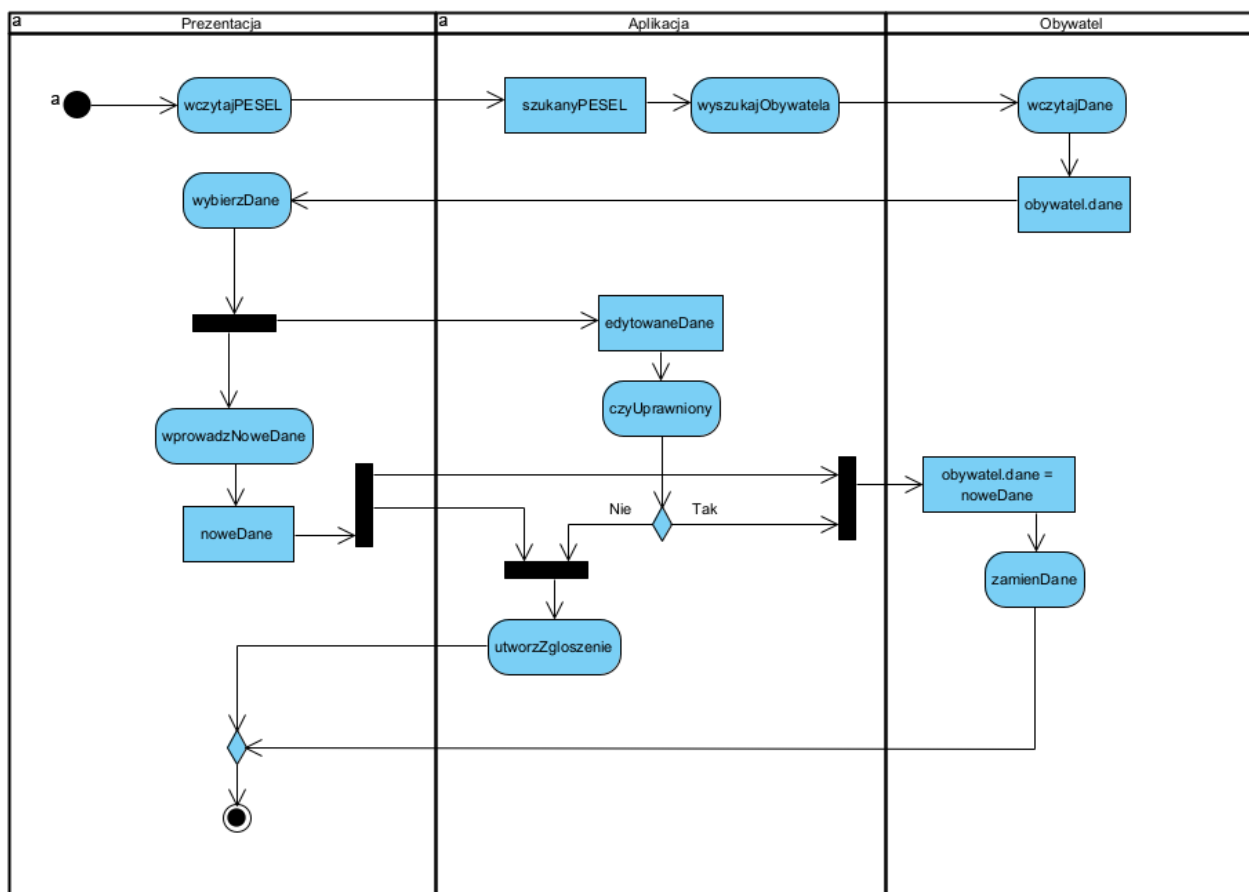
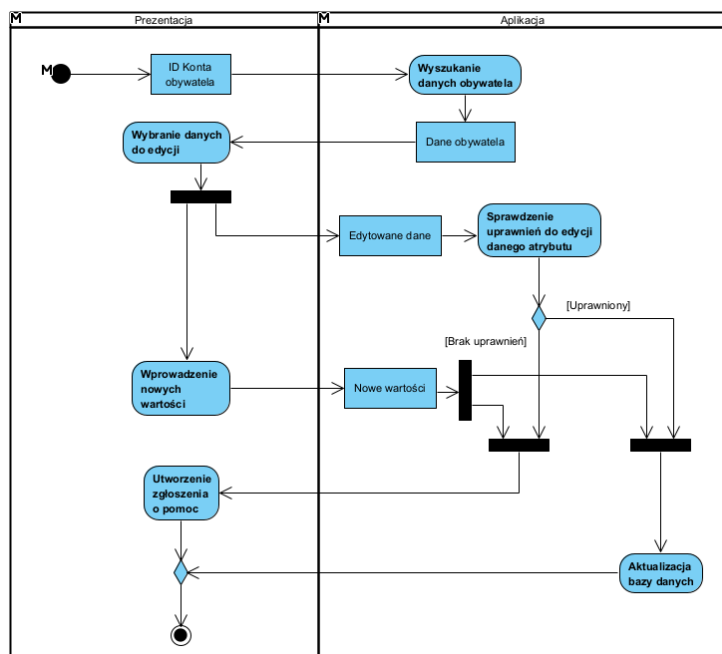
1. Rozwiązywanie problemów

Pierwszym przedstawionym przez nas diagramem będzie diagram dotyczący rozbudowanego przypadku użycia o nazwie „Rozwiązywanie problemów”. Jako pierwszy element przedstawimy jego wersję świata rzeczywistego – a jako drugi jego wersję logiczną dla komputera.



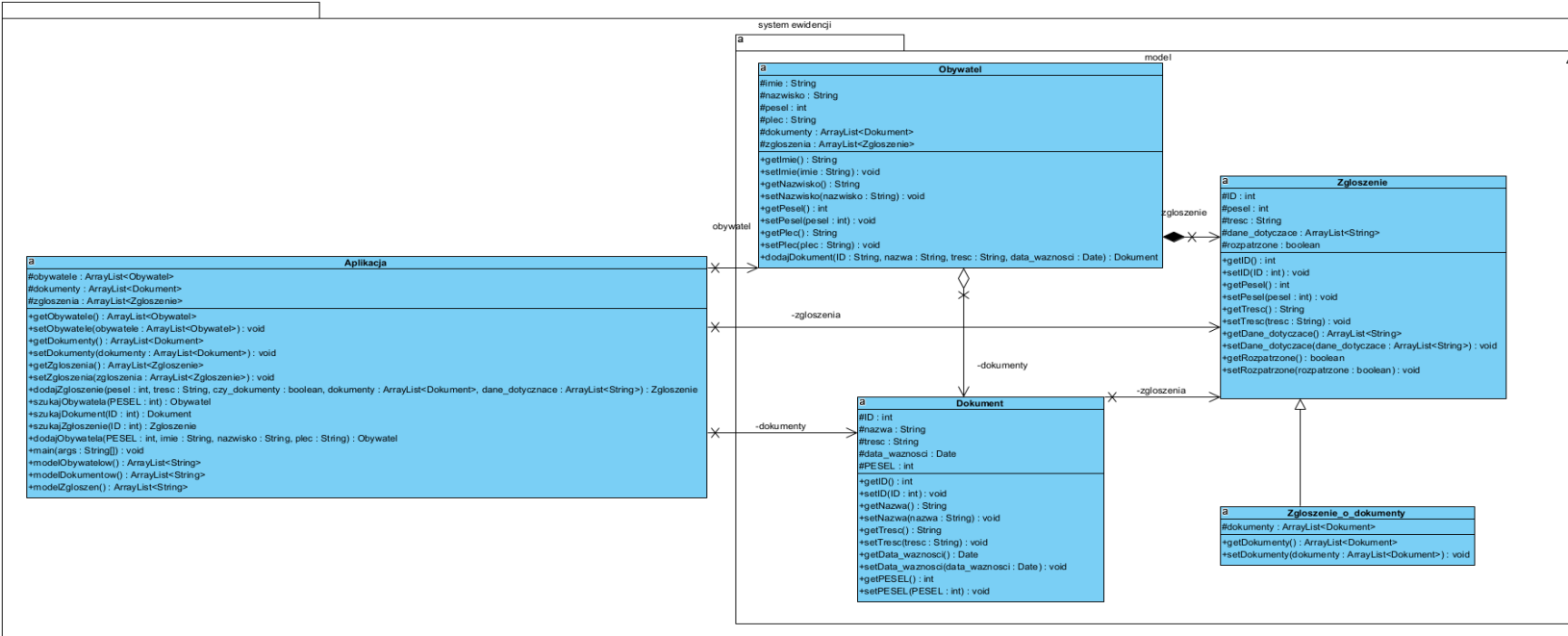
2. Edytowanie swoich danych

W podobny sposób przedstawimy drugi przypadek użycia.



8. Diagram klas

W ramach projektu, kluczowym składnikiem jest diagram klas, który precyzyjnie oddaje strukturę klas, obiektów oraz funkcji, z jakimi mamy do czynienia. W naszym podejściu postanowiliśmy rozdzielić je na pięć klas głównych: "aplikacja", "obywatel", "dokument" oraz "zgłoszenie". Ponadto, zdecydowaliśmy się wprowadzić klasę dziedziczną o nazwie "zgłoszenie_o_dokument", aby lepiej odzwierciedlić hierarchię związaną z zgłoszeniami związanymi z dokumentami. Każdą z tych klas wypełniliśmy odpowiednimi atrybutami i metodami, a całą strukturę przedstawimy w poniższym diagramie klas, który stanowi integralną część naszego kompleksowego projektu.

