Studio Projektowe

Projekt:

IDE RE

AGH Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Autorzy
Michał Burda
Radosław Barszczak

Spis treści

---

1. Wprowadzenie

Projekt zakłada stworzenie systemu umożliwiającego automatyczne generowanie kodu oraz diagramów aktywności na podstawie scenariuszy przypadków użycia. W ramach realizacji wykorzystano język programowania **Java**, narzędzie do parsowania gramatyk **ANTLR**, bibliotekę **PlantUML** do tworzenia diagramów UML oraz **Graphviz**, który wspomaga wizualizację struktur. System analizuje dostarczone scenariusze przypadków użycia, przekształcając je w kod źródłowy w różnych językach programowania oraz generując przejrzyste diagramy aktywności, co wspomaga proces projektowania i dokumentacji systemów informatycznych.

---

2. Technologia i narzędzia

W projekcie wykorzystano następujące technologie i narzędzia:

- **ANTLR 4** – generowanie analizatorów składniowych i leksykalnych na podstawie definicji gramatyki.
- **Java 17** – główny język programowania projektu.
- **PlantUML** – generowanie diagramów
- **Graphviz** – konwertowanie diagramów do plików png
- **Maven** – narzędzia do zarządzania zależnościami i budowy projektu.

- **Git** – system kontroli wersji.

---

3. Struktura Plików Projektu

2. Struktura katalogu `src/main`:

- **`java/org/example`** – Główna przestrzeń nazw dla kodu aplikacji.
  - `DiagramGenerator.java` – Klasa odpowiedzialna za generowanie diagramów aktywności na podstawie scenariuszy przypadków użycia.
  - `Main.java` – Główna klasa aplikacji, która zarządza procesem generowania kodu i diagramów.
- **`output`** – Katalog przechowujący wyniki działania aplikacji.
  - **`code`** – katalog przechowujący wygenerowany kod źródłowy w różnych językach programowania.
  - **`diagrams`** – Katalog zawierający wygenerowane diagramy aktywności:
    - Pliki `.png` – Grafiki przedstawiające diagramy aktywności dla poszczególnych scenariuszy.
    - Pliki `.puml` – Pliki źródłowe diagramów w składni PlantUML.
- **`parser`** – Katalog zawierający parser składniowy do przetwarzania scenariuszy przypadków użycia. Ten katalog zawiera również generowanie kodu w pliku
- **`resources`** – Katalog przechowujący plik potrzebny do PlantUML.
- **`scenarios`** – Katalog zawierający scenariusze przypadków użycia zapisane w plikach tekstowych (`.txt`), które są analizowane przez aplikację.
  - `scenario1.txt` – Przykładowy scenariusz przypadków użycia.

---

4. Opis działania aplikacji

4.1 Generowanie Diagramów

- Aplikacja wczytuje pliki wejściowe.
- Analizator scenariusza zapisuje wszystkie aktywności z głównego i alternatywnego scenariusza.
- Generator tworzy diagram aktywności.
- Aplikacja zapisuje diagramy w odpowiednich plikach .puml i .png

4.2 Generowanie kodu na podstawie scenariuszy

- Aplikacja wczytuje pliki wejściowe.
- Analizator leksykalny dzieli treść na tokeny.
- Parser przetwarza strukturę składniową na podstawie reguł gramatycznych.
- Listener tworzy kod w czterech różnych językach programowania(Java, C#, C++, Python)
- Aplikacja zapisuje wygenerowany kod w odpowiednich plikach.

4.3 Wyjaśnienie gramatyki

Najważniejszy jest główny i alternatywny przepływ i to one odpowiadają za wygląd wygenerowanego kodu i diagramu. Gramatyka i tokeny znajdują się w plikach odpowiednio
ucGrammar.g4 i ucTokens.g4. Do gramatyki dodane są też tagi <aktywnosc> … i <parametr> … </p> które zostały użyte w celu oznaczenia aktywności i parametrów, z których później będą tworzone diagramy i kod przypadku użycia.

