

```
// ===== File 1/70: DelegationApp/App/AppContainer.swift =====
import Foundation
import SwiftUI

@MainActor
final class AppContainer: ObservableObject {
    let taskService: TaskService
    let chatService: ChatService
    let profileService: ProfileService
    let announcementService: AnnouncementService
    let authService: AuthService

    // ВАЖНО: lazy – чтобы инициализация SessionStore произошла уже на MainActor
    lazy var session: SessionStore = SessionStore(auth: authService)

    init(
        taskService: TaskService,
        chatService: ChatService,
        profileService: ProfileService,
        announcementService: AnnouncementService,
        authService: AuthService
    ) {
        self.taskService = taskService
        self.chatService = chatService
        self.profileService = profileService
        self.announcementService = announcementService
        self.authService = authService
    }
}

extension AppContainer {
    @MainActor
    static let live = AppContainer(
        taskService: MockTaskService(),           // пока можно оставить мок
        chatService: MockChatService(),           // пока мок
        profileService: MockProfileService(),      // пока мок
        announcementService: NetworkAnnouncementService(), // ВАЖНО: сеть
        authService: NetworkAuthService()
    )
}
```

```
// ===== File 2/70: DelegationApp/App/AppRouter.swift =====
import SwiftUI
```

```
struct AppRouter: View {
    @EnvironmentObject var container: AppContainer

    var body: some View {
        rootView()
            .environmentObject(container)
            .environmentObject(container.session)
    }
}
```

```
struct RootView: View {
    @EnvironmentObject var container: AppContainer
    @EnvironmentObject var session: SessionStore

    var body: some View {
        Group {
            if !AppConfig.authEnabled {
                MainTabView()
            } else {
                if session.isRestoring {
                    VStack(spacing: 12) {
                        ProgressView()
                    }
                }
            }
        }
    }
}
```

```

                Text("Проверяем сессию...")
                    .font(.system(size: 14))
            }
        } else {
            if session.isAuthenticated {
                MainTabView()
            } else {
                AuthScreen()
            }
        }
    }
}
}
}
}
}

```

```

private struct MainTabView: View {
    @EnvironmentObject var container: AppContainer
    @EnvironmentObject var session: SessionStore
    @State private var tab: AppTab = .map

```

```

    var body: some View {
        ZStack { contentView }
            .safeAreaInset(edge: .bottom) {
                LiquidTabBar(selection: $tab)
                    .padding(.horizontal, 16)
                    .padding(.bottom, 8)
            }
            .tint(Theme.ColorToken.turquoise)
    }
}

```

```

@ViewBuilder

```

```

private var contentView: some View {
    switch tab {
    case .map:
        NavigationStack {
            MapScreen(
                vm: .init(
                    service: container.taskService,
                    announcementService: container.announcementService,
                    searchService: AddressSearchService()
                ),
                mapMode: MapDisplayConfig.defaultMode()
            )
        }
    case .route:
        NavigationStack {
            RouteScreen(vm: .init(service: container.taskService))
        }
    case .ads:
        NavigationStack {
            MyAdsScreen(
                vm: .init(
                    service: container.announcementService,
                    session: session
                )
            )
        }
    case .chats:
        NavigationStack {
            ChatsScreen(vm: .init(service: container.chatService))
        }
    case .profile:
        NavigationStack {
            ProfileScreen(vm: .init(service: container.profileService))
        }
    }
}

```

```

        .toolbar {
            if AppConfig.authEnabled {
                Button("Logout") { session.logout() }
            }
        }
    }
}
}
}
}
}

```

// ===== File 3/70: DelegationApp/App/DelegationApp.swift =====

```

import SwiftUI
import YandexMapsMobile

@main
struct DelegationApp: App {
    @StateObject private var container = AppContainer.live

    init() {
        YMKMapKit.setApiKey("df3f9145-2080-42b7-9b91-b879c34236bb")
        YMKMapKit.sharedInstance()
    }

    var body: some Scene {
        WindowGroup {
            rootView()
                .environmentObject(container)
                .environmentObject(container.session) // <- БОТ ЭТО ВАЖНО
        }
    }
}

```

// ===== File 4/70: DelegationApp/App/YandexMapView.swift =====

```

/// Обёртка над YMKMapView для использования в SwiftUI.
///
/// Поддерживает:
/// - интерактивные маркеры объявлений;
/// - выделение выбранного маркера;
/// - отрисовку маршрута;
/// - подстройку камеры под маршрут.

import SwiftUI
import YandexMapsMobile
import UIKit

struct YandexMapView: UIViewRepresentable {
    @Binding var centerPoint: YMKPoint?

    let pins: [MapAdPin]
    let selectedPinID: String?
    let routePolyline: YMKPolyline?
    let shouldFitRoute: Bool
    let onRouteFitted: () -> Void
    let onPinTap: (String) -> Void

    final class Coordinator: NSObject, YMKMapObjectTapListener {
        var mapView: YMKMapView?
        var pinPlacemarks: [String: YMKPlacemarkMapObject] = [:]
        var routeObject: YMKPolylineMapObject?
        var lastCenterPoint: YMKPoint?
        var imageCache: [String: UIImage] = [:]
        var onPinTap: ((String) -> Void)?
    }
}

```

```

func onMapObjectTap(with mapObject: YMKMapObject, point: YMKPoint) -> Bool {
    if let id = mapObject.userData as? String {
        onPinTap?(id)
        return true
    }
    if let id = mapObject.userData as? NSString {
        onPinTap?(id as String)
        return true
    }
    return false
}

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

func makeUIView(context: Context) -> UIView {
    let container = UIView()
    container.backgroundColor = .clear

    YandexMapConfigurator.configureIfNeeded()

    guard let mapView = makeNativeMapView() else {
        return container
    }
    mapView.translatesAutoresizingMaskIntoConstraints = false

    container.addSubview(mapView)
    NSLayoutConstraint.activate([
        mapView.topAnchor.constraint(equalTo: container.topAnchor),
        mapView.bottomAnchor.constraint(equalTo: container.bottomAnchor),
        mapView.leadingAnchor.constraint(equalTo: container.leadingAnchor),
        mapView.trailingAnchor.constraint(equalTo: container.trailingAnchor),
    ])

    context.coordinator.mapView = mapView
    context.coordinator.onPinTap = onPinTap

    return container
}

func updateUIView(_ uiView: UIView, context: Context) {
    guard let mapView = context.coordinator.mapView else { return }
    context.coordinator.onPinTap = onPinTap

    let map = mapView.mapWindow.map
    let mapObjects = map.mapObjects

    syncPins(
        pins: pins,
        selectedPinID: selectedPinID,
        mapObjects: mapObjects,
        coordinator: context.coordinator
    )
    syncRoute(routePolyline, mapObjects: mapObjects, coordinator: context.coordinator)

    if shouldFitRoute, let routePolyline {
        fitCameraToRoute(routePolyline, mapView: mapView)
        onRouteFitted()
        return
    }

    if let center = centerPoint {
        moveCameraIfNeeded(to: center, map: map, coordinator: context.coordinator)
    }
}

private func makeNativeMapView() -> YMKMapView? {
    #if targetEnvironment(simulator)

```

```

// For Apple Silicon simulators MapKit recommends Vulkan-backed view.
return YMKMapView(frame: .zero, vulkanPreferred: true)
#else
return YMKMapView(frame: .zero)
#endif
}

private func syncPins(
    pins: [MapAdPin],
    selectedPinID: String?,
    mapObjects: YMKMapObjectCollection,
    coordinator: Coordinator
) {
    let currentIDs = Set(pins.map(\.id))

    for (id, placemark) in coordinator.pinPlacemarks where !currentIDs.contains(id) {
        placemark.removeTapListener(with: coordinator)
        mapObjects.remove(with: placemark)
        coordinator.pinPlacemarks.removeValue(forKey: id)
    }

    for pin in pins {
        let placemark: YMKPlacemarkMapObject
        if let existing = coordinator.pinPlacemarks[pin.id] {
            placemark = existing
            placemark.geometry = pin.point
        } else {
            placemark = mapObjects.addPlacemark()
            placemark.geometry = pin.point
            placemark.userData = pin.id
            placemark.zIndex = 100
            placemark.addTapListener(with: coordinator)
            coordinator.pinPlacemarks[pin.id] = placemark
        }

        let isSelected = (selectedPinID == pin.id)
        applyMarkerStyle(
            for: placemark,
            label: pin.label,
            isSelected: isSelected,
            coordinator: coordinator
        )
    }
}

private func applyMarkerStyle(
    for placemark: YMKPlacemarkMapObject,
    label: String,
    isSelected: Bool,
    coordinator: Coordinator
) {
    let image = markerImage(label: label, isSelected: isSelected, cache: &coordinator.imageCache)
    let style = YMKIconStyle()
    style.anchor = NSValue(cgPoint: CGPoint(x: 0.5, y: 1.0))
    style.scale = 1.0
    style.zIndex = NSNumber(value: isSelected ? 10.0 : 0.0)
    placemark.setIconWith(image, style: style)
    placemark.zIndex = isSelected ? 200 : 100
}

private func markerImage(label: String, isSelected: Bool, cache: inout [String: UIImage]) -> UIImage {
    let key = "\(label)|\(isSelected ? "1" : "0")"
    if let cached = cache[key] { return cached }

    let text = label.isEmpty ? "Объявление" : label
    let font = UIFont.systemFont(ofSize: 12, weight: .semibold)
    let textSize = (text as NSString).size(withAttributes: [.font: font])

```

```

let paddingX: CGFloat = 12
let height: CGFloat = 34
let width = max(64, textSize.width + paddingX * 2)
let size = CGSize(width: width, height: height)
let radius = height / 2

let renderer = UIGraphicsImageRenderer(size: size)
let image = renderer.image { _ in
    let rect = CGRect(origin: .zero, size: size)
    let fillColor = isSelected ? UIColor.systemBlue : UIColor.white
    let strokeColor = isSelected ? UIColor.systemBlue : UIColor.systemTeal.withAlphaComponent(0.4)
    let textColor = isSelected ? UIColor.white : UIColor.label

    let path = UIBezierPath(roundedRect: rect, cornerRadius: radius)
    fillColor.setFill()
    path.fill()

    strokeColor.setStroke()
    path.lineWidth = 1
    path.stroke()

    let textRect = CGRect(
        x: (width - textSize.width) / 2,
        y: (height - textSize.height) / 2,
        width: textSize.width,
        height: textSize.height
    )
    (text as NSString).draw(
        in: textRect,
        withAttributes: [
            .font: font,
            .foregroundColor: textColor,
        ]
    )
}

cache[key] = image
return image
}

private func syncRoute(
    _ polyline: YMKPolyline?,
    mapObjects: YMKMapObjectCollection,
    coordinator: Coordinator
) {
    guard let polyline else {
        if let routeObject = coordinator.routeObject {
            mapObjects.remove(with: routeObject)
            coordinator.routeObject = nil
        }
        return
    }

    if let routeObject = coordinator.routeObject {
        routeObject.geometry = polyline
    } else {
        let routeObject = mapObjects.addPolyline(with: polyline)
        routeObject.strokeWidth = 5
        routeObject.outlineWidth = 2
        routeObject.outlineColor = UIColor.systemBlue.withAlphaComponent(0.2)
        routeObject.setStrokeColorWith(.systemBlue)
        routeObject.zIndex = 50
        coordinator.routeObject = routeObject
    }
}

private func moveCameraIfNeeded(

```

```

        to point: YMKPoint,
        map: YMKMap,
        coordinator: Coordinator
    ) {
        let shouldMove: Bool
        if let last = coordinator.lastCenterPoint {
            let dLat = abs(last.latitude - point.latitude)
            let dLon = abs(last.longitude - point.longitude)
            shouldMove = dLat > 0.000001 || dLon > 0.000001
        } else {
            shouldMove = true
        }

        guard shouldMove else { return }

        coordinator.lastCenterPoint = point
        let position = YMKCameraPosition(target: point, zoom: 14, azimuth: 0, tilt: 0)
        let animation = YMKAnimation(type: .smooth, duration: 0.6)
        map.move(with: position, animation: animation, cameraCallback: nil)
    }

private func fitCameraToRoute(_ polyline: YMKPolyline, mapView: YMKMapView) {
    let map = mapView.mapWindow.map
    let width = Float(max(1, mapView.bounds.width))
    let height = Float(max(1, mapView.bounds.height))

    let insetX: Float = 24
    let insetTop: Float = 80
    let insetBottom: Float = 220

    let topLeft = YMKScreenPoint(x: insetX, y: insetTop)
    let bottomRight = YMKScreenPoint(
        x: max(insetX + 1, width - insetX),
        y: max(insetTop + 1, height - insetBottom)
    )
    let focusRect = YMKScreenRect(topLeft: topLeft, bottomRight: bottomRight)

    let geometry = YMKGeometry(polyline: polyline)
    let camera = map.cameraPosition(
        with: geometry,
        azimuth: 0.0,
        tilt: 0.0,
        focus: focusRect
    )
    map.move(with: camera, animation: YMKAnimation(type: .smooth, duration: 0.65), cameraCallback: nil)
}
}

```

```

// ===== File 5/70: DelegationApp/Core/Components/FilterChip.swift =====
import SwiftUI

```

```

/// Универсальный чип-фильтр.
/// Поддерживает два режима:
/// 1) Binding<Bool> – чип сам переключает состояние.
/// 2) Bool + action – состояние вычисляется снаружи, чип только вызывает action.
struct FilterChip: View {
    let title: String

    // Внутри всегда есть Binding, но во "внешнем" режиме он будет .constant(...)
    @Binding private var isSelected: Bool

    // Нужно ли самому делать toggle()
    private let togglesSelection: Bool

    // Доп. действие при тапе (например, выбрать фильтр)
    private let action: (() -> Void)?

```

```

// MARK: - Init (Binding mode)
init(
    title: String,
    isSelected: Binding<Bool>,
    action: (() -> Void)? = nil
) {
    self.title = title
    self._isSelected = isSelected
    self.togglesSelection = true
    self.action = action
}

// MARK: - Init (Computed Bool mode)
init(
    title: String,
    isSelected: Bool,
    action: @escaping () -> Void
) {
    self.title = title
    self._isSelected = .constant(isSelected)
    self.togglesSelection = false
    self.action = action
}

var body: some View {
    Button {
        if togglesSelection {
            isSelected.toggle()
        }
        action?()
    } label: {
        Text(title)
            .font(.system(size: 14, weight: .semibold))
            .foregroundColor(isSelected ? Color.white : Color.primary)
            .padding(.vertical, 8)
            .padding(.horizontal, 12)
            .background(
                Capsule()
                    .fill(isSelected ? Theme.ColorToken.turquoise : Color.white.opacity(0.7))
            )
            .overlay(
                Capsule()
                    .stroke(Theme.ColorToken.turquoise.opacity(0.3), lineWidth: 1)
            )
    }
    .buttonStyle(.plain)
    .shadow(color: Color.black.opacity(0.05), radius: 4, x: 0, y: 2)
}
}

```

```

// ===== File 6/70: DelegationApp/Core/Components/FloatingPlusButton.swift =====
//
// FloatingPlusButton.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```
import SwiftUI
```

```

struct FloatingPlusButton: View {
    var action: () -> Void
    var body: some View {
        Button(action: action) {
            Image(systemName: "plus")
        }
    }
}

```



```

        .font(.system(size: 24, weight: .bold))
        .foregroundColor(Color.white)
        .frame(width: 64, height: 64)
        .background(Circle().fill(Theme.ColorToken.turquoise))
        .softCardShadow()
    }
    .buttonStyle(.plain)
    .accessibilityLabel("Создать")
}
}
}

```

```

// ===== File 7/70: DelegationApp/Core/Components/LiquidTabBar.swift =====
import SwiftUI

```

```

/// Прозрачный TabBar в стиле iOS 26 / Telegram
/// с «жидким» индикатором, который плавно переезжает между иконками.
struct LiquidTabBar: View {
    @Binding var selection: AppTab
    var badges: [AppTab: Int] = [:]

    @Namespace private var indicatorNamespace

    // Размеры – их теперь легко править
    private let barCornerRadius: CGFloat = 26
    private let barHeight: CGFloat = 74
    private let bubbleSize: CGFloat = 54

    var body: some View {
        HStack(spacing: 10) {
            ForEach(AppTab.allCases) { tab in
                tabButton(for: tab)
            }
        }
        .padding(.horizontal, 14)
        .frame(height: barHeight)
        .background(
            RoundedRectangle(cornerRadius: barCornerRadius, style: .continuous)
                .fill(.ultraThinMaterial) // стекло
                .overlay(
                    RoundedRectangle(cornerRadius: barCornerRadius, style: .continuous)
                        .stroke(Color.white.opacity(0.15), lineWidth: 1)
                )
                .shadow(color: Color.black.opacity(0.10),
                    radius: 22,
                    x: 0,
                    y: 10)
        )
        // Плавный переезд «капли» между иконками
        .animation(
            .spring(response: 0.45,
                dampingFraction: 0.85,
                blendDuration: 0.25),
            value: selection
        )
    }
}

// MARK: - Одна кнопка таба

private func tabButton(for tab: AppTab) -> some View {
    Button {
        if selection != tab {
            selection = tab
        }
    } label: {
        ZStack {
            // «Liquid Glass» пузырьрёк под выбранной иконкой

```

```

        if selection == tab {
            Circle()
                .fill(.ultraThinMaterial)
                .overlay(
                    Circle()
                        .stroke(Color.white.opacity(0.45), lineWidth: 1)
                )
                .shadow(color: Color.black.opacity(0.20),
                    radius: 14,
                    x: 0,
                    y: 8)
                .matchedGeometryEffect(id: "LIQUID_INDICATOR",
                    in: indicatorNamespace)
                .frame(width: bubbleSize, height: bubbleSize)
                .transition(.opacity)
        }

        VStack(spacing: 4) {
            Image(systemName: tab.iconName(selected: selection == tab))
                .font(.system(size: 18, weight: .semibold)) // иконка немного меньше
                .foregroundColor(
                    selection == tab
                    ? Theme.ColorToken.turquoise
                    : Theme.ColorToken.textSecondary
                )
                .scaleEffect(selection == tab ? 1.08 : 1.0)
                .frame(height: 20)

            Text(tab.title)
                .font(.system(size: 11, weight: .semibold)) // текст поменьше
                .foregroundColor(
                    selection == tab
                    ? Theme.ColorToken.turquoise
                    : Theme.ColorToken.textSecondary
                )
                .lineLimit(1) // всегда в одну строку
                .minimumScaleFactor(0.7) // «Объявления» сжимается, но не переносится
        }
        .frame(maxWidth: .infinity)
    }
    .contentShape(Rectangle())
}
.buttonStyle(.plain)
.frame(maxWidth: .infinity)
.overlay(alignment: .topTrailing) {
    // Красный бейдж (например, на профиле «2»)
    if let count = badges[tab], count > 0 {
        Text("\(count)")
            .font(.system(size: 11, weight: .bold))
            .padding(5)
            .background(
                Circle()
                    .fill(Color.red)
            )
            .foregroundColor(.white)
            .offset(x: 8, y: -10)
    }
}
}
}
}

