КАФЕДРА

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Отчет о лабораторной работе №9

Работа с сетевыми сервисами в мобильных приложениях

По дисциплине: Программирование мобильных устройств

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № _____ _____ _____

подпись, дата — инициалы, фамилия

Санкт-Петербург 2024

Задание:

Дополнить приложение разработанное в рамках одной из предыдущих лабораторных работ, добавив в него работу с внешним сетевым сервисом (взаимодействие с сервером).
Обязательная часть: Реализовать взаимодействие с внешним сервисом. Если не выполняется рекомендательная часть работы, то реализуется взаимодействие с уже готовым внешним сервисом (например, «Вконтакте», «Instagram», «Twitter»)
Рекомендуется: Реализовать серверную часть самомостоятельно на любом языке высокого уровня. При работе сервер должен открывать REST сервис для работы с внешними клиентами. При такой реализации сервер будет выступать в роли backend’a, а мобильное приложение в роли frontend'a.
Не обязательная часть: Реализовать всю работу с серверной частью асинхронно.

Цель работы:

Получение практических навыков использования сетевых сервисов при разработке приложений для мобильных устройств.

Выполнение задания:

Описание задачи: Необходимо дополнить существующее приложение, включив в него возможность определения текущей страны пользователя на основе его геолокации и отображения информации о стране и её флаге.
Выполненные шаги:

1. **Настройка геолокации:**
    - Создан класс **LocationManager**, использующий **CLLocationManager** для получения текущих координат пользователя.
    - Добавлено обновление пользовательского интерфейса при изменении геопозиции.
2. **Определение страны по геопозиции:**
    - Использование **CLGeocoder** для преобразования координат пользователя в код страны.
    - Обработка ошибок и предоставление отладочной информации.
3. **Запрос к API для получения данных о стране:**
    - Реализован метод **fetchCountryData(for:)**, который выполняет запрос к REST Countries API.
    - Декодирование полученных данных в структуру **CountryInfo**.
4. **Обновление структуры CountryInfo:**
    - Адаптация структуры **CountryInfo** под формат данных, предоставляемый API.

- Обработка и отображение названия страны и изображения флага.

5. **Интеграция в пользовательский интерфейс:**
   - Добавление в **SettingsView** компонентов для отображения информации о стране и флаге.
   - Обновление интерфейса при изменении геопозиции и получении новых данных.
6. **Оптимизация запросов:**
   - Применение **debounce** для снижения количества запросов при частом обновлении положения.

Трудности и решения: В процессе разработки возникли проблемы с соответствием структуры данных приложения формату ответа API, а также с задержками обновления данных. Проблемы были решены путём корректировки структуры **CountryInfo** и оптимизации вызовов функции **fetchCountryData** через **debounce**.

В результате проделанной работы в приложение успешно интегрирован функционал определения страны пользователя на основе его геопозиции и отображения соответствующей информации. Это расширяет функциональность приложения и повышает его интерактивность.

```
Код страны: US
Страна: Name(common: "United States"), Флаг: https://flagcdn.com/w320/us.png
Код страны: US
Страна: Name(common: "United States"), Флаг: https://flagcdn.com/w320/us.png
Код страны: US
Страна: Name(common: "United States"), Флаг: https://flagcdn.com/w320/us.png
Код страны: US
Страна: Name(common: "United States"), Флаг: https://flagcdn.com/w320/us.png
Код страны: US
Страна: Name(common: "United States"), Флаг: https://flagcdn.com/w320/us.png
```

