

Analiza i testowanie systemów informatycznych

PROJEKT SPA

Dokumentacja

Autorzy:

- Łukasz Janowicz 109665
- Patryk Wójtowicz 109960
- Michał Wołosewicz 109958
- Dawid Majewski 109770
- Piotr Szumowski 109933
- Magda Zaborowska 109962

Rodzaj studiów: Studia dzienne

Kierunek: Informatyka II stopień

Semestr: I

Grupa zajęciowa: PS4

Prowadzący ćwiczenie: dr hab. inż. prof. PB Stanisław Jarząbek

Data oddania projektu: 11.06.2024

1. Osiągnięcia i problemy w danych iteracjach

a. Zakres SPA w iteracji 1

- i. Implementacja *Parsera* dla następującego podzbioru *SIMPLE*: Programy z pojedynczą procedurą, przypisaniem i *while* oraz z wyrażeniami w formie uproszczonej
- ii. *Parser* i *Design Extractor* ładuje do PKB: ***Follows, Parent, Modifies, Uses*** (statements only), ***ATS***
- iii. Implementacja *QueryProcessor* dla pojedynczych relacji: ***Follows, Follows*, Modifies, Uses, Calls, Calls*, With***

b. Osiągnięcia i problemy w iteracji 1

- i. Osiągnięto zakres 1 iteracji

c. Zakres SPA w iteracji 2

- i. Rozwinięcie *Parsera* tak, aby odczytywał *SIMPLE* bez ograniczeń
- ii. *Parser* i *DesignExtractor* ładuje do PKB: ***ATS*** (pełen *SIMPLE*) ***Follows, Parent, Modifies, Uses*** (statements only)
- iii. Implementacja *QueryParser* z pojedynczą, podwójną i potrójną relacją oraz z klauzulą *with*. Implementacja ***BOOLEAN***

d. Osiągnięcia i problemy w iteracji 2

- i. Klauzula *with* działa jedynie gdy podamy konkretną liczbę lub znak. Pozostałe zagadnienia z Iteracji 2 zostały zaimplementowane.

e. Zakres SPA w iteracji 3

- i. *Parser* oraz *DesignExtractor* przyjmujący dane do PKB (**wszystkie relacje z pominięciem *Affects* oraz *Affects****)
- ii. Implementacja w *QueryProcessor* klauzuli ***Pattern***, klauzula *such that* z wieloma relacjami, zapytania nie mogą posiadać wiele klauzuli *such that*

f. Osiągnięcia i problemy w iteracji 3

- i. Zaimplementowano *such that* w wieloma *and*
- ii. Nie zaimplementowano obsługi zapytania z wieloma *such that* oraz obsługi ***Pattern***

2. Podział pracy

Poniższa tabela podziału pracy przedstawia ogólny zarys zadań i obowiązków. Zespół w trakcie planowania i implementacji projektu wymieniał się zadaniami. Komunikowanie się w zespole było dynamiczne, a po zgłoszeniu nowych błędów lub problemów członkowie z wolnym czasem pomagali ich rozwiązywaniu czynnie lub dając wskazówki.

Członek zespołu	Zakres głównych obowiązków
Łukasz Janowicz	<ul style="list-style-type: none">➤ Architektura PKB➤ Architektura PQL➤ Implementacja PKB➤ Implementacja PQL➤ Pisanie testów➤ Dokumentacja➤ Testowanie
Patryk Wójtowicz	<ul style="list-style-type: none">➤ Architektura PQL➤ Architektura PKB➤ Implementacja PKB➤ Implementacja PQL➤ Pisanie testów➤ Dokumentacja➤ Testowanie
Michał Wołosewicz	<ul style="list-style-type: none">➤ Implementacja PQL➤ Debugging PQL➤ Debugging PKB➤ Testowanie➤ Dokumentacja
Dawid Majewski	<ul style="list-style-type: none">➤ Implementacja PKB➤ Debugging PKB➤ Testowanie➤ Dokumentacja
Magda Zaborowska	<ul style="list-style-type: none">➤ Implementacja PQL➤ Debugging PQL➤ Dokumentacja
Piotr Szumowski	<ul style="list-style-type: none">➤ Implementacja PKB➤ Debugging PKB➤ Dokumentacja

