# iOS native mobile coursework

Name: Dilan Jayamanne

IIT ID: 2019284

UOW ID: w1761267

//

// AirPollutionModelData.swift

// CourseWork2Starter

//

// Created by Dilan Jayamanne on 02/04/23.

//

**import** Foundation

// **MARK: - AirPollutionModelData**

**struct** AirPollutionModelData: Codable {

**let** coord: Coord

**let** list: [AirPollutionDataList]

}

// **MARK: - Coord**

**struct** Coord:Codable{

**let** lon : Double

**let** lat : Double

}

// **MARK: - AirPollutionDataList**

**struct** AirPollutionDataList:Codable {

**let** dt: Int

**let** main: AirPollutionDataMain

**let** components: AirPollutionDataComponents

}

// **MARK: - AirPollutionDataMain**

**struct** AirPollutionDataMain:Codable {

**let** aqi:Int

}

// **MARK: - AirPollutionDataComponents**

**struct** AirPollutionDataComponents:Codable {

**let** co:Double

**let** no:Double

**let** no2:Double

**let** o3:Double

**let** so2:Double

**let** pm2\_5:Double

**let** pm10:Double

**let** nh3:Double

}

//

// PollutionView.swift

// Coursework2

//

// Created by GirishALukka on 30/12/2022.

//

**import** SwiftUI

**struct** PollutionView: View {

// @EnvironmentObject and @State varaibles here

//@EnvironmentObject var PollutionData: AirPollutionModelData

@EnvironmentObject **var** modelData: ModelData

@State **var** locationString: String = "Location loading..."

**var** body: **some** View {

ZStack {

// Use ZStack for background images

Image("background")

.resizable()

.ignoresSafeArea()

.blur(radius: 7)

VStack {

Text("\(locationString)")

.padding(EdgeInsets(top: 20, leading: 10, bottom: 50, trailing: 10))

.font(.title)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

.multilineTextAlignment(.center)

ScrollView {

VStack(spacing: 30){

Text("\((Int)(modelData.forecast!.current.temp))ºC")

.padding()

.font(.largeTitle)

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color(red: 30.0/255, green: 41.0/255, blue: 193.0/255) : Color(red: 186.0/255, green: 75.0/255, blue: 15.0/255))

.background(

RoundedRectangle(cornerRadius: 10)

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color.blue.opacity(0.1) : Color.orange.opacity(0.1))

)

HStack{

AsyncImage(url: URL(string: "https://openweathermap.org/img/wn/\(modelData.forecast!.current.weather[0].icon)@2x.png"))

.frame(width: 70, height: 70)

Text("\(modelData.forecast!.current.weather[0].weatherDescription.rawValue)")

.padding()

.font(.title2)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

}

Text("Feels Like: \((Int)(modelData.forecast!.current.feelsLike))ºC")

.padding()

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color(red: 30.0/255, green: 41.0/255, blue: 193.0/255) : Color(red: 186.0/255, green: 75.0/255, blue: 15.0/255))

.background(

RoundedRectangle(cornerRadius: 10)

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color.blue.opacity(0.1) : Color.orange.opacity(0.1))

)

Text("Air Quality Data:").bold().font(.title2)

HStack{

VStack{

Image("so2")

.resizable()

.aspectRatio(contentMode: .fit)

.clipShape(RoundedRectangle(cornerRadius: 10))

Text("\(String(format: "%.2f",modelData.airPollution?.list[0].components.so2 ?? 0.0))")

}

VStack{

Image("no")

.resizable()

.aspectRatio(contentMode: .fit)

.clipShape(RoundedRectangle(cornerRadius: 10))

Text("\((String)(format: "%.2f",modelData.airPollution?.list[0].components.no2 ?? 0.0))")

}

VStack{

Image("voc")

.resizable()

.aspectRatio(contentMode: .fit)

.clipShape(RoundedRectangle(cornerRadius: 10))

Text("\((String)(format: "%.2f",modelData.airPollution?.list[0].components.co ?? 0.0))")

}

VStack{

Image("pm")

.resizable()

.aspectRatio(contentMode: .fit)

.clipShape(RoundedRectangle(cornerRadius: 10))

Text("\((String)(format: "%.2f",modelData.airPollution?.list[0].components.pm10 ?? 0.0))")

}

}

}

}

}

}.onAppear{

Task.init {

**let** \_ = **try** **await** modelData.loadAirPollutionData()

**self**.locationString = **await** getLocFromLatLong(lat: modelData.forecast!.lat, lon: modelData.forecast!.lon)

}

}

}

}

**struct** Pollution\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

PollutionView().environmentObject(ModelData())

}

}

//

// LocationHelper.swift

// Coursework2

//

// Created by G Lukka.