Рисунок 1 – вывод в консоли

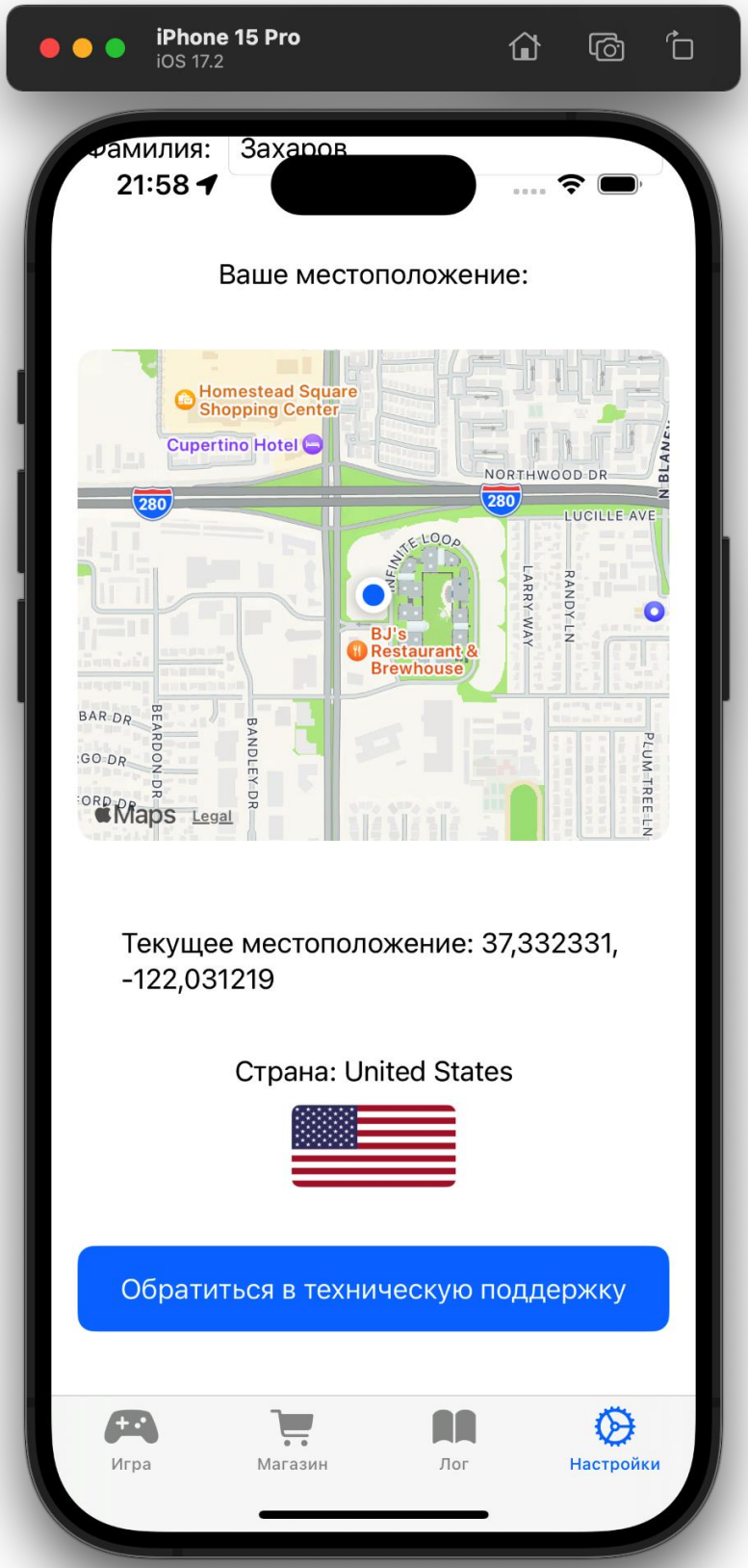

Рисунок 2 – определение страны по местоположению

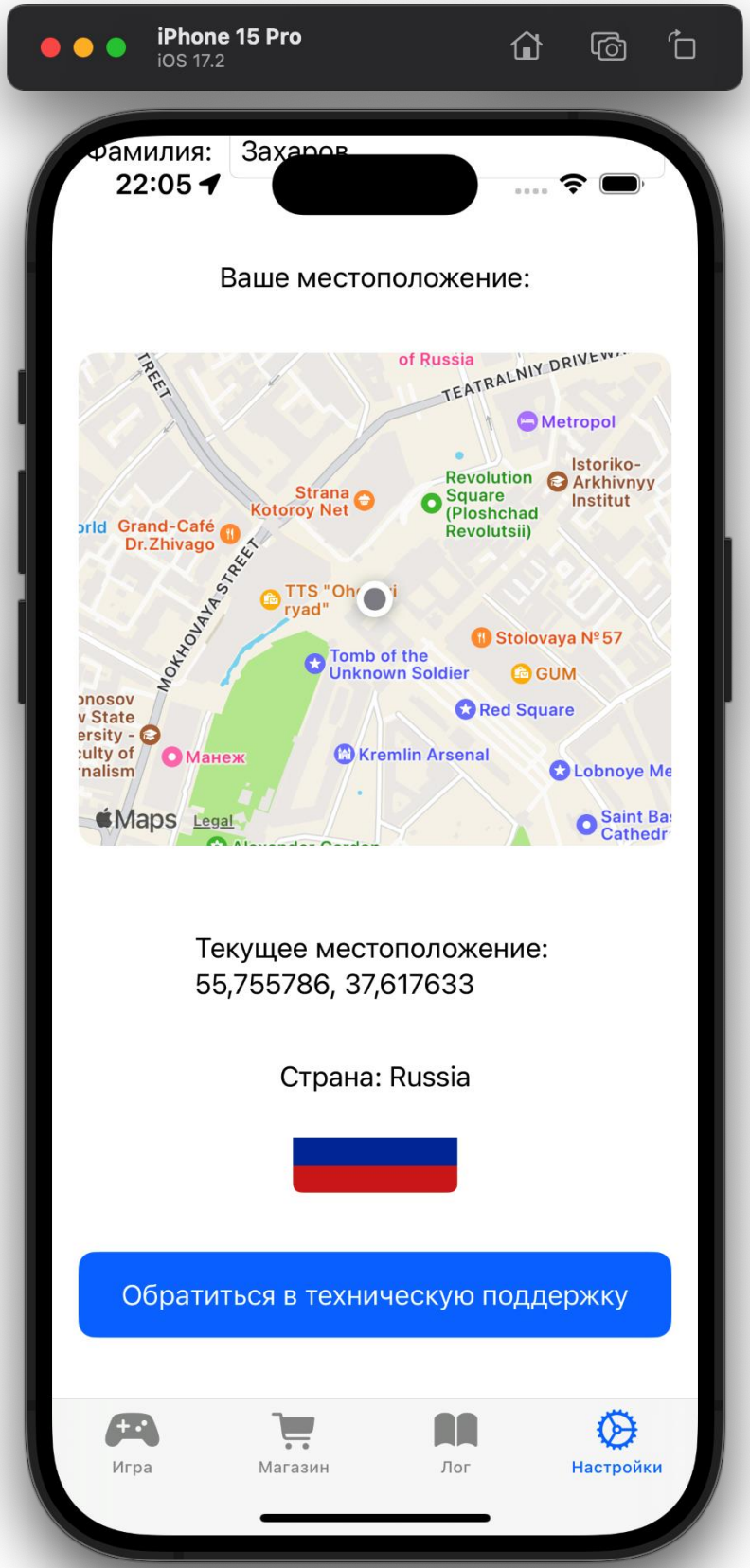

Рисунок 3 - определение страны по местоположению

Листинг:

Gameapp

```
import SwiftUI

@main
struct GameApp: App {
   var body: some Scene {
      WindowGroup {
         ContentView()
      }
   }
}
```

Contentview

```
import SwiftUI
import AVFoundation

struct ContentView: View {
   @State private var userGuess: Double = 5
   @State private var randomNumber = Int.random(in: 1...10)
   @State private var showAlert = false
   @State private var alertTitle = ""
   @State private var isHintEnabled: Bool = false
   @State private var numberRange = "1 - 10"
   let numberRanges = ["1 - 5", "1 - 10", "1 - 20"]
   @State private var balance: Int = 10000
   @State private var betAmount: Int = 100
   @State private var logRecords: [String] = []

   var body: some View {
      TabView {
         // Вкладка игрового экрана
         gameView()
            .tabItem {
               Label("Игра", systemImage: "gamecontroller")
            }
            .tag(1)

         // Вкладка магазина
         ShopView(balance: $balance, logRecords: $logRecords)
            .tabItem {
               Label("Магазин", systemImage: "cart")
            }
            .tag(2)

         // Вкладка лога операций
```

```
        LogView(logs: logRecords)
            .tabItem {
                Label("Лог", systemImage: "book")
            }
            .tag(3)

        SettingsView()
            .tabItem {
                Label("Настройки", systemImage: "gear")
            }
            .tag(4)

    }
}

func gameView() -> some View {
    NavigationView {
        VStack {
            Text("□")
                .font(.largeTitle)
                .padding()
            Text("Игра")
                .font(.title)
                .padding()

            HStack {
                VStack {
                    Text("Баланс:")
                        .font(.headline)
                    Text("\(balance)□")
                        .font(.title2)
                }
                Spacer()
                VStack {
                    Text("Ставка:")
                        .font(.headline)
                    TextField("100", value: $betAmount, format: .number)
                        .textFieldStyle(RoundedBorderTextFieldStyle())
                        .keyboardType(.numberPad)
                        .frame(width: 80)
                }
            }
            .padding()
            Spacer(minLength: 10)

            Text("Угадайте число")
                .font(.title)
            Spacer(minLength: 20)

            Picker("Выберите диапазон чисел", selection: $numberRange) {
                ForEach(numberRanges, id: \.self) {
```

```
                Text($0)
            }
        }
        .pickerStyle(SegmentedPickerStyle())
        .padding()
        .onChange(of: numberRange) { _ in
            updateRandomNumber()
        }
