POLITECHNIKA LUBELSKA

Wydział Elektrotechniki i Informatyki

Kierunek Informatyka

PROJEKT

Programowanie aplikacji mobilnych na platformę iOS

Schronisko dla zwierząt

Wykonał:
Tomasz Wilkołazki
nr albumu: 94520

Lublin 2022

Słowny opis projektu

Celem projektu było stworzenie aplikacji do przechowywania danych o zwierzętach znajdujących się w schronisku. Aplikacja skierowana jest do pracowników ośrodka, a nie jego potencjalnych klientów.

Projekt zawiera cztery widoki i spełnia wymagania projektowe:

I. ContentView – widok startowy aplikacji, zawiera logo oraz podstawowe informacje o schronisku. Implementuje mapę wraz z pinezką wskazującą na lokalizację ośrodka. Na dole widoki znajduje się przycisk kierujący do kolejnego widoku.

II. ListView – widok pobierający dane z CoreData i wyświetla je w pogrupowanej na sekcje liscie. Nad listą widoczny jest tytuł implementujący gest *LongPress*, który po przytrzymaniu zmienia wielkość czcionki. Gest *Swipe* umożliwia usunięcie elementu z listy poprzez wywołanie funkcji deleteAnimal(). Gest *Tap* na elemencie listy spowoduje przeniesienie do widoku ze szczegółowymi informacjami o danym zwierzęciu oraz możliwość edycji jego danych. Na pasku nawigacji widoczny jest przycisk **Dodaj,** który przenosi do widoku dodawania nowego zwierzęcia.

III. addAnimalView – widok dodania zwierzęcia do CoreData poprzez formularz. Gdy w formularzu nie można znaleźć odpowiednich typów(types) wyświetlany jest przycisk dodający dane typy do CoreData. Formularz jest walidowany: data nie może być z przyszłości, wiek musi być cyfrą (wykorzystuje bibliotekę Combine). Przycisk **Dodaj zwierzę** dodaję dane do CoreData. Przed ich dodaniem sprawdzana jest ich poprawność, zmieniany jest ich typ (jeśli jest to konieczne). Po dodaniu pustych danych dodane zostaną odpowiednie informacje a typ zwierzęcia określony zostanie domyślnie na „Inne”.

IV. editAnimalView – widok szczegółowych informacji oraz edycji danych. Widok zapisuje zmiany formularza do kontekstu. Na górze strony wyświetlany jest obrazek pobierany z Assets na kolejno na podstawie Rasy>>Typu, W przypadku wystąpienia błędu wyświetlone zostanie logo programu. Tak jak w widoku dodania jest tu przycisk do inicjalizacji typów zwierzat.

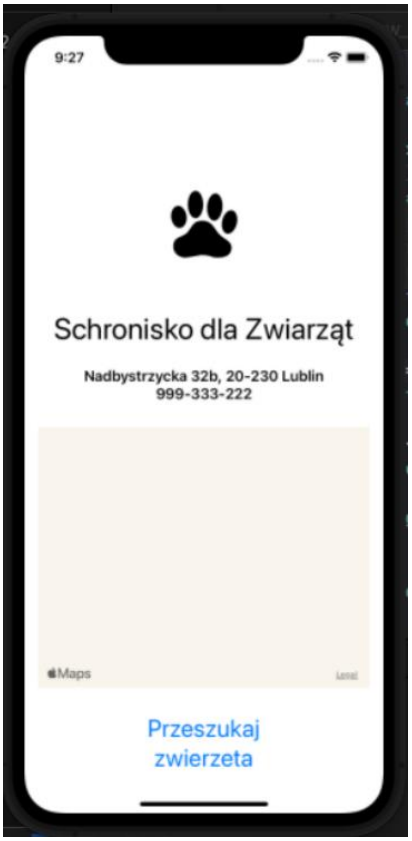
Widok I

Klasa **MyAnnotation** tworzy pinezkę dla mapy.

Struktura **MapView** tworzy i steruje akcjami dla mapy.

W **ContentView** tworzę zmienną tworzącą pinezkę na mapie.