4.4 Przykłady wygenerowanych diagramów i kodów

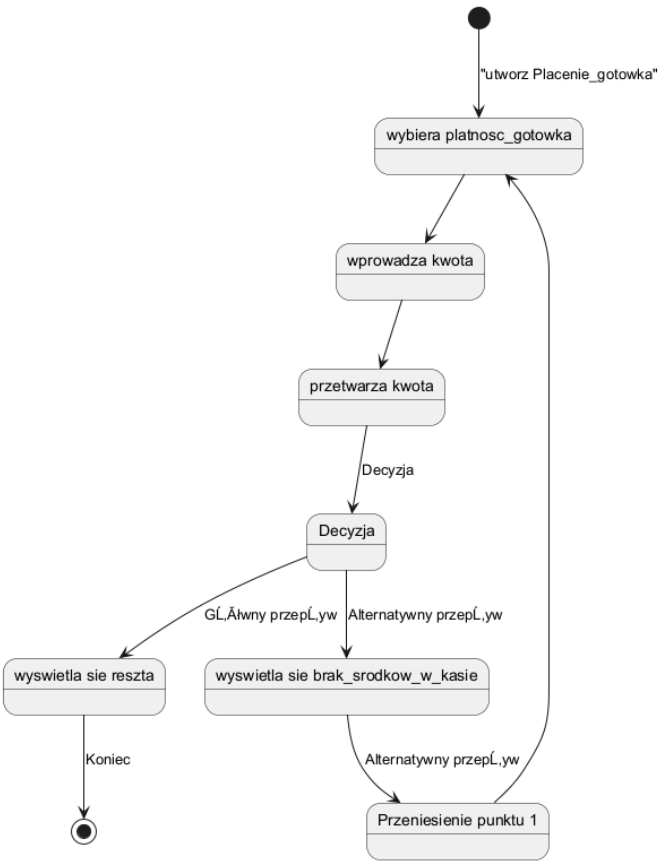
Przykładowy Scenariusz nr 1

```
UC 1 utworz Placenie_gotowka
Opis: Pracownik tworzy Formularz_platnosci. Klient oraz System_platyniczy dostarcza lub weryfikuje istotne informacje.
Warunki wstepne: 1. Tankowanie jest w stanie Zakonczony. Uruchomiony przez UC 11 utworz Placenie.
Aktorzy: Pracownik (typ: tworca), Klient (typ: towarzyszacy)
Glowny Przeplyw:
1. Pracownik <aktywnosc> wybiera </a> dane z listy <parametr> platnosc_gotowka </p>.
2. Pracownik <aktywnosc> wprowadza </a> parametr Formularza_platnosci <parametr> kwota </p>.
3. System <aktywnosc> przetwarza </a> wprowadzone dane Formularza_platnosci <parametr> kwota </p>.
4. Pracownikowi <aktywnosc> wyswietla sie </a> <parametr> reszta </p>.
Alternatywny Przeplyw:
4.1. Pracownikowi <aktywnosc> wyswietla sie </a> <parametr> brak_srodkow_w_kasie </p>.
4.2. <aktywnosc> Przeniesienie </a> Pracownika do <parametr> punktu 1 </p>.
Wyjatki: 2.1. <parametr> kwota </p> z Formularza_platnosci jest niepoprawna: Nieprawidlowe dane wejsciowe.
Warunki koncowe: Formularz_platnosci jest w stanie Zakonczony.
```

Wygenerowany kod w języku Java

```java
public class Placenie_gotowka {

    public void wybiera(String argument){}

    public void wprowadza(String argument){}

    public void przetwarza(String argument){}

    public void wyswietla_sie(String argument){}

    public void Przeniesienie(String argument){}

    public void run(){

        String platnosc_gotowka, kwota, reszta, brak_srodkow_w_kasie, punktu_1;

        wybiera(platnosc_gotowka);
        wprowadza(kwota);
        przetwarza(kwota);

        if(brak_srodkow_w_kasie){
            wyswietla sie(brak_srodkow_w_kasie);
            Przeniesienie(punktu_1);
            return;
        }
        wyswietla_sie(reszta);
    }
}
```

