Projekt iOS: „Kalkulator”

1. Cel

Projekt miał na celu stworzenia funkcjonalnego kalkulatora w języku Swift dla platformy iOS przy użyciu Framework Xcode.

2. Kod źródłowy

Do każdego przycisku liczbowego dodaliśmy odpowiedni tag, który odpowiadał wartości danego przycisku (np. dla liczby 5 tag wynosił „5”, dzięki czemu łatwo mogliśmy dopisać kolejną wartość na ekranie). Wciśnięcie każdego przycisku z numerem, wywoływało funkcję akcji *numbers().* Ponadto, w klasie zainicjowaliśmy flagi określające poszczególne operacje, przetrzymujące wartość oraz informujące o obecnym stanie programu (np. *globalValueButton* informuje o wciśnięciu przez użytkownika wykonania jednej z podstawowych operacji. Ekran kalkulatora jest pole tekstowym, w kodzie reprezentowany przez outlet *calculationScreen*.

```
import UIKit
import Foundation

class ViewController: UIViewController {

    var calculatorValue: Double? = 0.0;
    var multiplyOperation: Bool = false;
    var addOperation: Bool = false;
    var minusOperation: Bool = false;
    var divideOperation: Bool = false;
    var moduloOperation: Bool = false;

    var globalValueButton: Bool = false;

    var firstValue: Double? = 0.0;
    var secondValue: Double? = 0.0;

    @IBOutlet weak var calculationScreen: UITextField!
    @IBAction func numbers(_ sender: UIButton) {

            if globalValueButton
            {
            calculationScreen.text = "";
                globalValueButton = false;
            }


            calculationScreen.text = calculationScreen.text! + String(sender.tag);
            calculatorValue = Double(calculationScreen.text!)


    }


    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
```

Rys1.1 Zmienne w klasie ViewController oraz funkcja wykonująca akcję

Funkcja akcji *operationalButton()* wywoływana jest w momencie wciśnięcia przycisku operacji dodawania, odejmowania, dzielenia, mnożenia lub modulo. Przypisuje wartości do zmiennych, przypisuje znak oraz wykonuje niezbędne operacje, aby zawsze były maksymalnie dwie wartości w programie (czyli np. w momencie gdy użytkownik wpisze 5+5+, wartości zostaną dodane i zapisane w zmiennej *firstValue*, pomimo że nie został wciśnięty przycisk =). Takie podejście znacząco ułatwia obliczanie długich równań, gdyż nie musimy trzymać wszystkich wartości i znaków operacji w pamięci.

```
@IBAction func operationalButton(_ sender: UIButton) {

    if (firstValue == 0 || firstValue == 0.0)
    {
        firstValue = Double (calculationScreen.text!);
        addSign(tag: sender.tag);
    }
    else
    {
        secondValue = Double(calculationScreen.text!);
        isEqual();
        addSign(tag: sender.tag);
        secondValue = 0;
    }
}
```

Rys1.2 Funkcja akcji operationalButton()

Funkcja *addSign()* określa charakter operacji – dodawanie, odejmowanie, mnożenie, dzielenie albo modulo. Poszczególne operacje identyfikowane są za pomocą tagów. Funkcja ustawia również flagi, informując o obecnie wciśniętym przycisku operacji.

```
222     func addSign(tag: Int) -> Void
223     {
224         globalValueButton = true;
225         if tag == 20
226         {
227             calculationScreen.text = calculationScreen.text! + "+";
228             addOperation  = true;
229         }
230         else if tag == 21
231         {
232             calculationScreen.text = calculationScreen.text! + "-";
233             minusOperation = true;
234             //calculationScreen.text = calculationScreen.text! + "-";
235         }
236         else if tag == 22
237         {
238             calculationScreen.text = calculationScreen.text! + "*";
239             multiplyOperation = true;
240
241         }
242         else if tag == 23
243         {
244             calculationScreen.text = calculationScreen.text! + "/";
245             divideOperation = true;
246         }
247         else if tag == 26
248         {
249             calculationScreen.text = calculationScreen.text! + "%";
250             moduloOperation = true;
251
252         }
253     }
254 }
255
256
```

Rys1.3 Funkcja addSign()

Funkcja *isEqual()* wykonuje poszczególne działania.

```
188     func isEqual() -> Void
189     {
190         if addOperation == true
191         {
192             firstValue=firstValue! + secondValue!;
193             calculationScreen.text = String (describing: firstValue!);
194             addOperation = false;
195         }
196         else if minusOperation == true
197         {
198             firstValue=firstValue! - secondValue!;
199             calculationScreen.text = String (describing: firstValue!);
200             minusOperation = false;
201         }
202         else if multiplyOperation == true
203         {
204             firstValue=firstValue! * secondValue!;
205             calculationScreen.text = String (describing: firstValue!);
206             multiplyOperation = false;
207         }
208         else if divideOperation == true
209         {
210             firstValue=firstValue! / secondValue!;
211             calculationScreen.text = String (describing: firstValue!);
212             divideOperation = false;
213         }
214         else if moduloOperation == true
215         {
216             firstValue = firstValue!.truncatingRemainder(dividingBy: secondValue!);
217             calculationScreen.text = String (describing: firstValue!);
218             moduloOperation = false;
219         }
220     }
```