        Text("Выбранное число: \(Int(userGuess))")

        Slider(value: $userGuess, in: 1...CGFloat(getUpperRangeLimit()), step: 1)
            .padding()

        Toggle("Включить подсказки", isOn: $isHintEnabled)
            .padding()
        Spacer(minLength: 20)

        Button("Проверить") {
            checkGuess()
        }
        .padding()
        .background(Color.blue)
        .foregroundColor(.white)
        .cornerRadius(10)
        Spacer(minLength: 20)

            .alert(isPresented: $showAlert) {
                Alert(title: Text(alertTitle), dismissButton: .default(Text("OK")))
            }
        Spacer(minLength: 20)
    }
    .padding()
  }
}

func updateRandomNumber() {
  let limit = getUpperRangeLimit()
  randomNumber = Int.random(in: 1...limit)
  userGuess = min(userGuess, Double(limit))
}

func getUpperRangeLimit() -> Int {
  switch numberRange {
  case "1 - 20":
    return 20
  case "1 - 5":
    return 5
  default: // "1 - 10"
    return 10
  }
}
```

```
func checkGuess() {
    if betAmount > balance {
        alertTitle = "Ваша ставка превышает баланс!"
        showAlert = true
        return
    }

    balance -= betAmount
    let guess = Int(userGuess)

    if guess == randomNumber {
        var winMultiplier = 1.0
        switch numberRange {
        case "1 - 5":
            winMultiplier = 2.0
        case "1 - 10":
            winMultiplier = 5.0
        case "1 - 20":
            winMultiplier = 10.0
        default:
            break
        }

        let winAmount = Int(Double(betAmount) * winMultiplier)
        balance += winAmount
        alertTitle = "Правильно! Вы угадали число!"
        logRecords.append("✅Выигрыш: \(winAmount)□ в режиме \(numberRange)")
        flashLight() // Вызов функции мигания фонарика

        updateRandomNumber()

    } else {
        if isHintEnabled {
            if guess < randomNumber {
                alertTitle = "Слишком мало! Попробуйте число побольше."
            } else {
                alertTitle = "Слишком много! Попробуйте число поменьше."
            }
        } else {
            alertTitle = "Не угадали! Попробуйте еще раз."
        }
        logRecords.append("❌Проигрыш: \(betAmount)□ в режиме \(numberRange)")
    }

    showAlert = true
}
func flashLight() {
    guard let device = AVCaptureDevice.default(for: .video), device.hasTorch else {
        NSLog("Torch (Flashlight) is not available on this device.")
        return
```

```
        }
        do {
            try device.lockForConfiguration()
            for _ in 1...3 {
                NSLog("Torch (Flashlight) state changing.")
                device.torchMode = .on
                device.torchMode = .off
                Thread.sleep(forTimeInterval: 0.1)
            }
            device.unlockForConfiguration()
        } catch {
            NSLog("Error occurred while trying to access torch (flashlight):
\(error.localizedDescription)")
        }
    }
}

// Preview section
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Shopview