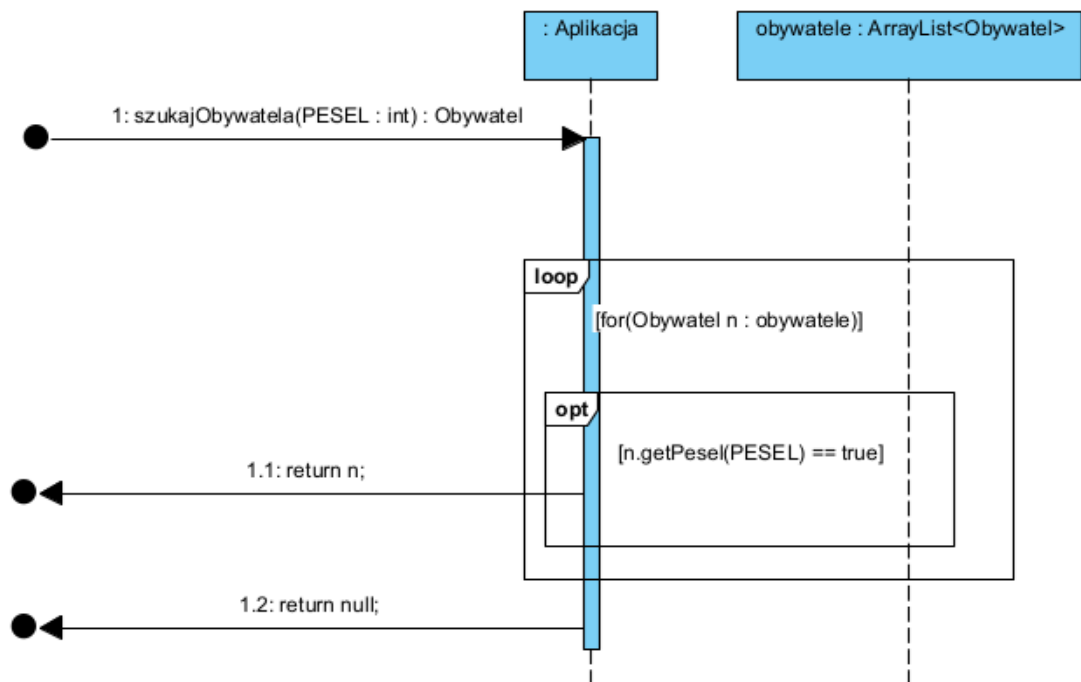


Projekt zakłada stworzenie systemu, który zarządza informacjami dotyczącymi obywateli, dokumentów oraz zgłoszeń. W klasie "Aplikacja" zaimplementowano metody umożliwiające dodawanie nowych obywateli, dokumentów oraz zgłoszeń do odpowiednich list. Dodatkowo, funkcje wyszukiwania obywateli, dokumentów i zgłoszeń zostały zaimplementowane w statycznych metodach, co pozwala na skuteczne odnajdywanie danych w kolekcjach. W zależności od typu zgłoszenia (czy wymaga ono dokumentów), tworzony jest odpowiedni obiekt klasy "Zgłoszenie" lub jej dziedziczącej klasy "Zgłoszenie_o_dokumenty". Dzięki temu hierarchia klas pozwala na elastyczne zarządzanie różnymi rodzajami zgłoszeń. Projekt skupia się na reprezentacji danych dotyczących obywateli, dokumentów oraz zgłoszeń w systemie. Klasa "Dokument" zawiera informacje o identyfikatorze, PESEL, nazwie, treści oraz dacie ważności dokumentu. Natomiast klasa "Zgłoszenie" odpowiada za przechowywanie zgłoszeń, zawierając informacje takie jak identyfikator, PESEL zgłaszającego, treść zgłoszenia, dane dotyczące oraz status rozpatrzenia. Klasa "Obywatel" natomiast agreguje dane dotyczące konkretnego obywatela, takie jak imię, nazwisko, PESEL, płeć oraz listy dokumentów i zgłoszeń przypisanych do tego obywatela. Dodatkowo, umożliwia ona dodawanie nowych dokumentów z wykorzystaniem metody "dodajDokument", która zarówno dodaje dokument do listy obywatela, jak i globalnej listy dokumentów w klasie "Aplikacja". Warto zauważyć, że identyfikatory dokumentów i zgłoszeń są generowane w sposób automatyczny, co eliminuje konieczność ręcznego ich nadawania, co może zwiększyć spójność danych w systemie.

9. Diagram klas

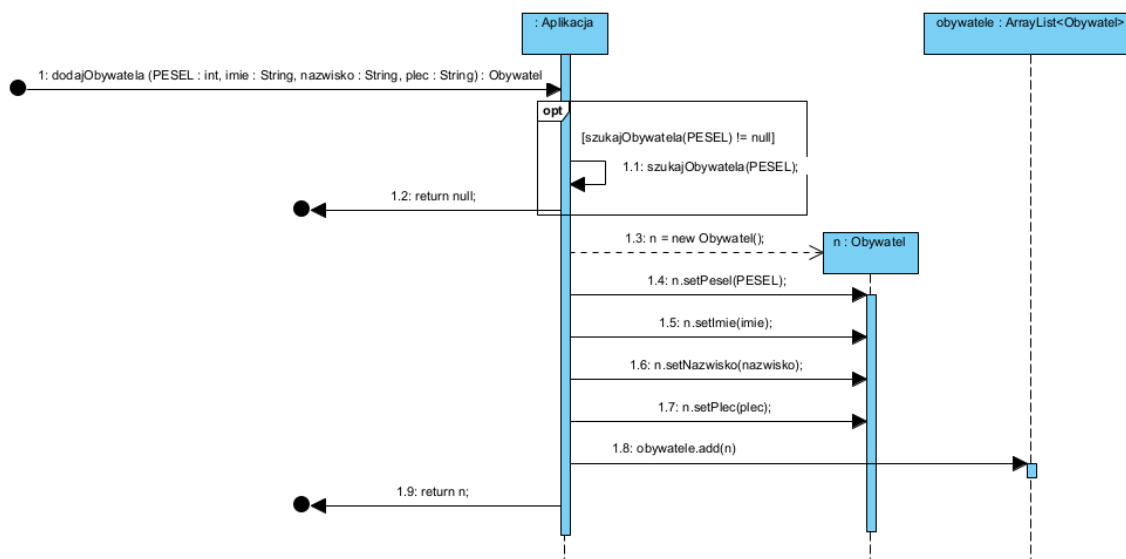
W niniejszym rozdziale skoncentrujemy się na opracowaniu diagramów sekwencji dla wybranych przypadków użycia, reprezentujących kluczowe usługi oprogramowania. Analiza tych przypadków, wsparta diagramami czynności, stanowi bazę dla definiowania operacji klas w języku Java. Ponadto, skupimy się na zastosowaniu projektowych wzorców zachowania, aby zoptymalizować strukturę systemu i zapewnić elastyczność w dostosowywaniu się do zmieniających się wymagań. W kolejnych sekcjach raportu przedstawimy diagramy sekwencji. Celem jest dostarczenie czytelnikowi kompleksowego obrazu procesu projektowania, implementacji i optymalizacji systemu.