```

```

// ===== File 8/70: DelegationApp/Core/Components/PriceTag.swift =====
//
// PriceTag.swift
// iCuno test
//

```

```
// Created by maftuna murtazaeva on 07.11.2025.
//
```

```
import SwiftUI
```

```
struct PriceTag: View {
    let price: Int
    let eta: Int
    var isHighlighted: Bool = false

    var body: some View {
        VStack(spacing: 4) {
            Text("\(price) ")
                .font(.system(size: 16, weight: .semibold))
            Text("\(eta) мин")
                .font(.system(size: 12, weight: .regular))
                .foregroundColor(Theme.ColorToken.textSecondary)
        }
        .padding(.horizontal, 16)
        .padding(.vertical, 10)
        .background(
            RoundedRectangle(cornerRadius: Theme.Radius.l, style: .continuous)
                .fill(Theme.ColorToken.white)
                .overlay(
                    RoundedRectangle(cornerRadius: Theme.Radius.l)
                        .stroke(isHighlighted ? Theme.ColorToken.turquoise : Color.clear, lineWidth: 2)
                )
        )
        .softCardShadow()
    }
}
```

```
// ===== File 9/70: DelegationApp/Core/Components/StarsView.swift =====
```

```
//
// StarsView.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//
```

```
import SwiftUI
```

```
struct StarsView: View {
    let rating: Double
    let max: Int = 5

    var body: some View {
        HStack(spacing: 4) {
            ForEach(0..
```

```
// ===== File 10/70: DelegationApp/Core/Config/AppConfig.swift =====
```

```
//
// AppConfig.swift
// iCuno test
//
// Created by maftuna murtazaeva on 22.01.2026.
//
```

```

import Foundation

enum AppConfig {

    // =====
    // DEV-ПЕРЕКЛЮЧАТЕЛЬ АВТОРИЗАЦИИ
    // =====
    // Хочешь разрабатывать приложение БЕЗ регистрации/логина?
    // Просто поставь false, и приложение будет сразу пускать внутрь.
    //
    // Когда будешь полноценно тестировать – верни true.
    static let authEnabled: Bool = {
        #if DEBUG
            return false // <-- МЕНЯЕШЬ ТУТ: false = без авторизации
        #else
            return true
        #endif
    }()

    // =====
    // BASE URL ДЛЯ API
    // =====
    // ВАЖНО:
    // - В симуляторе iOS часто можно ходить на 127.0.0.1 (это Mac).
    // - На РЕАЛЬНОМ iPhone 127.0.0.1 = сам iPhone, поэтому бэкэнд "на маке" не доступен.
    //
    // Правильный вариант для iPhone: http://<IP_твоего_Mac_в_WiFi>:8000
    // Например: http://192.168.1.10:8000
    static let apiBaseUrl: URL = {
        // Если захочешь – можешь положить API_BASE_URL в Info.plist,
        // тогда здесь подхватится автоматически (удобно для разных конфигов).
        if let s = Bundle.main.object(forKey: "API_BASE_URL") as? String,
            !s.isEmpty,
            let url = URL(string: s) {
            return url
        }

        let fallback: String
        #if targetEnvironment(simulator)
            fallback = "http://127.0.0.1:8000"
        #else
            // !!! Поменяй на IP твоего Mac (в той же сети Wi-Fi)
            fallback = "http://192.168.1.10:8000"
        #endif

        guard let url = URL(string: fallback) else {
            preconditionFailure("Invalid API base URL fallback: \(fallback)")
        }
        return url
    }()
}

// ===== File 11/70: DelegationApp/Core/Models/AdModels.swift =====
//
// AdModels.swift
// iCuno test
//
// Создано для экрана объявлений.
//

import Foundation

/// Модель объявления. Пока используется только для мок-данных
/// на экране "Мои объявления".
struct AdItem: Identifiable {

```

```

    let id: UUID = .init()
    let title: String
    let priceDescription: String
    let isExpired: Bool
    let views: Int
    let responses: Int
    let favorites: Int
}

// ===== File 12/70: DelegationApp/Core/Models/AnnouncementModels.swift =====
//
// AnnouncementModels.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

import Foundation

// MARK: - Network models

struct AnnouncementDTO: Codable, Identifiable {
    let id: String
    let user_id: String
    let category: String
    let title: String
    let status: String
    let data: [String: JSONValue]
    let created_at: String
}

struct CreateAnnouncementRequest: Codable {
    let category: String
    let title: String
    let status: String
    let data: [String: JSONValue]

    init(category: String, title: String, status: String = "active", data: [String: JSONValue]) {
        self.category = category
        self.title = title
        self.status = status
        self.data = data
    }
}

//-----

struct MediaModerationResponse: Codable {
    let announcement: AnnouncementDTO
    let max_nsfw: Double
    let decision: String
    let can_appeal: Bool
    let message: String
}

struct AppealRequest: Codable {
    let reason: String?
}

extension AnnouncementDTO {
    var moderationCanAppeal: Bool {
        guard
            let mod = data["moderation"]?.objectValue,
            let img = mod["image"]?.objectValue,
            let can = img["can_appeal"]?.boolValue
        else {
            return false
        }
        return can
    }
}

```

```

        else { return false }
        return can
    }
}

```

```

// ===== File 13/70: DelegationApp/Core/Models/AppTab.swift =====
import SwiftUI

```

```

/// Вкладки нижнего TabBar
enum AppTab: Int, CaseIterable, Identifiable {
    case map
    case route
    case ads
    case chats
    case profile

    var id: Int { rawValue }

    /// Текст под иконкой
    var title: String {
        switch self {
            case .map:      return "Карта"
            case .route:    return "Маршрут"
            case .ads:      return "Объявления"
            case .chats:    return "Чаты"
            case .profile:  return "Профиль"
        }
    }

    /// Названия системных иконок (для выбранного/не выбранного состояния)
    func iconName(selected: Bool) -> String {
        switch self {
            case .map:
                return selected ? "map.fill" : "map"

            case .route:
                // Ваша «ветка маршрута»
                return "point.topleft.down.curvedto.point.bottomright.up"

            case .ads:
                return selected
                ? "rectangle.stack.badge.plus.fill"
                : "rectangle.stack.badge.plus"

            case .chats:
                return selected
                ? "bubble.left.and.bubble.right.fill"
                : "bubble.left.and.bubble.right"

            case .profile:
                return selected ? "person.circle.fill" : "person.circle"
        }
    }
}

```

```

// ===== File 14/70: DelegationApp/Core/Models/AuthModels.swift =====
import Foundation

```

```

struct RegisterRequest: Codable {
    let email: String
    let password: String
}

```

```

struct LoginRequest: Codable {

```

```

    let email: String
    let password: String
}

struct TokenResponse: Codable {
    let access_token: String
    let token_type: String? // <-- стало optional (чтобы не падало при декоде)
}

struct MeResponse: Codable {
    let id: String
    let email: String
    let role: String
}

```

```

// ===== File 15/70: DelegationApp/Core/Models/ChatModels.swift =====
import Foundation

```

```

struct ChatPreview: Identifiable {
    let id: UUID = .init()
    let initials: String
    let name: String
    let lastMessage: String
    let time: String
    let unreadCount: Int
}

```

```

// ===== File 16/70: DelegationApp/Core/Models/JSONValue+Numbers.swift =====
//
// JSONValue+Numbers.swift
// iCuno test
//
// Created by maftuna murtazaeva on 25.02.2026.
//

```

```

import Foundation

extension JSONValue {
    var doubleValue: Double? {
        switch self {
            case .double(let d): return d
            case .int(let i): return Double(i)
            case .string(let s): return Double(s)
            default: return nil
        }
    }

    var objectValue: [String: JSONValue]? {
        if case .object(let o) = self { return o }
        return nil
    }
}

```

```

// ===== File 17/70: DelegationApp/Core/Models/JSONValue.swift =====
//
// JSONValue.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//
import Foundation

```

```

/// Универсальный тип для передачи "любого" JSON (строка/число/булев/массив/объект/null).
/// Нужен, чтобы сохранять форму объявления в БД без жёсткой схемы на iOS.
enum JSONValue: Codable, Equatable {
    case string(String)
    case int(Int)
    case double(Double)
    case bool(Bool)
    case object([String: JSONValue])
    case array([JSONValue])
    case null

    init(from decoder: Decoder) throws {
        let c = try decoder.singleValueContainer()

        if c.decodeNil() {
            self = .null
            return
        }
        if let b = try? c.decode(Bool.self) {
            self = .bool(b)
            return
        }
        if let i = try? c.decode(Int.self) {
            self = .int(i)
            return
        }
        if let d = try? c.decode(Double.self) {
            self = .double(d)
            return
        }
        if let s = try? c.decode(String.self) {
            self = .string(s)
            return
        }
        if let o = try? c.decode([String: JSONValue].self) {
            self = .object(o)
            return
        }
        if let a = try? c.decode([JSONValue].self) {
            self = .array(a)
            return
        }

        throw DecodingError.dataCorruptedError(in: c, debugDescription: "Unsupported JSON value")
    }

    func encode(to encoder: Encoder) throws {
        var c = encoder.singleValueContainer()
        switch self {
            case .string(let s): try c.encode(s)
            case .int(let i): try c.encode(i)
            case .double(let d): try c.encode(d)
            case .bool(let b): try c.encode(b)
            case .object(let o): try c.encode(o)
            case .array(let a): try c.encode(a)
            case .null: try c.encodeNil()
        }
    }
}

extension JSONValue {
    var stringValue: String? {
        if case .string(let s) = self { return s }
        return nil
    }
    var intValue: Int? {
        if case .int(let i) = self { return i }
    }
}

```



```

        return nil
    }
    var boolValue: Bool? {
        if case .bool(let b) = self { return b }
        return nil
    }
}

```

```

// ===== File 18/70: DelegationApp/Core/Models/ProfileModels.swift =====
//
// ProfileModels.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```
import Foundation
```

```

struct Profile {
    let name: String
    let phone: String
    let rating: Double
    let completed: Int
    let cancelled: Int
}

```

```

struct Review: Identifiable {
    let id: UUID = .init()
    let authorInitial: String
    let authorName: String
    let text: String
    let ago: String
    let stars: Int
}

```

```

// ===== File 19/70: DelegationApp/Core/Models/TaskModels.swift =====
//
// TaskModels.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```
import Foundation
```

```

struct TaskItem: Identifiable {
    let id: UUID = .init()
    let title: String
    let price: Int //
    let etaMinutes: Int // мин
    let distanceKm: Double
}

```

```

// ===== File 20/70: DelegationApp/Core/Services/AnnouncementService.swift =====
//
// AnnouncementService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```
import Foundation
```

```

//protocol AnnouncementService {
//      func createAnnouncement(token: String, request: CreateAnnouncementRequest) async throws ->
AnnouncementDTO
//      func myAnnouncements(token: String) async throws -> [AnnouncementDTO]
//
//      // Для карты: публичные активные объявления (без токена)
//      func publicAnnouncements() async throws -> [AnnouncementDTO]
//}

protocol AnnouncementService {
    func createAnnouncement(token: String, request: CreateAnnouncementRequest) async throws -> AnnouncementDTO
    func myAnnouncements(token: String) async throws -> [AnnouncementDTO]
    func publicAnnouncements() async throws -> [AnnouncementDTO]

    func uploadAnnouncementMedia(token: String, announcementId: String, images: [Data]) async throws ->
MediaModerationResponse
    func appealAnnouncement(token: String, announcementId: String, reason: String?) async throws ->
AnnouncementDTO
}

final class NetworkAnnouncementService: AnnouncementService {
    private let api: APIClient

    init(api: APIClient = APIClient()) {
        self.api = api
    }

    func createAnnouncement(token: String, request: CreateAnnouncementRequest) async throws -> AnnouncementDTO
{
        try await api.request(.createAnnouncement, body: request, token: token)
    }

    func myAnnouncements(token: String) async throws -> [AnnouncementDTO] {
        try await api.request(.myAnnouncements, token: token)
    }

    func publicAnnouncements() async throws -> [AnnouncementDTO] {
        try await api.request(.publicAnnouncements)
    }

    //-----

    func uploadAnnouncementMedia(token: String, announcementId: String, images: [Data]) async throws ->
MediaModerationResponse {
        let items = images.enumerated().map { idx, d in
            APIClient.UploadFileItem(
                data: d,
                filename: "image_\(idx).jpg",
                mimeType: "image/jpeg"
            )
        }
        return try await api.upload(.uploadAnnouncementMedia(id: announcementId), token: token, files: items)
    }

    func appealAnnouncement(token: String, announcementId: String, reason: String?) async throws ->
AnnouncementDTO {
        let req = AppealRequest(reason: reason)
        return try await api.request(.appealAnnouncement(id: announcementId), body: req, token: token)
    }
}

final class MockAnnouncementService: AnnouncementService {
    func createAnnouncement(token: String, request: CreateAnnouncementRequest) async throws -> AnnouncementDTO
{
        let now = ISO8601DateFormatter().string(from: Date())
        return AnnouncementDTO(
            id: UUID().uuidString,

```

```

        user_id: "dev",
        category: request.category,
        title: request.title,
        status: request.status,
        data: request.data,
        created_at: now
    )
}

func myAnnouncements(token: String) async throws -> [AnnouncementDTO] {
    let now = ISO8601DateFormatter().string(from: Date())
    return [
        AnnouncementDTO(
            id: UUID().uuidString,
            user_id: "dev",
            category: "delivery",
            title: "Доставка по пути: забрать посылку",
            status: "active",
            data: [
                "pickup_address": .string("Москва, Красная площадь"),
                "point": .object(["lat": .double(55.75393), "lon": .double(37.620795)])
            ],
            created_at: now
        )
    ]
}

func publicAnnouncements() async throws -> [AnnouncementDTO] {
    // Для превью/моков можно вернуть то же самое
    try await myAnnouncements(token: "DEV_TOKEN")
}
}

```

```

// ===== File 21/70: DelegationApp/Core/Services/AuthService.swift =====
//
// AuthService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 21.01.2026.
//

```

```
import Foundation
```

```

protocol AuthService {
    func register(email: String, password: String) async throws -> TokenResponse
    func login(email: String, password: String) async throws -> TokenResponse
    func me(token: String) async throws -> MeResponse
}

final class NetworkAuthService: AuthService {
    private let api: APIClient

    init(api: APIClient = APIClient()) {
        self.api = api
    }

    func register(email: String, password: String) async throws -> TokenResponse {
        let req = RegisterRequest(email: email, password: password)
        return try await api.request(.register, body: req)
    }

    func login(email: String, password: String) async throws -> TokenResponse {
        let req = LoginRequest(email: email, password: password)
        return try await api.request(.login, body: req)
    }
}

```

```

    }

    func me(token: String) async throws -> MeResponse {
        return try await api.request(.me, token: token)
    }
}

// ===== File 22/70: DelegationApp/Core/Services/ChatService.swift =====
//
// ChatService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

import Foundation

protocol ChatService {
    func loadChats() -> [ChatPreview]
}

// ===== File 23/70: DelegationApp/Core/Services/Mock/MockAnnouncementService.swift =====
////
//// MockAnnouncementService.swift
//// iCuno test
////
//// Created by maftuna murtazaeva on 18.02.2026.
////
//
//import Foundation
//
//final class MockAnnouncementService: AnnouncementService {
//    func createAnnouncement(token: String, request: CreateAnnouncementRequest) async throws ->
//AnnouncementDTO {
//        let now = ISO8601DateFormatter().string(from: Date())
//        return AnnouncementDTO(
//            id: UUID().uuidString,
//            user_id: "dev",
//            category: request.category,
//            title: request.title,
//            status: request.status,
//            data: request.data,
//            created_at: now
//        )
//    }
//
//    func myAnnouncements(token: String) async throws -> [AnnouncementDTO] {
//        let now = ISO8601DateFormatter().string(from: Date())
//        return [
//            AnnouncementDTO(
//                id: UUID().uuidString,
//                user_id: "dev",
//                category: "delivery",
//                title: "Доставка по пути: забрать посылку",
//                status: "active",
//                data: ["budget": .int(300)],
//                created_at: now
//            ),
//            AnnouncementDTO(
//                id: UUID().uuidString,
//                user_id: "dev",
//                category: "help",
//                title: "Помощь с мелким поручением",
//                status: "draft",

```

```
//          data: [:],
//          created_at: now
//      )
//  ]
//  }
//}
```

```
// ===== File 24/70: DelegationApp/Core/Services/Mock/MockChatService.swift =====
import Foundation
```

```
final class MockChatService: ChatService {
    func loadChats() -> [ChatPreview] {
        [
            .init(initials: "С", name: "Саша", lastMessage: "Ман сасилдок", time: "14:30", unreadCount: 1),
            .init(initials: "П", name: "Павел", lastMessage: "Посылку доставил", time: "Вчера", unreadCount:
0)
        ]
    }
}
```

```
// ===== File 25/70: DelegationApp/Core/Services/Mock/MockProfileService.swift =====
//
// MockProfileService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//
```

```
import Foundation

final class MockProfileService: ProfileService {
    func loadProfile() -> Profile {
        .init(name: "Алексей Иванов",
            phone: "+7 999 123-45-67",
            rating: 4.9,
            completed: 127,
            cancelled: 3)
    }
    func loadReviews() -> [Review] {
        [
            .init(authorInitial: "М", authorName: "Мария К.",
                text: "Отличный исполнитель! Всё сделал быстро и качественно. Рекомендую!",
                ago: "2 дня назад", stars: 5),
            .init(authorInitial: "Д", authorName: "Дмитрий С.",
                text: "Очень доволен! Приехал раньше срока, всё аккуратно.",
                ago: "неделю назад", stars: 5)
        ]
    }
}
```

```
// ===== File 26/70: DelegationApp/Core/Services/Mock/MockTaskService.swift =====
//
// MockTaskService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//
```

```
import Foundation

final class MockTaskService: TaskService {
    func loadNearbyTasks() -> [TaskItem] {
```

```

        [
            .init(title: "Купить молоко", price: 200, etaMinutes: 14, distanceKm: 1.1),
            .init(title: "Забрать посылку", price: 400, etaMinutes: 10, distanceKm: 2.0),
            .init(title: "Доставить цветы", price: 500, etaMinutes: 18, distanceKm: 3.5),
            .init(title: "Помочь донести", price: 250, etaMinutes: 7, distanceKm: 0.6)
        ]
    }
    func loadRouteTasks() -> [TaskItem] {
        [
            .init(title: "Подхватить письмо", price: 350, etaMinutes: 8, distanceKm: 0.9),
            .init(title: "Купить кофе", price: 150, etaMinutes: 12, distanceKm: 0.5)
        ]
    }
}

// ===== File 27/70: DelegationApp/Core/Services/Moderation/TextModerationService.swift =====
//
// TextModerationService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 27.02.2026.
//

import Foundation

/// Быстрая локальная проверка (на устройстве), чтобы мгновенно показать UI-ошибку.
/// Это НЕ заменяет серверную модерацию (на сервере всё равно проверяем).
final class TextModerationService {
    static let shared = TextModerationService()

    enum Verdict {
        case allow
        case reject(reason: String)
        case reviewable(reason: String) // можно оспорить/унести в черновик
    }

    // Минимальные "триггеры". Дальше ты можешь расширять списки под свои правила.
    private let hardBlockTokens: [String] = [
        "наркот", "оруж", "взрыв", "поддел", "порно"
    ]

    private let reviewTokens: [String] = [
        "продам", "куплю", "срочно", "быстро", "без документов"
    ]

    private init() {}

    func check(text: String) -> Verdict {
        let t = normalize(text)
        if t.isEmpty { return .allow }

        if containsAnyToken(in: t, tokens: hardBlockTokens) {
            return .reject(reason: "Текст похож на запрещённый контент. Измените формулировку.")
        }

        if containsAnyToken(in: t, tokens: reviewTokens) {
            return .reviewable(reason: "Текст выглядит спорным. Объявление будет сохранено как черновик (можно оспорить).")
        }

        return .allow
    }

    // MARK: - Helpers

    private func normalize(_ s: String) -> String {