Wygenerowany diagram aktywności



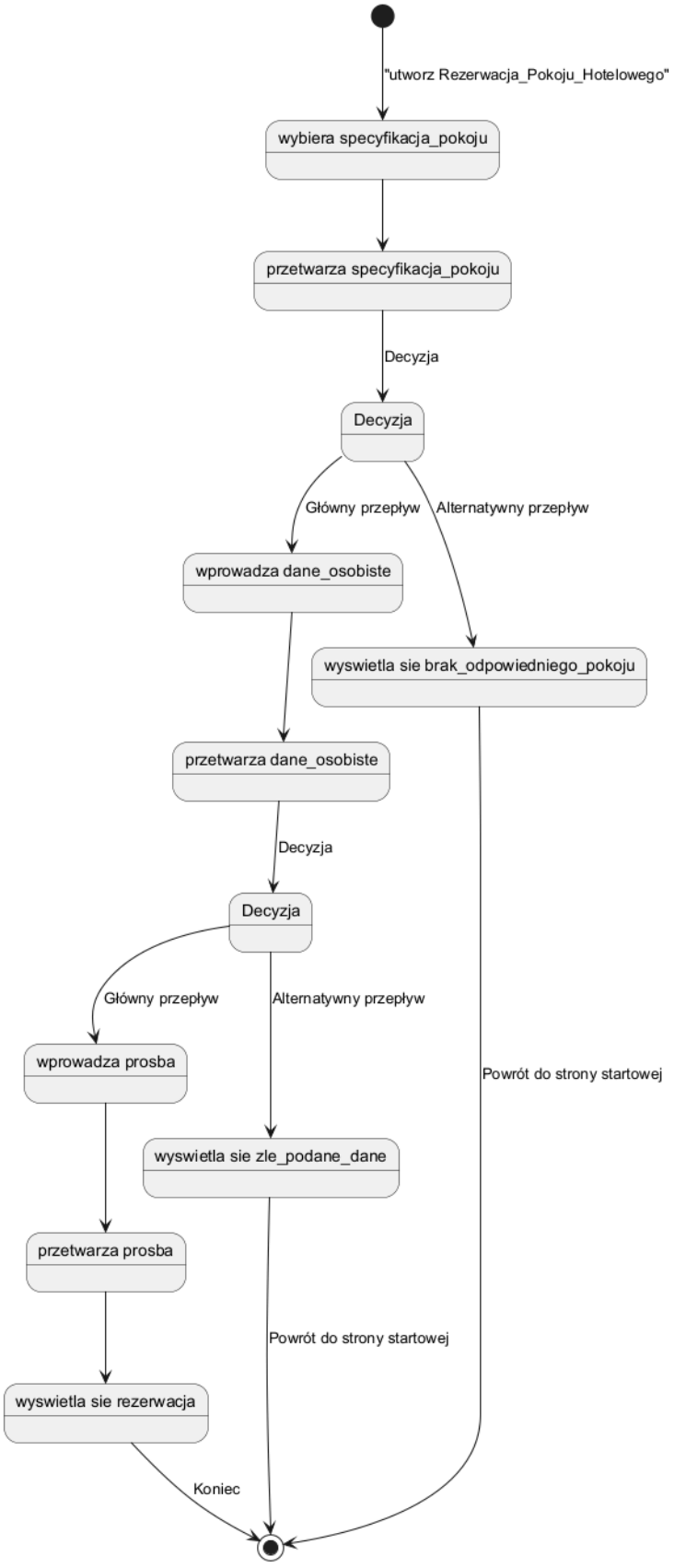
Przykładowy Scenariusz nr 2

```
UC 1 utworz Rezerwacja_Pokoju_Hotelowego
Opis: Klient tworzy Rezerwacje. Klient oraz System_hotelowy dostarcza lub weryfikuje istotne informacje.
Warunki wstepne: 1. Utworzenie_konta jest w stanie Zakonczony. Uruchomiony przez UC 11 utworz Rezerwacja_Pokoju_Hotelowego.
Aktorzy: Klient (typ: tworca)
Glowny Przeplyw:
1. Klient <aktywnosc> wybiera </a> dane z listy <parametr> specyfikacja_pokoju </p>.
2. System <aktywnosc> przetwarza </a> wprowadzone dane Listy <parametr> specyfikacja_pokoju </p>.
3. Klient <aktywnosc> wprowadza </a> parametr Formularza_dane_osobiste <parametr> dane_osobiste </p>.
4. System <aktywnosc> przetwarza </a> wprowadzone dane Formularz_dane_osobiste <parametr> dane_osobiste </p>.
5. Klient <aktywnosc> wprowadza </a> parametr Formularz_spcecjalnych_prosb <parametr> prosba </p>.
6. System <aktywnosc> przetwarza </a> wprowadzone dane Formularz_spcecjalnych_prosb <parametr> prosba </p>.
7. Klientowi <aktywnosc> wyswietla sie </a> <parametr> rezerwacja </p>.
Alternatywny Przeplyw:
3.1. Klientowi <aktywnosc> wyswietla sie </a> <parametr> brak_odpowiedniego_pokoju </p>.
3.2. <aktywnosc> Przeniesienie </a> Klienta do <parametr> strony startowej </p>.
5.1. Klientowi <aktywnosc> wyswietla sie </a> <parametr> zle_podane_dane </p>.
5.2. <aktywnosc> Przeniesienie </a> Klienta do <parametr> strony startowej </p>.
Wyjatki: 7.1. <parametr> prosba </p> z Formularza_specjalnych_prosb
jest niepoprawna: Nieprawidlowe dane wejsciowe.
Warunki koncowe: Formularz_rejestracji jest w stanie Zakonczony.
```