Wyświetlam podstawowe informację o Schronisku oraz mapę z pinezką. Przycisk „Przeszukaj zwierzęta” przenosi do kolejnego widoku – ListView().

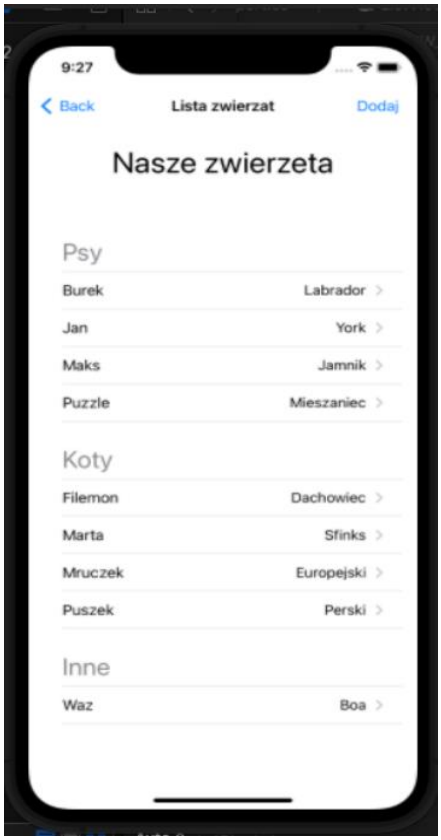



```
import CoreData
import SwiftUI
import MapKit

class MyAnnotation: NSObject, MKAnnotation {
    var title: String?
    var subtitle: String?
    var coordinate: CLLocationCoordinate2D

    init(title: String?, subtitle: String, coordinate: CLLocationCoordinate2D)
{
        self.title = title
        self.subtitle = title
        self.coordinate = coordinate
    }
}

struct MapView: UIViewRepresentable {
    @Binding var myAnnotation: MyAnnotation

    func makeUIView(context: UIViewRepresentableContext<MapView>) -> MKMapView
{
        let mapView = MKMapView(frame: .zero)
        return mapView
    }

    func updateUIView(_ uiView: MKMapView, context: UIViewRepresentableCon-
text<MapView>) {

        let span = MKCoordinateSpan(latitudeDelta: 0.1, longitudeDelta: 0.1)
        let region = MKCoordinateRegion(center:
        myAnnotation.coordinate, span: span)
        uiView.setRegion(region, animated: true)
        uiView.addAnnotations([myAnnotation])
    }
}
```

```
struct ContentView: View {

    @State var myAnnotation = MyAnnotation(title: "Nasza lokacja", subtitle:
"Schronisko dla zwierząt",
        coordinate: CLLocationCoordinate2D(latitude:51.2353112433304,
longitude: 22.55289822681853))


    var body: some View {
        NavigationView{
            VStack{
                Image("LOGO")
                    .resizable()
                    .scaledToFit()
                    .frame(width: 150, height: 150, alignment: .center)
                    .clipShape(Circle())

                Text("Schronisko dla Zwiarząt")
                    .font(.largeTitle)
                    .padding()
                Text("Nadbystrzycka 32b, 20-230 Lublin")
                    .font(.headline)
                Text("999-333-222")
                    .font(.headline)

                MapView(myAnnotation: $myAnnotation)
                .gesture(TapGesture()
                    .onEnded(){
                    })
                .frame(maxWidth: .infinity)
                .frame(height: 300)
                .padding()

                NavigationLink(
                    destination: ListView(), label: {
                        Text("Przeszukaj zwierzeta")
                            .font(.title).multilineTextAlignment(.center)
                        .frame(width: 200, height: 80)
                }
                )
                Spacer()
            }
            }.navigationBarTitle("Schronisko dla zwierzat")
    }
}
```

Widok II

Utworzenie zmiennych animals oraz types i połączenie ich z CoreData.
Utworzenie gestu długiego przyciśnięcia oraz zwrócenie VStack-a zawierającego wszystkie widoczne na ekranie elementy. Tytuł wykorzystuje ten gest i na jego podstawie zmienia na chwilę wielkość czcionki.
Następnie wyświetlana jest lista podzielona na sekcje według typów. Lista iteruje po elementach CoreData w celu wyświetlenia ich. Przyciśnięcie elementu listy wywoła **NavigationLink** przekierowujący do widoku edycji przekazując do niego dane tego obiektu.
Gest Swipe na liscie umożliwi wywołanie funkcji usunięcia danego obiektu.
Kolejno ustawiany jest styl listy tytuł nawigacji oraz dodany jest przycisk „Dodaj" przekierowujący do widoku dodania nowego obiektu.