//

**import** Foundation

**import** CoreLocation

**func** getLocFromLatLong(lat: Double, lon: Double) **async** -> String

{

**var** locationString: String

**var** placemarks: [CLPlacemark]

**let** center: CLLocationCoordinate2D = CLLocationCoordinate2D(latitude: lat, longitude: lon)

**let** ceo: CLGeocoder = CLGeocoder()

**let** loc: CLLocation = CLLocation(latitude: center.latitude, longitude: center.longitude)

**do** {

placemarks = **try** **await** ceo.reverseGeocodeLocation(loc)

**if** placemarks.count > 0 {

**if** (!placemarks[0].name!.isEmpty) {

locationString = placemarks[0].name!

} **else** {

locationString = (placemarks[0].locality ?? "No City")

}

**return** locationString

}

} **catch** {

print("Reverse geodecoe fail: \(error.localizedDescription)")

locationString = "No City, No Country"

**return** locationString

}

**return** "Error getting Location"

}

//

// WindDirHelper.swift

// Coursework2

//

// Created by G Lukka.

//

**import** Foundation

**func** convertDegToCardinal(deg: Int) -> String {

**let** cardinalDir = ["N","NNE","NE","ENE","E","ESE","SE","SSE","S","SSW","SW","WSW","W","WNW","NW","NNW","N"]

**return** cardinalDir[Int(round(((Double)(deg % 360)) / 22.5).nextDown) + 1]

}

**import** Foundation

**class** ModelData: ObservableObject {

@Published **var** forecast: Forecast?

@Published **var** userLocation: String = ""

@Published **var** airPollution : AirPollutionModelData?

**init**() {

**self**.forecast = load("london.json")

}

// function to load weather data from API

**func** loadData(lat: Double, lon: Double) **async** **throws** -> Forecast {

**let** url = URL(string: "https://api.openweathermap.org/data/3.0/onecall?lat=\(lat)&lon=\(lon)&units=metric&appid=85b4fd8ba41c782d11db5c52f15fcbf1")

**let** session = URLSession(configuration: .default)

**let** (data, \_) = **try** **await** session.data(from: url!)

**do** {

//print(data)

**let** forecastData = **try** JSONDecoder().decode(Forecast.**self**, from: data)

DispatchQueue.main.async {

**self**.forecast = forecastData

}

**return** forecastData

} **catch** {

**throw** error

}

}

// function to load air pollution data from API

**func** loadAirPollutionData() **async** **throws** {

**guard** **let** lat = forecast?.lat, **let** lon = forecast?.lon **else** {

fatalError("Couldn't find in lat & lon in forcast data.")

}

**let** url = URL(string: "https://api.openweathermap.org/data/2.5/air\_pollution?lat=\(lat)&lon=\(lon)&appid=85b4fd8ba41c782d11db5c52f15fcbf1"

)

**let** session = URLSession(configuration: .default)

**let** (data,\_) = **try** **await** session.data(from:url!)

**do**{

**let** pollutionData = **try** JSONDecoder().decode(AirPollutionModelData.**self**, from: data)

DispatchQueue.main.async {

**self**.airPollution = pollutionData

print("testing",**self**.airPollution?.list[0].components)

}

} **catch** {

**throw** error

}

}

**func** load<Forecast: Decodable>(\_ filename: String) -> Forecast {

**let** data: Data

**guard** **let** file = Bundle.main.url(forResource: filename, withExtension: **nil**)

**else** {

fatalError("Couldn't find \(filename) in main bundle.")