Wygenerowany kod w języku C++

```cpp
class Rezerwacja_Pokoju_Hotelowego {
public:
    void wybiera(const std::string& argument){}

    void przetwarza(const std::string& argument){}

    void wprowadza(const std::string& argument){}

    void wyswietla_sie(const std::string& argument){}

    void Przeniesienie(const std::string& argument){}

    void run(){
        std::string specyfikacja_pokoju;
        std::string dane_osobiste;
        std::string prosba;
        std::string rezerwacja;
        std::string brak_odpowiedniego_pokoju;
        std::string strona_glowna;
        std::string zle_podane_dane;

        wybiera(specyfikacja_pokoju);
        przetwarza(specyfikacja_pokoju);

        if(brak_odpowiedniego_pokoju){
            wyswietla sie(brak_odpowiedniego_pokoju);
            Przeniesienie(strona_glowna);
            return;
        }
        wprowadza(dane_osobiste);
        przetwarza(dane_osobiste);

        if(zle_podane_dane){
            wyswietla sie(zle_podane_dane);
            Przeniesienie(strona_glowna);
            return;
        }
        wprowadza(prosba);
        przetwarza(prosba);
        wyswietla_sie(rezerwacja);
    }
}
```

Wygenerowany diagram aktywności



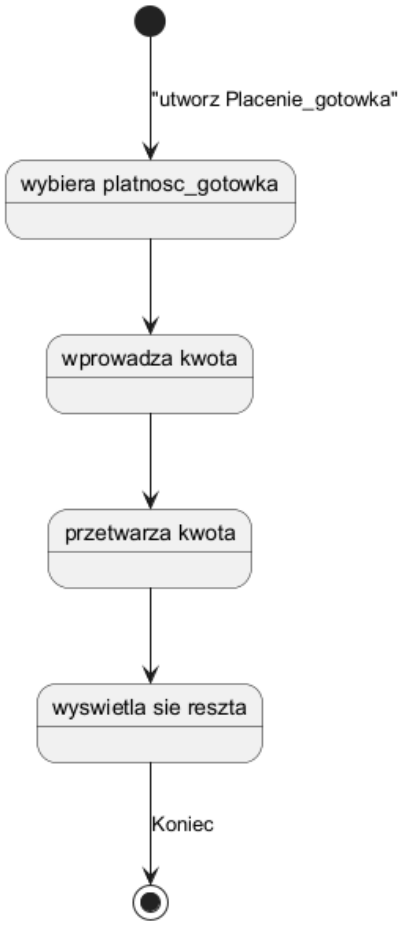
Przykładowy Scenariusz nr 3

```
UC 1 utworz Placenie_gotowka
Opis: Pracownik tworzy Formularz_platnosci. Klient oraz System_platyniczy dostarcza lub weryfikuje istotne informacje.
Warunki wstepne: 1. Tankowanie jest w stanie Zakonczony. Uruchomiony przez UC 11 utworz Placenie.
Aktorzy: Pracownik (typ: tworca), Klient (typ: towarzyszacy)
Glowny Przeplyw:
1. Pracownik <aktywnosc> wybiera </a> dane z listy <parametr> platnosc_gotowka </p>.
2. Pracownik <aktywnosc> wprowadza </a> parametr Formularza_platnosci <parametr> kwota </p>.
3. System <aktywnosc> przetwarza </a> wprowadzone dane Formularza_platnosci <parametr> kwota </p>.
4. Pracownikowi <aktywnosc> wyswietla sie </a> <parametr> reszta </p>.
Alternatywny Przeplyw:
Wyjatki: 2.1. <parametr> kwota </p> z Formularza_platnosci jest niepoprawna: Nieprawidlowe dane wejsciowe.
Warunki koncowe: Formularz_platnosci jest w stanie Zakonczony.
```