```

```

        s.lowercased()
        .trimmingCharacters(in: .whitespacesAndNewlines)
        .replacingOccurrences(of: "\\s+", with: " ", options: .regularExpression)
    }

    private func containsAnyToken(in text: String, tokens: [String]) -> Bool {
        for token in tokens {
            if text.contains(token) { return true }
        }
        return false
    }
}

```

```

// ===== File 28/70: DelegationApp/Core/Services/Networking/APIClient.swift =====
import Foundation

```

```

struct APIClient {

    struct APIError: LocalizedError {
        let statusCode: Int
        let message: String
        var errorDescription: String? {
            message.isEmpty ? "HTTP \(statusCode)" : message
        }
    }

    //-----
    struct UploadFileItem {
        let data: Data
        let filename: String
        let mimeType: String
    }

    // FastAPI error: {"detail": "..."} или {"detail":[{"loc":..., "msg":...}]}
    private struct FastAPIError: Decodable {
        let detail: Detail

        enum Detail: Decodable {
            case string(String)
            case validation([ValidationItem])
            case unknown

            init(from decoder: Decoder) throws {
                let container = try decoder.singleValueContainer()
                if let str = try? container.decode(String.self) {
                    self = .string(str)
                    return
                }
                if let arr = try? container.decode([ValidationItem].self) {
                    self = .validation(arr)
                    return
                }
                self = .unknown
            }
        }

        struct ValidationItem: Decodable {
            let loc: [String]?
            let msg: String?
            let type: String?
        }

        var humanMessage: String {
            switch detail {
            case .string(let s):
                return s
            }
        }
    }
}

```

```

        case .validation(let items):
            let msgs = items.compactMap { $0.msg }
            if msgs.isEmpty { return "Некорректные данные" }
            let joined = msgs.joined(separator: "\n")
            if joined.contains("value is not a valid email address") {
                return "Неверный email. Пример: name@mail.com"
            }
            return joined
        case .unknown:
            return "Ошибка запроса"
    }
}

// Используем свою сессию (таймауты!)
private let session: URLSession

init() {
    let config = URLSessionConfiguration.default
    config.timeoutIntervalForRequest = 8 // быстро падаем, а не "вечная загрузка"
    config.timeoutIntervalForResource = 15
    self.session = URLSession(configuration: config)
}

func request<T: Decodable, B: Encodable>(
    _ endpoint: APIEndpoint,
    body: B? = nil,
    token: String? = nil
) async throws -> T {

    var req = URLRequest(url: endpoint.url)
    req.httpMethod = endpoint.method.rawValue
    req.setValue("application/json", forHTTPHeaderField: "Content-Type")

    if let token {
        req.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    }

    if let body {
        req.httpBody = try JSONEncoder().encode(body)
    }

    do {
        let (data, response) = try await session.data(for: req)

        guard let http = response as? HTTPURLResponse else {
            throw APIError(statusCode: -1, message: "Нет HTTP ответа")
        }

        if (200..<300).contains(http.statusCode) {
            return try JSONDecoder().decode(T.self, from: data)
        }

        // пробуем красиво распарсить FastAPI error
        if let apiErr = try? JSONDecoder().decode(FastAPIError.self, from: data) {
            throw APIError(statusCode: http.statusCode, message: apiErr.humanMessage)
        }

        let raw = String(data: data, encoding: .utf8) ?? ""
        throw APIError(statusCode: http.statusCode, message: raw)

    } catch let e as URLError {
        // Человеческая ошибка сети (например, если baseUrl не доступен с iPhone)
        throw APIError(statusCode: -1, message: "Сеть недоступна: \(e.localizedDescription)")
    }
}

// Удобно для GET без body

```



```

func request<T: Decodable>(
    _ endpoint: APIEndpoint,
    token: String? = nil
) async throws -> T {
    try await request(endpoint, body: Optional<Int>.none, token: token)
}

//-----
private static func makeMultipartBody(files: [UploadFileItem], boundary: String, fieldName: String) ->
Data {
    var body = Data()
    let lineBreak = "\r\n"

    for f in files {
        body.append("--\((boundary)\)\(lineBreak)".data(using: .utf8!))
        body.append("Content-Disposition: form-data; name=\"\((fieldName)\";
filename=\"\((f.filename)\)\(lineBreak)".data(using: .utf8!))
        body.append("Content-Type: \((f.mimeType)\)\(lineBreak)\(lineBreak)".data(using: .utf8!))
        body.append(f.data)
        body.append(lineBreak.data(using: .utf8!))
    }

    body.append("--\((boundary)\)\(lineBreak)".data(using: .utf8!))
    return body
}

func upload<T: Decodable>(
    _ endpoint: APIEndpoint,
    token: String,
    files: [UploadFileItem],
    fieldName: String = "files"
) async throws -> T {
    var req = URLRequest(url: endpoint.url)
    req.httpMethod = endpoint.method.rawValue

    let boundary = "Boundary-\(UUID().uuidString)"
    req.setValue("multipart/form-data; boundary=\(boundary)", forHTTPHeaderField: "Content-Type")
    req.setValue("Bearer \((token)", forHTTPHeaderField: "Authorization")

    let body = Self.makeMultipartBody(files: files, boundary: boundary, fieldName: fieldName)

    let (data, response) = try await session.upload(for: req, from: body)
    guard let http = response as? HTTPURLResponse else {
        throw APIError(statusCode: -1, message: "Нет HTTP ответа")
    }

    if (200..<300).contains(http.statusCode) {
        return try JSONDecoder().decode(T.self, from: data)
    }

    if let apiErr = try? JSONDecoder().decode(FastAPIError.self, from: data) {
        throw APIError(statusCode: http.statusCode, message: apiErr.humanMessage)
    }
    let raw = String(data: data, encoding: .utf8) ?? ""
    throw APIError(statusCode: http.statusCode, message: raw)
}
}

```

```

// ===== File 29/70: DelegationApp/Core/Services/Networking/AddressSearchService.swift =====
// AddressSearchService.swift
// iCuno test / DelegationApp

```

```

import Foundation
import YandexMapsMobile

final class AddressSearchService {

    private let searchManager: YMKSearchManager?
    private var searchSession: YMKSearchSession?
    private let isEnabled: Bool

    init() {
        //      // В превью отключаем сервис.
        //      if RuntimeEnvironment.isPreview {
        //          self.searchManager = nil
        //          self.isEnabled = false
        //          return
        //      }

        YandexMapConfigurator.configureIfNeeded()

        let managerType: YMKSearchManagerType = .combined
        let search = YMKSearch.sharedInstance()
        self.searchManager = search?.createSearchManager(with: managerType)
        self.isEnabled = (self.searchManager != nil)
    }

    func searchAddress(
        _ text: String,
        completion: @escaping (YMKPoint?) -> Void
    ) {
        // Yandex Search API must be called on UI thread.
        guard Thread.isMainThread else {
            DispatchQueue.main.async { [weak self] in
                guard let self else {
                    completion(nil)
                    return
                }
                self.searchAddress(text, completion: completion)
            }
            return
        }

        let trimmed = text.trimmingCharacters(in: .whitespacesAndNewlines)
        guard !trimmed.isEmpty else {
            completion(nil)
            return
        }

        // В превью просто ничего не ищем.
        guard isEnabled, let searchManager else {
            completion(nil)
            return
        }

        let bbox = YMKBoundingBox(
            southWest: YMKPoint(latitude: -85.0, longitude: -180.0),
            northEast: YMKPoint(latitude: 85.0, longitude: 180.0)
        )
        let geometry = YMKGeometry(boundingBox: bbox)

        let options = YMKSearchOptions()
        options.geometry = true

        searchSession = searchManager.submit(
            withText: trimmed,
            geometry: geometry,
            searchOptions: options
        ) { [weak self] response, error in
            DispatchQueue.main.async {

```

```

        self?.searchSession = nil

        if let error {
            print("Search error: \(error)")
            completion(nil)
            return
        }

        guard
            let collection = response?.collection,
            let firstItem = collection.children.first,
            let obj = firstItem.obj,
            let point = obj.geometry.first?.point
        else {
            completion(nil)
            return
        }

        completion(point)
    }
}

func searchAddress(_ text: String) async -> YMKPoint? {
    await withCheckedContinuation { continuation in
        searchAddress(text) { point in
            continuation.resume(returning: point)
        }
    }
}
}

// ===== File 30/70: DelegationApp/Core/Services/Networking/Endpoints.swift =====
import Foundation

enum Endpoints {
    static let baseUrl: URL = AppConfig.apiBaseUrl
}

enum HTTPMethod: String {
    case GET, POST
}

enum APIEndpoint {
    case register
    case login
    case me

    // Announcements (Ads)
    case createAnnouncement
    case myAnnouncements
    case publicAnnouncements

    //-----
    case uploadAnnouncementMedia(id: String)
    case appealAnnouncement(id: String)

    var path: String {
        switch self {
            case .register: return "auth/register"
            case .login: return "auth/login"
            case .me: return "me"

            case .createAnnouncement: return "announcements"
            case .myAnnouncements: return "announcements/me"
            case .publicAnnouncements: return "announcements/public"
        }
    }
}

```

```

        //-----
        case .uploadAnnouncementMedia(let id): return "announcements/\(id)/media"
        case .appealAnnouncement(let id): return "announcements/\(id)/appeal"
    }
}

var method: HTTPMethod {
    switch self {
    case .register, .login, .createAnnouncement:
        return .POST
    case .me, .myAnnouncements, .publicAnnouncements:
        return .GET

        //-----
        case .uploadAnnouncementMedia, .appealAnnouncement:
            return .POST
    }
}

var url: URL {
    Endpoints.baseURL.appendingPathComponent(path)
}
}

```

```

// ===== File 31/70: DelegationApp/Core/Services/ProfileService.swift =====
//
// ProfileService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```

import Foundation

protocol ProfileService {
    func loadProfile() -> Profile
    func loadReviews() -> [Review]
}

```

```

// ===== File 32/70: DelegationApp/Core/Services/SessionStore.swift =====
import Foundation
import Security

```

```

@MainActor
final class SessionStore: ObservableObject {

    // MARK: - Published state
    @Published private(set) var token: String?
    @Published private(set) var me: MeResponse?
    @Published var errorText: String?
    @Published private(set) var isRestoring: Bool = true
    @Published private(set) var isBusy: Bool = false

    // MARK: - Private
    private let auth: AuthService
    private let keychainKey = "icuno.jwt.access_token"

    // MARK: - Init
    init(auth: AuthService) {
        self.auth = auth
    }
}

```

```

// =====
// DEV: если авторизация выключена –
// сразу пускаем внутрь и НИЧЕГО не проверяем
// =====
if !AppConfig.authEnabled {
    self.token = "DEV_TOKEN"
    self.me = MeResponse(id: "dev", email: "dev@local", role: "user")
    self.isRestoring = false
    return
}

// Читаем токен при старте
self.token = Keychain.readString(key: keychainKey)

// При запуске – пробуем восстановить (НО без “вечной” блокировки UI)
Task { await self.restoreSession() }

}

// MARK: - Computed
var isAuthorized: Bool { token != nil }

// MARK: - Public actions

/// Восстановление сессии при старте приложения.
/// ВАЖНО: UI не должен “висеть” пока сеть думает.
func restoreSession() async {
    // Сразу снимаем экран “Проверяем сессию...”
    isRestoring = false

    guard token != nil else { return }

    // Проверку токена делаем отдельной задачей (не блокируем rootView)
    Task { [weak self] in
        guard let self else { return }
        await self.validateTokenInBackground()
    }
}

func register(email: String, password: String) async {
    await runAuthFlow(email: email, password: password) {
        let t = try await auth.register(email: email, password: password)
        setTokenAndStore(t.access_token)
        try await loadMe()
    }
}

func login(email: String, password: String) async {
    await runAuthFlow(email: email, password: password) {
        let t = try await auth.login(email: email, password: password)
        setTokenAndStore(t.access_token)
        try await loadMe()
    }
}

func loadMe() async throws {
    guard let token else {
        throw NSError(domain: "SessionStore", code: 0,
            userInfo: [NSLocalizedStringDescriptionKey: "Нет токена"])
    }
    let profile = try await auth.me(token: token)
    self.me = profile
}

func logout() {
    clearSession()
}

```

```

// MARK: - Background validation

private func validateTokenInBackground() async {
    guard token != nil else { return }

    do {
        // Если сервер недоступен (например, baseUrl не тот) – не выкидываем сразу пользователя,
        // а просто показываем ошибку, чтобы не было бесконечной загрузки.
        // try await withTimeout(seconds: 5) {
        //     try await loadMe()
        // }
        try await withTimeout(seconds: 5) {
            try await self.loadMe()
        }

    } catch is TimeoutError {
        self.errorText = "Не удалось проверить сессию (таймаут). Проверь API Base URL."
    } catch {
        // Если токен реально невалиден/просрочен – чистим
        clearSession()
        self.errorText = error.localizedDescription
    }
}

// MARK: - Helpers

private func runAuthFlow(
    email: String,
    password: String,
    action: () async throws -> Void
) async {
    errorText = nil

    let trimmedEmail = email.trimmingCharacters(in: .whitespacesAndNewlines)
    let trimmedPass = password.trimmingCharacters(in: .whitespacesAndNewlines)

    guard !trimmedEmail.isEmpty, !trimmedPass.isEmpty else {
        errorText = "Заполни email и пароль"
        return
    }

    isBusy = true
    defer { isBusy = false }

    do {
        try await action()
    } catch {
        errorText = error.localizedDescription
    }
}

private func setTokenAndStore(_ value: String) {
    token = value
    Keychain.saveString(value, key: keychainKey)
}

private func clearSession() {
    token = nil
    me = nil
    Keychain.delete(key: keychainKey)
}

// MARK: - Timeout helper

private struct TimeoutError: LocalizedError {
    var errorDescription: String? { "Таймаут" }
}

```

```

private func withTimeout<T>(
    seconds: Double,
    operation: @escaping @Sendable () async throws -> T
) async throws -> T {
    try await withThrowingTaskGroup(of: T.self) { group in
        group.addTask {
            try await operation()
        }
        // group.addTask {
        //     try await Task.sleep(nanoseconds: UInt64(seconds * 1_000_000_000))
        //     throw TimeoutError()
        // }

        group.addTask { [seconds] in
            try await Task.sleep(nanoseconds: UInt64(seconds * 1_000_000_000))
            throw Self.TimeoutError()
        }

        guard let result = try await group.next() else {
            group.cancelAll()
            throw TimeoutError()
        }
        group.cancelAll()
        return result
    }
}

enum Keychain {

    static func saveString(_ value: String, key: String) {
        guard let data = value.data(using: .utf8) else { return }

        // 1) Сначала удаляем старое значение (если было)
        delete(key: key)

        // 2) Создаём запись
        let query: [String: Any] = [
            kSecClass as String: kSecClassGenericPassword,
            kSecAttrAccount as String: key,
            kSecValueData as String: data,
            // можно добавить доступность, чтобы работало предсказуемо

            kSecAttrAccessible as String: kSecAttrAccessibleAfterFirstUnlock
        ]

        SecItemAdd(query as CFDictionary, nil)
    }

    static func readString(key: String) -> String? {
        let query: [String: Any] = [
            kSecClass as String: kSecClassGenericPassword,
            kSecAttrAccount as String: key,
            kSecReturnData as String: true,
            kSecMatchLimit as String: kSecMatchLimitOne
        ]

        var item: AnyObject?
        let status = SecItemCopyMatching(query as CFDictionary, &item)

        guard status == errSecSuccess,
            let data = item as? Data,
            let str = String(data: data, encoding: .utf8)
        else {
            return nil
        }

        return str
    }
}

```

```

    }

    static func delete(key: String) {
        let query: [String: Any] = [
            kSecClass as String: kSecClassGenericPassword,
            kSecAttrAccount as String: key
        ]

        SecItemDelete(query as CFDictionary)
    }
}

// ===== File 33/70: DelegationApp/Core/Services/TaskService.swift =====
//
// TaskService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

import Foundation

protocol TaskService {
    func loadNearbyTasks() -> [TaskItem]
    func loadRouteTasks() -> [TaskItem]
}

// ===== File 34/70: DelegationApp/Core/Theme/Theme.swift =====
//
// Theme.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

import SwiftUI

enum Theme {
    enum ColorToken {
        static let turquoise = Color.hex("#3CC8C4")
        static let white = Color.hex("#FFFFFF")
        static let milk = Color.hex("#F7F3E9")
        static let peach = Color.hex("#FFC9A6")
        static let textPrimary = Color.black.opacity(0.9)
        static let textSecondary = Color.black.opacity(0.6)
        static let shadow = Color.black.opacity(0.08)
    }

    enum Radius {
        static let s: CGFloat = 10
        static let m: CGFloat = 16
        static let l: CGFloat = 24
        static let xl: CGFloat = 28
    }

    enum Spacing {
        static let xs: CGFloat = 6
        static let s: CGFloat = 8
        static let m: CGFloat = 12
        static let l: CGFloat = 16
        static let xl: CGFloat = 20
        static let xxl: CGFloat = 24
    }
}

```



```

enum Shadow {
    static let soft = ShadowStyle(radius: 16, y: 8, opacity: 0.10)
    struct ShadowStyle {
        let radius: CGFloat
        let y: CGFloat
        let opacity: Double
    }
}

extension View {
    /// Мягкая карточная тень под iOS
    func softCardShadow() -> some View {
        shadow(color: Theme.ColorToken.shadow, radius: Theme.Shadow.soft.radius, x: 0, y: Theme.Shadow.soft.y)
    }
}

```

```

// ===== File 35/70: DelegationApp/Core/Utils/Extensions/Color+Hex.swift =====
//
// Color+Hex.swift
// iCuno test
//
// Created by maftuna murtazaeva on 08.11.2025.
//

```

```

import SwiftUI

extension Color {
    static func hex(_ hex: String) -> Color {
        let hex = hex.trimmingCharacters(in: CharacterSet.alphanumerics.inverted)
        var int: UInt64 = 0; Scanner(string: hex).scanHexInt64(&int)
        let a, r, g, b: UInt64
        switch hex.count {
            case 3: (a,r,g,b) = (255, (int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) * 17)
            case 6: (a,r,g,b) = (255, int >> 16, int >> 8 & 0xFF, int & 0xFF)
            case 8: (a,r,g,b) = (int >> 24, int >> 16 & 0xFF, int >> 8 & 0xFF, int & 0xFF)
            default: (a,r,g,b) = (255,0,0,0)
        }
        return Color(.sRGB,
                    red: Double(r)/255, green: Double(g)/255,
                    blue: Double(b)/255, opacity: Double(a)/255)
    }
}