```
import SwiftUI

struct ShopView: View {
    @Binding var balance: Int
    @Binding var logRecords: [String]
    @State private var showAlert = false
    @State private var alertTitle = ""

    struct Product {
        var emoji: String
        var price: Int
        var count: Int
    }

    @State private var products: [Product] = [
        Product(emoji: "□", price: 50, count: 0),
        Product(emoji: "□", price: 100, count: 0),
        Product(emoji: "□", price: 5000, count: 0),
        Product(emoji: "□", price: 10000, count: 0)
    ]

    var body: some View {
```

```
        VStack {
            Text("□")
                .font(.largeTitle)
                .padding()
            Text("Магазин")
                .font(.title)
                .padding()
            Text("Баланс: \(balance)□")
                .font(.title2)
                .padding()

            // Список товаров
            ForEach($products.indices, id: \.self) { index in
                HStack {
                    Text(products[index].emoji)
                        .font(.largeTitle)
                        .frame(width: 50, alignment: .leading)
                    Text("\(products[index].price)□")
                        .frame(width: 100, alignment: .leading)
                    Button("Купить") {
                        buyProduct(index: index)
                    }
                    .padding(8)
                    .background(Color.blue)
                    .foregroundColor(.white)
                    .cornerRadius(8)
                    Spacer()
                    Text("x\(products[index].count)")
                }
                .padding()
            }

            Spacer()
        }
        .alert(isPresented: $showAlert) {
            Alert(title: Text(alertTitle), dismissButton: .default(Text("OK")))
        }
    }

    func buyProduct(index: Int) {
        if products[index].price <= balance {
            balance -= products[index].price
            products[index].count += 1
            logRecords.append("□ Покупка: \(products[index].emoji) за
\(products[index].price)□")
        } else {
            alertTitle = "Недостаточно средств для покупки!"
            showAlert = true
        }
    }
}
```

```
struct ShopView_Previews: PreviewProvider {
    @State static var tempBalance = 10000
    @State static var tempLogRecords: [String] = []

    static var previews: some View {
        ShopView(balance: $tempBalance, logRecords: $tempLogRecords)
    }
}
```

Logview