1. Szukaj obywatela



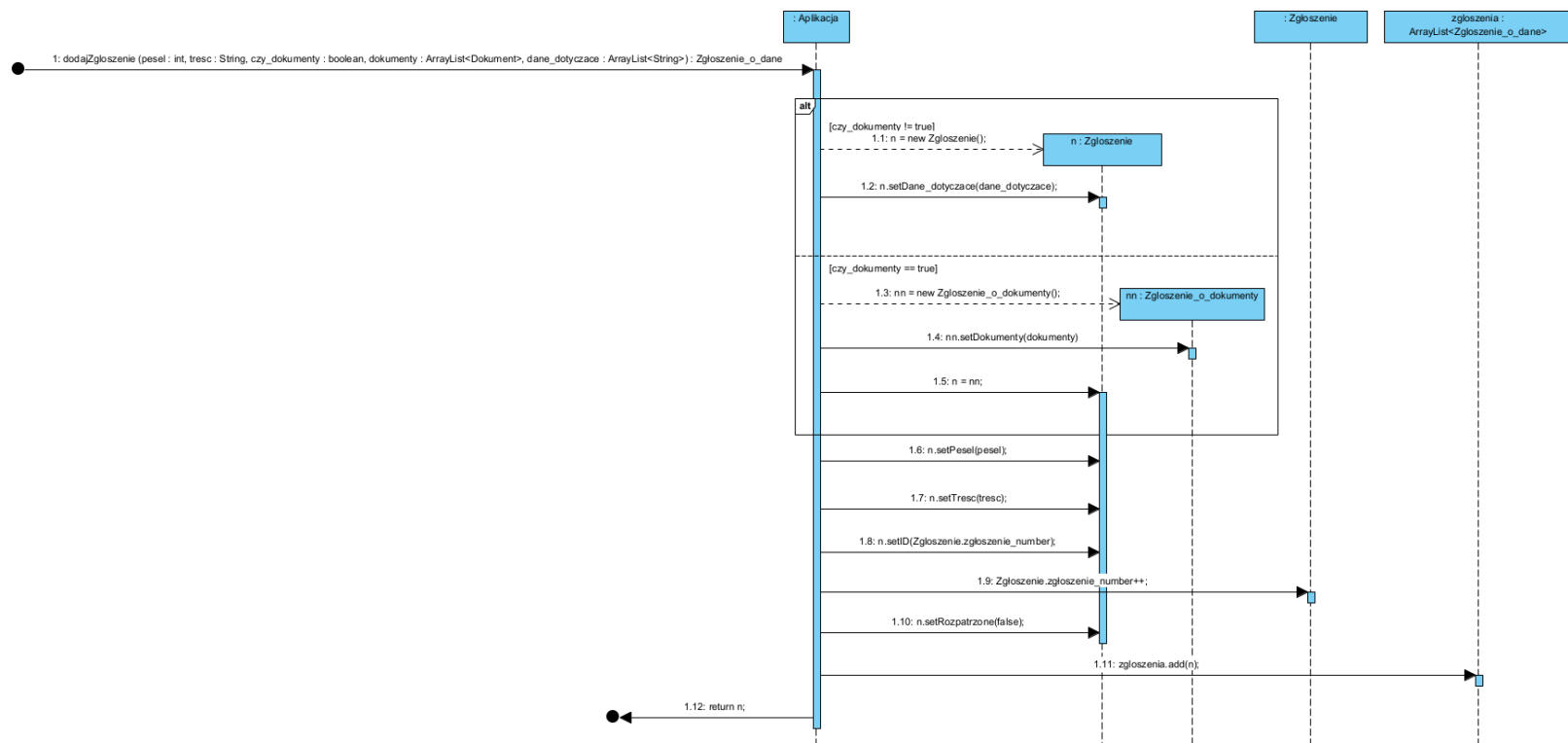
```
static public Obywatel szukajObywatela(int PESEL) {
    for(Obywatel n : obywatele) {
        if (n.getPesel() == PESEL) return n;
    }
    return null;
}
```

2. Dodaj obywatela



```
public static Obywatel dodajObywatela(int PESEL, String imie, String nazwisko, String plec) {  
    if(Aplikacja.szukajObywatela(PESEL)!=null) return null;  
    Obywatel n = new Obywatel();  
    n.setPesel(PESEL);  
    n.setImie(imie);  
    n.setNazwisko(nazwisko);  
    n.setPlec(plec);  
    obywatele.add(n);  
    return n;  
}
```

3. Dodaj zgłoszenie



```

public static Zgłoszenie dodajZgłoszenie(int pesel, String tresc,
boolean czy_dokumenty, ArrayList<Dokument> dokumenty, ArrayList<String>
dane_dotyczace) {

    Zgłoszenie n;

    if(czy_dokumenty) {

        Zgłoszenie_o_dokumenty nn = new
Zgłoszenie_o_dokumenty();

        nn.setDokumenty(dokumenty);

        n = nn;

    }

    else{

        n = new Zgłoszenie();

        n.setDane_dotyczace(dane_dotyczace);

    }

    n.setPesel(pesel);

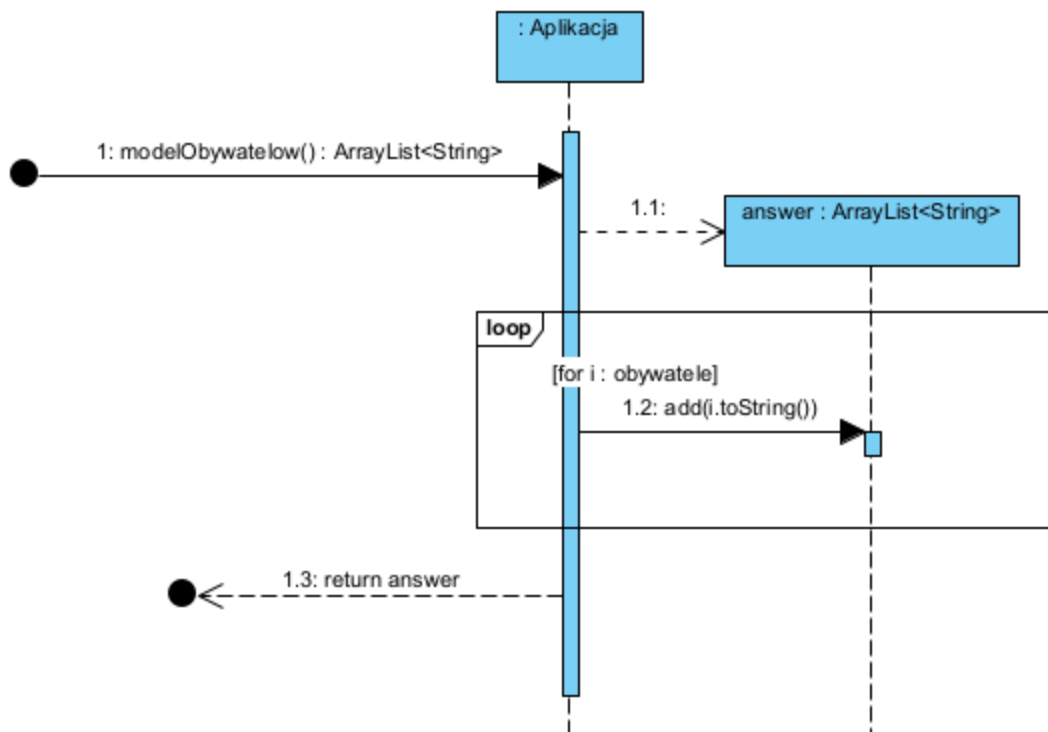
    n.setTresc(tresc);
  
```

```

        n.setID(Zgloszenie.zgłoszenie_number);
        Zgloszenie.zgłoszenie_number++;
        n.setRozpatrzone(false);
        zgloszenia.add(n);
        return n;
    }

```

4. Model Obywatelów

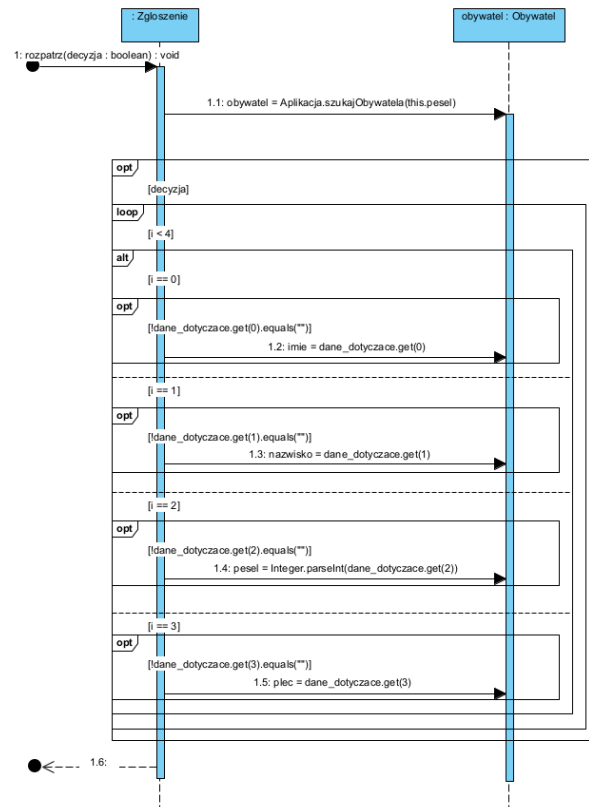


```

public static ArrayList<String> modelObywatelow() {
    ArrayList<String> answer = new ArrayList<String>();
    for (Obywatel i : obywatele) answer.add(i.toString());
    return answer;
}