```

```

// ===== File 36/70: DelegationApp/Features/Ads/AdsView/AdsScreen.swift =====
import SwiftUI

```

```

/// Экран "Мои объявления".
struct MyAdsScreen: View {
    @StateObject private var vm: MyAdsViewModel
    @State private var selectedFilter: AdsFilter = .active
    @State private var showNewAdSheet: Bool = false

    init(vm: MyAdsViewModel) {
        _vm = StateObject(wrappedValue: vm)
    }

    var body: some View {
        ZStack(alignment: .bottom) {
            content
            newAdButton
        }
        .navigationTitle("Мои объявления")
        .navigationBarTitleDisplayMode(.inline)
    }
}

```

```

        .background(Theme.ColorToken.milk.ignoresSafeArea())
        .task { await vm.reload() }
        .refreshable { await vm.reload() }
        .sheet(isPresented: $showNewAdSheet) {
            CreateAdFlowHost(
                service: vm.service,
                session: vm.session
            ) { _ in
                Task { await vm.reload() }
            }
        }
        .alert(
            "Ошибка",
            isPresented: Binding(
                get: { vm.errorText != nil },
                set: { _ in vm.errorText = nil }
            )
        ) {
            Button("Ок", role: .cancel) {}
        } message: {
            Text(vm.errorText ?? "")
        }
    }

private var content: some View {
    ScrollView {
        VStack(alignment: .leading, spacing: Theme.Spacing.l) {
            summarySection
            filtersSection
            listSection
        }
        .padding(.horizontal, Theme.Spacing.l)
        .padding(.top, Theme.Spacing.l)
        .padding(.bottom, 100)
    }
}

private var summarySection: some View {
    VStack(alignment: .leading, spacing: 10) {
        Text("Статистика")
            .font(.system(size: 18, weight: .bold))
        HStack(spacing: 12) {
            StatPill(title: "Активные", value: vm.activeCount)
            StatPill(title: "Черновики", value: vm.draftCount)
            StatPill(title: "Архив", value: vm.archivedCount)
        }
    }
    .padding(Theme.Spacing.l)
    .background(
        RoundedRectangle(cornerRadius: Theme.Radius.l, style: .continuous)
            .fill(Color.white.opacity(0.6))
    )
    .softCardShadow()
}

private var filtersSection: some View {
    ScrollView(.horizontal, showsIndicators: false) {
        HStack(spacing: 10) {
            ForEach(AdsFilter.allCases) { f in
                FilterChip(
                    title: "\(f.rawValue) \(count(for: f))",
                    isSelected: selectedFilter == f
                ) {
                    selectedFilter = f
                }
            }
        }
        .padding(.vertical, 2)
    }
}

```

```

    }
}

private fun count(for filter: AdsFilter) -> Int {
    switch filter {
    case .active: return vm.activeCount
    case .drafts: return vm.draftCount
    case .archived: return vm.archivedCount
    }
}

private var listSection: some View {
    Group {
        if vm.isLoading && vm.items.isEmpty {
            VStack(spacing: 12) {
                ProgressView()
                Text("Загружаем объявления...")
                    .foregroundColor(.secondary)
            }
            .frame(maxWidth: .infinity)
            .padding(.top, 30)
        } else {
            let list = vm.filteredItems(for: selectedFilter)

            if list.isEmpty {
                EmptyState()
            } else {
                VStack(spacing: 12) {
                    ForEach(list) { item in
                        AnnouncementCard(item: item)
                    }
                }
            }
        }
    }
}

private var newAdButton: some View {
    Button {
        showNewAdSheet = true
    } label: {
        HStack(spacing: 10) {
            Image(systemName: "plus.circle.fill")
                .font(.system(size: 18, weight: .bold))
            Text("Новое объявление")
                .font(.system(size: 16, weight: .bold))
        }
        .foregroundColor(.white)
        .padding(.vertical, 14)
        .frame(maxWidth: .infinity)
        .background(
            RoundedRectangle(cornerRadius: Theme.Radius.l, style: .continuous)
                .fill(Theme.ColorToken.turquoise)
        )
        .softCardShadow()
    }
    .padding(.horizontal, Theme.Spacing.l)
    .padding(.bottom, 90)
}

// MARK: - UI bits

private struct StatPill: View {
    let title: String
    let value: Int

    var body: some View {

```

```

VStack(alignment: .leading, spacing: 4) {
    Text(title)
        .font(.system(size: 12, weight: .semibold))
        .foregroundColor(.secondary)
    Text("\(value)")
        .font(.system(size: 18, weight: .bold))
}
.padding(.vertical, 10)
.padding(.horizontal, 12)
.background(
    RoundedRectangle(cornerRadius: 14, style: .continuous)
        .fill(Theme.ColorToken.milk.opacity(0.65))
)
}
}

//private struct FilterChip: View {
//    let title: String
//    let isSelected: Bool
//    let tap: () -> Void
//
//    var body: some View {
//        Button(action: tap) {
//            Text(title)
//                .font(.system(size: 14, weight: .semibold))
//                .foregroundColor(isSelected ? .white : .primary)
//                .padding(.vertical, 8)
//                .padding(.horizontal, 12)
//                .background(
//                    RoundedRectangle(cornerRadius: 14, style: .continuous)
//                        .fill(isSelected ? Theme.ColorToken.turquoise : Color.white.opacity(0.55))
//                )
//                .overlay(
//                    RoundedRectangle(cornerRadius: 14, style: .continuous)
//                        .stroke(Theme.ColorToken.turquoise.opacity(0.25), lineWidth: 1)
//                )
//        }
//        .buttonStyle(.plain)
//    }
//}

private struct AnnouncementCard: View {
    let item: AnnouncementDTO

    var body: some View {
        VStack(alignment: .leading, spacing: 10) {
            HStack(alignment: .top) {
                VStack(alignment: .leading, spacing: 6) {
                    Text(item.title)
                        .font(.system(size: 16, weight: .bold))
                        .foregroundColor(.primary)

                    Text(categorySubtitle(item.category))
                        .font(.system(size: 13, weight: .semibold))
                        .foregroundColor(.secondary)
                }

                Spacer()

                StatusBadge(status: item.status)
            }
        }
        .padding(Theme.Spacing.l)
        .background(
            RoundedRectangle(cornerRadius: Theme.Radius.l, style: .continuous)
                .fill(Color.white.opacity(0.6))
        )
    }
}

```

```

        .softCardShadow()
    }

    private func categorySubtitle(_ raw: String) -> String {
        switch raw {
        case "delivery": return "Доставка и поручения"
        case "help": return "Помощь"
        default: return raw
        }
    }
}

private struct StatusBadge: View {
    let status: String

    var body: some View {
        Text(title)
            .font(.system(size: 12, weight: .bold))
            .foregroundColor(.white)
            .padding(.vertical, 6)
            .padding(.horizontal, 10)
            .background(Capsule().fill(color))
    }

    private var title: String {
        switch status {
        case "draft": return "Черновик"
        case "archived": return "Архив"
        default: return "Активно"
        }
    }

    private var color: Color {
        switch status {
        case "draft": return Color.gray.opacity(0.75)
        case "archived": return Color.black.opacity(0.65)
        default: return Theme.ColorToken.turquoise
        }
    }
}

private struct EmptyState: View {
    var body: some View {
        VStack(spacing: 10) {
            Image(systemName: "doc.text.magnifyingglass")
                .font(.system(size: 34, weight: .bold))
                .foregroundColor(.secondary)
            Text("Здесь пока пусто")
                .font(.system(size: 16, weight: .bold))
            Text("Создайте первое объявление – оно появится в списке.")
                .font(.system(size: 14, weight: .medium))
                .foregroundColor(.secondary)
                .multilineTextAlignment(.center)
                .padding(.horizontal, 24)
        }
        .frame(maxWidth: .infinity)
        .padding(.top, 30)
    }
}

// ===== File 37/70: DelegationApp/Features/Ads/AdsView/NewAdCategoryScreen.swift =====
import SwiftUI

struct NewAdCategoryScreen: View {
    @ObservedObject var draft: CreateAdDraft
    let onClose: () -> Void

```

```

let onFinish: (AnnouncementDTO) -> Void

private let accent = Theme.ColorToken.turquoise

var body: some View {
    ScrollView {
        VStack(alignment: .leading, spacing: 16) {
            header

            VStack(spacing: 16) {
                NavigationLink {
                    NewDeliveryAdFormScreen(
                        draft: draft,
                        accent: accent,
                        onFinish: onFinish
                    )
                    .onAppear { draft.category = .delivery }
                } label: {
                    CategoryCard(
                        title: CreateAdDraft.Category.delivery.title,
                        subtitle: CreateAdDraft.Category.delivery.subtitle,
                        systemImage: "shippingbox.fill",
                        tint: accent
                    )
                }
                .buttonStyle(.plain)

                NavigationLink {
                    NewHelpAdFormScreen(
                        draft: draft,
                        accent: accent,
                        onFinish: onFinish
                    )
                    .onAppear { draft.category = .help }
                } label: {
                    CategoryCard(
                        title: CreateAdDraft.Category.help.title,
                        subtitle: CreateAdDraft.Category.help.subtitle,
                        systemImage: "hands.sparkles.fill",
                        tint: accent
                    )
                }
                .buttonStyle(.plain)
            }
        }
        .padding(.horizontal, 20)
        .padding(.top, 12)
        .padding(.bottom, 24)
    }
    .navigationBarBackButtonHidden(true)
    .toolbar {
        ToolbarItem(placement: .topBarLeading) {
            Button(action: onClose) {
                Image(systemName: "xmark")
                    .font(.system(size: 16, weight: .bold))
                    .foregroundColor(.primary)
                    .padding(8)
                    .background(Circle().fill(Color.white.opacity(0.65)))
            }
        }
    }
}

private var header: some View {
    VStack(alignment: .leading, spacing: 6) {
        Text("Новое объявление")
            .font(.system(size: 28, weight: .bold))
        Text("Выберите тип объявления")
    }
}

```

```

        .font(.system(size: 16, weight: .medium))
        .foregroundColor(.secondary)
    }
}

// MARK: - Category Card

private struct CategoryCard: View {
    let title: String
    let subtitle: String
    let systemImage: String
    let tint: Color

    var body: some View {
        HStack(alignment: .top, spacing: 14) {
            ZStack {
                RoundedRectangle(cornerRadius: 14, style: .continuous)
                    .fill(tint.opacity(0.18))
                Image(systemName: systemImage)
                    .font(.system(size: 20, weight: .bold))
                    .foregroundColor(tint)
            }
            .frame(width: 54, height: 54)

            VStack(alignment: .leading, spacing: 6) {
                Text(title)
                    .font(.system(size: 18, weight: .bold))
                    .foregroundColor(.primary)
                Text(subtitle)
                    .font(.system(size: 14, weight: .medium))
                    .foregroundColor(.secondary)
            }

            Spacer()

            Image(systemName: "chevron.right")
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(Color.secondary)
        }
        .padding(16)
        .background(
            RoundedRectangle(cornerRadius: 18, style: .continuous)
                .fill(Color.white.opacity(0.6))
        )
        .overlay(
            RoundedRectangle(cornerRadius: 18, style: .continuous)
                .stroke(tint.opacity(0.18), lineWidth: 1)
        )
    }
}

// ===== File 38/70: DelegationApp/Features/Ads/CreateAdFlow/AdAudienceScreen.swift =====
//
// AdAudienceScreen.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

import SwiftUI

struct AdAudienceScreen: View {
    @EnvironmentObject private var vm: CreateAdFlowViewModel

    @ObservedObject var draft: CreateAdDraft

```

```

let accent: Color
let onFinish: (AnnouncementDT0) -> Void

@State private var showError: Bool = false
@State private var errorText: String = ""

//-----

@State private var showModeration: Bool = false
@State private var moderationText: String = ""
@State private var pending: AnnouncementDT0?

var body: some View {
    VStack(spacing: 0) {
        ScrollView {
            VStack(alignment: .leading, spacing: 16) {
                titleBlock
                optionsCard
                statusHint
            }
            .padding(20)
            .padding(.top, 6)
        }

        CreateAdBottomButton(
            title: vm.isSubmitting ? "Публикуем..." : "Опубликовать",
            accent: accent
        ) {
            Task { @MainActor in
                let created = await vm.submit(draft: draft)
                if let created {
                    if let w = vm.warningText, !w.isEmpty || created.status == "draft" {
                        pending = created
                        moderationText = vm.warningText ?? "Объявление сохранено как черновик."
                        showModeration = true
                    } else {
                        onFinish(created)
                    }
                } else if let txt = vm.errorText {
                    errorText = txt
                    showError = true
                } else {
                    errorText = "Не удалось опубликовать"
                    showError = true
                }
            }
        }
        .disabled(vm.isSubmitting)
    }
    .navigationBarBackButtonHidden(true)
    .toolbar { backToolbar }
    .alert("Ошибка", isPresented: $showError) {
        Button("Ок", role: .cancel) {}
    } message: {
        Text(errorText)
    }
    .alert("Модерация", isPresented: $showModeration) {
        if let ann = pending, ann.moderationCanAppeal {
            Button("Оспорить") {
                Task { @MainActor in
                    guard let token = vm.session.token else { return }
                    _ = try? await vm.service.appealAnnouncement(token: token, announcementId: ann.id,
reason: nil)

                    onFinish(ann)
                }
            }
        }
    }
    Button("Ок") {

```



```

        if let ann = pending { onFinish(ann) }
    }
} message: {
    Text(moderationText)
}
}

private var titleBlock: some View {
    VStack(alignment: .leading, spacing: 6) {
        Text("Кому показывать объявление")
            .font(.system(size: 28, weight: .bold))
        Text("Выберите аудиторию (можно поменять позже).")
            .font(.system(size: 15, weight: .medium))
            .foregroundColor(.secondary)
    }
}

private var optionsCard: some View {
    CreateAdSectionCard(
        title: "Аудитория",
        subtitle: nil,
        accent: accent
    ) {
        VStack(alignment: .leading, spacing: 10) {
            ForEach(CreateAdDraft.Audience.allCases) { a in
                AudienceRow(title: a.title, isSelected: draft.audience == a) {
                    draft.audience = a
                }
            }
        }
    }
}

private var statusHint: some View {
    CreateAdSectionCard(
        title: "Статус",
        subtitle: "Сейчас публикуем как активное объявление. Черновики добавим позже.",
        accent: accent
    ) {
        Text("После публикации объявление появится в разделе \"Мои объявления\".")
            .font(.system(size: 14, weight: .medium))
            .foregroundColor(.secondary)
    }
}

private var backToolbar: some ToolbarContent {
    ToolbarItem(placement: .topBarLeading) {
        CreateAdBackButton()
    }
}

private struct AudienceRow: View {
    let title: String
    let isSelected: Bool
    let tap: () -> Void

    var body: some View {
        Button(action: tap) {
            HStack(spacing: 10) {
                Image(systemName: isSelected ? "checkmark.circle.fill" : "circle")
                    .foregroundColor(isSelected ? Theme.ColorToken.turquoise : Color.secondary)
                Text(title)
                    .font(.system(size: 16, weight: .semibold))
                    .foregroundColor(.primary)
                Spacer()
            }
        }
        .padding(.vertical, 6)
    }
}

```

```

    }
    .buttonStyle(.plain)
}
}

```

```
// ===== File 39/70: DelegationApp/Features/Ads/CreateAdFlow/AdContactScreen.swift =====
```

```
//
// AdContactScreen.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//
```

```
import SwiftUI
```

```

struct AdContactScreen: View {
    @ObservedObject var draft: CreateAdDraft
    let accent: Color
    let onFinish: (AnnouncementDTO) -> Void

    @State private var goAudience: Bool = false
    @State private var showValidationAlert: Bool = false
    @State private var validationText: String = ""

    var body: some View {
        VStack(spacing: 0) {
            ScrollView {
                VStack(alignment: .leading, spacing: CreateAdUI.Spacing.l) {
                    Text("Как с вами связаться")
                        .font(.system(size: 28, weight: .bold))
                        .padding(.top, 6)

                    CreateAdSectionCard(
                        title: "Контакты",
                        subtitle: "Можно оставить пустым и заполнить позже в профиле – пока MVP.",
                        accent: accent
                    ) {
                        CreateAdTextField(
                            label: "Имя",
                            placeholder: "Введите имя",
                            text: $draft.contactName
                        )
                        CreateAdTextField(
                            label: "Телефон",
                            placeholder: "+7 ...",
                            text: $draft.contactPhone,
                            keyboard: .phonePad
                        )
                    }

                    CreateAdSectionCard(
                        title: "Предпочтительный способ связи",
                        subtitle: nil,
                        accent: accent
                    ) {
                        Picker("", selection: $draft.contactMethod) {
                            ForEach(CreateAdDraft.ContactMethod.allCases) { m in
                                Text(m.title).tag(m)
                            }
                        }
                        .pickerStyle(.segmented)
                    }

                    CreateAdSectionCard(
                        title: "Каналы (позже)",
                        subtitle: "Например: Telegram/WhatsApp – добавишь позже.",

```

```

        accent: accent
    ) {
        Text("Пока оставляем как точку расширения.")
            .font(.system(size: 14, weight: .medium))
            .foregroundColor(.secondary)
    }
}
.padding(.horizontal, 20)
.padding(.bottom, 24)
}

CreateAdBottomButton(title: "Продолжить", accent: accent) {
    if let error = draft.validateContactStep() {
        validationText = error
        showValidationAlert = true
    } else {
        goAudience = true
    }
}

}
.navigationBarBackButtonHidden(true)
.toolbar { backToolbar }
.alert("Проверьте данные", isPresented: $showValidationAlert) {
    Button("Ок", role: .cancel) {}
} message: {
    Text(validationText)
}
.navigationBarDestination(isPresented: $goAudience) {
    AdAudienceScreen(
        draft: draft,
        accent: accent,
        onFinish: onFinish
    )
}
}

private var backToolbar: some ToolbarContent {
    ToolbarItem(placement: .topBarLeading) {
        CreateAdBackButton()
    }
}
}
}

```

```

// ===== File 40/70: DelegationApp/Features/Ads/CreateAdFlow/AdMediaPickerSection.swift =====
//
// AdMediaPickerSection.swift
// iCuno test
//
// Created by maftuna murtazaeva on 27.02.2026.
//

```

```

import SwiftUI
import PhotosUI

struct AdMediaPickerSection: View {
    @ObservedObject var draft: CreateAdDraft
    @State private var pickerItems: [PhotosPickerItem] = []

    var body: some View {
        VStack(alignment: .leading, spacing: 12) {
            PhotosPicker(
                selection: $pickerItems,
                maxSelectionCount: 3,
                matching: .images
            ) {
                HStack(spacing: 10) {