}

**do** {

data = **try** Data(contentsOf: file)

} **catch** {

fatalError("Couldn't load \(filename) from main bundle:\n\(error)")

}

**do** {

**let** decoder = JSONDecoder()

**return** **try** decoder.decode(Forecast.**self**, from: data)

} **catch** {

fatalError("Couldn't parse \(filename) as \(Forecast.**self**):\n\(error)")

}

}

}

// This file was generated from JSON Schema using quicktype, do not modify it directly.

// To parse the JSON, add this file to your project and do:

//

// let forecast = try? newJSONDecoder().decode(Forecast.self, from: jsonData)

**import** Foundation

// **MARK: - Forecast**

**struct** Forecast: Codable, Identifiable {

**let** id = UUID()

**let** lat, lon: Double

**let** timezone: String

**let** timezoneOffset: Int

**let** current: Current

**let** minutely: [Minutely]?

**let** hourly: [Current]

**let** daily: [Daily]

**enum** CodingKeys: String, CodingKey {

**case** lat, lon, timezone

**case** timezoneOffset = "timezone\_offset"

**case** current, minutely, hourly, daily

}

}

// **MARK: - Current**

**struct** Current: Codable, Identifiable {

**let** id = UUID()

**let** dt: Int

**let** sunrise, sunset: Int?

**let** temp, feelsLike: Double

**let** pressure, humidity: Int

**let** dewPoint, uvi: Double

**let** clouds, visibility: Int

**let** windSpeed: Double

**let** windDeg: Int

**let** weather: [Weather]

**let** windGust, pop: Double?

**let** rain: Rain?

**enum** CodingKeys: String, CodingKey {

**case** dt, sunrise, sunset, temp

**case** feelsLike = "feels\_like"

**case** pressure, humidity

**case** dewPoint = "dew\_point"

**case** uvi, clouds, visibility

**case** windSpeed = "wind\_speed"

**case** windDeg = "wind\_deg"

**case** weather

**case** windGust = "wind\_gust"

**case** pop, rain

}

}

// **MARK: - Rain**

**struct** Rain: Codable {

**let** the1H: Double

**enum** CodingKeys: String, CodingKey {

**case** the1H = "1h"

}

}

// **MARK: - Weather**

**struct** Weather: Codable {

**let** id: Int

**let** main: Main

**let** weatherDescription: Description

**let** icon: String

**enum** CodingKeys: String, CodingKey {

**case** id, main

**case** weatherDescription = "description"

**case** icon

}

}

// **MARK: - Main**

**enum** Main: String, Codable {

**case** clear = "Clear"

**case** clouds = "Clouds"

**case** rain = "Rain"

**case** mist = "Mist"

**case** smoke = "Smoke"

**case** haze = "Haze"

**case** dust = "Dust"

**case** fog = "Fog"

**case** sand = "Sand"

**case** ash = "Ash"

**case** squall = "Squall"

**case** tornado = "Tornado"

**case** snow = "Snow"

**case** drizzle = "Drizzle"

**case** thunderstorm = "Thunderstorm"

}

// **MARK: - Description**

**enum** Description: String, Codable {

**case** brokenClouds = "broken clouds"

**case** clearSky = "clear sky"

**case** fewClouds = "few clouds"

**case** lightRain = "light rain"

**case** moderateRain = "moderate rain"

**case** overcastClouds = "overcast clouds"

**case** scatteredClouds = "scattered clouds"

**case** thunderstormWithLightRain = "thunderstorm with light rain"

**case** thunderstormWithRain = "thunderstorm with rain"

**case** thunderstormWithHeavyRain = "thunderstorm with heavy rain"

**case** lightThunderstorm = "light thunderstorm"

**case** thunderstorm = "thunderstorm"

**case** heavyThunderstorm = "heavy thunderstorm"

**case** raggedThunderstorm = "ragged thunderstorm"

**case** thunderstormWithLightDrizzle = "thunderstorm with light drizzle"

**case** thunderstormWithDrizzle = "thunderstorm with drizzle"

**case** thunderstormWithHeavyDrizzle = "thunderstorm with heavy drizzle"

**case** heavyIntensityDrizzle = "heavy intensity drizzle"

**case** lightIntensityDrizzleRain = "light intensity drizzle rain"

**case** drizzleRain = "drizzle