Wygenerowany kod w języku C#

```
class Placenie_gotowka {
public:
    void wybiera(const std::string& argument){}

    void wprowadza(const std::string& argument){}

    void przetwarza(const std::string& argument){}

    void wyswietla_sie(const std::string& argument){}

    void run(){
        std::string platnosc_gotowka;
        std::string kwota;
        std::string reszta;

        wybiera(platnosc_gotowka);
        wprowadza(kwota);
        przetwarza(kwota);
        wyswietla_sie(reszta);
    }
}
```

Wygenerowany diagram aktywności



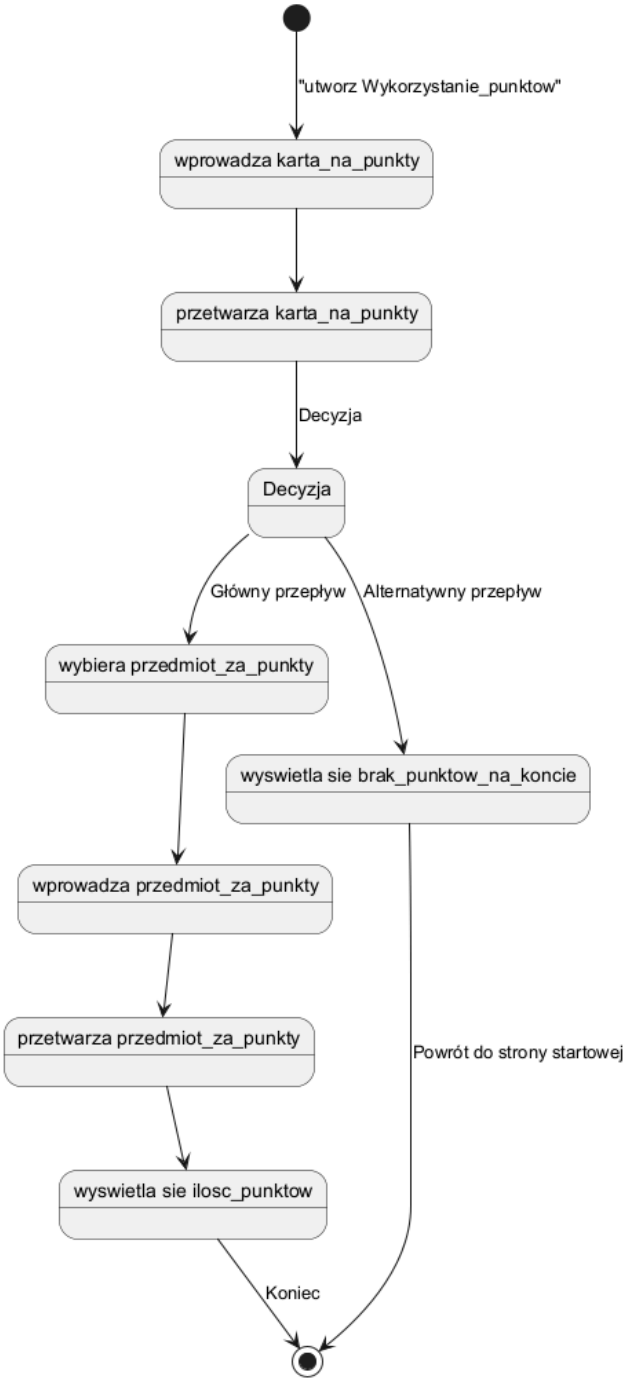
Przykładowy Scenariusz nr 4

```
UC 6 utworz Wykorzystanie_punktow
Opis: Pracownik tworzy Formularz_punktow. Klient oraz System_podliczenia dostarcza lub weryfikuje istotne informacje.
Warunki wstepne: 1.  Pracownik jest upowazniony.
Aktorzy: Pracownik (typ: tworca), Klient (typ: towarzyszacy)
Glowny Przeplyw:
1. Pracownik <aktywnosc> wprowadza </a> parametr Formularz_punktow <parametr> karta_na_punkty </p>.
2. System <aktywnosc> przetwarza </a> wprowadzone dane Formularza_punktow <parametr> karta_na_punkty </p>.
3. Klient <aktywnosc> wybiera </a> dane z listy <parametr> przedmiot_za_punkty </p>.
4. Pracownik <aktywnosc> wprowadza </a> parametr Formularza_punktow <parametr> przedmiot_za_punkty </p>.
5. System <aktywnosc> przetwarza </a> wprowadzone dane Formularza_punktow <parametr> przedmiot_za_punkty </p>.
6. Pracownikowi <aktywnosc> wyswietla sie </a> <parametr> ilosc_punktow </p>.
Alternatywny Przeplyw:
3.1. Pracownikowi <aktywnosc> wyswietla sie </a> <parametr> brak_punktow_na_koncie </p>.
3.2. <aktywnosc> Przeniesienie </a> Pracownika do <parametr> strony startowej </p>.
Wyjatki: 2.1. <parametr> karta_na_punkty </p> z Formularza_platnosci jest niepoprawna: Nieprawidlowe dane wejsciowe.
Warunki koncowe: Formularz_punktow jest w stanie Zakonczony.
```