```

```

        Image(systemName: "photo.on.rectangle.angled")
        Text(draft.mediaImages.isEmpty ? "Добавить фото (до 3)" : "Изменить фото")
    }
    .font(.system(size: 15, weight: .semibold))
}

if !draft.mediaImages.isEmpty {
    ScrollView(.horizontal, showsIndicators: false) {
        HStack(spacing: 10) {
            ForEach(Array(draft.mediaImages.enumerated()), id: \.offset) { idx, img in
                ZStack(alignment: .topTrailing) {
                    Image(uiImage: img)
                        .resizable()
                        .scaledToFill()
                        .frame(width: 86, height: 86)
                        .clipShape(RoundedRectangle(cornerRadius: 14, style: .continuous))

                    Button {
                        draft.removeMedia(at: idx)
                    } label: {
                        Image(systemName: "xmark.circle.fill")
                            .font(.system(size: 18, weight: .bold))
                    }
                        .padding(6)
                }
            }
        }
    }
} else {
    Text("Фото проверяются на сервере. Если спорно – объявление уйдёт в черновики.")
        .font(.system(size: 13, weight: .medium))
        .foregroundColor(.secondary)
}
}

.onChange(of: pickerItems) { _, newItem in
    Task {
        var datas: [Data] = []
        for item in newItem {
            if let d = try? await item.loadTransferable(type: Data.self) {
                datas.append(d)
            }
        }
        await MainActor.run {
            draft.setMediaFromPicker(datas: datas)
        }
    }
}
}
}

```

```

// ===== File 41/70: DelegationApp/Features/Ads/CreateAdFlow/CreateAdBackButton.swift =====
//
// CreateAdBackButton.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```
import SwiftUI
```

```
/// Единая кнопка "Назад" для create-flow, чтобы не дублировать код по экранам.
```

```
struct CreateAdBackButton: View {
    @Environment(\.dismiss) private var dismiss
```

```

    var body: some View {
        Button(action: { dismiss() }) {

```

```

        Image(systemName: "chevron.left")
            .font(.system(size: 16, weight: .bold))
            .foregroundColor(.primary)
            .padding(8)
            .background(Circle().fill(Color.white.opacity(0.65)))
    }
}

```

```

// ===== File 42/70: DelegationApp/Features/Ads/CreateAdFlow/CreateAdFlowHost.swift =====
//
// CreateAdFlowHost.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```

import SwiftUI

struct CreateAdFlowHost: View {
    @Environment(\.dismiss) private var dismiss

    @StateObject private var draft = CreateAdDraft()
    @StateObject private var vm: CreateAdFlowViewModel

    let onCreate: (AnnouncementDTO) -> Void

    init(
        service: AnnouncementService,
        session: SessionStore,
        searchService: AddressSearchService = AddressSearchService(),
        onCreate: @escaping (AnnouncementDTO) -> Void
    ) {
        _vm = StateObject(
            wrappedValue: CreateAdFlowViewModel(
                service: service,
                session: session,
                searchService: searchService
            )
        )
        self.onCreate = onCreate
    }

    var body: some View {
        NavigationStack {
            NewAdCategoryScreen(
                draft: draft,
                onClose: { dismiss() },
                onFinish: { created in
                    onCreate(created)
                    dismiss()
                }
            )
        }
        .environmentObject(vm)
        .background(Theme.ColorToken.milk.ignoresSafeArea())
    }
}

```

```

// ===== File 43/70: DelegationApp/Features/Ads/CreateAdFlow/NewDeliveryAdFormScreen.swift =====
//
// NewDeliveryAdFormScreen.swift
// iCuno test
//

```

```
// Created by maftuna murtazaeva on 18.02.2026.
```

```
//
```

```
import SwiftUI
```

```
struct NewDeliveryAdFormScreen: View {
    @ObservedObject var draft: CreateAdDraft
    let accent: Color
    let onFinish: (AnnouncementDTO) -> Void

    @State private var goContact: Bool = false
    @State private var isValidating: Bool = false
    @State private var showValidationAlert: Bool = false
    @State private var validationText: String = ""
    private let searchService = AddressSearchService()

    var body: some View {
        ScrollView {
            VStack(alignment: .leading, spacing: CreateAdUI.Spacing.l) {
                header

                CreateAdSectionCard(
                    title: "Основное",
                    subtitle: "Коротко опишите задачу и бюджет (опционально).",
                    accent: accent
                ) {
                    CreateAdTextField(
                        label: "Название",
                        placeholder: "Например: Забрать посылку и привезти",
                        text: $draft.title
                    )

                    CreateAdValueField(
                        label: "Бюджет (опционально)",
                        placeholder: "0",
                        trailing: "",
                        text: $draft.budget
                    )
                }

                CreateAdSectionCard(
                    title: "Маршрут",
                    subtitle: "Откуда и куда нужно доставить.",
                    accent: accent
                ) {
                    CreateAdTextField(
                        label: "Адрес забора",
                        placeholder: "Введите адрес",
                        text: $draft.pickupAddress
                    )

                    CreateAdTextField(
                        label: "Адрес доставки",
                        placeholder: "Введите адрес",
                        text: $draft.dropoffAddress
                    )
                }

                CreateAdSectionCard(
                    title: "Время",
                    subtitle: "Когда можно выполнить поручение.",
                    accent: accent
                ) {
                    DatePicker("Начало", selection: $draft.startDate, displayedComponents: [.date,
.hourAndMinute])

                    .font(.system(size: 16, weight: .semibold))

                    CreateAdToggleRow(title: "Указать время окончания", isOn: $draft.hasEndTime)
```

```

        if draft.hasEndTime {
            DatePicker("Окончание", selection: $draft.endDate, displayedComponents: [.date,
.hourAndMinute])
                .font(.system(size: 16, weight: .semibold))
        }
    }

    CreateAdSectionCard(
        title: "Габариты (опционально)",
        subtitle: "Если важно – укажите размеры.",
        accent: accent
    ) {
        HStack(spacing: 12) {
            CreateAdTextField(label: "Длина", placeholder: "см", text: $draft.cargoLength,
keyboard: .decimalPad)
            CreateAdTextField(label: "Ширина", placeholder: "см", text: $draft.cargoWidth,
keyboard: .decimalPad)
            CreateAdTextField(label: "Высота", placeholder: "см", text: $draft.cargoHeight,
keyboard: .decimalPad)
        }
    }

    CreateAdSectionCard(
        title: "Подъём (опционально)",
        subtitle: "Если нужно поднять/спустить груз.",
        accent: accent
    ) {
        CreateAdTextField(label: "Этаж", placeholder: "Например: 5", text: $draft.floor, keyboard:
.numberPad)

        CreateAdToggleRow(title: "Есть лифт", isOn: $draft.hasElevator)
        CreateAdToggleRow(title: "Нужен грузчик", isOn: $draft.needLoader)
    }

    CreateAdSectionCard(
        title: "Описание",
        subtitle: "Любые детали, которые помогут исполнителю.",
        accent: accent
    ) {
        CreateAdTextArea(
            label: "Комментарий",
            placeholder: "Например: позвонить за 10 минут до приезда...",
            text: $draft.notes
        )
    }

    // Заглушка под медиа – оставляем "задел", но без логики загрузки
    CreateAdSectionCard(
        title: "Фото",
        subtitle: "Загрузите фото (до 3). Модерация – на сервере.",
        accent: accent
    ) {
        AdMediaPickerSection(draft: draft)
    }
}
.padding(.horizontal, 20)
.padding(.top, 10)
.padding(.bottom, 24)
}
.navigationBarBackButtonHidden(true)
.toolbar { backToolbar }
.navigationDestination(isPresented: $goContact) {
    AdContactScreen(
        draft: draft,
        accent: accent,
        onFinish: onFinish
    )
}
}

```

```

        .alert("Проверьте данные", isPresented: $showValidationAlert) {
            Button("Ок", role: .cancel) {}
        } message: {
            Text(validationText)
        }
        .safeAreaInset(edge: .bottom) {
            CreateAdBottomButton(
                title: isValidating ? "Проверяем адреса..." : "Продолжить",
                accent: accent
            ) {
                guard !isValidating else { return }
                Task { @MainActor in
                    isValidating = true
                    defer { isValidating = false }

                    if let error = await draft.validateAndGeocodeMainStep(searchService: searchService) {
                        validationText = error
                        showValidationAlert = true
                        return
                    }
                    goContact = true
                }
            }
            .disabled(isValidating)
        }
    }

    private var header: some View {
        Text("Доставка и поручения")
            .font(.system(size: 28, weight: .bold))
            .padding(.top, 6)
    }

    private var backToolbar: some ToolbarContent {
        ToolbarItem(placement: .topBarLeading) {
            CreateAdBackButton()
        }
    }
}

```

```

// ===== File 44/70: DelegationApp/Features/Ads/CreateAdFlow/NewHelpAdFormScreen.swift =====
//
// NewHelpAdFormScreen.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```
import SwiftUI
```

```

struct NewHelpAdFormScreen: View {
    @ObservedObject var draft: CreateAdDraft
    let accent: Color
    let onFinish: (AnnouncementDTO) -> Void

    @State private var goContact: Bool = false
    @State private var isValidating: Bool = false
    @State private var showValidationAlert: Bool = false
    @State private var validationText: String = ""
    private let searchService = AddressSearchService()

    var body: some View {
        ScrollView {
            VStack(alignment: .leading, spacing: CreateAdUI.Spacing.l) {
                header
            }

```



```

CreateAdSectionCard(
    title: "Основное",
    subtitle: "Коротко опишите задачу и бюджет (опционально).",
    accent: accent
) {
    CreateAdTextField(
        label: "Название",
        placeholder: "Например: Помочь донести сумки",
        text: $draft.title
    )

    CreateAdValueField(
        label: "Бюджет (опционально)",
        placeholder: "0",
        trailing: "",
        text: $draft.budget
    )
}

CreateAdSectionCard(
    title: "Адрес",
    subtitle: "Где нужна помощь.",
    accent: accent
) {
    CreateAdTextField(
        label: "Адрес",
        placeholder: "Введите адрес",
        text: $draft.helpAddress
    )
}

CreateAdSectionCard(
    title: "Время",
    subtitle: "Когда можно выполнить поручение.",
    accent: accent
) {
    DatePicker("Начало", selection: $draft.startDate, displayedComponents: [.date,
.hourAndMinute])

    .font(.system(size: 16, weight: .semibold))

    CreateAdToggleRow(title: "Указать время окончания", isOn: $draft.hasEndTime)

    if draft.hasEndTime {
        DatePicker("Окончание", selection: $draft.endDate, displayedComponents: [.date,
.hourAndMinute])

        .font(.system(size: 16, weight: .semibold))
    }
}

CreateAdSectionCard(
    title: "Описание",
    subtitle: "Любые детали, которые помогут исполнителю.",
    accent: accent
) {
    CreateAdTextArea(
        label: "Комментарий",
        placeholder: "Например: работа на 30 минут, перчатки не нужны...",
        text: $draft.notes
    )
}

CreateAdSectionCard(
    title: "Фото",
    subtitle: "Загрузите фото (до 3). Модерация – на сервере.",
    accent: accent
) {
    AdMediaPickerSection(draft: draft)
}

```

```

        }
        .padding(.horizontal, 20)
        .padding(.top, 10)
        .padding(.bottom, 24)
    }
    .navigationBarBackButtonHidden(true)
    .toolbar { backToolbar }
    .navigationDestination(isPresented: $goContact) {
        AdContactScreen(
            draft: draft,
            accent: accent,
            onFinish: onFinish
        )
    }
    .alert("Проверьте данные", isPresented: $showValidationAlert) {
        Button("Ок", role: .cancel) {}
    } message: {
        Text(validationText)
    }
    .safeAreaInset(edge: .bottom) {
        CreateAdBottomButton(
            title: isValidating ? "Проверяем адрес..." : "Продолжить",
            accent: accent
        ) {
            guard !isValidating else { return }
            Task { @MainActor in
                isValidating = true
                defer { isValidating = false }

                if let error = await draft.validateAndGeocodeMainStep(searchService: searchService) {
                    validationText = error
                    showValidationAlert = true
                    return
                }
                goContact = true
            }
        }
        .disabled(isValidating)
    }
}

private var header: some View {
    Text("Помощь")
        .font(.system(size: 28, weight: .bold))
        .padding(.top, 6)
}

private var backToolbar: some ToolbarContent {
    ToolbarItem(placement: .topBarLeading) {
        CreateAdBackButton()
    }
}
}
}

```

```

// ===== File 45/70: DelegationApp/Features/Ads/CreateAdUI/CreateAdUI.swift =====
//
// CreateAdUI.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```

import SwiftUI
import UIKit

```

```

enum CreateAdUI {

```

```

enum Spacing {
    static let xs: CGFloat = 8
    static let s: CGFloat = 12
    static let m: CGFloat = 16
    static let l: CGFloat = 20
}

enum Radius {
    static let card: CGFloat = 18
}

enum Palette {
    static let turquoise = Theme.ColorToken.turquoise
    static let beige = Theme.ColorToken.milk
}
}

// MARK: - Section Card

struct CreateAdSectionCard<Content: View>: View {
    let title: String
    let subtitle: String?
    let accent: Color
    @ViewBuilder let content: Content

    var body: some View {
        VStack(alignment: .leading, spacing: CreateAdUI.Spacing.m) {
            VStack(alignment: .leading, spacing: 6) {
                Text(title)
                    .font(.system(size: 18, weight: .bold))

                if let subtitle {
                    Text(subtitle)
                        .font(.system(size: 14, weight: .medium))
                        .foregroundColor(.secondary)
                }
            }

            content
        }
        .padding(CreateAdUI.Spacing.l)
        .background(
            RoundedRectangle(cornerRadius: CreateAdUI.Radius.card, style: .continuous)
                .fill(Color.white.opacity(0.55))
        )
        .overlay(
            RoundedRectangle(cornerRadius: CreateAdUI.Radius.card, style: .continuous)
                .stroke(accent.opacity(0.25), lineWidth: 1)
        )
    }
}

// MARK: - Fields

struct CreateAdTextField: View {
    let label: String
    let placeholder: String
    @Binding var text: String
    var keyboard: UIKeyboardType = .default

    var body: some View {
        VStack(alignment: .leading, spacing: 6) {
            Text(label)
                .font(.system(size: 12, weight: .semibold))
                .foregroundColor(.secondary)

            TextField(placeholder, text: $text)
                .keyboardType(keyboard)
        }
    }
}

```

```

        .textInputAutocapitalization(.sentences)
        .font(.system(size: 16, weight: .semibold))
        .padding(.vertical, 10)

        Divider().opacity(0.35)
    }
}

struct CreateAdValueField: View {
    let label: String
    let placeholder: String
    let trailing: String
    @Binding var text: String

    var body: some View {
        VStack(alignment: .leading, spacing: 6) {
            Text(label)
                .font(.system(size: 12, weight: .semibold))
                .foregroundColor(.secondary)

            HStack(spacing: 8) {
                TextField(placeholder, text: $text)
                    .keyboardType(.decimalPad)
                    .font(.system(size: 16, weight: .semibold))

                Spacer(minLength: 0)

                Text(trailing)
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(.secondary)
            }
            .padding(.vertical, 10)

            Divider().opacity(0.35)
        }
    }
}

struct CreateAdToggleRow: View {
    let title: String
    @Binding var isOn: Bool

    var body: some View {
        HStack {
            Text(title)
                .font(.system(size: 16, weight: .semibold))
            Spacer()
            Toggle("", isOn: $isOn)
                .labelsHidden()
        }
        .padding(.vertical, 6)
    }
}

struct CreateAdTextArea: View {
    let label: String
    let placeholder: String
    @Binding var text: String

    var body: some View {
        VStack(alignment: .leading, spacing: 6) {
            Text(label)
                .font(.system(size: 12, weight: .semibold))
                .foregroundColor(.secondary)

            ZStack(alignment: .topLeading) {
                if text.isEmpty {

```

```

        Text(placeholder)
            .foregroundColor(Color.secondary.opacity(0.8))
            .padding(.top, 10)
            .padding(.leading, 4)
    }

    TextEditor(text: $text)
        .frame(minHeight: 110)
        .scrollContentBackground(.hidden)
        .font(.system(size: 16, weight: .regular))
    }
    .padding(.vertical, 6)

    Divider().opacity(0.35)
}
}
}

// MARK: - Bottom Button

struct CreateAdBottomButton: View {
    let title: String
    let accent: Color
    let action: () -> Void

    var body: some View {
        Button(action: action) {
            Text(title)
                .font(.system(size: 16, weight: .bold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding(.vertical, 16)
                .background(
                    RoundedRectangle(cornerRadius: 16, style: .continuous)
                    .fill(accent)
                )
                .shadow(color: accent.opacity(0.25), radius: 12, x: 0, y: 6)
        }
        .padding(.horizontal, 20)
        .padding(.vertical, 12)
        .background(.ultraThinMaterial)
    }
}

```

```

// ===== File 46/70: DelegationApp/Features/Ads/Model/CreateAdDraft.swift =====
//
// CreateAdDraft.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```

import UIKit
import Foundation
import YandexMapsMobile

```

```

@MainActor
final class CreateAdDraft: ObservableObject {

    enum Category: String, CaseIterable, Identifiable {
        case delivery = "delivery"
        case help = "help"
        var id: String { rawValue }

        var title: String {
            switch self {

```

```

        case .delivery: return "Доставка и поручения"
        case .help: return "Помощь"
    }
}

var subtitle: String {
    switch self {
    case .delivery: return "Посылки, покупки, курьерские услуги"
    case .help: return "Помощь руками, поддержка, бытовые задачи"
    }
}
}

enum ContactMethod: String, CaseIterable, Identifiable {
    case callsAndMessages = "calls_and_messages"
    case messagesOnly = "messages_only"
    case callsOnly = "calls_only"
    var id: String { rawValue }

    var title: String {
        switch self {
        case .callsAndMessages: return "Звонки и сообщения"
        case .messagesOnly: return "Только сообщения"
        case .callsOnly: return "Только звонки"
        }
    }
}

enum Audience: String, CaseIterable, Identifiable {
    case individuals = "individuals"
    case business = "business"
    case both = "both"
    var id: String { rawValue }

    var title: String {
        switch self {
        case .individuals: return "Частные лица"
        case .business: return "Бизнес"
        case .both: return "Частные лица и бизнес"
        }
    }
}

// MARK: - Common
@Published var category: Category?
@Published var title: String = ""
@Published var budget: String = ""

// MARK: - Delivery fields
@Published var pickupAddress: String = ""
@Published var dropoffAddress: String = ""
@Published var pickupPoint: YMKPoint?
@Published var dropoffPoint: YMKPoint?
@Published var startDate: Date = Date().addingTimeInterval(10 * 60)
@Published var hasEndTime: Bool = false
@Published var endDate: Date = Date().addingTimeInterval(40 * 60)

@Published var cargoLength: String = ""
@Published var cargoWidth: String = ""
@Published var cargoHeight: String = ""

@Published var floor: String = ""
@Published var hasElevator: Bool = true
@Published var needLoader: Bool = false

// MARK: - Help fields
@Published var helpAddress: String = ""
@Published var helpPoint: YMKPoint?