rain"

**case** heavyIntensityDrizzleRain = "heavy intensity drizzle rain"

**case** showerRainAndDrizzle = "shower rain and drizzle"

**case** heavyShowerRainAndDrizzle = "heavy shower rain and drizzle"

**case** showerDrizzle = "shower drizzle"

**case** heavyIntensityRain = "heavy intensity rain"

**case** veryHeavyRain = "very heavy rain"

**case** extremeRain = "exteme rain"

**case** freezingRain = "freezing rain"

**case** lightIntensityShowerRain = "light intensity shower rain"

**case** showerRain = "shower rain"

**case** heavyIntensityShowerRain = "heavy intensity shower rain"

**case** raggedShowerRain = "ragged shower rain"

**case** lightSnow = "light snow"

**case** Snow = "Snow"

**case** HeavySnow = "Heavy snow"

**case** Sleet = "Sleet"

**case** LightShowerSleet = "Light shower sleet"

**case** ShowerSleet = "Shower sleet"

**case** LightRainAndSnow = "Light rain and snow"

**case** RainAndSnow = "Rain and snow"

**case** LightShowerSnow = "Light shower snow"

**case** ShowerSnow = "Shower snow"

**case** HeavyShowerSnow = "Heavy shower snow"

**case** mist = "mist"

**case** Smoke = "Smoke"

**case** Haze = "Haze"

**case** sandDustWhirls = "sand/dust whirls"

**case** fog = "fog"

**case** sand = "sand"

**case** dust = "dXust"

**case** volcanicAsh = "volcanic ash"

**case** squalls = "squalls"

**case** tornado = "tornado"

**case** fewClouds1125 = "few clouds: 11-25%"

**case** scatteredClouds2550 = "scattered clouds: 25-50%"

**case** brokenClouds5184 = "broken clouds: 51-84%"

**case** overcastClouds85100 = "overcast clouds: 85-100%"

}

// **MARK: - Daily**

**struct** Daily: Codable, Identifiable {

**let** id = UUID()

**let** dt, sunrise, sunset, moonrise: Int

**let** moonset: Int

**let** moonPhase: Double

**let** temp: Temp

**let** feelsLike: FeelsLike

**let** pressure, humidity: Int

**let** dewPoint, windSpeed: Double

**let** windDeg: Int

**let** windGust: Double

**let** weather: [Weather]

**let** clouds: Int

**let** pop: Double

**let** rain: Double?

**let** uvi: Double

**enum** CodingKeys: String, CodingKey {

**case** dt, sunrise, sunset, moonrise, moonset

**case** moonPhase = "moon\_phase"

**case** temp

**case** feelsLike = "feels\_like"

**case** pressure, humidity

**case** dewPoint = "dew\_point"

**case** windSpeed = "wind\_speed"

**case** windDeg = "wind\_deg"

**case** windGust = "wind\_gust"

**case** weather, clouds, pop, rain, uvi

}

}

// **MARK: - FeelsLike**

**struct** FeelsLike: Codable {

**let** day, night, eve, morn: Double

}

// **MARK: - Temp**

**struct** Temp: Codable {

**let** day, min, max, night: Double

**let** eve, morn: Double

}

// **MARK: - Minutely**

**struct** Minutely: Codable {

**let** dt: Int

**let** precipitation: Double

}

//

// NavBar.swift

// Coursework2

//

// Created by G Lukka.

//

**import** SwiftUI

**struct** NavBar: View {

@EnvironmentObject **var** modelData: ModelData

// Tab view accent color changes to orange if the temp is > 50 otherwise its blue

**var** body: **some** View {

TabView{

Home()

.tabItem{

Label("City", systemImage: "magnifyingglass")

}

CurrentWeatherView()

.tabItem {

Label("WeatherNow", systemImage: "sun.max")

}

HourlyView()

.tabItem{

Label("HourlyView", systemImage: "clock.fill")

}

ForecastView()

.tabItem {

Label("ForecastView", systemImage: "calendar")

}

PollutionView()

.tabItem {

Label("PollutionView", systemImage: "aqi.high")

}

}.onAppear {

UITabBar.appearance().isTranslucent = **false**

}

.accentColor(modelData.forecast!.current.temp < 15 ? Color(.blue) : Color(.orange))

}

}

**struct** NavBar\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

NavBar().environmentObject(ModelData())

}

}

//

// DailyView.swift

// Coursework2

//

// Created by G Lukka.