```
import SwiftUI

struct LogView: View {
    var logs: [String] // Массив строк с записями лога

    var body: some View {
        VStack {
            Text("□")
                .font(.largeTitle)
                .padding()
            Text("Лог операций")
                .font(.title)
                .padding()

            // Проверяем, есть ли записи в логе
            if logs.isEmpty {
                Text("Записей в логе нет")
                    .padding()
            } else {
                List(logs.reversed(), id: \.self) { log in
                    Text(log)
                }
            }
        }
    }
}

// Предпросмотр
struct LogView_Previews: PreviewProvider {
    static var previews: some View {
        // Создание примера данных для предпросмотра
        let sampleLogs = ["Покупка: □ за 50□", "Выигрыш: 500 в режиме 1 - 10"]

        // Возвращаем LogView с примерными данными
        LogView(logs: sampleLogs)
```

```
    }
}
```

Settingsview

```
import SwiftUI
import CoreLocation
import MapKit
import Combine


// Структура для хранения информации о стране
struct CountryInfo: Codable {
    struct Name: Codable {
        var common: String
    }

    var name: Name
    var alpha2Code: String
    var flags: Flags

    struct Flags: Codable {
        var png: String
    }

    var flagImageUrl: String {
        return flags.png
    }

    enum CodingKeys: String, CodingKey {
        case name
        case alpha2Code = "cca2"
        case flags
    }
}

class CustomPointAnnotation: NSObject, MKAnnotation, Identifiable {
    dynamic var coordinate: CLLocationCoordinate2D
    let id = UUID()  // Добавлен идентификатор

    init(coordinate: CLLocationCoordinate2D) {
        self.coordinate = coordinate
    }
}

// Класс для управления местоположением пользователя
class LocationManager: NSObject, ObservableObject, CLLocationManagerDelegate {
```

```
    private let locationManager = CLLocationManager()
    @Published var location: CLLocation?
    @Published var userAnnotation = CustomPointAnnotation(coordinate:
CLLocationCoordinate2D(latitude: 0, longitude: 0))
    private var cancellables: Set<AnyCancellable> = []
    private var locationSubject = PassthroughSubject<CLLocation, Never>()

    override init() {
        super.init()
        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters //
Увеличение точности
        locationManager.requestWhenInUseAuthorization()
        locationManager.startUpdatingLocation()
    }

    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
[CLLocation]) {
        if let newLocation = locations.first {
            location = newLocation
            userAnnotation.coordinate = newLocation.coordinate
        }
    }

    func locationManager(_ manager: CLLocationManager, didFailWithError error: Error) {
        print("Ошибка при получении местоположения: \(error.localizedDescription)")
    }

    func locationManager(_ manager: CLLocationManager, didChangeAuthorization status:
CLAuthorizationStatus) {
        switch status {
        case .notDetermined:
            manager.requestWhenInUseAuthorization()
        case .authorizedWhenInUse, .authorizedAlways:
            manager.startUpdatingLocation()
        case .restricted, .denied:
            print("Доступ к геолокации ограничен или отклонен")
        default:
            break
        }
    }

    func getCountryName(completion: @escaping (String?) -> Void) {
        guard let location = location else {
            completion(nil)
            return
        }

        let geocoder = CLGeocoder()
        geocoder.reverseGeocodeLocation(location) { placemarks, error in
            if let error = error {
```

```
                print("Ошибка обратного геокодирования: \(error)")
                completion(nil)
            } else {
                completion(placemarks?.first?.isoCountryCode)
            }
        }
    }

}

struct SettingsView: View {
    @StateObject private var locationManager = LocationManager()
    @State private var showingImagePicker = false
    @State private var inputImage: UIImage?
    @State private var avatarImage: Image?
    @State private var countryInfo: CountryInfo?
    @State var firstName = "Андрей"
    @State var lastName = "Захаров"

    var body: some View {
        NavigationView {
            ScrollView {
                VStack(spacing: 20) {
                    Text("□")
                        .font(.largeTitle)
                        .padding(.top, 20)
                    Text("Настройки профиля")
                        .font(.title)
                        .padding()

                    avatarSection
                    Spacer(minLength: 20)

                    Group {
                        HStack {
                            Text("Имя:")
                                .frame(width: 80, alignment: .leading)
                            TextField("Введите имя", text: $firstName)
                                .textFieldStyle(RoundedBorderTextFieldStyle())
                        }

                        HStack {
                            Text("Фамилия:")
                                .frame(width: 80, alignment: .leading)
                            TextField("Введите фамилию", text: $lastName)
                                .textFieldStyle(RoundedBorderTextFieldStyle())
                        }
                    }
                    .frame(maxWidth: .infinity)
                    .padding([.leading, .trailing], 20)
```

