

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

Bartosz Woliński

31 października 2016

# Spis treści

<b>1</b>	<b>WSTĘP</b>	<b>2</b>
<b>2</b>	<b>CEL I ZAKRES PRACY</b>	<b>3</b>
<b>3</b>	<b>ŹRÓDŁA I DEFINICJE</b>	<b>4</b>
3.1	Historia i charakterystyka systemu iOS . . . . .	4
3.2	Historia i charakterystyka języka Swift . . . . .	4
3.3	Paradygmat funkcyjny . . . . .	6
3.4	Paradygmat reaktywny . . . . .	6
3.4.1	Idea programowania reaktywnego . . . . .	6
3.4.2	Podejście reaktywne w praktyce . . . . .	6
3.4.3	Operatory reaktywne . . . . .	6
3.4.4	Współbieżność . . . . .	6
3.4.5	Wady i zalety podejścia reaktywnego . . . . .	6
3.4.6	Implementacja na platformie iOS . . . . .	6
<b>4</b>	<b>PRACA WŁASNA</b>	<b>7</b>
4.1	Czynności przygotowawcze . . . . .	7
4.1.1	Instalacja narzędzi . . . . .	7
4.1.2	Wstępna konfiguracja projektu . . . . .	7
4.1.3	Stworzenie repozytorium . . . . .	7
4.2	Budowa aplikacji właściwej . . . . .	8
4.2.1	Wymagania funkcjonalne . . . . .	8
4.2.2	Wymagania нефункционалне . . . . .	8
4.2.3	Diagram przypadków użycia aplikacji . . . . .	8
4.2.4	Zależności w bazie danych . . . . .	8
4.2.5	Diagram klas . . . . .	8
4.2.6	Opis wybranych klas . . . . .	8
4.2.7	Opis użytych bibliotek i frameworków . . . . .	8
4.2.8	Implementacja aplikacji mobilnej - zastosowana architektura . . . . .	8
4.2.9	Prezentacja aplikacji . . . . .	8
<b>5</b>	<b>PODSUMOWANIE</b>	<b>10</b>

## Rozdział 1

# WSTĘP

## Rozdział 2

# CEL I ZAKRES PRACY

## Rozdział 3

# ŹRÓDŁA I DEFINICJE

### 3.1 Historia i charakterystyka systemu iOS

iOS to mobilny system operacyjny stworzony przez firmę Apple Inc. na urządzenia iPod, iPhone i iPad. Został zaprezentowany w styczniu 2007 roku na konferencji Macworld. Jest jednym z dwóch<sup>1</sup> najpopularniejszych systemów mobilnych.

**Specyfikacje techniczne** System iOS oparty jest na jądrze Darwin. Darwin to unixowy system operacyjny typu open-source wypuszczony przez Apple Inc. w 2000 roku. Zbudowany jest w oparciu o projekty firm Apple, NeXTSTEP, BSD, Mach i kilka innych. System działa na architekturach PowerPC, Intel x86 i ARM.[4][5] Jądrzem systemu Darwin jest XNU. Jest to hybrydowe jądro łączące w swojej implementacji Mach 3 microkernel i elementy BSD t.j. stos sieciowy czy wirtualny system plików.[?]

**Wersje systemu iOS** **iPhone OS** - pierwsza iteracja mobilnego systemu Apple. Nie nadano żadnej oficjalnej nazwy. Jedyne co utrzymywano to, że iPhone korzysta z jednej z desktopowych wersji systemu OSX.[?] W marcu 2008 roku wypuszczono zestaw narzędzi do programowania tę platformę: iPhone SDK

### 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]

---

<sup>1</sup>dane firmy Gartner Inc. za rok 2015: urządzenia mobilne z systemem android - 1,3 mld urządzeń z systemem iOS - 297 milionów

Jeden z twórców <sup>2</sup> 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 struktury lub klasy. W Objective-C zwrócenie przez funkcję wartości pustej (pomimo innej spodziewanej) nie powoduje błędów kompilacji ani błędów w czasie działania. W Swiftcie zaś w takiej sytuacji mielibyśmy do czynienia z błędem kompilacji lub błędem krytycznym w czasie działania co chroni nas przed niespodziewanymi 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 weryfikację 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 Swiftcie, podejście do enumeratorów zostało bardzo rozbudowane. Mogą one zawierać metody i być przekazywane przez wartość.

**Podejście do funkcji** - Każda funkcja w Swiftcie posiada typ, który składa się z typów parametrów oraz typu zwracanego. To oznacza, że można przypisywać funkcje do zmiennych a nawet przysyłać 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 języków t.j. "try catch". [3]

---

<sup>2</sup>Chris Lattner - inżynier Apple

### **3.3 Paradygmat funkcyjny**

### **3.4 Paradygmat 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*
- [3] <https://redwerk.com/blog/10-differences-objective-c-swift> - *Różnice między Swiftem a Objective - C*
- [4] [www.puredarwin.com](http://www.puredarwin.com) - *informacje o systemie Darwin*
- [5] [www.opengroup.org](http://www.opengroup.org) - *informacje o systemach unixowych*

## Rozdział 5

# PODSUMOWANIE