3. Diagram UML

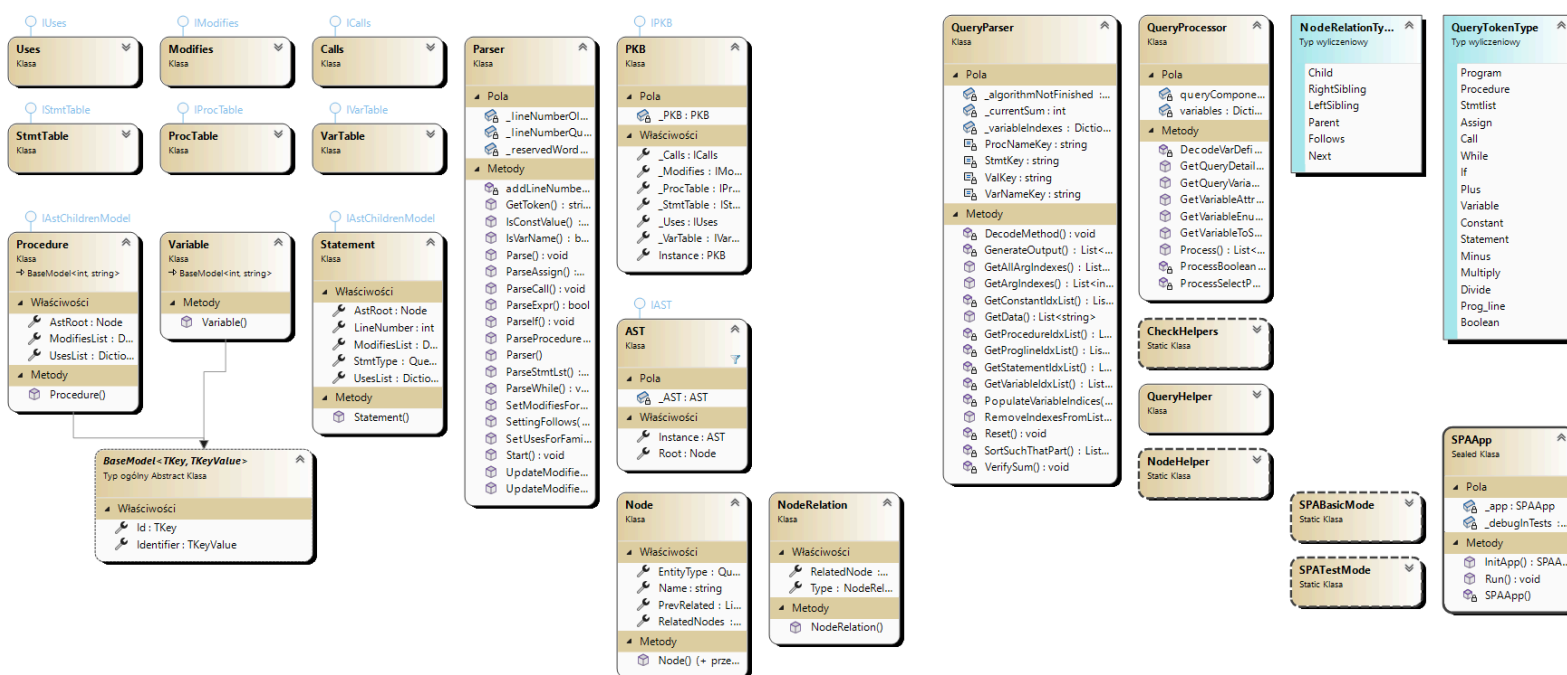


Diagram Klas

Diagram pozwolił na wizualizację struktury klas oraz ich zależności. W projekcie wykorzystano go do planowania struktury kodu, ułatwienia komunikacji między członkami zespołu oraz dokumentacji systemu. Dzięki wykorzystaniu diagramu klas, zwizualizowano jak różne elementy systemu mają ze sobą współpracować oraz określono które klasy i funkcjonalności będą wymagały potencjalnie najwięcej uwagi podczas implementacji.

4. Ważne decyzje w projektowaniu

5. Query processing

a. Walidacja

Metoda walidująca sprawdza w pierwszej kolejności czy zapytanie zawiera wyłącznie zaimplementowane klauzule.

b. Query Evaluator

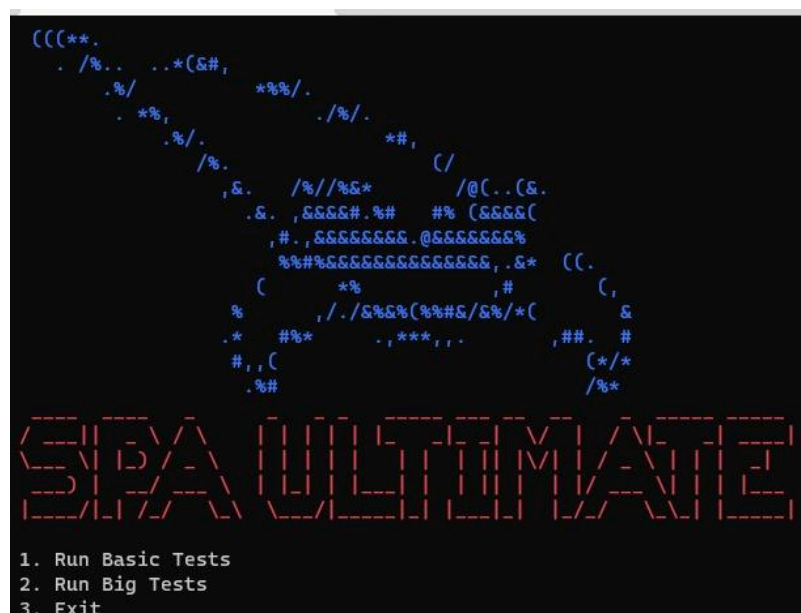
Reprezentacja danych:

