UBI-previev

1 Aplikacja z jednym oknem

Rozmieszczenie elementów

Wybieramy z listy elementy które chcemy np. *Label*, *TextField*, itp. Następnie trzeba pamiętać aby dodać ograniczenia - `Add missing constrains`. Potem trzymając `ctrl` przeciągamy nasze elementy do `ViewController`. W oknie które sie pokaże wybieramy rodzaj połączenia. Jeśli chcemy jakieś funke to też przy przeciąganiu zaznaczamy to. Resztę kodu piszemy w `viewController`.

Kod w viewController

```
import UIKit
```

Import biblioteki UIKit, która odpowiada za interfejs graficzny w aplikacjach iOS (przyciski, widoki, kontrolery, itp.).

```
class ViewController: UIViewController, UIPickerViewDataSource, UIPickerViewDelegate {
```

Stworzenie klasy `ViewController`, która dziedziczy po `UIViewController`, czyli odpowiada za ekran aplikacji. Jednocześnie ta klasa odpowiada za dostarczenie danych do pickera (`UIPickerViewDataSource`) i za obsługę pickera np. wyboru waluty (`UIPickerViewDelegate`).

```
    private let currencies = ["PLN", "USD", "EUR", "GBP", "CHF"]
    private var rates = [String:Double]() //słownik waluta kurs - EUR-PLN : 4.12
    private var from=0, to=0
```

Wczytywanie zmiennych (var) i stałych (let)

```
    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var pickerCurrencies: UIPickerView!
    @IBOutlet weak var labelOutput: UILabel!
```

To połączenia między kodem a elementami graficznymi (UI):
-`textField` – pole tekstowe do wpisania kwoty
-`pickerCurrencies` – picker do wyboru walut
-`labelOutput` – etykieta z wynikiem konwersji

```
    @IBAction func textChanged(_ sender: Any) {
    convertValue()
}
```

Akcja przy zmianie tekstu. Metoda `convertValue` jest wykonywana za każdym razem kiedy użytkownik zmieni tekst w polu tekstowym.

Funkcje i nadpisywanie funckji

```
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 2
}
```

Ta funkcja określa ile kolumn jest w pickerze, w tym wypadku dwie.

```
    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    return currencies.count
}
```

Tutaj ustawiamy ile wierszy ma każda kolumna w pickerze. Czyli liczymy ile walut mamy do wyboru w naszym przypadku.

```
    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int)
    -> String? {
    return currencies[row]
}
```

Ta funkcja odpowiada za etykiety, czyli dco ma byś wyświetlane na pickerze W przypadku kiedy chcemy mieć rożne dane w każdej kolumnie:

```
    let currenciesFrom = ["PLN", "USD", "EUR"]
    let currenciesTo = ["JPY", "GBP", "CHF", "CAD"]
    private var rates = [String:Double]()
    private var from = 0, to = 0
    \\...
    \\...
    \\...
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 2 // dwie kolumny
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return component == 0 ? currenciesFrom.count : currenciesTo.count
    }

    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int)
    -> String? {
        return component == 0 ? currenciesFrom[row] : currenciesTo[row]
    }
```

```
    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
    switch component {
    case 0:
        from = row
        textField.placeholder = "Wprowadź kwotę w \(currencies[row])"
    case 1:
        to = row
        labelOutput.text = "Kwota w \(currencies[row])"
    default:
        break
    }
    convertValue()
}
```

W tej funkcji `component` to kolumny (kolumna 0 i 1) i wybieramy że from to wartość z kolumny 0 a to z kolumy 1. Oraz jednocześnie określamy co ma się wyświetlać na pleceholderze pola tekstowego i jaka informacja ma sie wyświetlić w `Label`

Przeliczanie kwoty

```
func convertValue() {
    var rate = 1.0
    var key = "\(currencies[from])-\(currencies[to])"
    if rates.keys.contains(key) {
        rate = rates[key]!
    } else {
        key = "\(currencies[to])-\(currencies[from])"
        if rates.keys.contains(key) {
            rate = 1/rates[key]!
        }
    }
    if let input = textField.text {
        if let value = Double(input) {
            let result = value * rate
            labelOutput.text = "\(String(format: "%.2f", result)) \(currencies[to])"
        } else {
            labelOutput.text = "Wprowadź prawidłową kwotę"
        }
    } else {
        labelOutput.text = "Wprowadź prawidłową kwotę"
    }
}
```

Na początku kurs ustawiany jest na 1 i budowany jest klucz np. `USD-PLN`. Jeżeli taki klucz jest dostępny to przypisujemy nową wartość kursu, jeżeli nie ma takiego klucza to klucz obliczany jest na podstawie odwrtotności przeciwnego klucza.