```
            Text("Ваше местоположение:")
                .padding(.top, 30)

            UserLocationMapView(locationManager: locationManager)
                .frame(height: 300)
                .cornerRadius(12)
                .padding()

            if let location = locationManager.location {
                Text("Текущее местоположение: \(location.coordinate.latitude),
\(location.coordinate.longitude)")
                    .padding()
            } else {
                Text("Местоположение не определено")
                    .padding()
            }

            // Добавление информации о стране и флага под координатами
            if let countryInfo = countryInfo {
                VStack {
                    Text("Страна: \(countryInfo.name.common)") // Использование
countryInfo.name.common
                    AsyncImage(url: URL(string: countryInfo.flagImageUrl)) { phase in
                        if let image = phase.image {
                            image.resizable()
                        } else if phase.error != nil {
                            Text("Ошибка загрузки изображения")
                        } else {
                            ProgressView()
                        }
                    }
                    .frame(width: 100, height: 50)
                    .clipShape(Rectangle())
                    .cornerRadius(5)
                }
            }


            NavigationLink(destination: AskView()) {
                Text("Обратиться в техническую поддержку")
                    .foregroundColor(.white)
                    .frame(maxWidth: .infinity)
                    .padding()
                    .background(Color.blue)
                    .cornerRadius(10)
                    .contentShape(Rectangle())
            }
            .padding([.top, .horizontal])
            .padding(.bottom, 40)
        }
        .padding(.bottom, 20)
```

```
            .onReceive(locationManager.$location) { newLocation in
                if let newLocation = newLocation {
                    locationManager.getCountryName { countryCode in
                        if let countryCode = countryCode {
                            print("Код страны: \(countryCode)")
                            self.fetchCountryData(for: countryCode)
                        } else {
                            print("Код страны не получен")
                        }
                    }
                }
            }
        }
        .sheet(isPresented: $showingImagePicker, onDismiss: loadImage) {
            ImagePicker(image: self.$inputImage)
        }
    }
}

var avatarSection: some View {
    ZStack {
        if let avatarImage = avatarImage {
            avatarImage
                .resizable()
                .scaledToFit()
                .frame(width: 200, height: 200)
                .clipShape(Circle())
                .overlay(Circle().stroke(Color.gray, lineWidth: 4))
        } else {
            Circle()
                .frame(width: 200, height: 200)
                .overlay(Circle().stroke(Color.gray, lineWidth: 4))
        }
    }
    .onTapGesture {
        self.showingImagePicker = true
    }
}

func loadImage() {
    guard let inputImage = inputImage else { return }
    avatarImage = Image(uiImage: inputImage)
}

func fetchCountryData(for countryCode: String) {
    let urlString = "https://restcountries.com/v3.1/alpha/\(countryCode)"
    guard let url = URL(string: urlString) else {
        print("Некорректный URL")
        return
    }
```

```
        URLSession.shared.dataTask(with: url) { data, response, error in
            if let error = error {
                print("Ошибка запроса данных о стране: \(error.localizedDescription)")
                return
            }
            guard let data = data else {
                print("Данные не получены")
                return
            }
            if let countryInfos = try? JSONDecoder().decode([CountryInfo].self, from: data),
               let countryInfo = countryInfos.first {
                DispatchQueue.main.async {
                    self.countryInfo = countryInfo
                    print("Страна: \(countryInfo.name), Флаг: \(countryInfo.flagImageUrl)")
                }
            } else {
                print("Ошибка декодирования данных")
            }
        }.resume()
    }

}

struct UserLocationMapView: View {
    @ObservedObject var locationManager: LocationManager

    var body: some View {
        Map(coordinateRegion: .constant(
            MKCoordinateRegion(
                center: locationManager.location?.coordinate ?? CLLocationCoordinate2D(latitude:
0, longitude: 0),
                span: MKCoordinateSpan(latitudeDelta: 0.01, longitudeDelta: 0.01)
            )
        ),
            showsUserLocation: true,
            userTrackingMode: .constant(.follow),
            annotationItems: [locationManager.userAnnotation] // Используйте userAnnotation
напрямую
        ) { annotation in
            MapPin(coordinate: annotation.coordinate) // Теперь coordinate это прямой доступ
к CLLocationCoordinate2D
        }
    }
}


struct ImagePicker: UIViewControllerRepresentable {
    @Environment(\.presentationMode) var presentationMode
    @Binding var image: UIImage?