- Typ wyliczeniowy *QueryTokenType*

- Reprezentuje rodzaj słowa kluczowego występującego w *query*. Pomaga zidentyfikować istotne dla działania programu elementy zapytania.

- Słownik deklaracji zmiennych

- Zbiór zmiennych zadeklarowanych w *query* z kluczem będącym nazwą zmiennej np. *n* oraz wartością *QueryTokenType* np. *Variable*



Została napisana prosta aplikacja konsolowa pozwalająca na wykonywanie testów. Jeśli wystąpiły błędy i SPA zwrócił błąd, wtedy wypisane zostanie zapytanie, co zwróciło SPA oraz oczekiwany, poprawny wynik. Dodatkowo wypisana zostanie informacja o liczbie udanych i nieudanych testów oraz liczba niewspieranych przez aplikację, rodzajów zapytań.

Ponadto w procesie debugowania używano wbudowanego w środowisko Visual Studio debuggera, który umożliwia szybkie lokalizowanie błędów i przyspiesza proces ich naprawiania. Oferuje on zaawansowane funkcje, takie jak śledzenie krok po kroku, ustawienie punktów przerwania, inspekcję zmiennych oraz podgląd stosu wywołań, co pozwala na precyzyjną analizę zachowania programu i precyzyjne znajdowanie źródła problemów.

W trakcie procesu debugowania rozpoczęto od identyfikacji i naprawy najbardziej oczywistych i krytycznych błędów co umożliwiło osiągnięcie stabilności podstawowej wersji aplikacji. Następnie przeprowadzano bardziej szczegółowe testy, skupiając się na mniej oczywistych problemach oraz na optymalizacji algorytmów. W tym celu wykorzystano opisaną wcześniej aplikację testującą i debugger oraz symulowano różne scenariusze użytkowania aby upewnić się że rozwiązanie działa poprawnie w różnych warunkach.

```
| Select BOOLEAN such that Next (17,18) |
| true |
| Correct answer: |
| false |
```

Przykład wpisania błędnego wyniku query

```
Correct: 574, Incorrect: 9
```

Przykład odpowiedzi na przetworzone testy


7. Testy

Program został poddany szczegółowym testom, aby zapewnić jego niezawodność i dokładność w działaniu. Testowanie przeprowadzono za pomocą dwóch metod: własnych testów oraz specjalistycznego oprogramowania Pipe Tester v1.5.

W ramach własnych testów stworzono przypadki testowe osadzone bezpośrednio w kodzie, które generowały na konsoli tabele z błędnymi odpowiedziami oraz wartościami, jakie powinny być prawidłowe. Dzięki temu możliwe było szybkie identyfikowanie i korygowanie wszelkich nieprawidłowości. Analiza tych wyników pozwoliła na systematyczne polepszanie programu oraz dokładne badanie jego stabilności pod kątem występowania bugów. Proces ten zapewnił, że program stawał się coraz bardziej precyzyjny i odporny na błędy.

Dodatkowo Simple został przetestowany przy użyciu Pipe Tester v1.5, zaawansowanego narzędzia do automatycznego testowania oprogramowania. Pipe Tester v1.5 umożliwił przeprowadzenie

skomplikowanych analiz wydajnościowych, porównawczych oraz wykrycia bugów. Dzięki użyciu Pipe Tester v1.5 możliwe było uzyskanie poprawnych wyników testów.

App Path	Status	IsA...	Index	Progress	OK	Fail	Time	Ex	Cr
C:\Users\lukasz\Desktop\VAITS\VAITS-Projec...	 Done	False	583	<div style="width: 100%; height: 10px; background-color: green;"></div>	570	13	0	0	0

Przykładowy rezultat testów w Pipe Tester v1.5 dla pliku SIMPLE udostępnionego w ramach kursu

Zarówno testy własne, jak i te przeprowadzone za pomocą Pipe Tester v1.5, pomogły zweryfikować skuteczność programu oraz jego przydatności na tle działającego programu dostarczonego wraz z kursem.

8. Przyszły rozwój projektu

1. Implementacja obsługi pozostałych klauzuli występujących w zapytaniach query, jak Pattern, Affects, Affects* oraz wielokrotnego such that.
2. Optymalizacja kodu