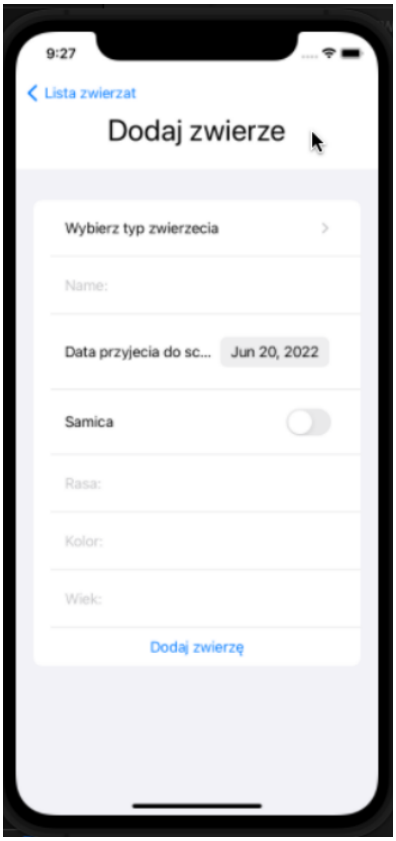

```
import SwiftUI
import CoreData

struct ListView: View {

    @Environment(\.managedObjectContext) private var viewContext
    @FetchRequest(sortDescriptors: [NSSortDescriptor(keyPath:
        \Animal.name, ascending: true)], animation: .default)
    private var animals: FetchedResults<Animal>

    @FetchRequest(sortDescriptors: [NSSortDescriptor(keyPath:
        \Type.name, ascending: true)], animation: .default)
    private var types: FetchedResults<Type>

    @GestureState var isLongPress = false

    var body: some View {

        let longPress = LongPressGesture()
            .updating(self.$isLongPress){value,state,transaction in
            state = value
        }

        return VStack{

            Spacer()
            Text("Nasze zwierzeta")
                .padding()
                .font(isLongPress ? .headline : .largeTitle)
                .gesture(longPress)

            List{
                Section(header: Text("Psy").font(.title)){
                ForEach(animals){animal in
                    if animal.type == "Pies"{
                        NavigationLink(
                            destination: editAnimalView(animal:animal), label:
{

                                Text(animal.name!)
                                Spacer()
                                Text(animal.breed!)
                                    .font(.callout)
                            })
                    }}
                .onDelete(perform: self.deleteAnimal)
                }
```

```
                Section(header: Text("Koty").font(.title)){
                ForEach(animals){animal in
                    if animal.type == "Kot"{
                        NavigationLink(
                            destination: editAnimalView(animal:animal), label:
{

                                Text(animal.name!)
                                Spacer()
                                Text(animal.breed!)
                                    .font(.callout)
                            })}
                }
                .onDelete(perform: self.deleteAnimal)
                }

                Section(header: Text("Inne").font(.title)){
                ForEach(animals){animal in
                    if animal.type == "Inne"{
                        NavigationLink(
                            destination: editAnimalView(animal:animal), label:
{

                                Text(animal.name!)
                                Spacer()
                                Text(animal.breed!)
                                    .font(.callout)
                            })}
                }
                .onDelete(perform: self.deleteAnimal)
                }
            }
            .padding()
            .listStyle(PlainListStyle())
            .navigationBarTitle("Lista zwierzat",displayMode: .inline)
            .navigationBarItems(
                trailing:NavigationLink("Dodaj", destination: addAnima-
lView()))
        }
    }
```

```
    private func deleteAnimal(offset: IndexSet){
        withAnimation{
            offset.map{ animals[$0]}.forEach(viewContext.delete)

            do {
                try viewContext.save()
            }
            catch {
                let err = error as NSError
                fatalError("\(err)")
            }
        }
```

