Użycie języka Swift i elementów paradygmatu reaktywnego na przykładzie aplikacji mobilnej do polecania filmów i seriali

Bartosz Woliński

26 października 2016

Spis treści

Rozdział 1

WSTĘP

Rozdział 2

CEL I ZAKRES PRACY

Rozdział 3

ŹRÓDŁA I DEFINICJE

3.1 Historia systemu iOS

3.2 Historia i charakterystyka języka Swift

Swift to kompilowany, hybrydowy język programowania stworzony przez firmę Apple Inc. Jego premiera odbyła się podczas Worldwide Developers Conference w czerwcu 2014 roku.[1]

Jeden z twórców [1] opisuje Swifta jako narzędzie czerpiące idee z wielu innych języków t.j. Objective-C, Rust, Haskell, Ruby, Python, C#, CLU i wielu innych.

Swift został stworzony jako nowoczesny następca Objective-C na platformy MacOS i iOS, ale w grudniu 2015 roku stał się językiem open source. Oznacza to, że została stworzona społeczność przy użyciu serwisu Swift.org a oprócz tego udostępniono publiczne repozytorium Gita. Ponadto uwolniono narzędzia takie jak kompilator LLVM, biblioteki standardowe czy menedżer zależności projektu. Dodatkowo Swift otrzymał wsparcie na platformę Linux.[2]

Główne różnice między Swiftem a Objective-C Swift jako język mający na celu zastąpienie leciwego już Objective-C oferuje wiele nowych mechanizmów. **Wartości opcjonalne -** pozwalają funkcjom, które nie zawsze zwrócą konkretną wartość lub obiekt na zwrócenie obiektu enkapsulowanego w wartość opcjonalną bądź wartość nil. W języku C i Objective-C funkcje mogą zwrócić wartość pustą (nil) nawet jeżeli spodziewana wartość jest typu stuktury lub klasy. W Objective-C zwrócenie przez funkcję wartości pustej (pomimo innej spodziewanej) nie powoduje błędów kompilacji ani błędów w czasie działania. W Swficie zaś w takiej sytuacji mielibyśmy do czynienia z błędem kompilacji lub błędem krytycznym w czasie działania co chroni nas przed niespodziewanymi

[1] Chris Lattner - inżynier Apple

zachowaniami.
Wnioskowanie typów - kompilator języka Swift jest w stanie wywnioskować typ tworzonej zmiennej. Ponadto zmienna o zadeklarowanym (wywnioskowanym) typie nie może go zmienić.
Krotki - Swift wspiera obiekty krotkowe, czyli takie, które mogą przechowywać na raz kilka wartości różnych typów. Dzięki temu możliwe jest zwracanie przez funkcji wielu wartości.
Guard - wyrażenie warunkowe w składni Swifta. Zapewnia weryfikacje poprawności oczekiwanego typu zmiennej a w razie błędu, może spowodować wcześniejsze wyjście z funkcji.
Elementy programowania funkcyjnego - Swift posiada możliwość programowania funkcyjnego co niejednokrotnie jest dużo bardziej czytelne i wydajne od tradycyjnego podejścia. Z tego powodu oferuje on operatory funkcyjne typu **map** czy **filter**.
Enumeratory - W Swficie, podejście do enumeratorów zostało bardzo rozbudowane. Mogą one zawierać metody i być przekazywane przez wartość.
Podejście do funkcji - Każda funkcja w Swifcie posiada typ, który składa się z typów parametrów oraz typu zwracanego. To oznacza, że można przypisywać funkcje do zmiennych a nawet przesyłać je jako parametry innych funkcji.
Słowo kluczowe "do" - pozwala na utworzenie nowego zakresu w kodzie a ponadto może zawierać mechanizm obsługi błędów, znany z innych jęzków t.j. "try catch".

3.3 Paragydmat funkcyjny

3.4 Paragydmat reaktywny

3.4.1 Idea programowania reaktywnego

3.4.2 Podejście reaktywne w praktyce

3.4.3 Operatory reaktywne

3.4.4 Współbieżność

3.4.5 Wady i zalety podejścia reaktywnego

3.4.6 Implementacja na platformie iOS

Rozdział 4

PRACA WŁASNA

4.1 Czynności przygotowawcze

Stworzenie aplikacji mobilnej było możliwe dzięki wykonaniu uprzednio czynności przygotowawczych. Do czynności tych należało: wybór środowiska programistycznego, instalacja bibliotek i frameworków, wstępna konfiguracja projektu oraz stworzenie repozytorium.

4.1.1 Instalacja narzędzi

Program XCode jest jednym z niewielu dostępnych środowisk programistycznych na platformę iOS. Jest on darmowy i dostarczany wraz systemem MacOS. XCode posiada wbudowany kompilator LLVM oraz szereg przydatnych narzędzi tj. interface builder - graficzny edytor do tworzenia elementów interfejsu, dynamiczną kontrolę składni czy menedżer systemu kontroli wersji. Ze względu na powyższe właściwości zdecydowałem się użyć właśnie tego środowiska programistycznego. Ponadto użyłem także narzędzia do rozwiązywania zależności - CocoaPods. Jest ono bardzo przydatne szczególnie przy instalacji bibliotek i frameworków do projektu.

4.1.2 Wstępna konfiguracja projektu

Podczas konfigurowania projektu w środowisku XCode istotne jest abyśmy stworzyli go z użyciem CoreData. Jest to baza danych środowiska iOS i może stanowić integralną część aplikacji.

4.1.3 Stworzenie repozytorium

Dostępnych jest wiele systemów kontroli wersji, ale najpopularniejszym z nich jest GIT. Zdecydowałem się na jego wykorzystanie, ponieważ jest dosyć prosty w użyciu a większość serwerów GITa jest darmowa. Użycie systemu typu GIT pozwoli mi na kontrolę postępów mojej pracy jak i na dokumentowanie jej.

Oprócz tego użycia GITa powoduje, że błąd popełniony przeze mnie na dalszym etapie mogę odwrócić przywracając poprzedni stan projektu.

4.2 Budowa aplikacji właściwej

4.2.1 Wymagania funkcjonalne

4.2.2 Wymagania niefunkcjonalne

4.2.3 Diagram przypadków użycia aplikacji

4.2.4 Zależności w bazie danych

4.2.5 Diagram klas

4.2.6 Opis wybranych klas

4.2.7 Opis użytych bibliotek i frameworków

4.2.8 Implementacja aplikacji mobilnej - zastosowana architektura

4.2.9 Prezentacja aplikacji

Bibliografia

[1] https://developer.apple.com/swift/blog/?id=14 - *historia języka Swift*

[2] https://developer.apple.com/swift/blog/?id=34 - *Swift jako język open-source*

Rozdział 5

PODSUMOWANIE