```

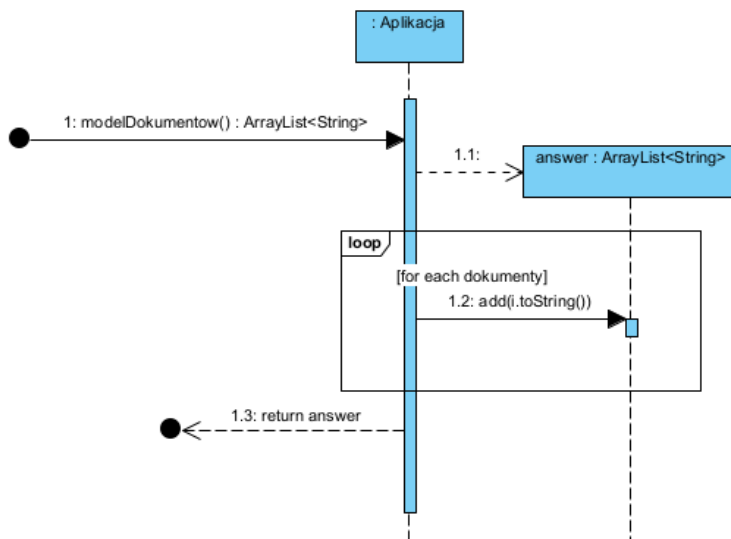

5. Rozpatrz zgłoszenia



```

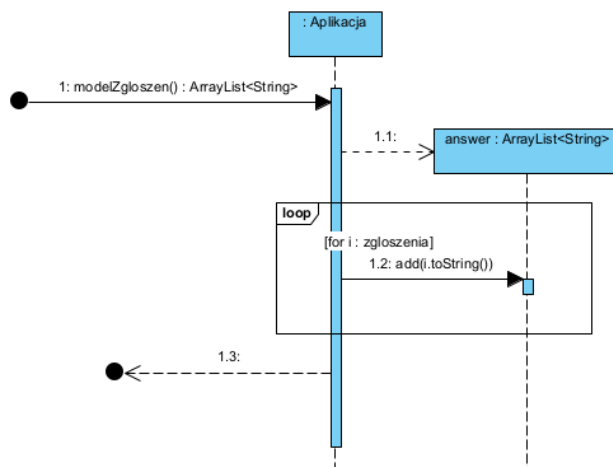
public void rozpatrz(boolean decyzja) {
    Obywatel obywatel = Aplikacja.szukajObywatela(this.pesel);
    if(decyzja) {
        for(int i = 0; i < 4; i++) {
            switch(i) {
                case 0:
                    if(!dane_dotyczace.get(0).equals(""))
                        obywatel.imie = dane_dotyczace.get(0);
                    break;
                case 1:
                    if(!dane_dotyczace.get(1).equals(""))
                        obywatel.nazwisko = dane_dotyczace.get(1);
                    break;
                case 2:
                    if(!dane_dotyczace.get(2).equals(""))
                        obywatel.pesel = Integer.parseInt(dane_dotyczace.get(2));
                    break;
                case 3:
                    if(!dane_dotyczace.get(3).equals(""))
                        obywatel.plec = dane_dotyczace.get(3);
                    break;
            }
        }
        this.rozpatrzone = true;
    }
}
  
```

6. Model dokumentów



```
public static ArrayList<String> modelDokumentow() {
    ArrayList<String> answer = new ArrayList<String>();
    for (Dokument i : dokumenty) answer.add(i.toString());
    return answer;
}
```

7. Model zgłoszeń



```
public static ArrayList<String> modelZgloszen() {
    ArrayList<String> answer = new ArrayList<String>();
    for (Zgloszenie i : zgloszenia) answer.add(i.toString());
    return answer;
}
```

10. Diagram stanów

Diagramy stanów są niezwykle istotnym narzędziem w dziedzinie inżynierii oprogramowania oraz modelowania procesów. Stanowią graficzną reprezentację różnych stanów, które obiekt lub system może przyjąć w odpowiedzi na różne zdarzenia. W kontekście konkretnego projektu, takie diagramy pozwalają nam zobaczyć, jak dokument ewoluuje w czasie, odzwierciedlając jego możliwe stany. Poprzez graficzne przedstawienie przejść między tymi stanami oraz uwzględnienie szczegółów, takich jak związki z zgłoszeniem czy możliwość przedłużenia ważności, diagramy stanów stają się nieocenionym narzędziem analizy i projektowania systemów. Umożliwiają one skupienie uwagi na kluczowych aspektach procesu, co przekłada się na zwiększoną przejrzystość, zrozumienie oraz efektywność w pracy nad danym systemem.

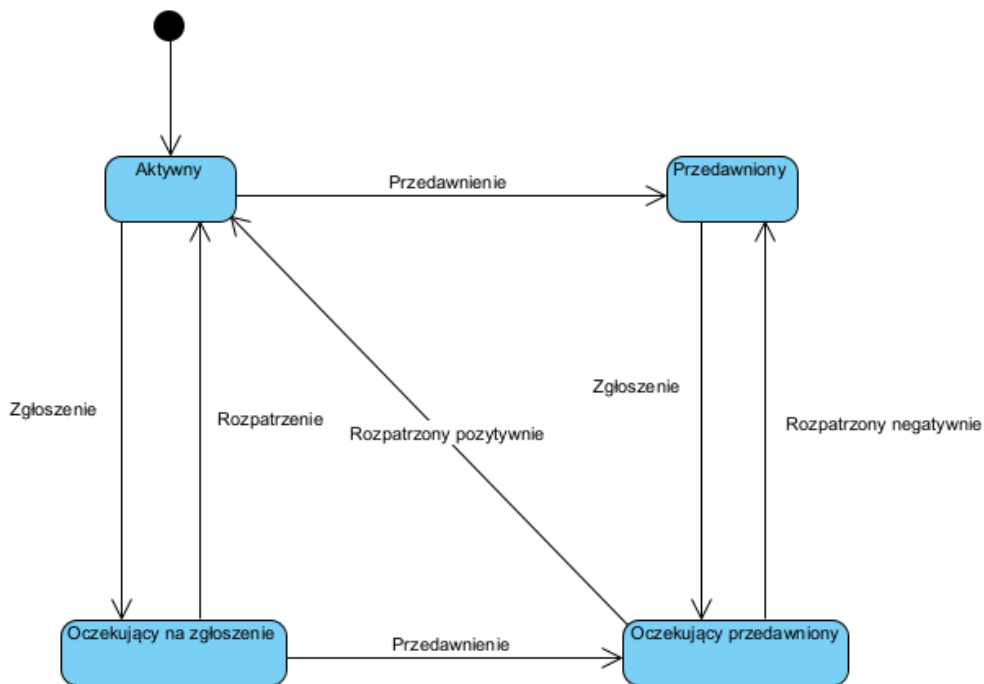


Diagram stanu przedstawiający proces dokumentu obejmuje różne stany, które mogą wpływać na jego aktualność i związki z ewentualnym zgłoszeniem. Istnieją dwa główne stany, a mianowicie "Przedawniony" oraz "Aktywny", które są odzwierciedleniem statusu dokumentu w czasie. W przypadku każdego stanu istnieje możliwość, że dokument jest powiązany z zgłoszeniem lub nie. Strzałki skierowane w prawo reprezentują możliwość przeterminowania dokumentu, niezależnie od obecności związanych zgłoszeń. Ważny jest fakt, że strzałki w dół symbolizują elastyczność procesu, gdzie użytkownik ma możliwość zgłoszenia przedłużenia dokumentu w dowolnej chwili. Natomiast strzałki skierowane w górę podkreślają, że dokument, który jeszcze nie jest przeterminowany, nie ulegnie przeterminowaniu nawet po otrzymaniu negatywnej decyzji ze strony urzędnika. Jednak, jeśli dokument jest już przeterminowany, istnieje szansa na jego przywrócenie do ważności, ale tylko w przypadku pozytywnego orzeczenia urzędnika. Takie graficzne przedstawienie procesu dokumentu pozwala klarownie zrozumieć zależności między poszczególnymi stanami oraz elastyczność i warunki, które wpływają na ich przejścia.

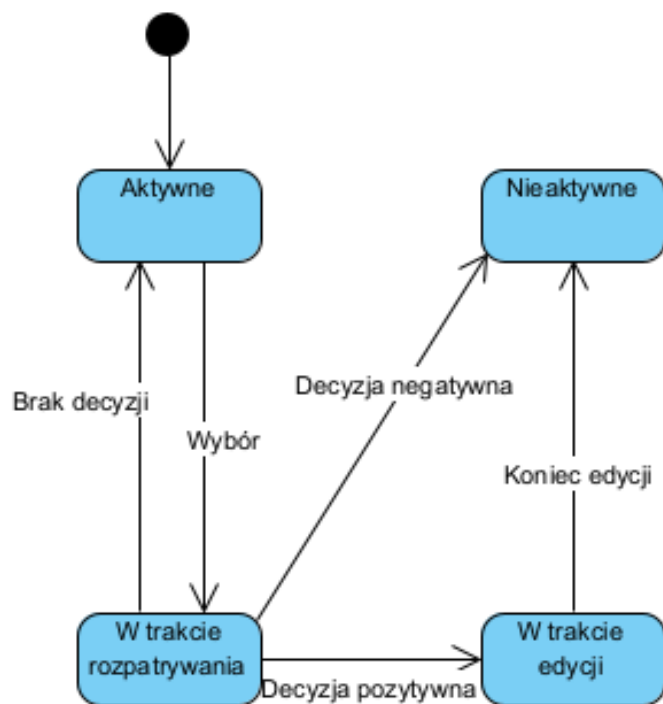


Diagram stanu przedstawia proces zgłoszenia w systemie, zaczynając od stanu "Aktywne". W tym stanie zgłoszenie jest aktywne i dostępne do wyboru przez urzędnika z listy. Istnieje możliwość, że urzędnik nie podejmie żadnych działań, co jednak nie wpływa na status zgłoszenia - pozostaje ono nadal w stanie "Aktywne". W przypadku negatywnej decyzji urzędnika, zgłoszenie zostaje wyłączone z listy, co oznacza, że nie będzie dłużej brane pod uwagę w danym kontekście. W przypadku pozytywnej decyzji urzędnika, zgłoszenie przechodzi do kolejnego stanu, czyli "W trakcie edycji". W tym stanie urzędnik odczytuje zgłoszenie, dokonuje ewentualnych zmian, a następnie przejmuje kontrolę nad wyłączeniem zgłoszenia. Po wykonaniu zmian i przetworzeniu zgłoszenia, następuje wyjście z tego stanu, a zgłoszenie zostaje wyłączone.