//

**import** SwiftUI

**struct** DailyView: View {

**var** day : Daily

**var** body: **some** View {

HStack {

AsyncImage(url: URL(string: "https://openweathermap.org/img/wn/\(day.weather[0].icon)@2x.png"))

.frame(width: 70, height: 70, alignment: .leading)

Spacer()

VStack {

Text("\(day.weather[0].weatherDescription.rawValue)")

.font(.subheadline)

Text(Date(timeIntervalSince1970: TimeInterval(((Int)(day.dt)))).formatted(.dateTime.weekday().day()))

.font(.subheadline)

}

Spacer()

// color changes to orange if the temperature > 15

Text("\((Int)(day.temp.max))°C/\((Int)(day.temp.min))°C")

.foregroundColor(day.temp.max < 15 ? Color(red: 30.0/255, green: 41.0/255, blue: 193.0/255) : Color(red: 186.0/255, green: 75.0/255, blue: 15.0/255))

.background(

RoundedRectangle(cornerRadius: 10)

.foregroundColor(day.temp.max < 15 ? Color.blue.opacity(0.1) : Color.orange.opacity(0.1))

)

.font(.subheadline)

.frame(alignment: .trailing)

}

.padding()

.foregroundColor(.black)

.frame(maxWidth: .infinity, alignment: .leading)

.background(RoundedRectangle(cornerRadius: 10).foregroundColor(.white).opacity(0.7))

.shadow(color: .white, radius: 0.5)

}

}

**struct** DailyView\_Previews: PreviewProvider {

**static** **var** day = ModelData().forecast!.daily

**static** **var** previews: **some** View {

DailyView(day: day[0])

}

}

//

// SearchView.swift

// CWK2\_23\_GL

//

// Created by GirishALukka on 11/03/2023.

//

**import** SwiftUI

**import** CoreLocation

**struct** SearchView: View {

@EnvironmentObject **var** modelData: ModelData

@Binding **var** isSearchOpen: Bool

@State **var** location = ""

@Binding **var** userLocation: String

**var** body: **some** View {

ZStack {

// Blur(style: .systemUltraThinMaterial)

// .ignoresSafeArea()

// RoundedRectangle(cornerRadius: 20)

// .fill(Color.teal)

// .opacity(0.8)

// .ignoresSafeArea()

// color changes to orange if the temperature > 15

**if**(modelData.forecast!.current.feelsLike < 15){

Color.blue.opacity(0.3)

.background(.ultraThinMaterial.opacity(0.7))

.ignoresSafeArea()

}**else**{

Color.orange.opacity(0.3)

.background(.ultraThinMaterial.opacity(0.7))

.ignoresSafeArea()

}

VStack{

HStack {

TextField("Enter New Location", text: **self**.$location, onCommit: {

CLGeocoder().geocodeAddressString(location) { (placemarks, error) **in**

**if** **let** lat = placemarks?.first?.location?.coordinate.latitude,

**let** lon = placemarks?.first?.location?.coordinate.longitude {

Task{

**let** \_ = **try** **await** modelData.loadData(lat: lat, lon: lon)

userLocation = location

}

isSearchOpen.toggle()

}

}

}

)

.padding(10)

.shadow(color: .blue, radius: 10)

.cornerRadius(10)

.fixedSize()

.font(.custom("Ariel", size: 26))

.textFieldStyle(RoundedBorderTextFieldStyle())

.cornerRadius(15)

Button(action: {

CLGeocoder().geocodeAddressString(location) { (placemarks, error) **in**

**if** **let** lat = placemarks?.first?.location?.coordinate.latitude,

**let** lon = placemarks?.first?.location?.coordinate.longitude {

Task{

**let** \_ = **try** **await** modelData.loadData(lat: lat, lon: lon)

userLocation = location

}

isSearchOpen.toggle()