Funkcja pozwalająca usunąć obiekt z listy. Otrzymuje ona Index na którym element się znajduję. Na tym elemencie iteruje po wszystkich jego polach i usuwa je, po czym zapisuje kontekst – co oznacza zatwierdzenie zmian w CoreData. Jeśli operacja się nie powiedzie program zgłosi błąd.

Widok III



Na początku łączę się z CoreData tworząc zmienne animals opra types. Następnie tworze pola w których będę przechowywać dane z formularza oraz zmienną środowiskową dzięki której po dodaniu nowego obiektu do listy, przycisk przeniesie użytkownika z powrotem. Podczas gdy nie widoczne są typy zwierząt należy zainicjalizować te wartości do CoreData przyciskiem widocznym tylko gdy w CoreData nie ma żadnych wartości.
Następnie widoczne są pola formularza w których wykorzystane są takie kontrolki jak: TextField, Picker, DatePicker oraz Toggle.
Pola formularza są odpowiednia walidowane podczas wpisywania oraz przed dodaniem do CoreData.
Kliknięcie przycisku powoduje wywołanie funkcji addAnimal().

```
import CoreData
import SwiftUI
import Combine

struct addAnimalView: View {

    @Environment(\.managedObjectContext) private var viewContext
    @FetchRequest(sortDescriptors: [NSSortDescriptor(keyPath:
        \Animal.name, ascending: true)], animation: .default)
    private var animals: FetchedResults<Animal>

    @FetchRequest(sortDescriptors: [NSSortDescriptor(keyPath:
        \Type.name, ascending: true)], animation: .default)
    private var types: FetchedResults<Type>

    @State private var name: String = ""
    @State private var sex: Bool = false
    @State private var date_admission: Date = Date()
    @State private var breed: String = ""
    @State private var colour: String = ""
    @State private var age: String = ""
    @State private var type: Type?
    @Environment(\.presentationMode) private var presentationMode: Bin-
ding<PresentationMode>


    var body: some View {
        VStack{
            Text("Dodaj zwierze")
                .font(.largeTitle)
            if (types.isEmpty){
                Button("Dodaj typy zwierzat"){self.initTypes()}
            }
```

```
Form
            {
                Picker(selection: $type,
                    label: Text("Wybierz typ zwierzecia")){
                    ForEach(types, id: \.self) { (type: Type) in
                    Text(type.name!).tag(type as Type?)
                    }}.padding()
                TextField("Name:",text: $name).padding()
                    DatePicker(selection: $date_admission, in: ...Date(), di-
splayedComponents: .date){
                    Text("Data przyjecia do schroniska")
                }.padding()
                Toggle(isOn: $sex){
                        if (sex == true) {Text("Samiec")}
                        else{Text("Samica")}
                }.padding()

                TextField("Rasa:",text: $breed).padding()

                TextField("Kolor:",text: $colour).padding()
                TextField("Wiek:",text: $age).padding()
                        .keyboardType(.numberPad)
                        .onReceive(Just(age)) {
                            newValue in
                            let filtered = newValue.filter{"0123456789".conta-
ins($0)}
                            if filtered != newValue {
                                self.age = filtered
                            }
                            }

                        HStack{
                        Spacer()
                    Button("Dodaj zwierzę"){
                        self.addAnimal()
                        self.presentationMode.wrappedValue.dismiss()
                    }
                        Spacer()
                }

                }
```