11. Testy jednostkowe

Testy jednostkowe są nieodłącznym narzędziem w procesie programistycznym, mającym na celu zwiększenie pewności co do poprawności kodu oraz ułatwienie utrzymania aplikacji. W Apache NetBeans, zaawansowanym środowisku programistycznym, deweloperzy korzystają z wbudowanego wsparcia dla popularnych frameworków testowych, takich jak JUnit czy TestNG. Proces tworzenia testów jednostkowych polega na izolowaniu fragmentów kodu, zwanych jednostkami, i sprawdzaniu, czy zachowują się zgodnie z oczekiwaniami. To kluczowy element praktyki Test Driven Development (TDD), gdzie testy są tworzone przed właściwą implementacją kodu. Dzięki temu podejściu programiści zyskują pewność co do poprawności i niezawodności swojego kodu, co przekłada się na łatwiejsze utrzymanie aplikacji w dłuższej perspektywie czasowej. Wsparcie frameworków testowych oraz łatwość generowania, zarządzania i uruchamiania testów w Apache NetBeans sprawiają, że proces tworzenia oprogramowania staje się bardziej efektywny i bezpieczny.

Niżej przedstawiamy zestaw testów jednostkowych dla klasy Aplikacja w języku Java przy użyciu frameworka JUnit obejmuje kilka testów funkcji. Metoda setUp jest używana do przywrócenia stanu aplikacji przed każdym testem poprzez wywołanie Aplikacja.aplikacjaClear(). Pierwszy test, testDodajObywatela, sprawdza poprawność funkcji dodajObywatela, która dodaje obywatela do systemu. Tworzy się oczekiwany obiekt Obywatel, dodaje go do systemu, a następnie porównuje wyniki, aby upewnić się, że dodanie obywatela działa zgodnie z oczekiwaniami. Drugi test, testModelZgloszen, ocenia funkcję modelZgloszen, która generuje model zgłoszeń na podstawie dodanych obywateli i zgłoszeń. Test sprawdza, czy utworzony model zgłoszeń jest zgodny z oczekiwaniami. Warto zauważyć, że używane są asercje z frameworka JUnit, takie jak assertNotNull i assertEquals, do weryfikacji poprawności wyników testów.

```
package com.mycompany.inzynieriaoprogramowania;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import java.util.Date;
import static org.junit.jupiter.api.Assertions.*;
import java.text.SimpleDateFormat;

public class AplikacjaTest {

    public AplikacjaTest() {

    }

    @BeforeAll
    public static void setUpClass() {

    }

    @AfterAll
    public static void tearDownClass() {

    }
```

```

@BeforeEach
public void setUp() {
    Aplikacja.aplikacjaClear();
}

@AfterEach
public void tearDown() {
}

@Test
public void testDodajObywatela() {
    System.out.println("dodajObywatela");
    int PESEL = 123456789;
    String imie = "Jan";
    String nazwisko = "Kowalski";
    String plec = "M";

    Obywatel expectedResult = new Obywatel();
    expectedResult.setPesel(PESEL);
    expectedResult.setImie(imie);
    expectedResult.setNazwisko(nazwisko);
    expectedResult.setPlec(plec);

    Obywatel result = Aplikacja.dodajObywatela(PESEL, imie,
nazwisko, plec);

    assertNotNull(result);
    assertEquals(expectedResult.getPesel(), result.getPesel());
    assertEquals(expectedResult.getImie(), result.getImie());
    assertEquals(expectedResult.getNazwisko(), result.getNazwisko());
    assertEquals(expectedResult.getPlec(), result.getPlec());
}

@Test
public void testModelZgloszen() {
    System.out.println("modelZgloszen");
    Aplikacja.dodajObywatela(123456789, "Jan", "Kowalski", "M");
    ArrayList<String> zmiany = new ArrayList<>();
    zmiany.add("");
    zmiany.add("Iksinski");
    zmiany.add("2");
    zmiany.add("");
    Aplikacja.dodajZgloszenie(123456789, "Testowe zgloszenie",
false, null, zmiany);

    ArrayList<String> expectedResult = new ArrayList<>();
    expectedResult.add("0 - obywatel 123456789 - zmiana danych:
nazwisko > Iksinski, PESEL > 2, - rozpatrzone: false");

    ArrayList<String> result = Aplikacja.modelZgloszen();

```

```
        assertNotNull(result);
        assertEquals(expResult, result);
    }
}
```

```
package com.mycompany.inzynieriaoprogramowania;
```

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
```

```
import java.util.Date;
```

```
public class DokumentTest {
```

```
    @Test
```

```
    public void testToString() {
```

```
        System.out.println("toString");
```

```
        Dokument instance = new Dokument();
```

```
        instance.setID(1);
```

```
        instance.setPesel(123456789);
```

```
        instance.setNazwa("Dowód");
```

```
        instance.setTresc("Treść dokumentu");
```

```
        instance.setData_waznosci(new Date());
```

```
        String expResult = "1 Dowód - obywatel 123456789, wazne do " +
instance.getData_waznosci() + ": Treść dokumentu";
```

```
        String result = instance.toString();
```

```
        assertEquals(expResult, result);
```

```
    }
```

```
    @Test
```

```
    public void testGetID() {
```

```
        System.out.println("getID");
```

```
        Dokument instance = new Dokument();
        instance.setID(1);
        int expectedResult = 1;
        int result = instance.getID();
        assertEquals(expectedResult, result);
    }
}
```

```
@Test
public void testSetID() {
    System.out.println("setID");
    int ID = 1;
    Dokument instance = new Dokument();
    instance.setID(ID);
    assertEquals(ID, instance.getID());
}
}
```

```
@Test
public void testGetPesel() {
    System.out.println("getPesel");
    Dokument instance = new Dokument();
    instance.setPesel(123456789);
    int expectedResult = 123456789;
    int result = instance.getPesel();
    assertEquals(expectedResult, result);
}
}
```

```
@Test
public void testSetPesel() {
    System.out.println("setPesel");
    int pesel = 123456789;
    Dokument instance = new Dokument();
    instance.setPesel(pesel);
}
```



```
        assertEquals(pesel, instance.getPesel());  
    }  
}
```

```
@Test  
public void testGetNazwa() {  
    System.out.println("getNazwa");  
    Dokument instance = new Dokument();  
    instance.setNazwa("Dowód");  
    String expectedResult = "Dowód";  
    String result = instance.getNazwa();  
    assertEquals(expectedResult, result);  
}
```

```
@Test  
public void testSetNazwa() {  
    System.out.println("setNazwa");  
    String nazwa = "Dowód";  
    Dokument instance = new Dokument();  
    instance.setNazwa(nazwa);  
    assertEquals(nazwa, instance.getNazwa());  
}
```

```
@Test  
public void testGetTresc() {  
    System.out.println("getTresc");  
    Dokument instance = new Dokument();  
    instance.setTresc("Treść dokumentu");  
    String expectedResult = "Treść dokumentu";  
    String result = instance.getTresc();  
    assertEquals(expectedResult, result);  
}
```

```

@Test
public void testSetTresc() {
    System.out.println("setTresc");
    String tresc = "Treść dokumentu";
    Dokument instance = new Dokument();
    instance.setTresc(tresc);
    assertEquals(tresc, instance.getTresc());
}

@Test
public void testGetData_waznosci() {
    System.out.println("getData_waznosci");
    Dokument instance = new Dokument();
    Date expResult = new Date();
    instance.setData_waznosci(expResult);
    Date result = instance.getData_waznosci();
    assertEquals(expResult, result);
}

@Test
public void testSetData_waznosci() {
    System.out.println("setData_waznosci");
    Date data_waznosci = new Date();
    Dokument instance = new Dokument();
    instance.setData_waznosci(data_waznosci);
    assertEquals(data_waznosci, instance.getData_waznosci());
}
}

```