}

}

}) {

Image(systemName: "magnifyingglass")

.foregroundColor(.white)

.font(.system(size: 26))

.padding()

.background(

RoundedRectangle(cornerRadius: 20)

.fill(modelData.forecast!.current.temp < 15 ? Color.blue.opacity(0.7) : Color.orange.opacity(0.7))

.shadow(color: Color.black.opacity(0.3), radius: 10, x: 0, y: 5)

)

}

}

Button(action: {

**self**.isSearchOpen.toggle()

}, label: {

Text("Cancel")

.font(.title2)

})

.padding()

.foregroundColor(.white)

.background(

RoundedRectangle(cornerRadius: 20)

.fill(modelData.forecast!.current.temp < 15 ? Color.blue.opacity(0.7) : Color.orange.opacity(0.7))

.shadow(color: Color.black.opacity(0.3), radius: 10, x: 0, y: 5)

)

}

}

}

}

// Custom blur effect

**struct** Blur: UIViewRepresentable {

**var** style: UIBlurEffect.Style = .systemUltraThinMaterial

**func** makeUIView(context: Context) -> UIVisualEffectView {

**let** view = UIVisualEffectView(effect: UIBlurEffect(style: style))

**return** view

}

**func** updateUIView(\_ uiView: UIVisualEffectView, context: Context) {}

}

//

// HomeView.swift

// CWK2\_23\_GL

//

// Created by GirishALukka on 10/03/2023.

//

**import** SwiftUI

**import** CoreLocation

**struct** Home: View {

@EnvironmentObject **var** modelData: ModelData

@State **var** isSearchOpen: Bool = **false**

@State **var** userLocation: String = ""

**var** body: **some** View {

ZStack {

Image("background2")

.resizable()

.ignoresSafeArea()

.blur(radius: 7)

VStack {

Button {

**self**.isSearchOpen.toggle()

} label: {

HStack (spacing: 8){

Image(systemName: "location.fill")

.font(.system(size: 24))

// .foregroundColor(.white)

Text("Change Location")

.bold()

.font(.system(size: 30))

// .foregroundColor(.white)

}

.padding(.vertical, 12)

.padding(.horizontal, 16)

.foregroundColor(.white)

.background(

RoundedRectangle(cornerRadius: 20)

.fill(modelData.forecast!.current.temp < 15 ? Color.blue.opacity(0.7) : Color.orange.opacity(0.7))

.shadow(color: Color.black.opacity(0.3), radius: 10, x: 0, y: 5)

)

}

.sheet(isPresented: $isSearchOpen) {

SearchView(isSearchOpen: $isSearchOpen, userLocation: $userLocation)

.background(BackgroundClearView())

}.padding()

ScrollView {

VStack{

Text(userLocation)

.font(.title)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

.multilineTextAlignment(.center)

Text(Date(timeIntervalSince1970: TimeInterval(((Int)(modelData.forecast?.current.dt ?? 0))))

.formatted(.dateTime.year().hour().month().day()))

.padding()

.font(.largeTitle)

.foregroundColor(.black)

.shadow(color: .black, radius: 1)

.padding(EdgeInsets(top: 10, leading: 10, bottom: 20, trailing: 10))

Spacer()

Text("Temp: \((Int)(modelData.forecast!.current.temp))ºC")

.padding()

.font(.title2)

.shadow(color: .black, radius: 0.5)

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color(red: 30.0/255, green: 41.0/255, blue: 193.0/255) : Color(red: 186.0/255, green: 75.0/255, blue: 15.0/255))

.background(

RoundedRectangle(cornerRadius: 10)

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color.blue.opacity(0.1) : Color.orange.opacity(0.1))

)

Text("Humitidy: \((Int)(modelData.forecast!.current.humidity))%")

.padding()

.font(.title2)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

Text("Pressure: \((Int)(modelData.forecast!.current.pressure)) hpa")

.padding()

.font(.title2)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

HStack{

AsyncImage(url: URL(string: "https://openweathermap.org/img/wn/\(modelData.forecast!.current.weather[0].icon)@2x.png"))

.frame(width: 70, height: 70)