Wygenerowany kod Python

```python
class Wykorzystanie_punktow:

    def wprowadza(self, argument):
        pass

    def przetwarza(self, argument):
        pass

    def wybiera(self, argument):
        pass

    def wyswietla_sie(self, argument):
        pass

    def Przeniesienie(self, argument):
        pass

    def run(self):
        karta_na_punkty = None
        przedmiot_za_punkty = None
        ilosc_punktow = None
        brak_punktow_na_koncie = None
        strona_glowna = None

        self.wprowadza(karta_na_punkty)
        self.przetwarza(karta_na_punkty)

        if brak_punktow_na_koncie:
            self.wyswietla_sie(brak_punktow_na_koncie)
            self.Przeniesienie(strona_glowna)
            return

        self.wybiera(przedmiot_za_punkty)
        self.wprowadza(przedmiot_za_punkty)
        self.przetwarza(przedmiot_za_punkty)
        self.wyswietla_sie(ilosc_punktow)
```

Wygenerowany diagram aktywności



5. Jak uruchomić projekt

Aby uruchomić projekt na lokalnej maszynie, wykonaj następujące kroki:

1. **Instalacja Javy**
   Upewnij się, że masz zainstalowaną Javę w wersji 17 lub wyższej.
2. **Pobierz Graphiz** z oficjalnej strony https://graphviz.gitlab.io/download. Wybierz odpowiednią wersję dla swojego urządzenia.
3. **Dodaj ścieżkę do pliku dot.ex.** Przykładowa ścieżka C:\Program Files\Graphviz\bin\dot.exe .
4. **Dodanie scenariuszy.** Dodaj do folderu scenarios pliki txt swoich scenariuszy.

Budowanie projektu
Przy użyciu Maven:

```
mvn clean install
```

Uruchomienie aplikacji
Projekt uruchamia się w Main.java który znajduje się w folderze src/main/java/org/example/Main.java.

---

6. Co można rozwinąć w tym projekcie.

1. **Rozszerzenie funkcjonalności analizatora**:
   - Umożliwienie automatycznego wykrywania błędów w scenariuszach (np. brakujące aktywności, niespójność przepływów).
   - Rozbudowa obsługi alternatywnych i wyjątkowych przepływów.
2. **Dodanie kolejnych rozszerzeń do których byłby eksportowany diagram**:
   - Eksportowanie diagramów nie tylko do pliku .png ale również do .jpg i .pdf
   - Eksportowanie diagramów do specjalnego rozszerzenia XML reprezentującego strukturę diagramu, który może być użyty w innych narzędziach UML.
3. **Automatyczne testy jednostkowe**:
   - Generowanie testów jednostkowych na podstawie wygenerowanego kodu.
   - Rozszerzenie o możliwość walidacji danych wyjściowych zgodnie z predefiniowanymi warunkami końcowymi.
4. **Rozszerzenie gramatyki**:
   - Rozbudowa gramatyki ANTLR o dodatkowe reguły i tokeny, które mogą obsługiwać bardziej złożone przypadki użycia lub specyficzne przepływy w systemach.

---

7. Podsumowanie

Projekt miał na celu przygotowanie narzędzi generujących kod i diagramy przypadków użycia dla kolejnych grup które będą rozwijać IDE RE. Staraliśmy się osiągnąć jak największą prostotę w użytkowaniu tych narzędzi a równocześnie skupiliśmy się na poprawności diagramów jak i generowanego kodu.