```

```

// MARK: - Notes + future hooks
@Published var notes: String = ""
/// Заглушка под будущую загрузку фото/видео (позже: PhotosPicker / upload).
@Published var mediaLocalIdentifiers: [String] = []
/// Заглушка под будущую обработку текста/фото моделями.
@Published var aiHints: [String] = []

// MARK: - Contacts
@Published var contactName: String = ""
@Published var contactPhone: String = ""
@Published var contactMethod: ContactMethod = .callsAndMessages

// MARK: - Audience
@Published var audience: Audience = .both

@Published var mediaImages: [UIImage] = []
@Published var mediaJPEGData: [Data] = []

private var pickupAddressSnapshot: String?
private var dropoffAddressSnapshot: String?
private var helpAddressSnapshot: String?

// MARK: - Validation
func validateMainStepSync() -> String? {
    guard let category else { return "Выберите категорию" }

    if let e = AdValidators.validateTitle(title) { return e }
    if let e = AdValidators.validateBudget(budget) { return e }
    if let e = AdValidators.validateTimeWindow(
        startDate: startDate,
        hasEndTime: hasEndTime,
        endDate: endDate
    ) { return e }

    switch category {
    case .delivery:
        if let e = AdValidators.validateAddress(pickupAddress, fieldName: "Адрес забора") { return e }
        if let e = AdValidators.validateAddress(dropoffAddress, fieldName: "Адрес доставки") { return e }
        if let e = AdValidators.validateDifferentAddresses(pickupAddress, dropoffAddress) { return e }
        if let e = AdValidators.validateDimension(cargoLength, fieldName: "Длина", max: 85) { return e }
        if let e = AdValidators.validateDimension(cargoWidth, fieldName: "Ширина", max: 57) { return e }
        if let e = AdValidators.validateDimension(cargoHeight, fieldName: "Высота", max: 57) { return e }
        if let e = AdValidators.validateFloor(floor) { return e }
    case .help:
        if let e = AdValidators.validateAddress(helpAddress, fieldName: "Адрес") { return e }
    }

    return nil
}

func validateAndGeocodeMainStep(searchService: AddressSearchService) async -> String? {
    if let e = validateMainStepSync() { return e }

    guard let category else { return "Выберите категорию" }

    switch category {
    case .delivery:
        let pickupNormalized = AdValidators.normalizedAddress(pickupAddress)
        let dropoffNormalized = AdValidators.normalizedAddress(dropoffAddress)

        if pickupAddressSnapshot != pickupNormalized {
            pickupPoint = await searchService.searchAddress(pickupNormalized)
            pickupAddressSnapshot = pickupPoint == nil ? nil : pickupNormalized
        }
        guard pickupPoint != nil else {
            return "Адрес забора не найден. Уточните адрес"
        }
    }
}

```

```

        if dropoffAddressSnapshot != dropoffNormalized {
            dropoffPoint = await searchService.searchAddress(dropoffNormalized)
            dropoffAddressSnapshot = dropoffPoint == nil ? nil : dropoffNormalized
        }
        guard dropoffPoint != nil else {
            return "Адрес доставки не найден. Уточните адрес"
        }

    case .help:
        let helpNormalized = AdValidators.normalizedAddress(helpAddress)
        if helpAddressSnapshot != helpNormalized {
            helpPoint = await searchService.searchAddress(helpNormalized)
            helpAddressSnapshot = helpPoint == nil ? nil : helpNormalized
        }
        guard helpPoint != nil else {
            return "Адрес не найден. Уточните адрес"
        }
    }

    return nil
}

func validateContactStep() -> String? {
    let phone = AdValidators.validateOptionalPhone(contactPhone)
    if let e = phone.error { return e }
    contactPhone = phone.normalized ?? ""
    contactName = AdValidators.trimmed(contactName)
    return nil
}

func validateForSubmit(searchService: AddressSearchService) async -> String? {
    if let e = await validateAndGeocodeMainStep(searchService: searchService) { return e }
    if let e = validateContactStep() { return e }
    return nil
}

// MARK: - Mapping to request
func toCreateRequest(statusOverride: String? = nil) -> CreateAnnouncementRequest {
    let iso = ISO8601DateFormatter()
    var data: [String: JSONValue] = [:]
    let titleTrimmed = AdValidators.trimmed(title)

    //      var data: [String: JSONValue] = [:]
    //      let titleTrimmed = AdValidators.trimmed(title)

    data["category"] = .string(category?.rawValue ?? "")
    data["budget"] = jsonOptionalDecimal(budget)

    let contactNameTrimmed = AdValidators.trimmed(contactName)
    data["contact_name"] = contactNameTrimmed.isEmpty ? .null : .string(contactNameTrimmed)
    if let phoneNormalized = AdValidators.validateOptionalPhone(contactPhone).normalized {
        data["contact_phone"] = .string(phoneNormalized)
    } else {
        data["contact_phone"] = .null
    }
    data["contact_method"] = .string(contactMethod.rawValue)

    data["audience"] = .string(audience.rawValue)

    let notesTrimmed = AdValidators.trimmed(notes)
    data["notes"] = notesTrimmed.isEmpty ? .null : .string(notesTrimmed)
    data["media_local_identifiers"] = .array(mediaLocalIdentifiers.map { .string($0) })
    data["ai_hints"] = .array(aiHints.map { .string($0) })

    if let category {

```



```

switch category {
case .delivery:
    let pickup = AdValidators.normalizedAddress(pickupAddress)
    let dropoff = AdValidators.normalizedAddress(dropoffAddress)

    data["pickup_address"] = .string(pickup)
    data["dropoff_address"] = .string(dropoff)
    data["start_at"] = .string(iso.string(from: startDate))
    data["has_end_time"] = .bool(hasEndTime)
    data["end_at"] = hasEndTime ? .string(iso.string(from: endDate)) : .null

    data["cargo_length"] = jsonOptionalDecimal(cargoLength)
    data["cargo_width"] = jsonOptionalDecimal(cargoWidth)
    data["cargo_height"] = jsonOptionalDecimal(cargoHeight)

    data["floor"] = jsonOptionalInt(floor)
    data["has_elevator"] = .bool(hasElevator)
    data["need_loader"] = .bool(needLoader)

    if let pickupPoint {
        let jsonPoint = Self.jsonPoint(pickupPoint)
        data["pickup_point"] = jsonPoint
        data["point"] = jsonPoint
    }
    if let dropoffPoint {
        data["dropoff_point"] = Self.jsonPoint(dropoffPoint)
    }

case .help:
    let address = AdValidators.normalizedAddress(helpAddress)
    data["address"] = .string(address)
    data["start_at"] = .string(iso.string(from: startDate))
    data["has_end_time"] = .bool(hasEndTime)
    data["end_at"] = hasEndTime ? .string(iso.string(from: endDate)) : .null

    if let helpPoint {
        let jsonPoint = Self.jsonPoint(helpPoint)
        data["help_point"] = jsonPoint
        data["point"] = jsonPoint
    }
}

}

//      let categoryRaw = category?.rawValue ?? "unknown"
//      return CreateAnnouncementRequest(
//          category: categoryRaw,
//          title: titleTrimmed,
//          status: "active",
//          data: data
//      )

let categoryRaw = category?.rawValue ?? "unknown"
let status = statusOverride ?? "active"

return CreateAnnouncementRequest(
    category: categoryRaw,
    title: titleTrimmed,
    status: status,
    data: data
)
}

private func jsonOptionalDecimal(_ raw: String) -> JSONValue {
    guard let value = AdValidators.parseDecimal(raw) else { return .null }
    return .double(value)
}

private func jsonOptionalInt(_ raw: String) -> JSONValue {

```

```

        let trimmed = AdValidators.trimmed(raw)
        guard let value = Int(trimmed) else { return .null }
        return .int(value)
    }

    private static func jsonPoint(_ point: YMKPoint) -> JSONValue {
        .object([
            "lat": .double(point.latitude),
            "lon": .double(point.longitude),
        ])
    }

    func setMediaFromPicker(datas: [Data]) {
        var images: [UIImage] = []
        var jpeg: [Data] = []

        for d in datas {
            guard let img = UIImage(data: d) else { continue }
            // сжимаем, чтобы не отправлять огромные файлы
            let out = img.jpegData(compressionQuality: 0.82) ?? d
            images.append(img)
            jpeg.append(out)
        }

        self.mediaImages = images
        self.mediaJPEGData = jpeg
    }

    func removeMedia(at index: Int) {
        guard mediaImages.indices.contains(index) else { return }
        mediaImages.remove(at: index)
        mediaJPEGData.remove(at: index)
    }
}

// ===== File 47/70: DelegationApp/Features/Ads/Model/Validators.swift =====
import Foundation

enum AdValidators {
    static let maxBudget: Double = 1_000_000
    static let minTitleLength = 3
    static let maxTitleLength = 200
    static let minAddressLength = 5
    static let minStartLeadSeconds: TimeInterval = 5 * 60
    static let minEndDeltaSeconds: TimeInterval = 10 * 60

    static func trimmed(_ value: String) -> String {
        value.trimmingCharacters(in: .whitespacesAndNewlines)
    }

    static func normalizedAddress(_ value: String) -> String {
        let compact = trimmed(value)
        .replacingOccurrences(of: "\\s+", with: " ", options: .regularExpression)
        return compact
    }

    static func validateTitle(_ value: String) -> String? {
        let v = trimmed(value)
        guard !v.isEmpty else {
            return "Заполните название объявления"
        }
        guard v.count >= minTitleLength else {
            return "Название должно быть не короче \(minTitleLength) символов"
        }
        guard v.count <= maxTitleLength else {
            return "Название должно быть не длинее \(maxTitleLength) символов"
        }
    }
}

```

```

    }
    return nil
}

static func parseDecimal(_ value: String) -> Double? {
    let raw = trimmed(value)
    guard !raw.isEmpty else { return nil }

    let normalized = raw
        .replacingOccurrences(of: " ", with: "")
        .replacingOccurrences(of: ",", with: ".")

    return Double(normalized)
}

static func validateBudget(_ value: String) -> String? {
    let raw = trimmed(value)
    guard !raw.isEmpty else { return nil }
    guard let number = parseDecimal(raw) else {
        return "Бюджет должен быть числом"
    }
    guard number >= 0 else {
        return "Бюджет не может быть отрицательным"
    }
    guard number <= maxBudget else {
        return "Бюджет слишком большой (максимум \(Int(maxBudget)))"
    }
    return nil
}

static func validateTimeWindow(startDate: Date, hasEndTime: Bool, endDate: Date) -> String? {
    let minStart = Date().addingTimeInterval(minStartLeadSeconds)
    guard startDate >= minStart else {
        return "Время начала должно быть не раньше чем через 5 минут"
    }

    guard hasEndTime else { return nil }
    guard endDate.timeIntervalSince(startDate) >= minEndDeltaSeconds else {
        return "Время окончания должно быть минимум на 10 минут позже начала"
    }
    return nil
}

static func validateAddress(_ value: String, fieldName: String) -> String? {
    let v = normalizedAddress(value)
    guard !v.isEmpty else {
        return "Укажите \(fieldName.lowercased())"
    }
    guard v.count >= minAddressLength else {
        return "\(fieldName) должен быть не короче \(minAddressLength) символов"
    }
    return nil
}

static func validateDifferentAddresses(_ pickup: String, _ dropoff: String) -> String? {
    let p = normalizedAddress(pickup).lowercased()
    let d = normalizedAddress(dropoff).lowercased()
    guard p != d else {
        return "Адрес забора и адрес доставки не должны совпадать"
    }
    return nil
}

static func validateDimension(_ value: String, fieldName: String, max: Double) -> String? {
    let raw = trimmed(value)
    guard !raw.isEmpty else { return nil }
    guard let number = parseDecimal(raw) else {
        return "\(fieldName) должен быть числом"
    }

```

```

    }
    guard number > 0 else {
        return "\(fieldName) должен быть больше 0"
    }
    guard number <= max else {
        return "\(fieldName) не может быть больше \((formatNumber(max)) см"
    }
    return nil
}

static func validateFloor(_ value: String) -> String? {
    let raw = trimmed(value)
    guard !raw.isEmpty else { return nil }
    guard let floor = Int(raw) else {
        return "Этаж должен быть целым числом"
    }
    guard (0...200).contains(floor) else {
        return "Этаж должен быть в диапазоне 0...200"
    }
    return nil
}

static func normalizePhone(_ raw: String) -> String? {
    let digits = raw.filter(\.isNumber)
    guard !digits.isEmpty else { return nil }
    guard digits.count == 11 else { return nil }

    if digits.first == "8" {
        return "+7" + String(digits.dropFirst())
    }

    guard digits.first == "7" else { return nil }
    return "+" + digits
}

static func validateOptionalPhone(_ raw: String) -> (normalized: String?, error: String?) {
    let trimmedPhone = trimmed(raw)
    guard !trimmedPhone.isEmpty else {
        return (nil, nil)
    }
    guard let normalized = normalizePhone(trimmedPhone) else {
        return (nil, "Телефон должен быть в формате +7XXXXXXXXX или 8XXXXXXXXX")
    }
    return (normalized, nil)
}

private static func formatNumber(_ value: Double) -> String {
    if floor(value) == value {
        return String(Int(value))
    }
    return String(format: "%.2f", value)
}
}

```

```

// ===== File 48/70: DelegationApp/Features/Ads/ViewModel/CreateAdFlowViewModel.swift =====
//
// CreateAdFlowViewModel.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```
import Foundation
```

```
@MainActor
```

```
final class CreateAdFlowViewModel: ObservableObject {
```

```

@Published var isSubmitting: Bool = false
@Published var errorText: String?

//-----
@Published var warningText: String?

private let service: AnnouncementService
private let session: SessionStore
private let searchService: AddressSearchService

init(
    service: AnnouncementService,
    session: SessionStore,
    searchService: AddressSearchService = AddressSearchService()
) {
    self.service = service
    self.session = session
    self.searchService = searchService
}

func submit(draft: CreateAdDraft) async -> AnnouncementDT0? {
    errorText = nil
    warningText = nil

    guard let token = session.token else {
        errorText = "Нет токена сессии"
        return nil
    }

    if let validationError = await draft.validateForSubmit(searchService: searchService) {
        errorText = validationError
        return nil
    }

    // -----
    // LOCAL TEXT MODERATION (instant UI)
    // -----
    let combined = [draft.title, draft.notes].joined(separator: "\n\n")
    let verdict = TextModerationService.shared.check(text: combined)

    var statusOverride: String = "active"
    switch verdict {
    case .allow:
        statusOverride = "active"
    case .reject(let reason):
        errorText = reason
        return nil
    case .reviewable(let reason):
        statusOverride = "draft"
        warningText = reason
        draft.aiHints.append("LOCAL_TEXT_REVIEW")
    }

    isSubmitting = true
    defer { isSubmitting = false }

    do {
        let req = draft.toCreateRequest(statusOverride: statusOverride)
        var created = try await service.createAnnouncement(token: token, request: req)

        // -----
        // IMAGES (server moderation)
        // -----
        if !draft.mediaJPEGData.isEmpty {
            let resp = try await service.uploadAnnouncementMedia(
                token: token,
                announcementId: created.id,
                images: draft.mediaJPEGData
            )
        }
    }
}

```

```

        )
        created = resp.announcement

        if resp.decision == "draft" {
            warningText = resp.message
        }
    }

    // если сервер вернул draft (например, из-за текста UNKNOWN) – предупреждаем
    if created.status == "draft", warningText == nil {
        warningText = "Объявление сохранено как черновик (модерация)."

```

```

// ===== File 49/70: DelegationApp/Features/Ads/ViewModel/MyAdsViewModel.swift =====
//
// MyAdsViewModel.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

```

```

import Foundation

```

```

@MainActor

```

```

final class MyAdsViewModel: ObservableObject {
    @Published var isLoading: Bool = false
    @Published var errorText: String?
    @Published private(set) var items: [AnnouncementDT0] = []

    let service: AnnouncementService
    let session: SessionStore

    init(service: AnnouncementService, session: SessionStore) {
        self.service = service
        self.session = session
    }

    func reload() async {
        guard let token = session.token else {
            errorText = "Нет токена сессии"
            return
        }

        isLoading = true
        defer { isLoading = false }

        do {
            items = try await service.myAnnouncements(token: token)
        } catch {
            errorText = error.localizedDescription
        }
    }

    // MARK: - Derived counts
    var activeCount: Int { items.filter { $0.status == "active" }.count }
    var draftCount: Int { items.filter { $0.status == "draft" }.count }
    var archivedCount: Int { items.filter { $0.status == "archived" }.count }
}

```

```

func filteredItems(for filter: AdsFilter) -> [AnnouncementDTO] {
    switch filter {
    case .active:
        return items.filter { $0.status == "active" }
    case .drafts:
        return items.filter { $0.status == "draft" }
    case .archived:
        return items.filter { $0.status == "archived" }
    }
}
}

```

```

enum AdsFilter: String, CaseIterable, Identifiable {
    case active = "Активные"
    case drafts = "Черновики"
    case archived = "Архив"

    var id: String { rawValue }
}

```

```

// ===== File 50/70: DelegationApp/Features/Auth/AuthScreen.swift =====
import SwiftUI

```

```

struct AuthScreen: View {
    @EnvironmentObject var container: AppContainer

    @State private var email: String = ""
    @State private var password: String = ""
    @State private var isLoginMode: Bool = false

    var body: some View {
        VStack(spacing: 16) {
            Text(isLoginMode ? "Вход" : "Регистрация")
                .font(.system(size: 24, weight: .bold))

            TextField("Email (например: name@mail.com)", text: $email)
                .keyboardType(.emailAddress)
                .textInputAutocapitalization(.never)
                .autocorrectionDisabled(true)
                .textFieldStyle(.roundedBorder)

            SecureField("Пароль", text: $password)
                .textFieldStyle(.roundedBorder)

            if let err = container.session.errorText {
                Text(err)
                    .foregroundColor(.red)
                    .font(.system(size: 13))
                    .multilineTextAlignment(.center)
            }

            Button {
                Task {
                    if isLoginMode {
                        await container.session.login(email: email, password: password)
                    } else {
                        await container.session.register(email: email, password: password)
                    }
                }
            } label: {
                HStack(spacing: 10) {
                    if container.session.isBusy {
                        ProgressView()
                    }
                    Text(isLoginMode ? "Войти" : "Создать аккаунт")
                }
            }
        }
    }
}

```

```

        .frame(maxWidth: .infinity)
    }
    .buttonStyle(.borderedProminent)
    .disabled(container.session.isBusy)

    Button {
        isLoginMode.toggle()
        container.session.errorText = nil
    } label: {
        Text(isLoginMode ? "Нет аккаунта? Зарегистрироваться" : "Уже есть аккаунт? Войти")
        .font(.system(size: 14))
    }
}
.padding()
}
}

```

```

// ===== File 51/70: DelegationApp/Features/Chats/View/ChatsScreen.swift =====
import SwiftUI