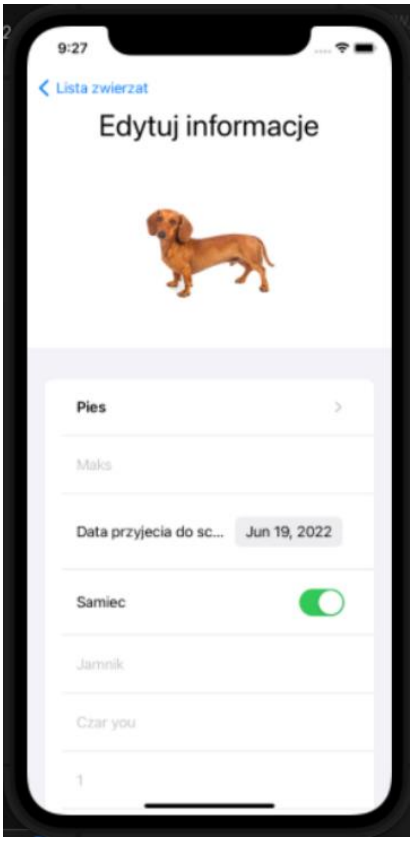
```swift
private func initTypes(){

        let newType1 = Type(context: viewContext)
            newType1.name = "Pies"

        let newType2 = Type(context: viewContext)
            newType2.name = "Kot"

        let newType3 = Type(context: viewContext)
            newType3.name = "Inne"

        do {
            try viewContext.save()
        }
        catch {
            let err = error as NSError
            fatalError("\(err)")
        }
}
private func addAnimal(){
       let newAnimal = Animal(context: viewContext)
        newAnimal.sex = sex
    if (name.isEmpty){ newAnimal.name = "Nie podano imienia" }
    else{newAnimal.name = name}
    if (breed.isEmpty){ newAnimal.breed = "Brak informacji o rasie" }
    else{newAnimal.breed = breed}
    newAnimal.date_admission = date_admission
    if (colour.isEmpty) {newAnimal.colour = "Brak informacji o kolorze"}
    else{ newAnimal.colour = colour }

    if (type?.name == nil) {newAnimal.type = "Inne"}
    else{ newAnimal.type = type?.name }

    if (age.isEmpty){
        newAnimal.age = -1
    }
    else{newAnimal.age = Int16(age)!}

       do {
           try viewContext.save()
       }
       catch {
           let err = error as NSError
           fatalError("\(err)")
       }
   }
}
```

Funkcja **initTypes()** populuje CoreData podstawowymi wartościami. Wywoływana jest za pomocą przycisku jednokrotnie podczas działania programu podczas jego pierwszego uruchomienia.

Funkcja **addAnimal()** tworzy zmienną typu Animal dla danego kontekstu.

Ustawia pola dla tej zmiennej jednocześnie weryfikując ich poprawność po czym zapisuje kontekst, dodając tymsamym dane do CoreData.