Dotknięcie ekranu

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    textField.endEditing(true)
}
```

Jeśli użytkownik dotknie ekranu gdzie koliwek poza klawiatura - klawiatura jest zamykana

didViewLoad()

```
override func viewDidLoad() {
    super.viewDidLoad()
    fetchData { (dict,error) in
        if let d = dict {
            for k in d.keys {
                let rate = Double(d[k]!)
                self.rates[k] = rate
            }
        }
    }
}
```

Metoda `didViewLoad()` jest wywoływana zaraz po tym jak widok się załaduje, wiec w niej ustawiamy startową konfiguracje.
W tej metodzie po załadow'niu vidoku ładowane są kursy za pomocą metody `detchData`.

Pobieranie danych

```
func fetchData(completion: @escaping ([String:String]?, Error?) -> Void) {
    let url = URL(string: "https://fcds.cs.put.poznan.pl/MyWeb/Media/currencies.json")!
    let task = URLSession.shared.dataTask(with: url) { (data,response,error) in
        guard let data = data else { return }
        do {
            if let dict = try JSONSerialization.jsonObject(with: data, options: .allowFragments) as?
                completion(dict,nil)
            }
        }
        catch {
            print(error)
            completion(nil,error)
        }
    }
    task.resume()
}
```

Ta funkcja jest odpowiedzialna za wczytywanie danych z serwera. Jeżeli nie ma błędu to zwracamy `completion(dict, nil)` słownik z kursami i brak errorów, a jeżeli pojawił się error to nie zwracamy słownika tylko error.

2 Dodatkowy przykład kiedy picker jest bardziej skomplikowany

```
return component == 0 ? currenciesFrom[row] : currenciesTo[row]
//to to samo co
if component == 0 {
    return currenciesFrom[row]
} else {
    return currenciesTo[row]
}
```

Załóżmy, że mamy dwie tablice z walutami:

```
let currenciesFrom = ["PLN", "USD", "EUR"]
let currenciesTo = ["JPY", "GBP", "CHF", "CAD"]
```

Teraz aby rozplanować pickera robimy to w taki sposób:

```
class ViewController: UIViewController, UIPickerViewDataSource, UIPickerViewDelegate {

    let currenciesFrom = ["PLN", "USD", "EUR"]      //Pierwsza tablica
    let currenciesTo = ["JPY", "GBP", "CHF", "CAD"] //Druga tablica
    private var rates = [String:Double]()
    private var from = 0, to = 0

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var pickerCurrencies: UIPickerView!
    @IBOutlet weak var labelOutput: UILabel!

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

```
    convertValue()
}

func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 2 //Ponownie mamy dwie kolumny w pikerze
}

func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    return component == 0 ? currenciesFrom.count : currenciesTo.count
    // Jeżeli component == 0 - kolumna pierwsza to tyle rzędów ile elementów tablicy
    // Jeżeli inna kolumna to tyle wierszy ile elmenoów tablicy drugiej
}

func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) ->
    return component == 0 ? currenciesFrom[row] : currenciesTo[row]
    // Etykiety rzędów w danych kolumnach
}

func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
    if component == 0 {
        from = row //Przypisujemy do from wartość z rzędu kolumny pierwszej
        textField.placeholder = "Wprowadź kwotę w \(currenciesFrom[row])"
        //Zmieniamy text na "Wprowadź kwotę w np. PLN" w wybranej walucie
    } else {
        to = row
        labelOutput.text = "Kwota w \(currenciesTo[row])"
    }
    convertValue()
}

func convertValue() {
    guard from < currenciesFrom.count, to < currenciesTo.count else { return }

    var rate = 1.0
    let fromCurrency = currenciesFrom[from]
    let toCurrency = currenciesTo[to]
    var key = "\(fromCurrency)-\(toCurrency)"

    if rates.keys.contains(key) {
        rate = rates[key]!
    } else {
        key = "\(toCurrency)-\(fromCurrency)"
        if rates.keys.contains(key) {
            rate = 1 / rates[key]!
        }
    }

    if let input = textField.text, let value = Double(input) {
        let result = value * rate
        labelOutput.text = "\(String(format: "%.2f", result)) \(toCurrency)"
    } else {
        labelOutput.text = "Wprowadź prawidłową kwotę"
```

```
        }
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        textField.endEditing(true)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        fetchData { (dict, error) in
            if let d = dict {
                for k in d.keys {
                    let rate = Double(d[k]!)
                    self.rates[k] = rate
                }
            }
        }
    }

    func fetchData(completion: @escaping ([String:String]?, Error?) -> Void) {
        let url = URL(string: "https://fcds.cs.put.poznan.pl/MyWeb/Media/currencies.json")!
        let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
            guard let data = data else { return }
            do {
                if let dict = try JSONSerialization.jsonObject(with: data, options: .allowFragments)
                    completion(dict, nil)
                }
            } catch {
                print(error)
                completion(nil, error)
            }
        }
        task.resume()
    }
}
```