Rys1.4 Funkcja isEqual()

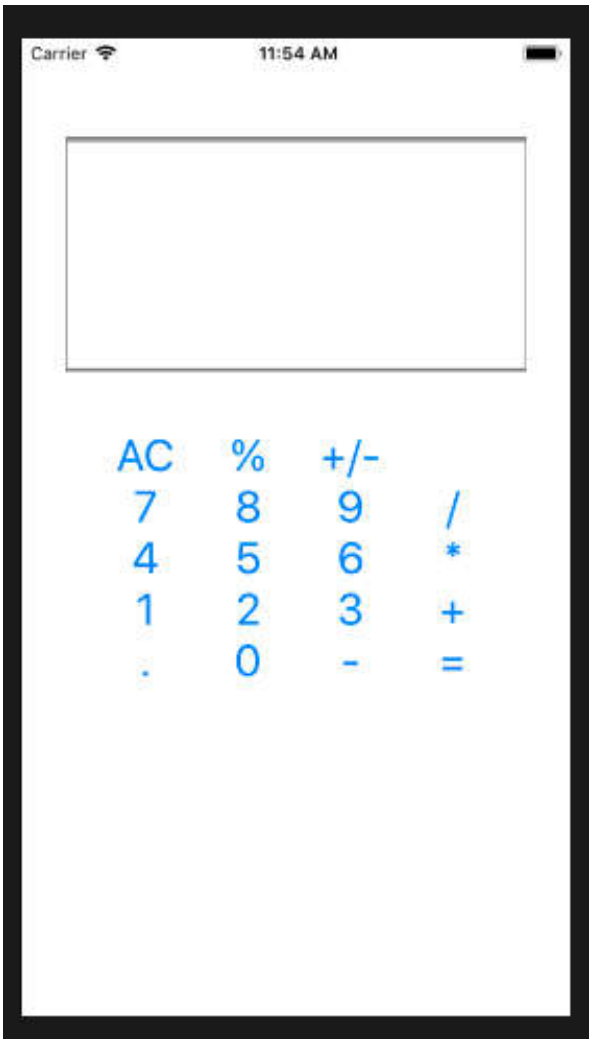
Kalkulator wykonuje również operację zamiany wartości na przeciwną, obliczania sinusa, cosinusa, tangensa, cotangensa, arcus sinusa, arcus cosinusa, arcus tangensa, logarytmu naturalnego, logarytmu dziesiętnego, pierwiastkowania, potęgowania do kwadratu oraz sześcianu. Z przycisków można również wyczyścić ekran oraz wyzerować wartości, wprowadzić kropkę oraz liczbę pi. Każdy z tych przycisków zdefiniowany jest osobnej funkcji akcji.

```
    @IBAction func cosButton(_ sender: Any) {

        if(Double(calculationScreen.text!) != nil){
            var tempValue = Double(calculationScreen.text!);
            tempValue = cos(tempValue!);
            calculationScreen.text = String(describing: tempValue!);
        }
    }
    @IBAction func tgButton(_ sender: Any) {
        if(Double(calculationScreen.text!) != nil){
            var tempValue = Double(calculationScreen.text!);
            tempValue = tan(tempValue!);
            calculationScreen.text = String(describing: tempValue!);
        }
    }
    @IBAction func ctgButton(_ sender: Any) {
        if(Double(calculationScreen.text!) != nil){
            var tempValue = Double(calculationScreen.text!);

            tempValue = cos(tempValue!)/sin(tempValue!);
            calculationScreen.text = String(describing: tempValue!);
        }
    }
    @IBAction func asinButton(_ sender: Any) {
        if(Double(calculationScreen.text!) != nil){
            var tempValue = Double(calculationScreen.text!);

            tempValue = asin(tempValue!);
            calculationScreen.text = String(describing: tempValue!);
        }
    }
    @IBAction func acosButton(_ sender: Any) {
        if(Double(calculationScreen.text!) != nil){
            var tempValue = Double(calculationScreen.text!);

            tempValue = acos(tempValue!);
            calculationScreen.text = String(describing: tempValue!);
        }
    }
```