```

package com.mycompany.inzynieriaoprogramowania;

```

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class ObywatelTest {

    @Test
    public void testToString() {
        System.out.println("toString");
        Obywatel instance = new Obywatel();
        String expectedResult = "0: Jan Kowalski, ";
        String result = instance.toString();
        assertEquals(expectedResult, result);
    }

    @Test
    public void testGetImie() {
        System.out.println("getImie");
        Obywatel instance = new Obywatel();
        instance.setImie("Jan"); // Set the expected value for the test
        String expectedResult = "Jan";
        String result = instance.getImie();
        assertEquals(expectedResult, result);
    }

    @Test
    public void testSetImie() {
        System.out.println("setImie");
        String imie = "";
        Obywatel instance = new Obywatel();
        instance.setImie(imie);
        assertEquals(imie, instance.getImie());
    }
}
```

```
@Test
public void testGetNazwisko() {
    System.out.println("getNazwisko");
    Obywatel instance = new Obywatel();
    instance.setNazwisko("Kowalski"); // Set the expected value for
the test
    String expResult = "Kowalski";
    String result = instance.getNazwisko();
    assertEquals(expResult, result);
}
```

```
@Test
public void testSetNazwisko() {
    System.out.println("setNazwisko");
    String nazwisko = "";
    Obywatel instance = new Obywatel();
    instance.setNazwisko(nazwisko);
    assertEquals(nazwisko, instance.getNazwisko());
}
```

```
@Test
public void testGetPesel() {
    System.out.println("getPesel");
    Obywatel instance = new Obywatel();
    int expResult = 0;
    int result = instance.getPesel();
    assertEquals(expResult, result);
}
```

```
@Test
public void testSetPesel() {
```

```

        System.out.println("setPesel");
        int pesel = 123456789;
        Obywatel instance = new Obywatel();
        instance.setPesel(pesel);
        assertEquals(pesel, instance.getPesel());
    }

    @Test
    public void testGetPlec() {
        System.out.println("getPlec");
        Obywatel instance = new Obywatel();
        instance.setPlec(""); // Set the expected value for the test
        String expectedResult = "";
        String result = instance.getPlec();
        assertEquals(expectedResult, result);
    }

    @Test
    public void testSetPlec() {
        System.out.println("setPlec");
        String plec = "M";
        Obywatel instance = new Obywatel();
        instance.setPlec(plec);
        assertEquals(plec, instance.getPlec());
    }

    @Test
    public void testDodajDokument() {
        System.out.println("dodajDokument");
        int ID = 1;
        String nazwa = "Dowód";
        String tresc = "Treść dokumentu";
    }

```

```

        Obywatel instance = new Obywatel();
        Dokument expResult = new Dokument();
        expResult.setID(ID);
        expResult.setPesel(instance.getPesel());
        expResult.setNazwa(nazwa);
        expResult.setTresc(tresc);

        Dokument result = instance.dodajDokument(ID, nazwa, tresc,
null);

        assertEquals(expResult.toString(), result.toString());
    }
}

```

```

package com.mycompany.inzynieriaoprogramowania;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

import java.util.ArrayList;

public class ZgloszenieTest {

```

```

    @Test
    public void testToString() {
        System.out.println("toString");
        Zgloszenie instance = new Zgloszenie();
        instance.setID(1);
        instance.setPesel(123456789);
        instance.setTresc("Testowe zgłoszenie");
        ArrayList<String> dane_dotyczace = new ArrayList<>();
    }
}

```

```
dane_dotyczace.add("Jan");
dane_dotyczace.add("Kowalski");
dane_dotyczace.add("987654321");
dane_dotyczace.add("M");
instance.setDane_dotyczace(dane_dotyczace);
instance.setRozpatrzone(false);
```

```
String expResult = "1 - obywatel 123456789 - zmiana danych: imie  
> Jan, nazwisko > Kowalski, PESEL > 987654321, plec > M, - rozpatrzone:  
false";
```

```
String result = instance.toString();
assertEquals(expResult, result);
}
```

```
@Test
```

```
public void testGetID() {
    System.out.println("getID");
    Zgloszenie instance = new Zgloszenie();
    instance.setID(1);
    int expResult = 1;
    int result = instance.getID();
    assertEquals(expResult, result);
}
```

```
@Test
```

```
public void testSetID() {
    System.out.println("setID");
    int ID = 1;
    Zgloszenie instance = new Zgloszenie();
    instance.setID(ID);
    assertEquals(ID, instance.getID());
}
```

```
@Test
public void testGetPesel() {
    System.out.println("getPesel");
    Zgloszenie instance = new Zgloszenie();
    instance.setPesel(123456789);
    int expResult = 123456789;
    int result = instance.getPesel();
    assertEquals(expResult, result);
}
```

```
@Test
public void testSetPesel() {
    System.out.println("setPesel");
    int pesel = 123456789;
    Zgloszenie instance = new Zgloszenie();
    instance.setPesel(pesel);
    assertEquals(pesel, instance.getPesel());
}
```

```
@Test
public void testGetTresc() {
    System.out.println("getTresc");
    Zgloszenie instance = new Zgloszenie();
    instance.setTresc("Testowe zgłoszenie");
    String expResult = "Testowe zgłoszenie";
    String result = instance.getTresc();
    assertEquals(expResult, result);
}
```

```
@Test
public void testSetTresc() {
```



```

        System.out.println("setTresc");
        String tresc = "Testowe zgłoszenie";
        Zgloszenie instance = new Zgloszenie();
        instance.setTresc(tresc);
        assertEquals(tresc, instance.getTresc());
    }

```

@Test

```

public void testGetDane_dotyczace() {
    System.out.println("getDane_dotyczace");
    Zgloszenie instance = new Zgloszenie();
    ArrayList<String> dane_dotyczace = new ArrayList<>();
    dane_dotyczace.add("Jan");
    dane_dotyczace.add("Kowalski");
    dane_dotyczace.add("987654321");
    dane_dotyczace.add("M");
    instance.setDane_dotyczace(dane_dotyczace);
    ArrayList<String> expectedResult = dane_dotyczace;
    ArrayList<String> result = instance.getDane_dotyczace();
    assertEquals(expectedResult, result);
}

```

@Test

```

public void testSetDane_dotyczace() {
    System.out.println("setDane_dotyczace");
    ArrayList<String> dane_dotyczace = new ArrayList<>();
    dane_dotyczace.add("Jan");
    dane_dotyczace.add("Kowalski");
    dane_dotyczace.add("987654321");
    dane_dotyczace.add("M");
    Zgloszenie instance = new Zgloszenie();
    instance.setDane_dotyczace(dane_dotyczace);
}

```

```

        assertEquals(dane_dotyczace, instance.getDane_dotyczace());
    }

    @Test
    public void testGetRozpatrzone() {
        System.out.println("getRozpatrzone");
        Zgloszenie instance = new Zgloszenie();
        instance.setRozpatrzone(true);
        boolean expResult = true;
        boolean result = instance.getRozpatrzone();
        assertEquals(expResult, result);
    }

    @Test
    public void testSetRozpatrzone() {
        System.out.println("setRozpatrzone");
        boolean rozpatrzone = true;
        Zgloszenie instance = new Zgloszenie();
        instance.setRozpatrzone(rozpatrzone);
        assertEquals(rozpatrzone, instance.getRozpatrzone());
    }

}

package com.mycompany.inzynieriaoprogramowania;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

import java.util.ArrayList;

```

```

public class Zgloszenie_o_dokumentyTest {

    @Test
    public void testToString() {
        System.out.println("toString");
        Zgloszenie_o_dokumenty instance = new Zgloszenie_o_dokumenty();
        instance.setID(1);
        instance.setPesel(123456789);
        instance.setRozpatrzone(false);

        Dokument dokument1 = new Dokument();
        dokument1.setID(1);
        Dokument dokument2 = new Dokument();
        dokument2.setID(2);

        ArrayList<Dokument> dokumenty = new ArrayList<>();
        dokumenty.add(dokument1);
        dokumenty.add(dokument2);
        instance.setDokumenty(dokumenty);

        String expResult = "1 - obywatel 123456789 - zmiana dokumentow:
1 2 - rozpatrzone: false";
        String result = instance.toString();
        assertEquals(expResult, result);
    }

    @Test
    public void testGetDokumenty() {
        System.out.println("getDokumenty");
        Zgloszenie_o_dokumenty instance = new Zgloszenie_o_dokumenty();
        Dokument dokument1 = new Dokument();
        dokument1.setID(1);

```

```

        Dokument dokument2 = new Dokument();
        dokument2.setID(2);

        ArrayList<Dokument> dokumenty = new ArrayList<>();
        dokumenty.add(dokument1);
        dokumenty.add(dokument2);
        instance.setDokumenty(dokumenty);

        ArrayList<Dokument> expResult = dokumenty;
        ArrayList<Dokument> result = instance.getDokumenty();
        assertEquals(expResult, result);
    }

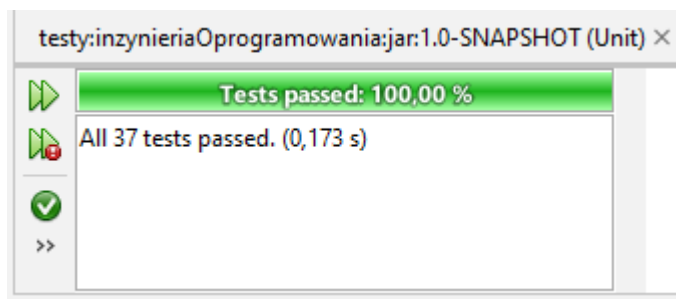
    @Test
    public void testSetDokumenty() {
        System.out.println("setDokumenty");
        ArrayList<Dokument> dokumenty = new ArrayList<>();
        Dokument dokument1 = new Dokument();
        dokument1.setID(1);
        Dokument dokument2 = new Dokument();
        dokument2.setID(2);
        dokumenty.add(dokument1);
        dokumenty.add(dokument2);

        Zgloszenie_o_dokumenty instance = new Zgloszenie_o_dokumenty();
        instance.setDokumenty(dokumenty);

        assertEquals(dokumenty, instance.getDokumenty());
    }
}

```

Po zaimplementowaniu kodu w Apache, możemy sprawdzić poprawność naszych testów. Program sprawdza osobno każdą klasę i zwraca błędy (jeżeli są) do odpowiednich metod. W tym przypadku wszystkie testy przeszły bez zarzutów.