```

```

struct ChatsScreen: View {
    @StateObject var vm: ChatsViewModel
    init(vm: ChatService) { _vm = StateObject(wrappedValue: .init(service: vm)) }
    init(vm: ChatsViewModel) { _vm = StateObject(wrappedValue: vm) }

    var body: some View {
        List {
            ForEach(vm.chats) { chat in
                HStack(spacing: 12) {
                    Circle()
                        .fill(LinearGradient(colors: [Theme.ColorToken.turquoise, Theme.ColorToken.peach],
                                                startPoint: .topLeading, endPoint: .bottomTrailing))
                        .frame(width: 44, height: 44)
                        .overlay(Text(chat.initials).foregroundColor(.white).font(.system(size: 17, weight:
13)))
                            .bold()))

                    VStack(alignment: .leading, spacing: 4) {
                        HStack {
                            Text(chat.name).font(.system(size: 16, weight: .semibold))
                            Spacer()
                            Text(chat.time).foregroundColor(Theme.ColorToken.textSecondary).font(.system(size:
13))
                        }
                        Text(chat.lastMessage)
                            .foregroundColor(Theme.ColorToken.textSecondary)
                            .lineLimit(1)
                            .font(.system(size: 14))
                    }
                    if chat.unreadCount > 0 {
                        Text("\(chat.unreadCount)")
                            .font(.system(size: 12, weight: .bold))
                            .padding(.vertical, 4).padding(.horizontal, 8)
                            .background(Capsule().fill(Theme.ColorToken.turquoise))
                            .foregroundColor(.white)
                    }
                }
                .listRowBackground(Theme.ColorToken.white)
            }
        }
        .scrollContentBackground(.hidden)
        .background(Theme.ColorToken.milk)
        .navigationTitle("Сообщения")
    }
}

```

```

// #Preview {

```



```
// ChatsScreen(
//     vm: ChatsViewModel(service: MockChatService())
// )
//}
```

```
// ===== File 52/70: DelegationApp/Features/Chats/ViewModel/ChatsViewModel.swift =====
import Foundation
```

```
final class ChatsViewModel: ObservableObject {
    @Published var chats: [ChatPreview] = []

    private let service: ChatService
    init(service: ChatService) {
        self.service = service
        self.chats = service.loadChats()
    }
}
```

```
// ===== File 53/70: DelegationApp/Features/Chats.swift =====
//
// Chats.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//
```

```
// ===== File 54/70: DelegationApp/Features/Map/MapScreen/AnnouncementSheetView.swift =====
import SwiftUI
```

```
struct AnnouncementSheetView: View {
    let announcement: AnnouncementDT0
    let onRespondTap: (() -> Void)?

    @State private var showSoonAlert: Bool = false

    init(
        announcement: AnnouncementDT0,
        onRespondTap: (() -> Void)? = nil
    ) {
        self.announcement = announcement
        self.onRespondTap = onRespondTap
    }

    var body: some View {
        ScrollView {
            VStack(alignment: .leading, spacing: 16) {
                header
                addressSection
                timeSection
                budgetSection
                detailsSection
                contactsSection
            }
            .padding(20)
            .padding(.bottom, 20)
        }
        .safeAreaInset(edge: .bottom) {
            respondButton
        }
        .alert("Чқоп", isPresented: $showSoonAlert) {
            Button("Оқ", role: .cancel) {}
        }
    }
}
```

```

    } message: {
        Text("Отклики появятся в следующем обновлении.")
    }
}

private var header: some View {
    VStack(alignment: .leading, spacing: 8) {
        Text(announcement.title)
            .font(.system(size: 24, weight: .bold))

        Text(categoryTitle)
            .font(.system(size: 13, weight: .semibold))
            .foregroundColor(.white)
            .padding(.horizontal, 10)
            .padding(.vertical, 6)
            .background(Capsule().fill(Theme.ColorToken.turquoise))
    }
}

private var addressSection: some View {
    Group {
        if isDelivery {
            SectionCard(title: "Адреса") {
                valueRow("Откуда", value: valueOrDash(string("pickup_address")))
                valueRow("Куда", value: valueOrDash(string("dropoff_address")))
            }
        } else {
            SectionCard(title: "Адрес") {
                valueRow("Где", value: valueOrDash(string("address")))
            }
        }
    }
}

private var timeSection: some View {
    SectionCard(title: "Время") {
        valueRow("Начало", value: valueOrDash(formattedDate(string("start_at"))))

        if bool("has_end_time") == true {
            valueRow("Окончание", value: valueOrDash(formattedDate(string("end_at"))))
        }
    }
}

private var budgetSection: some View {
    SectionCard(title: "Бюджет") {
        valueRow("Сумма", value: valueOrDash(formattedBudget))
    }
}

private var detailsSection: some View {
    Group {
        if isDelivery {
            SectionCard(title: "Детали") {
                if let dimensions = dimensionsText {
                    valueRow("Габариты", value: dimensions)
                }

                if let floorText = floorText {
                    valueRow("Подъём", value: floorText)
                }

                valueRow("Лифт", value: bool("has_elevator") == true ? "Есть" : "Нет")
                valueRow("Грузчик", value: bool("need_loader") == true ? "Нужен" : "Не нужен")
            }
        }
    }
}

```

```

private var contactsSection: some View {
    SectionCard(title: "Контакты") {
        if let name = string("contact_name"), !name.isEmpty {
            valueRow("Имя", value: name)
        }
        if let phone = string("contact_phone"), !phone.isEmpty {
            valueRow("Телефон", value: phone)
        }
        if let method = contactMethodTitle {
            valueRow("Связь", value: method)
        }

        if string("contact_name") == nil,
            string("contact_phone") == nil,
            contactMethodTitle == nil {
            valueRow("Контакты", value: "-")
        }
    }
}

private var respondButton: some View {
    Button {
        onRespondTap?()
        showSoonAlert = true
    } label: {
        Text("Откликнуться")
            .font(.system(size: 16, weight: .bold))
            .foregroundColor(.white)
            .frame(maxWidth: .infinity)
            .padding(.vertical, 14)
            .background(
                RoundedRectangle(cornerRadius: 16, style: .continuous)
                    .fill(Theme.ColorToken.turquoise)
            )
            .padding(.horizontal, 20)
            .padding(.vertical, 12)
    }
    .background(.ultraThinMaterial)
}

private var isDelivery: Bool {
    announcement.category.lowercased() == "delivery"
}

private var categoryTitle: String {
    switch announcement.category.lowercased() {
    case "delivery": return "Доставка и поручения"
    case "help": return "Помощь"
    default: return announcement.category
    }
}

private var formattedBudget: String? {
    if let number = announcement.data["budget"]?.doubleValue {
        return "\(Int(number)) "
    }
    if let str = string("budget"), !str.isEmpty {
        return "\(str) "
    }
    return nil
}

private var dimensionsText: String? {
    let length = announcement.data["cargo_length"]?.doubleValue
    let width = announcement.data["cargo_width"]?.doubleValue
    let height = announcement.data["cargo_height"]?.doubleValue

```

```

        var parts: [String] = []
        if let length { parts.append("Д \(\formatNumber(length)) см") }
        if let width { parts.append("Ш \(\formatNumber(width)) см") }
        if let height { parts.append("В \(\formatNumber(height)) см") }

        return parts.isEmpty ? nil : parts.join(separator: " • ")
    }

    private var floorText: String? {
        let floorValue = announcement.data["floor"]?.doubleValue
        guard let floorValue else { return nil }
        return "\(\Int(floorValue)) этаж"
    }

    private var contactMethodTitle: String? {
        guard let raw = string("contact_method") else { return nil }
        switch raw {
            case "calls_and_messages": return "Звонки и сообщения"
            case "messages_only": return "Только сообщения"
            case "calls_only": return "Только звонки"
            default: return raw
        }
    }

    private func string(_ key: String) -> String? {
        guard let raw = announcement.data[key]?.stringValue?.trimmingCharacters(in: .whitespacesAndNewlines)
    else {
        return nil
    }
    return raw.isEmpty ? nil : raw
}

    private func bool(_ key: String) -> Bool? {
        announcement.data[key]?.boolValue
    }

    private func formattedDate(_ iso: String?) -> String? {
        guard let iso, !iso.isEmpty else { return nil }

        let parser = ISO8601DateFormatter()
        parser.formatOptions = [.withInternetDateTime, .withFractionalSeconds]
        if let date = parser.date(from: iso) {
            return date.formatted(date: .abbreviated, time: .shortened)
        }

        parser.formatOptions = [.withInternetDateTime]
        if let date = parser.date(from: iso) {
            return date.formatted(date: .abbreviated, time: .shortened)
        }
        return iso
    }

    private func valueOrDash(_ value: String?) -> String {
        if let value, !value.isEmpty { return value }
        return "-"
    }

    private func formatNumber(_ value: Double) -> String {
        if floor(value) == value {
            return String(Int(value))
        }
        return String(format: "%.1f", value)
    }

    @ViewBuilder
    private func valueRow(_ title: String, value: String) -> some View {
        VStack(alignment: .leading, spacing: 2) {

```

```

        Text(title)
            .font(.system(size: 12, weight: .medium))
            .foregroundColor(.secondary)
        Text(value)
            .font(.system(size: 15, weight: .semibold))
    }
    .frame(maxWidth: .infinity, alignment: .leading)
}
}

private struct SectionCard<Content: View>: View {
    let title: String
    @ViewBuilder let content: Content

    var body: some View {
        VStack(alignment: .leading, spacing: 12) {
            Text(title)
                .font(.system(size: 17, weight: .bold))

            VStack(alignment: .leading, spacing: 12) {
                content
            }
        }
        .padding(14)
        .background(
            RoundedRectangle(cornerRadius: 14, style: .continuous)
                .fill(Color.white.opacity(0.8))
        )
        .overlay(
            RoundedRectangle(cornerRadius: 14, style: .continuous)
                .stroke(Theme.ColorToken.turquoise.opacity(0.15), lineWidth: 1)
        )
    }
}

```

// ===== File 55/70: DelegationApp/Features/Map/MapScreen/MapScreen.swift =====

import SwiftUI

```

struct MapScreen: View {
    @StateObject private var vm: MapViewModel
    private let mapMode: MapDisplayMode

    init(
        vm: MapViewModel,
        mapMode: MapDisplayMode = MapDisplayConfig.defaultMode()
    ) {
        _vm = StateObject(wrappedValue: vm)
        self.mapMode = mapMode
    }

    var body: some View {
        ZStack(alignment: .top) {
            mapArea

            VStack(spacing: 5) {
                Spacer().frame(height: 50)
                searchBar
                errorLabel
                chipsRow
                Spacer()
            }
            .padding(.horizontal, 16)
            .padding(.top, 8)
            .ignoresSafeArea()
        }
        .task {

```

```

        await vm.reloadPins()
    }
    .sheet(item: $vm.selectedAnnouncement, onDismiss: {
        vm.clearSelection()
    }) { announcement in
        AnnouncementSheetView(announcement: announcement)
            .presentationDetents([.medium, .large])
            .presentationDragIndicator(.visible)
    }
}

private var searchBar: some View {
    HStack(spacing: 8) {
        Image(systemName: "magnifyingglass")
            .foregroundColor(Theme.ColorToken.textSecondary)

        TextField("Введите адрес", text: $vm.searchText, onCommit: vm.performSearch)
            .textFieldStyle(.plain)

        if !vm.searchText.isEmpty {
            Button {
                vm.searchText = ""
            } label: {
                Image(systemName: "xmark.circle.fill")
                    .foregroundColor(Theme.ColorToken.textSecondary)
                    .imageScale(.medium)
            }
        }

        Button(action: vm.performSearch) {
            Text("Найти")
                .font(.system(size: 15, weight: .semibold))
        }
    }
    .padding(.horizontal, 12)
    .padding(.vertical, 10)
    .background(.ultraThinMaterial)
    .clipShape(RoundedRectangle(cornerRadius: 16, style: .continuous))
    .softCardShadow()
}

private var errorLabel: some View {
    Group {
        if let message = vm.errorMessage {
            Text(message)
                .font(.caption)
                .foregroundColor(.red)
                .frame(maxWidth: .infinity, alignment: .leading)
        }
    }
}

private var chipsRow: some View {
    ScrollView(.horizontal, showsIndicators: false) {
        HStack(spacing: Theme.Spacing.m) {
            ForEach(vm.chips, id: \.self) { chip in
                FilterChip(
                    title: chip,
                    isSelected: Binding(
                        get: { vm.selected.contains(chip) },
                        set: { isOn in
                            if isOn { vm.selected.insert(chip) }
                            else { vm.selected.remove(chip) }
                        }
                    )
                )
            }
        }
    }
}

```

```

    }
}

private var mapArea: some View {
    MapCanvasView(
        centerPoint: $vm.centerPoint,
        pins: vm.pins,
        selectedPinID: vm.selectedPinID,
        routePolyline: vm.routePolyline,
        shouldFitRoute: vm.shouldFitRoute,
        onRouteFitted: vm.consumeRouteFitRequest,
        onPinTap: { pinID in
            Task { @MainActor in
                await vm.selectAnnouncement(pinID: pinID)
            }
        },
        mode: mapMode
    )
    .ignoresSafeArea(edges: .top)
}
}

```

// ===== File 56/70: DelegationApp/Features/Map/MapViewModel/MapViewModel.swift =====

```

import SwiftUI
import YandexMapsMobile
import Foundation

struct MapAdPin: Identifiable {
    let id: String
    let point: YMKPoint
    let label: String
    let announcement: AnnouncementDTO
}

enum MapDisplayMode {
    case real
    case placeholder
}

enum MapDisplayConfig {
    static func defaultMode() -> MapDisplayMode {
        #if DEBUG
            return .real
        #else
            return .real
        #endif
    }
}

```

```

struct MapCanvasView: View {
    @Binding var centerPoint: YMKPoint?
    let pins: [MapAdPin]
    let selectedPinID: String?
    let routePolyline: YMKPolyline?
    let shouldFitRoute: Bool
    let onRouteFitted: () -> Void
    let onPinTap: (String) -> Void
    let mode: MapDisplayMode

    var body: some View {
        Group {
            switch mode {
            case .real:
                YandexMapView(
                    centerPoint: $centerPoint,
                    pins: pins,

```

```

        selectedPinID: selectedPinID,
        routePolyline: routePolyline,
        shouldFitRoute: shouldFitRoute,
        onRouteFitted: onRouteFitted,
        onPinTap: onPinTap
    )
    case .placeholder:
        Rectangle()
            .fill(Theme.ColorToken.milk)
            .overlay(
                VStack(spacing: 8) {
                    Image(systemName: "map")
                        .font(.system(size: 32))
                        .foregroundColor(Theme.ColorToken.textSecondary)
                    Text("Map placeholder")
                        .font(.system(size: 14, weight: .medium))
                        .foregroundColor(Theme.ColorToken.textSecondary)
                }
            )
    }
}
}
}
}

```

@MainActor

```

private final class DrivingRouteService {
    enum RouteError: LocalizedError {
        case unavailable
        case noRoute

        var errorDescription: String? {
            switch self {
            case .unavailable:
                return "Маршрутизация недоступна"
            case .noRoute:
                return "Маршрут не найден"
            }
        }
    }

    private let router: YMKDrivingRouter?
    private var session: YMKDrivingSession?

    init() {
        YandexMapConfigurator.configureIfNeeded()
        self.router = YMKDirections.sharedInstance()?.createDrivingRouter(withType: .combined)
    }

    func buildRoute(from: YMKPoint, to: YMKPoint) async throws -> YMKPolyline {
        guard let router else { throw RouteError.unavailable }

        let requestPoints = [
            YMKRequestPoint(point: from, type: .waypoint, pointContext: nil, drivingArrivalPointId: nil),
            YMKRequestPoint(point: to, type: .waypoint, pointContext: nil, drivingArrivalPointId: nil),
        ]
        let drivingOptions = YMKDrivingOptions()
        drivingOptions.routesCount = 1
        let vehicleOptions = YMKDrivingVehicleOptions()

        return try await withCheckedThrowingContinuation { continuation in
            var didResume = false
            session?.cancel()
            session = router.requestRoutes(
                with: requestPoints,
                drivingOptions: drivingOptions,
                vehicleOptions: vehicleOptions
            ) { [weak self] routes, error in
                guard !didResume else { return }

```



```

        didResume = true
        self?.session = nil

        if let error {
            continuation.resume(throwing: error)
            return
        }
        guard let route = routes?.first else {
            continuation.resume(throwing: RouteError.noRoute)
            return
        }
        continuation.resume(returning: route.geometry)
    }
}

func cancel() {
    session?.cancel()
    session = nil
}
}

@MainActor
final class MapViewModel: ObservableObject {
    // MARK: - Фильтры
    @Published var chips: [String] = [
        "Купить", "Доставить", "Забрать",
        "Помочь", "Перенести", "Другое"
    ]
    @Published var selected: Set<String> = []

    // MARK: - Моковые задачи (пока оставляем)
    @Published var tasks: [TaskItem] = []

    // MARK: - Объявления на карте
    @Published private(set) var announcements: [AnnouncementDTO] = []
    @Published private(set) var pins: [MapAdPin] = []
    @Published var selectedAnnouncement: AnnouncementDTO?
    @Published var selectedPinID: String?
    @Published var routePolyline: YMKPolyline?
    @Published var shouldFitRoute: Bool = false

    // MARK: - Поиск и карта
    @Published var searchText: String = ""
    @Published var centerPoint: YMKPoint?
    @Published var errorMessage: String?

    private let service: TaskService
    private let announcementService: AnnouncementService
    private let searchService: AddressSearchService
    private let routeService: DrivingRouteService
    private var geocodeCache: [String: YMKPoint] = [:]

    init(
        service: TaskService,
        announcementService: AnnouncementService,
        searchService: AddressSearchService = AddressSearchService()
    ) {
        self.service = service
        self.announcementService = announcementService
        self.searchService = searchService
        self.routeService = DrivingRouteService()

        self.tasks = service.loadNearbyTasks()
        self.centerPoint = YMKPoint(latitude: 55.751244, longitude: 37.618423)
    }

    func toggle(_ chip: String) {

```

```

        if selected.contains(chip) { selected.remove(chip) }
        else { selected.insert(chip) }
    }

    func performSearch() {
        let query = searchText.trimmingCharacters(in: .whitespacesAndNewlines)
        guard !query.isEmpty else {
            errorMessage = nil
            return
        }

        searchService.searchAddress(query) { [weak self] point in
            DispatchQueue.main.async {
                guard let self else { return }
                if let point {
                    self.centerPoint = point
                    self.errorMessage = nil
                } else {
                    self.errorMessage = "Адрес не найден"
                }
            }
        }
    }

    func reloadPins() async {
        do {
            let list = try await announcementService.publicAnnouncements()
            announcements = list
            pins = list.compactMap { Self.makePin(from: $0) }
        } catch {
            errorMessage = "Не удалось загрузить объявления: \(error.localizedDescription)"
        }
    }

    func selectAnnouncement(pinID: String) async {
        guard let pin = pins.first(where: { $0.id == pinID }) else { return }

        selectedPinID = pinID
        selectedAnnouncement = pin.announcement
        errorMessage = nil

        await buildRouteIfNeeded(for: pin.announcement, expectedPinID: pinID)
    }

    func clearSelection() {
        selectedAnnouncement = nil
        selectedPinID = nil
        clearRoute()
    }

    func consumeRouteFitRequest() {
        shouldFitRoute = false
    }

    private func clearRoute() {
        routeService.cancel()
        routePolyline = nil
        shouldFitRoute = false
    }

    private func buildRouteIfNeeded(for announcement: AnnouncementDTO, expectedPinID: String) async {
        clearRoute()
        guard announcement.category.lowercased() == "delivery" else { return }

        guard let points = await resolveDeliveryRoutePoints(for: announcement) else {
            guard selectedPinID == expectedPinID else { return }
            errorMessage = "Не удалось определить точки маршрута для объявления"
        }
    }