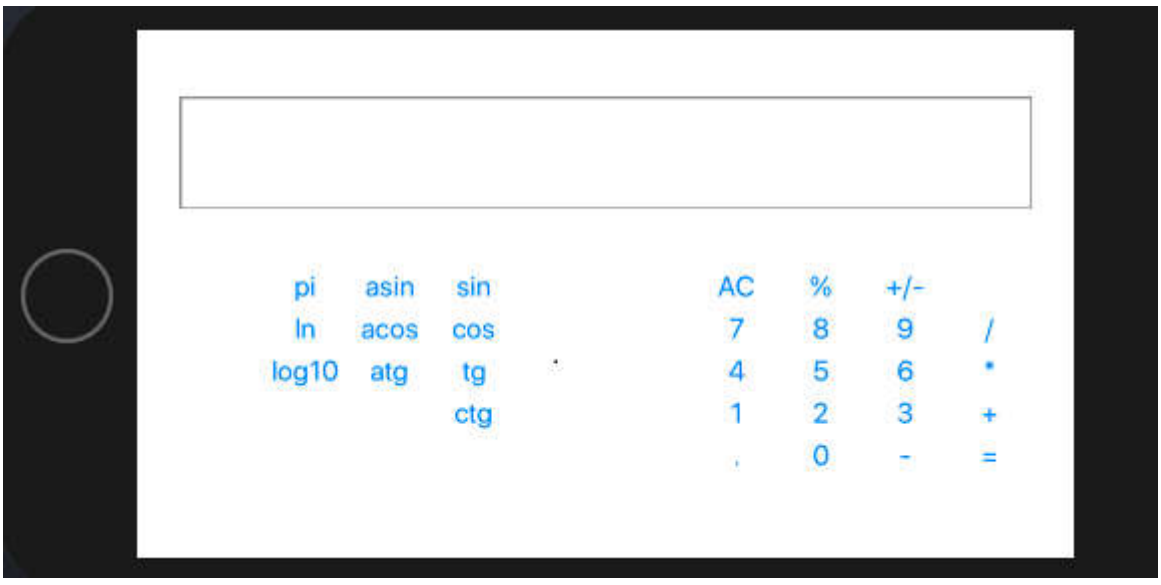
Rys1.5 Przykładowa implementacja funkcji matematycznych

3. Wygląd aplikacji

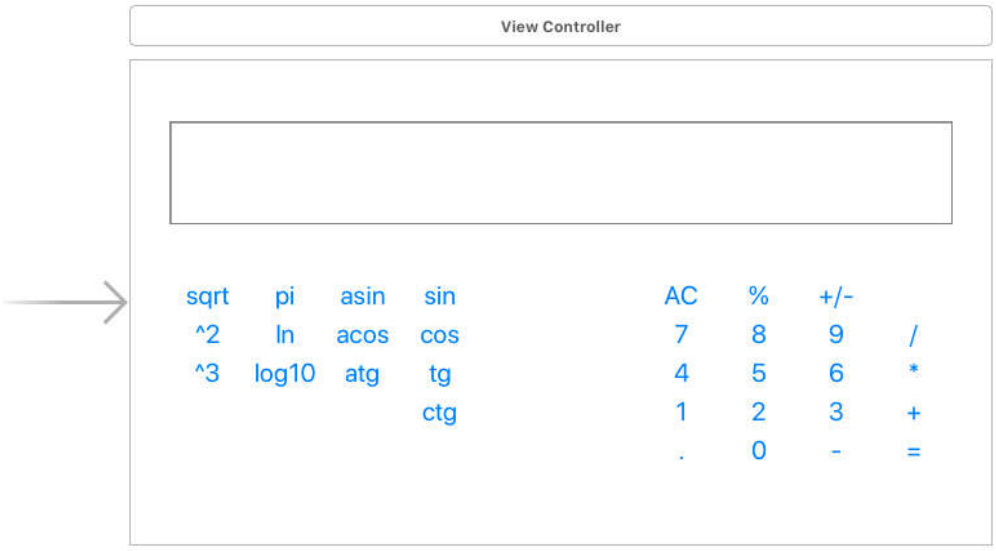
W tworzonej przez nas aplikacji przede wszystkim zmagaliśmy się z funkcjonalnością działania kalkulatora oraz pułapkami jakie powstawały w trakcie pisania kodu. Niemniej, choć kalkulator nasz nie doczekał się „ładnego” wyglądu, staraliśmy się aby ten był w miarę schludny, a wszystkie jego elementy zostały właściwie wypozycjonowane i nie były zbyt duże bądź małe co mogłoby utrudniać pracę z kalkulatorem oraz tworzyłoby złe wrażenie.