    func makeUIViewController(context: Context) -> UIImagePickerController {
```

```
        let picker = UIImagePickerController()
        if UIImagePickerController.isSourceTypeAvailable(.camera) {
            picker.sourceType = .camera
        } else {
            print("Камера недоступна на данном устройстве")
        }
        picker.delegate = context.coordinator
        return picker
    }

    func updateUIViewController(_ uiViewController: UIImagePickerController, context:
Context) {}

    func makeCoordinator() -> Coordinator {
        Coordinator(self)
    }

    class Coordinator: NSObject, UINavigationControllerDelegate,
UIImagePickerControllerDelegate {
        let parent: ImagePicker

        init(_ parent: ImagePicker) {
            self.parent = parent
        }

        func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
            if let uiImage = info[.originalImage] as? UIImage {
                parent.image = uiImage
            }
            parent.presentationMode.wrappedValue.dismiss()
        }

        func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
            parent.presentationMode.wrappedValue.dismiss()
        }
    }
}

struct SettingsView_Previews: PreviewProvider {
    static var previews: some View {
        NavigationView {
            SettingsView()
        }
    }
}
```

Askview

```
import SwiftUI

struct AskView: View {
    @State private var name: String = ""
    @State private var email: String = ""
    @State private var message: String = ""
    @State private var requests: [SupportRequest] = []
    @State private var selectedRequest: SupportRequest?
    @State private var showingDetail = false
    @State private var isEditing = false
    @State private var showingConfirmationDialog = false
    @State private var requestToDelete: SupportRequest?

    private let databaseManager = DatabaseManager()

    var body: some View {
        NavigationView {
            VStack {
                Form {
                    TextField("Name", text: $name)
                    TextField("Email", text: $email)
                    TextField("Message", text: $message)

                    Button("Send") {
                        let newRequest = SupportRequest(id: 0, name: name, email: email, message:
message)
                        databaseManager.createRequest(newRequest)
                        loadRequests()
                        name = ""
                        email = ""
                        message = ""
                    }
                }

                List(requests) { request in
                    HStack {
                        Text(request.message)
                        Spacer()
                        Button(action: {
                            self.selectedRequest = request
                            self.isEditing = false
                            self.showingDetail = true
                        }) {
                            Text("□")
                        }
                        .buttonStyle(BorderlessButtonStyle()) // Для улучшения взаимодействия
                        Button(action: {
                            self.selectedRequest = request
                            self.isEditing = true
```

```
                    self.showingDetail = true
                }) {
                    Text("□")
                }
                .buttonStyle(BorderlessButtonStyle()) // Для улучшения взаимодействия
                Button(action: {
                    self.requestToDelete = request
                    self.showingConfirmationDialog = true
                }) {
                    Text("□")
                }
                .buttonStyle(BorderlessButtonStyle()) // Для улучшения взаимодействия
            }

        }
        .onAppear(perform: loadRequests)
    }
    .navigationBarTitle("Support Requests")
    .sheet(isPresented: $showingDetail) {
        if isEditing {
            EditRequestView(request: $selectedRequest, databaseManager:
databaseManager, loadRequests: loadRequests)
        } else {
            DetailRequestView(request: selectedRequest)
        }
    }
    .alert(isPresented: $showingConfirmationDialog) {
        Alert(title: Text("Подтверждение удаления"),
              message: Text("Вы уверены, что хотите удалить этот запрос?"),
              primaryButton: .destructive(Text("Удалить")) {
            if let requestToDelete = self.requestToDelete {
                databaseManager.deleteRequest(id: requestToDelete.id)
                loadRequests()
            }
        },
              secondaryButton: .cancel()
        )
    }
  }
}

private func loadRequests() {
  requests = databaseManager.readRequests()
}
}

struct DetailRequestView: View {
  var request: SupportRequest?

  var body: some View {
    VStack {
```

```
            if let request = request {
                Text(request.name)
                Text(request.email)
                Text(request.message)
            }
        }
    }
}

struct EditRequestView: View {
    @Binding var request: SupportRequest?
    var databaseManager: DatabaseManager
    var loadRequests: () -> Void

    @State private var tempName: String = ""
    @State private var tempEmail: String = ""
    @State private var tempMessage: String = ""

    var body: some View {
        VStack {
            if let request = request {
                TextField("Name", text: $tempName)
                TextField("Email", text: $tempEmail)
                TextField("Message", text: $tempMessage)

                Button("Update") {
                    if var updatedRequest = self.request {
                        updatedRequest.name = tempName
                        updatedRequest.email = tempEmail
                        updatedRequest.message = tempMessage
                        databaseManager.updateRequest(updatedRequest)
                        loadRequests()
                    }
                }
            }
        }
        .onAppear {
            // Инициализация временных переменных, когда View появляется
            if let request = request {
                tempName = request.name
                tempEmail = request.email
                tempMessage = request.message
            }
        }
    }
}


struct AskView_Previews: PreviewProvider {
    static var previews: some View {
        AskView()
```