Testowane przez nas klasy i metody to:

```
modelGloszen
dodajObywatela
getNazwa
getPesel
getTresc
getID
setID
toString
getData_waznosci
setData_waznosci
setNazwa
setPesel
setTresc
getPesel
toString
setNazvisko
getNazvisko
getImie
getPlec
dodajDokument
setImie
setPlec
setPesel
getPesel
getTresc
getID
setID
getRozpatrzone
toString
setDane_dotyczace
setRozpatrzone
getDane_dotyczace
setPesel
setTresc
toString
getDokumenty
setDokumenty
```

12. Testy akceptacyjne

Testy akceptacyjne stanowią kluczowy etap w procesie rozwoju oprogramowania, zapewniając, że system spełnia oczekiwania i wymagania użytkowników oraz interesariuszy. To rodzaj testów, które sprawdzają, czy aplikacja działa zgodnie z założeniami biznesowymi i spełnia oczekiwania klienta. Głównym celem testów akceptacyjnych jest upewnienie się, że oprogramowanie jest gotowe do wdrożenia i może być zaakceptowane do użytku przez końcowego użytkownika. W ramach tych testów sprawdzana jest funkcjonalność systemu, zgodność z wymaganiami biznesowymi oraz interakcje z użytkownikiem, co pozwala zidentyfikować potencjalne problemy przed uruchomieniem systemu w produkcji. FitNesse to framework do testowania akceptacyjnego, który umożliwia łatwe tworzenie i zarządzanie testami, integrując je z dokumentacją projektową. Dzięki FitNesse możliwe jest skoncentrowanie się nie tylko na technicznej poprawności kodu, ale również na zapewnieniu zgodności z oczekiwaniami użytkownika końcowego. Narzędzie to pozwala na tworzenie testów w formie zrozumiałej dla osób niefachowych, co sprzyja lepszemu zrozumieniu wymagań biznesowych i interakcji z systemem. W ten sposób FitNesse staje się skutecznym narzędziem wspierającym proces testowania akceptacyjnego, który ma kluczowe znaczenie dla zapewnienia jakości oprogramowania przed wdrożeniem.

Rozpoczęliśmy zadanie od stworzenia odpowiedniej strony, która będzie obsługiwać klasę Obywatel. Użyliśmy kodu z testów jednostkowych, zgodnie z instrukcją.

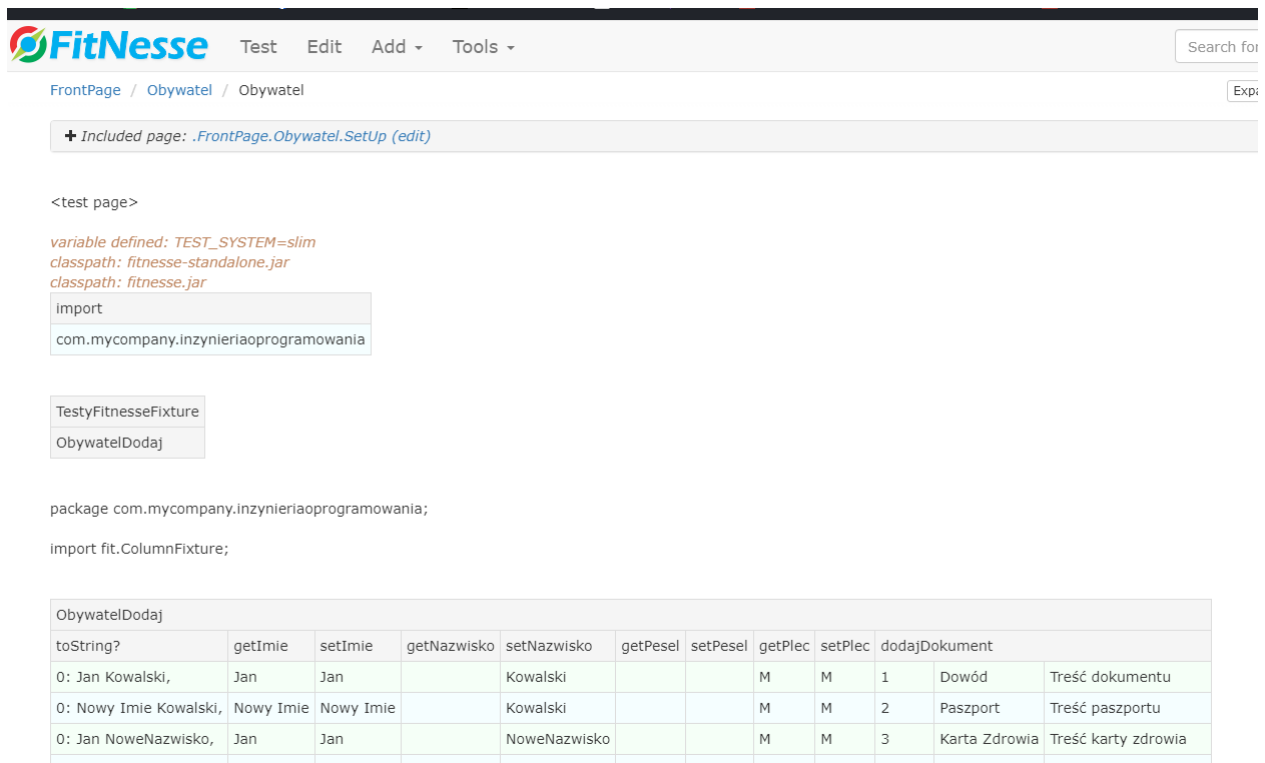
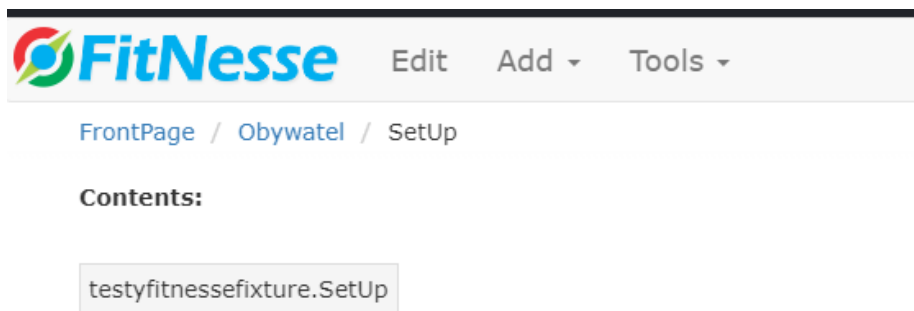
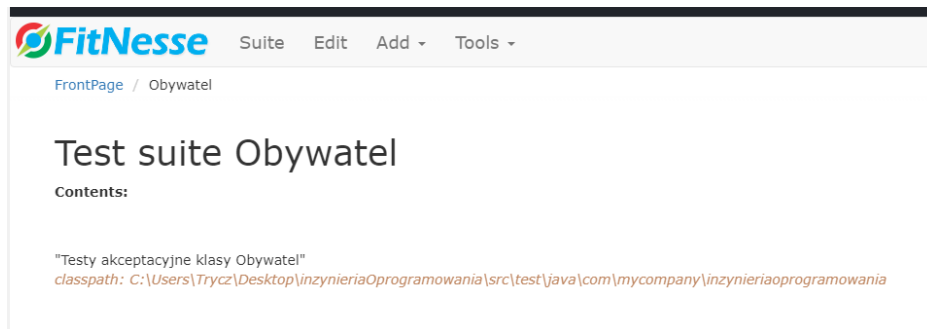
Welcome to Obywatel!

Testy akceptacyjne klasy Obywatel

To add your first "page", click the [Edit](#) button and add a [WikiWord](#) to the page.

To Learn More...	
User Guide	<i>Answer the rest of your questions here.</i>
A Two-Minute Example	<i>A brief example. Read this one next.</i>
Acceptance Tests	<i>FitNesse's suite of Acceptance Tests</i>
Release Notes	<i>Find out about FitNesse's new features</i>

Po zapoznaniu się z działaniem aplikacji, byliśmy w stanie stworzyć stronę, która będzie zajmowała się testami, co przedstawiamy na poniższych zrzutach.