Text("\(modelData.forecast!.current.weather[0].weatherDescription.rawValue)")

.padding()

.font(.title2)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

}

Spacer()

}

}

}

.onAppear {

Task.init {

**self**.userLocation = **await** getLocFromLatLong(lat: modelData.forecast!.lat, lon: modelData.forecast!.lon)

**self**.modelData.userLocation = userLocation

}

}

}

}

}

**struct** Home\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

Home().environmentObject(ModelData())

}

}

**struct** BackgroundClearView: UIViewRepresentable {

**func** makeUIView(context: Context) -> UIView {

**let** view = UIView()

DispatchQueue.main.async {

view.superview?.superview?.backgroundColor = .clear

}

**return** view

}

**func** updateUIView(\_ uiView: UIView, context: Context) {}

}

//

// Hourly.swift

// Coursework2

//

// Created by G Lukka.

//

**import** SwiftUI

**struct** HourlyView: View {

@EnvironmentObject **var** modelData: ModelData

@State **var** locationString: String = ""

**var** body: **some** View {

ZStack {

Image("background")

.resizable()

.ignoresSafeArea()

.blur(radius: 7)

VStack{

Text("\(locationString)").font(.title)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

.multilineTextAlignment(.center)

.padding(.top)

List {

// color changes to orange if the temperature > 15

ForEach(modelData.forecast!.hourly) { hour **in**

HourCondition(current: hour)

}

.listRowBackground(modelData.forecast!.current.temp < 15 ? Color(.blue).opacity(0.1) : Color(.orange).opacity(0.1))

}

.scrollContentBackground(.hidden)

}

}.onAppear {

Task.init {

**self**.locationString = **await** getLocFromLatLong(lat: modelData.forecast!.lat, lon: modelData.forecast!.lon)

}

}

}

}

**struct** Hourly\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

HourlyView().environmentObject(ModelData())

}

}

//

// HourCondition.swift

// Coursework2

//

// Created by G Lukka.

//

**import** SwiftUI

**struct** HourCondition: View {

**var** current : Current

**var** body: **some** View {

ZStack {

HStack( spacing: 8) {

Text(Date(timeIntervalSince1970: TimeInterval(((Int)(current.dt)))).formatted(.dateTime.hour().weekday()))

.font(.headline)

.frame(alignment: .leading)

Spacer()

AsyncImage(url: URL(string: "https://openweathermap.org/img/wn/\(current.weather[0].icon)@2x.png"))

.frame(width: 70, height: 70)

Text("\(current.weather[0].weatherDescription.rawValue)")

.font(.subheadline)

.frame(alignment: .leading)

Spacer()

// color changes to orange if the temperature > 15

Text("\((Int)(current.temp))°C")

.foregroundColor(current.temp < 15 ? Color(red: 30.0/255, green: 41.0/255, blue: 193.0/255) : Color(red: 186.0/255, green: 75.0/255, blue: 15.0/255))

.background(

RoundedRectangle(cornerRadius: 10)

.foregroundColor(current.temp < 15 ? Color.blue.opacity(0.1) : Color.orange.opacity(0.1))

)

.font(.subheadline)

.frame(alignment: .trailing)

}

.padding()

.foregroundColor(.black)

.frame(maxWidth: .infinity, alignment: .leading)

.background(RoundedRectangle(cornerRadius: 10).foregroundColor(.white).opacity(0.7))

.shadow(color: .white, radius: 0.5)

}

}

}

**struct** HourCondition\_Previews: PreviewProvider {

**static** **var** hourly = ModelData().forecast!.hourly

**static** **var** previews: **some** View {

HourCondition(current: hourly[0])

}

}

//

// Forecast.swift

// Coursework2

//

// Created by G Lukka.

//

**import** SwiftUI

**struct** ForecastView: View {

@EnvironmentObject **var** modelData: ModelData

@State **var** locationString: String = "Location loading..."