```
    }
}
```

Datebasemanager

```
import Foundation
import SQLite3

struct SupportRequest: Identifiable {
    let id: Int32
    var name: String
    var email: String
    var message: String
}


class DatabaseManager {
    var db: OpaquePointer?

    init() {
        openDatabase()
        createTable()
    }

    func openDatabase() {
        let fileURL = try! FileManager.default.url(for: .documentDirectory,
                                                   in: .userDomainMask,
                                                   appropriateFor: nil,
                                                   create: false)
            .appendingPathComponent("SupportDatabase.sqlite")

        if sqlite3_open(fileURL.path, &db) != SQLITE_OK {
            print("Error opening database")
            return
        }
    }

    func createTable() {
        let createTableString = """
        CREATE TABLE IF NOT EXISTS SupportRequests(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Name TEXT,
        Email TEXT,
        Message TEXT);
```

```
        """

        if sqlite3_exec(db, createTableString, nil, nil, nil) != SQLITE_OK {
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            print("Error creating table: \(errmsg)")
        }
    }

    func createRequest(_ request: SupportRequest) {
        let insertStatementString = "INSERT INTO SupportRequests (Name, Email, Message)
VALUES (?, ?, ?);"
        var insertStatement: OpaquePointer?

        if sqlite3_prepare_v2(db, insertStatementString, -1, &insertStatement, nil) ==
SQLITE_OK {
            sqlite3_bind_text(insertStatement, 1, (request.name as NSString).utf8String, -1, nil)
            sqlite3_bind_text(insertStatement, 2, (request.email as NSString).utf8String, -1, nil)
            sqlite3_bind_text(insertStatement, 3, (request.message as NSString).utf8String, -1,
nil)

            if sqlite3_step(insertStatement) == SQLITE_DONE {
                print("Successfully inserted row.")
            } else {
                print("Could not insert row.")
            }
        } else {
            print("INSERT statement could not be prepared.")
        }
        sqlite3_finalize(insertStatement)
    }

    func readRequests() -> [SupportRequest] {
        let queryStatementString = "SELECT * FROM SupportRequests;"
        var queryStatement: OpaquePointer?
        var requests = [SupportRequest]()

        if sqlite3_prepare_v2(db, queryStatementString, -1, &queryStatement, nil) ==
SQLITE_OK {
            while sqlite3_step(queryStatement) == SQLITE_ROW {
                let id = sqlite3_column_int(queryStatement, 0)
                let name = String(describing: String(cString: sqlite3_column_text(queryStatement,
1)))
                let email = String(describing: String(cString: sqlite3_column_text(queryStatement,
2)))
                let message = String(describing: String(cString:
sqlite3_column_text(queryStatement, 3)))

                requests.append(SupportRequest(id: id, name: name, email: email, message:
message))
            }
        } else {
```

```
            print("SELECT statement could not be prepared")
        }

        sqlite3_finalize(queryStatement)
        return requests
    }

    func updateRequest(_ request: SupportRequest) {
        let updateStatementString = "UPDATE SupportRequests SET Name = ?, Email = ?,
Message = ? WHERE Id = ?;"
        var updateStatement: OpaquePointer?

        if sqlite3_prepare_v2(db, updateStatementString, -1, &updateStatement, nil) ==
SQLITE_OK {
            sqlite3_bind_text(updateStatement, 1, (request.name as NSString).utf8String, -1, nil)
            sqlite3_bind_text(updateStatement, 2, (request.email as NSString).utf8String, -1, nil)
            sqlite3_bind_text(updateStatement, 3, (request.message as NSString).utf8String, -1,
nil)
            sqlite3_bind_int(updateStatement, 4, request.id)

            if sqlite3_step(updateStatement) == SQLITE_DONE {
                print("Successfully updated row.")
            } else {
                print("Could not update row.")
            }
        } else {
            print("UPDATE statement could not be prepared.")
        }

        sqlite3_finalize(updateStatement)
    }

    func deleteRequest(id: Int32) {
        let deleteStatementString = "DELETE FROM SupportRequests WHERE Id = ?;"
        var deleteStatement: OpaquePointer?

        if sqlite3_prepare_v2(db, deleteStatementString, -1, &deleteStatement, nil) ==
SQLITE_OK {
            sqlite3_bind_int(deleteStatement, 1, id)

            if sqlite3_step(deleteStatement) == SQLITE_DONE {
                print("Successfully deleted row.")
            } else {
                print("Could not delete row.")
            }
        } else {
            print("DELETE statement could not be prepared.")
        }

        sqlite3_finalize(deleteStatement)
    }
```

```
    deinit {
        sqlite3_close(db)
    }
}
```