Widok IV



Widok podobny do poprzedniego. Wywoływany poprzez kliknięcie elementu listy. Wyświetla obrazek na podstawie podanej rasy, lub jeśli nie jest ona wpisana lub obraz o nazwie danej klasy nie znajduje się w Assets zostanie wyświetlony obrazek dla danego typu zwierzęcia. Wyświielta dane o zwierzęciu pobrane z CoreData. Umożliwia modyfikację tych danych poprzez zmianę ich i wywołanie funkcji **editAnimal()** za pomocą przycisku. Funkcja ta zapisuje kontekst.

```
import CoreData
import SwiftUI
import Combine

struct editAnimalView: View {

    @Environment(\.managedObjectContext) private var viewContext
    @FetchRequest(sortDescriptors: [NSSortDescriptor(keyPath:
        \Animal.name, ascending: true)], animation: .default)
    private var animals: FetchedResults<Animal>

    @FetchRequest(sortDescriptors: [NSSortDescriptor(keyPath:
        \Type.name, ascending: true)], animation: .default)
    private var types: FetchedResults<Type>

    @State private var name: String = ""
    @State private var breed: String = ""
    @State private var colour: String = ""
    @State private var age: String = ""
    @State private var type: Type?
    @Environment(\.presentationMode) private var presentationMode: Bin-
ding<PresentationMode>

    var animal:Animal
    var body: some View {
        VStack{
            Text("Edytuj informacje")
                .font(.largeTitle)

            if (UIImage(named: animal.breed!.capitalized) != nil){
                Image(animal.breed!)
                    .resizable()
                    .scaledToFit()
                    .frame(width: 200, height: 200, alignment: .center)
                    .clipShape(Circle())            }
            else if (UIImage(named: animal.type!) != nil){
                Image(animal.type!)
                    .resizable()
                    .scaledToFit()
                    .frame(width: 200, height: 200, alignment: .center)
                    .clipShape(Circle())            }
            else {Image("LOGO")
                    .resizable()
                    .scaledToFit()
                    .frame(width: 200, height: 200, alignment: .center)
                    .clipShape(Circle())            }
            if (types.isEmpty){
                Button("Dodaj typy zwierzat"){self.initTypes()}
            }
```

```
Picker(selection: $type, label: Text(animal.type ?? "Brak informa-
cji").font(.headline)){
                    ForEach(types, id: \.self) { (type: Type) in
                    Text(type.name!).tag(type as Type?)
                    }}.padding()
            TextField(animal.name ?? "Imie psa:", text: $name).padding()

                DatePicker(selection: Binding<Date>(
                    get: {self.animal.date_admission ?? Date()},
                    set: {self.animal.date_admission = $0
                        try? self.viewContext.save()
                    }), in: ...Date(), displayedComponents: .date){
                    Text("Data przyjecia do schroniska")
                }.padding()

                Toggle(isOn: Binding<Bool>(
                    get: {self.animal.sex},
                    set: {
                        self.animal.sex = $0
                        try? self.viewContext.save()
                    })){
                        if (animal.sex == true) {Text("Samiec")}
                        else{Text("Samica")}
                    }.padding()
            TextField(animal.breed ?? "Rasa:", text: $breed).padding()
            TextField(animal.colour ?? "Kolor:",text: $colour).padding()
                TextField(String(animal.age) ,text: $age).padding()
                    .keyboardType(.numberPad)
                    .onReceive(Just(age)) {
                        newValue in
                        let filtered = newValue.filter{"0123456789".conta-
ins($0)}
                        if filtered != newValue {
                            self.age = filtered
                        }
                        }

                    HStack{
                    Spacer()
                Button("Zapisz"){
                    if (!self.name.isEmpty) {self.animal.name = self.name}
                    if (!self.age.isEmpty) {self.animal.age =
Int16(self.age)!}
                    if (!self.breed.isEmpty) {self.animal.breed = self.breed}
                    if (!self.colour.isEmpty) {self.animal.colour = self.co-
lour}
                    if (self.type != nil) {self.animal.type = self.type?.name}
                    self.editAnimal()
                    self.presentationMode.wrappedValue.dismiss()
```

```
    private func initTypes(){

        let newType1 = Type(context: viewContext)
            newType1.name = "Pies"

        let newType2 = Type(context: viewContext)
            newType2.name = "Kot"

        let newType3 = Type(context: viewContext)
            newType3.name = "Inne"

        do {
            try viewContext.save()
        }
        catch {
            let err = error as NSError
            fatalError("\(err)")
        }
}

private func editAnimal(){
       do {
           try viewContext.save()
       }
       catch {
           let err = error as NSError
           fatalError("\(err)")
       }
   }
}
```

Funkcja inicjalizująca typy oraz funkcja zapisująca kontekst po edycji obiektu.

Podsumowanie

Projekt w dużym stopniu wymusił powtórzenie materiału o programowaniu aplikacji na IOS realizowanego podczas laboratoriów. Jego napisanie sprawiło że moje podejście do tego przedmiotu uległo zmianie. Czuję się bardziej komfortowo pisząc kod w języku Swift, lepiej go rozumiem i szybciej odnajduję popełnione błędy. Tworzenie GUI – stylizacja poszczególnych elementów (kontrolek) przyniosły mi sporą satysfakcję i były zdecydowanie moim ulubionym elementem działania. Backend i CoreData są dla mnie zrozumiałe, ale wolałbym stosować inne technologie gdyby była taka możliwość w aplikacjach mobilnych dla IOS.