**var** body: **some** View {

ZStack {

Image("background2")

.resizable()

.ignoresSafeArea()

.blur(radius: 7)

VStack{

Text("\(locationString)").font(.title)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

.padding(EdgeInsets(top: 20, leading: 10, bottom: 10, trailing: 10))

List{

ForEach(modelData.forecast!.daily) { day **in**

DailyView(day: day)

}

.listRowBackground(modelData.forecast!.current.temp < 15 ? Color(.blue).opacity(0.2) : Color(.orange).opacity(0.2))

}

.scrollContentBackground(.hidden)

}

}

.onAppear{

Task.init {

**self**.locationString = **await** getLocFromLatLong(lat: modelData.forecast!.lat, lon: modelData.forecast!.lon)

}

}

}

}

**struct** Forecast\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

ForecastView().environmentObject(ModelData())

}

}

//

// Conditions.swift

// Coursework2

//

// Created by G Lukka.

//

**import** SwiftUI

**struct** CurrentWeatherView: View {

@EnvironmentObject **var** modelData: ModelData

@State **var** locationString: String = "Location loading..."

**var** body: **some** View {

ZStack {

Image("background2")

.resizable()

.ignoresSafeArea()

.blur(radius: 7)

VStack (){

Text("\(locationString)")

.font(.title)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

.multilineTextAlignment(.center)

.padding()

ScrollView {

VStack (spacing: 30){

// Text color changes according to the temperature

Text("\((Int)(modelData.forecast!.current.temp))ºC")

.padding()

.font(.system(size: 60))

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color(red: 30.0/255, green: 41.0/255, blue: 193.0/255) : Color(red: 186.0/255, green: 75.0/255, blue: 15.0/255))

.background(

RoundedRectangle(cornerRadius: 10)

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color.blue.opacity(0.1) : Color.orange.opacity(0.1))

)

HStack{

AsyncImage(url: URL(string: "https://openweathermap.org/img/wn/\(modelData.forecast!.current.weather[0].icon)@2x.png"))

.frame(width: 70, height: 70)

Text("\(modelData.forecast!.current.weather[0].weatherDescription.rawValue)")

.padding()

.font(.title2)

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

}

// color changes to orange if the temperature > 15

Text("Feels Like: \((Int)(modelData.forecast!.current.feelsLike))ºC").padding()

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color(red: 30.0/255, green: 41.0/255, blue: 193.0/255) : Color(red: 186.0/255, green: 75.0/255, blue: 15.0/255))

.font(Font.custom("Helvetica", size: 24))

.background(

RoundedRectangle(cornerRadius: 10)

.foregroundColor(modelData.forecast!.current.feelsLike < 15 ? Color.blue.opacity(0.1) : Color.orange.opacity(0.1))

)

HStack {

Text("Wind Speed: \(String(format: "%.0f", modelData.forecast!.current.windSpeed)) m/s")

Text("Direction: \(convertDegToCardinal(deg:modelData.forecast!.current.windDeg))")

}

HStack {

Text("Humidity: \(modelData.forecast!.current.humidity) %")

Spacer()

Text("Pressure: \(modelData.forecast!.current.pressure) hPg")

}.padding()

HStack {

Label {

Text("\(Date(timeIntervalSince1970: TimeInterval(((Int)(modelData.forecast?.current.sunrise ?? 0)))).formatted(.dateTime.hour().minute()))")

} icon: {

Image(systemName: "sun.and.horizon.fill").foregroundColor(.yellow)

}

Label {

Text("\(Date(timeIntervalSince1970: TimeInterval(((Int)(modelData.forecast?.current.sunset ?? 0)))).formatted(.dateTime.hour().minute()))")

} icon: {

Image(systemName: "sun.and.horizon.fill").foregroundColor(.yellow)

}

}.padding()

}.padding().font(.title2)

}

}

.foregroundColor(.black)

.shadow(color: .black, radius: 0.5)

}.onAppear {

Task.init {

**self**.locationString = **await** getLocFromLatLong(lat: modelData.forecast!.lat, lon: modelData.forecast!.lon)

}

}

}

}

**struct** Conditions\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

CurrentWeatherView()

.environmentObject(ModelData())

}

}

**import** SwiftUI

**import** SwiftUI

**struct** ContentView: View {

**var** body: **some** View {

NavBar()

}

}

**struct** ContentView\_Previews: PreviewProvider {

**static** **var** previews: **some** View {

ContentView()

}

}