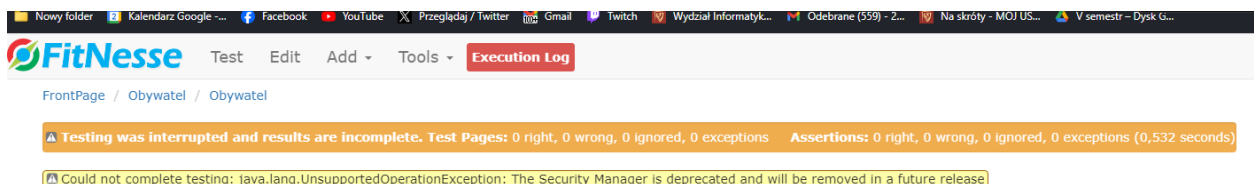
Test suite Obywatel

Contents:

- [Obywatel +](#)
- [Set Up](#)

"Testy akceptacyjne klasy Obywatel"

classpath: C:\Users\Trycz\Desktop\inzynieriaOprogramowania\src\test\java\com\mycompany\inzynieriaoprogramowania



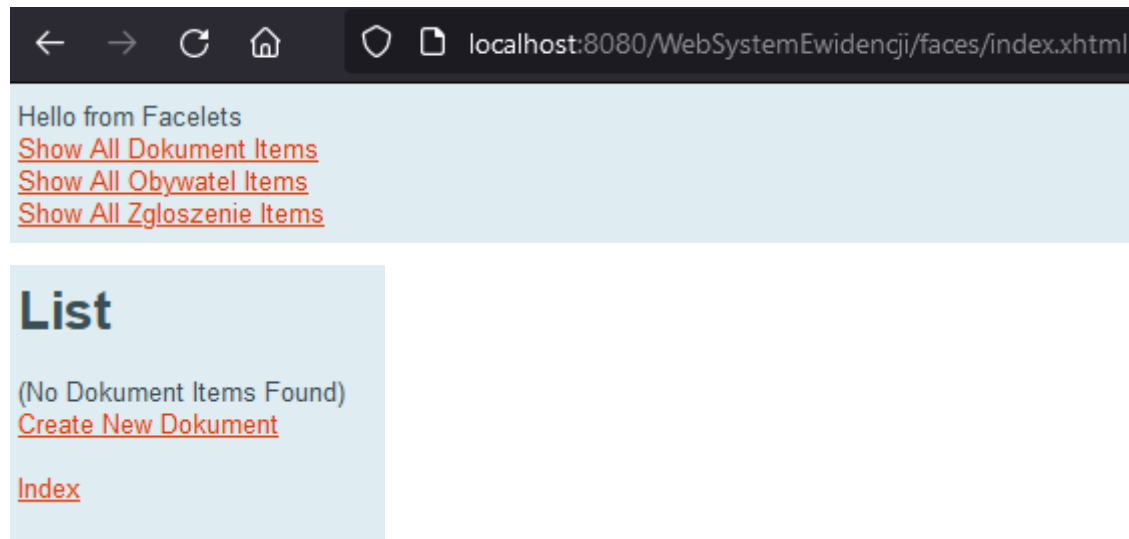
The screenshot shows the FitNesse interface with the 'Execution Log' tab selected. The log displays a message: 'Testing was interrupted and results are incomplete. Test Pages: 0 right, 0 wrong, 0 ignored, 0 exceptions Assertions: 0 right, 0 wrong, 0 ignored, 0 exceptions (0,532 seconds)'. Below this, a detailed error message is shown: 'Could not complete testing: java.lang.UnsupportedOperationException: The Security Manager is deprecated and will be removed in a future release'.

Niestety, podczas procesu tworzenia testów akceptacyjnych, napotkaliśmy na problem, którego wynik został uwieczniony na załączonym zrzutku ekranu. Ten błąd okazał się być nie do pokonania mimo wielu prób różnych strategii naprawczych. Szereg eksperymentów i testów został przeprowadzony, aby zidentyfikować i rozwiązać przyczynę tego problemu, jednak wszystkie podejścia okazały się nieskuteczne. W związku z tym, niestety, nie byliśmy w stanie stworzyć działających testów akceptacyjnych.

13. Testy funkcjonalne

Testy funkcjonalne odgrywają kluczową rolę w zapewnieniu poprawności działania oprogramowania poprzez sprawdzenie, czy poszczególne funkcje oraz zachowania aplikacji działają zgodnie z oczekiwaniami. Ich głównym celem jest weryfikacja, czy system spełnia określone wymagania funkcjonalne i zachowuje się zgodnie z założeniami projektowymi. W ramach tych testów sprawdzana jest m.in. poprawność interakcji użytkownika z interfejsem użytkownika, prawidłowe działanie funkcji systemowych oraz reakcja na różne scenariusze i dane wejściowe. Aby skutecznie przeprowadzić testy funkcjonalne, zdecydowaliśmy się wykorzystać narzędzie Selenium. Selenium jest popularnym narzędziem do automatyzacji testów, które umożliwia tworzenie skryptów testowych w języku programowania, takich jak Java czy Python, oraz ich uruchamianie w przeglądarkach internetowych. Dzięki temu narzędziu będziemy w stanie efektywnie przeprowadzić testy funkcjonalne, co pozwoli nam zweryfikować poprawność działania oprogramowania przed jego wdrożeniem.

Rozpoczęliśmy od poprawnego zainicjalizowania aplikacji naszego systemu ewidencji. W jej skład wchodziła pusta lista, która zaraz będzie wykorzystana.



Stworzyliśmy okienko, dzięki któremu program Selenium mógł w łatwy sposób testować naszą klasę Java.

Create New Dokument

Id:

Nazwa:

Tresc:

DataWaznosci:

Pesel:

[Save](#)
[Show All Dokument Items](#)
[Index](#)

Create New Dokument

Dokument was successfully created.

Id:

Nazwa:

Tresc:

DataWaznosci:

Pesel:

[Save](#)
[Show All Dokument Items](#)
[Index](#)

List

1..1/1

Id	Nazwa	Tresc	DataWaznosci	Pesel	
30	Sele	Nium	10/07/2025	23936877	View Edit Destroy

[Create New Dokument](#)

[Index](#)

Edit Dokument

Id:

Nazwa:

Tresc:

DataWaznosci:

Pesel:

[Save](#)

[View](#)

[Show All Dokument Items](#)

[Index](#)

View

Dokument was successfully updated.

Id: 30

Nazwa: Sele

Tresc: NiumIDE

DataWaznosci: 10/07/2025

Pesel: 23936877

[Destroy](#)

[Edit](#)

[Create New Dokument](#)

[Show All Dokument Items](#)

[Index](#)

List

1..1/1

Id	Nazwa	Tresc	DataWaznosci	Pesel	
30	Sele	NiumlDE	10/06/2025	23936877	View Edit Destroy

[Create New Dokument](#)

[Index](#)

Po wykonaniu wszystkich testów stwierdziliśmy, iż Selenium poprawnie wykonał oczekiwaną przez nas pracę i potwierdził testami działanie naszej aplikacji. Poniżej przedstawimy kroki programu Selenium, którym sprawdzał funkcjonalność i porównywał z UI.

	Command	Target	Value
1	<i>open</i>	http://localhost:8080/WebSystemEwidencji/	
2	<i>set window size</i>	798x1008	
3	<i>click</i>	linkText=Show All Dokument Items	
4	<i>click</i>	linkText=Create New Dokument	
5	<i>assert title</i>	Create New Dokument	
6	<i>assert text</i>	css=h1	Create New Dokument
7	<i>verify text</i>	css=tr:nth-child(1) label	Id:
8	<i>verify text</i>	css=tr:nth-child(2) label	Nazwa:
9	<i>verify text</i>	css=tr:nth-child(3) label	Tresc:
10	<i>verify text</i>	css=tr:nth-child(4) label	DataWaznosci:
11	<i>verify text</i>	css=tr:nth-child(5) label	Pesel:

12	click	id=j_idt12:id	
13	type	id=j_idt12:id	1
14	click	id=j_idt12:nazwa	
15	type	id=j_idt12:nazwa	Sele
16	click	id=j_idt12:tresc	
17	type	id=j_idt12:tresc	Nium
18	click	id=j_idt12:dataWaznosci	
19	type	id=j_idt12:dataWaznosci	10/07/2025
20	click	id=j_idt12:pesel	
21	select	id=j_idt12:pesel	label=entities.Obywatel[pesel=23936877]
22	click	css=option:nth-child(6)	
23	click	linkText=Save	
24	verify text	css=table:nth-child(1) td	Dokument was successfull y created.
25	click	linkText=Show All Dokumen t Items	
26	verify text	css=td:nth-child(2)	Sele
27	verify text	css=td:nth-child(3)	Nium
28	verify text	css=td:nth-child(4)	10/07/2025
29	verify text	css=td:nth-child(5)	23936877

	Command	Target	Value
1	open	http://localhost:8080/WebSystemEwidencji/	
2	set window size	798x1008	
3	click	linkText=Show All Dokument Items	
4	verify text	css=td:nth-child(3)	Nium
5	click	linkText=Edit	
6	assert title	Edit Dokument	
7	verify text	css=h1	Edit Dokument
8	verify text	css=tr:nth-child(1) label	Id:
9	verify text	css=tr:nth-child(2) label	Nazwa:
10	verify text	css=tr:nth-child(3) label	Tresc:
11	verify text	css=tr:nth-child(4) label	DataWaznosci:
12	verify text	css=tr:nth-child(5) label	Pesel:
13	click	id=j_idt12:tresc	
14	type	id=j_idt12:tresc	NiumIDE
15	click	linkText=Save	
16	click	linkText=Show All Dokument Items	
17	verify text	css=td:nth-child(3)	NiumIDE
18	click	linkText=View	
19	verify text	css=tr:nth-child(3) span	NiumIDE