Rys2.1. Główny ekran kalkulatora



Rys2.2. Główny ekran kalkulatora (wygląd boczny) – ***przed dodaniem ostatnich funkcjonalności***



Rys2.3. Główny ekran kalkulatora (wygląd boczny View Controller)

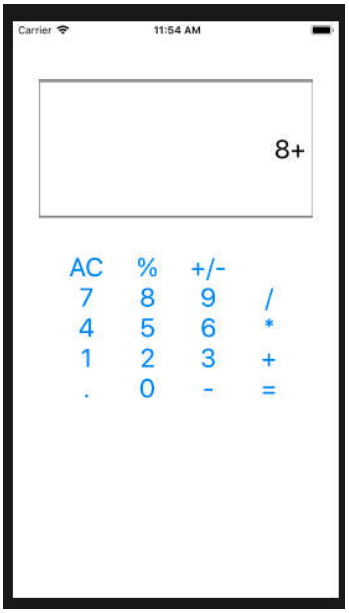
Widok pionowy oraz poziomy zostały od siebie odróżnione nie tylko poprzez układ oraz wielkość przycisków (w ekranie pionowym znaki są większe i umiejscowione na środku ekranu, podczas gdy w ekranie poziomym mniejsze i przesunięte w prawo aby zrobić miejsce na dodatkowe funkcjonalności), ale także dodaliśmy szereg dodatkowych opcji jak to poczynione jest w większości kalkulatorów. Niestety większość z nich została dodana na dodatkowym spotkaniu na samym końcu, dlatego ich ułożenie nie jest optymalne co widać chociażby poprzez trzy wolne pola wyróżniające się w lewej części widoku, a także zbyt dużą przerwą pomiędzy opcjami podstawowymi, a rozbudowanymi.

3.1 Opis przycisków znajdujących się w kalkulatorze

AC - Zerowanie kalkulatora (wraz z usunięciem zapamiętanej liczby)
% - Dzielenie modulo
+/- - Zmiana znaku liczby na ujemny/dodatni
sqrt - Wyznaczenie pierwiastka liczby
^2 - Podniesienie liczby do drugiej potęgi
^3 - Podniesienie liczby do trzeciej potęgi
pi - Liczba pi
ln - Wyznaczenie logarytmu naturalnego liczby (radiany)
log10 - Wyznaczenie logarytmu o podstawie 10 liczby (radiany)
asin/acos/atg - Wyznaczenie arcussinusa/arcuscosinusa/arcustangensa liczby (radiany)
sin/cos/tg/ctg -Wyznaczenie sinusa/cosinusa/tangensa/cotangensa liczby (radiany)

Z funkcjonalności dodatkowych jakie zawiera nasz kalkulator warto zwrócić uwagę na możliwość bezpośredniego wyznaczenia funkcji cotangens na co nie pozwala większość znanych kalkulatorów.

3.2 System wyświetlania liczb na ekranie aplikacji



Rys2.4. Ekran aplikacji podczas testowego działania z wykorzystaniem znaku „+”

Podczas pisania aplikacji stanęliśmy przed problemem zapisywania i odczytywania liczb z/na ekranie. Ostatecznie zdecydowaliśmy się wykorzystać obsługę zapisywania pierwszej liczby w znaku działania (plus/minus/mnożenie/dzielenie) co skutkowało odpowiednią reakcją w wyglądzie reprezentacji wyników i działań na ekranie.

Użytkownik po wprowadzeniu pierwszej liczby i znaku działania widzi oba te elementy na ekranie, w momencie rozpoczęcia wpisywania drugiej liczby jednak, poprzednie dwa elementy znikają zastępowane drugą liczbą działania co nie pozwala na podgląd całego zapisanego działania. Nie jest to opcja nieznana dla wielu kalkulatorów (wręcz wiele z nich korzysta
z podobnych metod zapisu i odczytu) jednak w wypadku rozbudowanej części kalkulatora przy uwzględnieniu jego pełnej funkcjonalności (np. dodanie nawiasów) element ten powinien być zmieniony. Na rzecz naszej aplikacji jednak był wystarczający dlatego zostaliśmy własnie przy nim.

4. Podsumowanie

Po przeprowadzeniu prostych testów manualnych, doszliśmy do wniosku, że udało się nam zrealizować podstawową funkcjonalność kalkulatora – wykonuje proste działania i zwraca oczekiwane wyniki. Na bardziej skomplikowane testy lub na zaimplementowanie testów jednostkowych/zintegrowanych zabrakło czasu. Moglibyśmy rozszerzyć funkcjonalność kalkulatora np. poprzez dodanie możliwości zamiany radianów na stopnie, obliczania wartości e, potęgowania oraz pierwiastkowania dowolnego stopnia. Dla komercyjnego produktu poprawilibyśmy również interfejs graficzny, lecz w czasie projektu skupiliśmy się bardziej na dodawaniu kolejnych funkcjonalności niż na całościowym wyglądzie aplikacji.