```

```

        return
    }

    do {
        let polyline = try await routeService.buildRoute(from: points.start, to: points.end)
        guard selectedPinID == expectedPinID else { return }
        routePolyline = polyline
        shouldFitRoute = true
    } catch {
        guard selectedPinID == expectedPinID else { return }
        errorMessage = "Не удалось построить маршрут: \"(error.localizedDescription)\"
    }
}

private func resolveDeliveryRoutePoints(
    for announcement: AnnouncementDTO
) async -> (start: YMKPoint, end: YMKPoint)? {

    let data = announcement.data

    var start: YMKPoint? = Self.extractPoint(from: data, keys: ["pickup_point", "point"])
    if start == nil {
        start = await geocodeFromData(data: data, key: "pickup_address")
    }

    var end: YMKPoint? = Self.extractPoint(from: data, keys: ["dropoff_point"])
    if end == nil {
        end = await geocodeFromData(data: data, key: "dropoff_address")
    }

    guard let start, let end else { return nil }
    return (start, end)
}

private func geocodeFromData(data: [String: JSONValue], key: String) async -> YMKPoint? {
    guard
        let address = data[key]?.stringValue?
            .trimmingCharacters(in: .whitespacesAndNewlines),
        !address.isEmpty
    else { return nil }

    if let cached = geocodeCache[address] {
        return cached
    }

    let point = await searchService.searchAddress(address)
    if let point {
        geocodeCache[address] = point
    }
    return point
}

private static func makePin(from announcement: AnnouncementDTO) -> MapAdPin? {
    guard let point = extractDisplayPoint(from: announcement) else { return nil }
    return MapAdPin(
        id: announcement.id,
        point: point,
        label: markerLabel(from: announcement),
        announcement: announcement
    )
}

private static func extractDisplayPoint(from announcement: AnnouncementDTO) -> YMKPoint? {
    let data = announcement.data
    if announcement.category.lowercased() == "delivery" {
        if let p = extractPoint(from: data, keys: ["pickup_point", "point"]) { return p }
    } else if announcement.category.lowercased() == "help" {
        if let p = extractPoint(from: data, keys: ["help_point", "point"]) { return p }
    }
}

```

```

        return extractPoint(from: data, keys: ["point", "pickup_point", "help_point"])
    }

    private static func extractPoint(
        from data: [String: JSONValue],
        keys: [String]
    ) -> YMKPoint? {
        for key in keys {
            guard let pointVal = data[key]?.objectValue else { continue }
            guard
                let lat = pointVal["lat"]?.doubleValue,
                let lon = pointVal["lon"]?.doubleValue
            else { continue }
            return YMKPoint(latitude: lat, longitude: lon)
        }
        return nil
    }

    private static func markerLabel(from announcement: AnnouncementDTO) -> String {
        if let budget = announcement.data["budget"]?.doubleValue, budget > 0 {
            return "\(Int(budget)) "
        }
        if let budgetText = announcement.data["budget"]?.stringValue,
            let parsed = Double(budgetText.replacingOccurrences(of: ",", with: ".")),
            parsed > 0 {
            return "\(Int(parsed)) "
        }
        return announcement.category.lowercased() == "delivery" ? "Доставка" : "Помощь"
    }
}

```

```

// ===== File 57/70: DelegationApp/Features/Profile/View/ProfileEditScreen.swift =====
////
//// ProfileEditScreen.swift
//// iCuno test
////
//// Created by maftuna murtazaeva on 23.01.2026.
////
//
//import SwiftUI
//import PhotosUI
//import Foundation
//
//struct UserSettingsView: View {
//    @State private var selectedItem: PhotosPickerItem? = nil
//    @State private var showAgePicker = false
//    @State private var selectedImage: UIImage? = nil
//
//    var body: some View {
//        VStack {
//            HStack { Spacer(); saveButton }
//            ScrollView {
//
//                profileImage
//
//                nicknameSection
//                randomNameButton.padding(.bottom, 20)
//
//                bioSection.padding(.bottom, 20)
//                ageSection.padding(.bottom, 20)
//
//                genderPicker.padding(.bottom, 20)
//                Spacer()
//            }
//        }
//    }
//
//}

```

```

////      .onAppear { viewModel.onAppear() }
////      .alert("Сохранено!", isPresented: $viewModel.showSaveBanner) {
////          Button("OK", role: .cancel) {}
////      }
////      .sheet(isPresented: $showAgePicker) { agePicker }
//  }
//
//  private var saveButton: some View {
//      Button("Сохранить") { }
//          .foregroundColor(.red)
//          .padding(.horizontal, 25)
//  }
//
//
//  var genderPicker: some View {
//      VStack(alignment: .leading) {
//          Text("Ваш пол: Мужской")
//          Text("Укажите свой пол для анкеты")
//              .font(.footnote)
//
//          Picker("", selection: Binding(
//              get: { viewModel.gender },
//              set: { viewModel.genderSelected($0) }
//          )) {
//              ForEach(Gender1.allCases) { g in
//                  Text(g.rawValue).tag(g)
//              }
//          }
//          .pickerStyle(.segmented)
//      }
//      .padding(.horizontal, 25)
//  }
//
//  var profileImage: some View {
//      VStack {
//          if let image = viewModel.image {
//              Image(uiImage: image)
//                  .resizable()
//                  .frame(width: 130, height: 130)
//                  .cornerRadius(65)
//                  .padding(.top, 25)
//          } else {
//              Text("Изображение не выбрано")
//                  .foregroundColor(.gray)
//                  .padding()
//          }
//
//          PhotosPicker(
//              selection: $selectedItem,
//              matching: .images,
//              photoLibrary: .shared()
//          ) {
//              Text("Изменить фотографию")
//                  .foregroundColor(Color(.label))
//          }
//          .onChange(of: selectedItem) { newItem in
//              Task {
//                  await viewModel.handleImageSelection(newItem)
//              }
//          }
//      }
//      .padding()
//  }
//
//
//  private var nicknameSection: some View {
//      VStack(alignment: .leading) {
//          Text("Никнейм: \(viewModel.name)")
//      }
//  }

```

```

//      Text("Никнейм будет отображён в анкете")
//      .font(.footnote)
//      TextField("Введите никнейм...",
//          text: Binding(get: { viewModel.name },
//              set: viewModel.nameChanged))
//      .textFieldStyle(.roundedBorder)
//      .overlay(RoundedRectangle(cornerRadius: 20)
//          .stroke(Color(.label).opacity(0.4), lineWidth: 1))
//  }
//  .padding(.horizontal, 25)
//  }
//  private var randomNameButton: some View {
//      Button("СЛУЧАЙНЫЙ НИКНЕЙМ") { viewModel.nicknameTapped() }
//          .frame(maxWidth: .infinity)
//          .padding(.vertical, 10)
//          .background(Color(.systemGroupedBackground))
//          .cornerRadius(20)
//          .overlay(RoundedRectangle(cornerRadius: 20)
//              .stroke(Color(.label).opacity(0.4), lineWidth: 1))
//          .padding(.horizontal, 25)
//          .foregroundColor(Color(.label))
//  }
//
//  private var bioSection: some View {
//      VStack(alignment: .leading) {
//          Text("0 себе")
//          Text("Напиши пару слов о себе, чтобы заинтересовать собеседника")
//              .font(.footnote)
//          TextField("0 себе",
//              text: Binding(get: { viewModel.bio },
//                  set: viewModel.bioChanged))
//              .textFieldStyle(.roundedBorder)
//              .overlay(RoundedRectangle(cornerRadius: 20)
//                  .stroke(Color(.label).opacity(0.4), lineWidth: 1))
//      }
//      .padding(.horizontal, 25)
//  }
//
//  private var ageSection: some View {
//      VStack(alignment: .leading) {
//          Text("Ваш возраст: \(viewModel.age)")
//          Text("Укажите свой возраст для анкеты")
//              .font(.footnote)
//          Button("УКАЖИТЕ ВОЗРАСТ") { showAgePicker = true }
//              .frame(maxWidth: .infinity)
//              .padding(.vertical, 10)
//              .background(Color(.systemGroupedBackground))
//              .cornerRadius(20)
//              .overlay(RoundedRectangle(cornerRadius: 20)
//                  .stroke(Color(.label).opacity(0.4), lineWidth: 1))
//              .foregroundColor(Color(.label))
//      }
//      .padding(.horizontal, 25)
//  }
//
//  private var agePicker: some View {
//      VStack(spacing: 0) {
//          Text("Выберите ваш возраст")
//              .font(.headline)
//              .padding()
//              .frame(maxWidth: .infinity)
//              .background(Color(.systemGray6))
//          Divider()
//          Picker("Возраст", selection: Binding(
//              get: { viewModel.age },
//              set: viewModel.ageSelected

```



```

HStack(spacing: 28) {
  VStack(alignment: .leading) {
    HStack(spacing: 6) {
      Image(systemName: "star.fill").foregroundStyle(Theme.ColorToken.peach)
      Text("\(vm.profile.rating, specifier: "%.1f")")
        .font(.system(size: 16, weight: .semibold))
    }
    Text("Рейтинг").foregroundStyle(Theme.ColorToken.textSecondary).font(.system(size: 12))
  }
  VStack(alignment: .leading) {
    Text("\(vm.profile.completed)").font(.system(size: 16, weight: .semibold))
    Text("Выполнено").foregroundStyle(Theme.ColorToken.textSecondary).font(.system(size: 12))
  }
  VStack(alignment: .leading) {
    Text("\(vm.profile.cancelled)").font(.system(size: 16, weight: .semibold))
    Text("Отменено").foregroundStyle(Theme.ColorToken.textSecondary).font(.system(size: 12))
  }
  Spacer()
}
}
.padding()

.background(LinearGradient(colors: [Theme.ColorToken.turquoise.opacity(0.85),
Theme.ColorToken.turquoise],
startPoint: .topLeading, endPoint: .bottomTrailing))
.foregroundStyle(.white)
.clipShape(RoundedRectangle(cornerRadius: Theme.Radius.xl, style: .continuous))
.padding(.horizontal)
.padding(.top, 12)
.softCardShadow()
}

private var settings: some View {
  SectionBox(title: "Настройки") {
    ToggleRow(title: "Тёмная тема", isOn: $vm.darkMode)
    NavRow(title: "Уведомления")
    NavRow(title: "Платежи и выплаты")
  }
}

private var support: some View {
  SectionBox(title: "Поддержка") {
    NavRow(title: "Помощь")
    NavRow(title: "Правила и условия")
  }
}

private var reviews: some View {
  SectionBox(title: "Отзывы") {
    ForEach(vm.reviews) { r in
      HStack(alignment: .top, spacing: 12) {
        Circle().fill(Theme.ColorToken.milk).frame(width: 40, height: 40)
          .overlay(Text(r.authorInitial).font(.system(size: 16, weight: .bold)))
        VStack(alignment: .leading, spacing: 6) {
          HStack {
            Text(r.authorName).font(.system(size: 15, weight: .semibold))
            StarsView(rating: Double(r.stars))
            Spacer()
          }
          Text(r.text).font(.system(size: 14)).fixedSize(horizontal: false, vertical: true)
          Text(r.ago).font(.system(size: 12)).foregroundStyle(Theme.ColorToken.textSecondary)
        }
        Spacer(minLength: 0)
      }
      .padding(.vertical, 8)
      .padding(.horizontal, 8)
    }
  }
  Button("Посмотреть все отзывы") { }
    .font(.system(size: 15, weight: .semibold))
}

```



```

        .frame(maxWidth: .infinity, alignment: .leading)
        .padding(.top, 6)
        .tint(Theme.ColorToken.turquoise)
        .padding()
    }
}

private struct SectionBox<Content: View>: View {
    let title: String
    @ViewBuilder var content: Content

    var body: some View {
        VStack(alignment: .leading, spacing: 8) {
            Text(title).font(.system(size: 12, weight: .bold))
                .foregroundColor(Theme.ColorToken.textSecondary)
                .padding(.horizontal)
            VStack(spacing: 0) { content }
                .background(RoundedRectangle(cornerRadius: Theme.Radius.l).fill(Theme.ColorToken.white))
                .softCardShadow()
                .padding(.horizontal)
        }
        .padding(.top, 4)
    }
}

private struct ToggleRow: View {
    let title: String
    @Binding var isOn: Bool
    var body: some View {
        HStack {
            Label(title, systemImage: "moon.fill")
                .labelStyle(.titleAndIcon)
            Spacer()
            Toggle("", isOn: $isOn).labelsHidden()
        }
        .padding()
        .background(Color.clear)
    }
}

private struct NavRow: View {
    let title: String
    var body: some View {
        HStack {
            Text(title)
            Spacer()
            Image(systemName: "chevron.right").foregroundColor(Theme.ColorToken.textSecondary)
        }
        .padding()
    }
}

// #Preview {
//     let service = MockProfileService()
//     let vm = ProfileViewModel(service: service)
//     ProfileScreen(vm: vm)
// }

// @StateObject var vm: ProfileViewModel
// init(vm: ProfileViewModel) { _vm = StateObject(wrappedValue: vm) }

// ===== File 59/70: DelegationApp/Features/Profile/ViewModel/ProfileViewModel.swift =====
import Foundation

```

```

final class ProfileViewModel: ObservableObject {
    @Published var profile: Profile
    @Published var reviews: [Review]
    @Published var darkMode: Bool = false

    private let service: ProfileService
    init(service: ProfileService) {
        self.service = service
        self.profile = service.loadProfile()
        self.reviews = service.loadReviews()
    }
}

// ===== File 60/70: DelegationApp/Features/Route/View/RouteScreen.swift =====
import SwiftUI

enum PreviewData {
    @MainActor static var container: AppContainer { .live }

    static let chatsVM = ChatsViewModel(service: MockChatService())

    @MainActor static var mapVM: MapViewModel {
        MapViewModel(
            service: MockTaskService(),
            announcementService: MockAnnouncementService()
        )
    }

    static let routeVM = RouteViewModel(service: MockTaskService())
    static let profileVM = ProfileViewModel(service: MockProfileService())
}

struct RouteScreen: View {
    @StateObject var vm: RouteViewModel
    init(vm: RouteViewModel) { _vm = StateObject(wrappedValue: vm) }

    var body: some View {
        ScrollView {
            VStack(spacing: Theme.Spacing.l) {
                VStack(spacing: Theme.Spacing.m) {
                    RouteRow(symbol: "a.circle.fill", text: vm.pointA)
                    RouteRow(symbol: "b.circle.fill", text: vm.pointB)
                    RouteRow(symbol: "clock.fill", text: vm.time)
                }
                .padding()
                .background(RoundedRectangle(cornerRadius: Theme.Radius.l)
                    .fill(Theme.ColorToken.white))
                .softCardShadow()
                .padding(.horizontal)

                HStack {
                    Image(systemName: "arrow.forward.circle")
                    Text("45 мин · 12.5 км")
                        .font(.system(size: 16, weight: .semibold))
                    Spacer()
                    Capsule()
                        .fill(Theme.ColorToken.milk)
                        .frame(width: 36, height: 28)
                        .overlay(Text("\(vm.tasks.count)").font(.system(size: 15, weight: .semibold)))
                }
                .padding()
                .background(RoundedRectangle(cornerRadius: Theme.Radius.l)
                    .fill(Theme.ColorToken.white))
                .softCardShadow()
                .padding(.horizontal)
            }
        }
    }
}

```

```

        // Карта заглушка
        RoundedRectangle(cornerRadius: Theme.Radius.l)
            .fill(Theme.ColorToken.milk)
            .frame(height: 220)
            .overlay(Text("Карта с маршрутом").foregroundColor(Theme.ColorToken.textSecondary))
            .padding(.horizontal)

        VStack(alignment: .leading, spacing: Theme.Spacing.m) {
            Text("Задания по пути")
                .font(.system(size: 18, weight: .semibold))
            ForEach(vm.tasks) { t in
                HStack {
                    VStack(alignment: .leading, spacing: 4) {
                        Text(t.title).font(.system(size: 16, weight: .semibold))
                        Text("~\\(t.distanceKm, specifier: "%.1f") км • \\(t.etaMinutes) мин")
                            .foregroundColor(Theme.ColorToken.textSecondary)
                            .font(.system(size: 13))
                    }
                    Spacer()
                    PriceTag(price: t.price, eta: t.etaMinutes)
                }
                .padding()
            }
            .background(RoundedRectangle(cornerRadius:
Theme.Radius.m).fill(Theme.ColorToken.white))
                .softCardShadow()
        }
        .padding(.horizontal)
        .padding(.bottom, 24)
    }
}
.navigationTitle("Маршрут")
}
}

private struct RouteRow: View {
    let symbol: String
    let text: String
    var body: some View {
        HStack(spacing: 12) {
            Image(systemName: symbol)
                .foregroundColor(Theme.ColorToken.turquoise)
            Text(text)
            Spacer()
        }
        .font(.system(size: 16))
    }
}

// #Preview("RouteScreen") {
//     NavigationStack {
//         RouteScreen(vm: PreviewData.routeVM)
//     }
//     .preferredColorScheme(.light)
// }

// ===== File 61/70: DelegationApp/Features/Route/View/RouteView.swift =====
////
//// RouteView.swift
//// iCuno test
////
//// RootView с таббаром.
////
//
```

```

//import SwiftUI
//
//struct RootView: View {
//    @EnvironmentObject var container: AppContainer
//    @State private var selectedTab = 0
//
//    var body: some View {
//        TabView(selection: $selectedTab) {
//
//            // Вкладка "Карта"
//            NavigationStack {
//                MapScreen(vm: .init(service: container.taskService), mapMode: .placeholder)
//            }
//            .tabItem {
//                Label("Карта", systemImage: "map")
//            }
//            .tag(0)
//
//            // Вкладка "Маршрут"
//            NavigationStack {
//                RouteScreen(vm: .init(service: container.taskService))
//            }
//            .tabItem {
//                Label("Маршрут", systemImage: "point.topleft.down.curvedto.point.bottomright.up")
//            }
//            .tag(1)
//
//            // Новая вкладка "Объявления"
//            NavigationStack {
//                MyAdsScreen()
//            }
//            .tabItem {
//                Label("Объявления", systemImage: "rectangle.stack.badge.plus")
//            }
//            .tag(2)
//
//            // Вкладка "Чаты"
//            NavigationStack {
//                ChatsScreen(vm: .init(service: container.chatService))
//            }
//            .tabItem {
//                Label("Чаты", systemImage: "bubble.left.and.bubble.right")
//            }
//            .tag(3)
//
//            // Вкладка "Профиль"
//            NavigationStack {
//                ProfileScreen(vm: .init(service: container.profileService))
//            }
//            .tabItem {
//                Label("Профиль", systemImage: "person.circle")
//            }
//            .tag(4)
//        }
//        .background(Color.black)
//        .ignoresSafeArea()
//        .tint(Theme.ColorToken.turquoise)
//        .cornerRadius(20)
//
//        .background(.ultraThinMaterial)
//        .clipShape(RoundedRectangle(cornerRadius: 16, style: .continuous))
//        .softCardShadow()
//    }
//}
//
//import SwiftUI
//

```

```

//// MARK: - RootView с кастомным «Liquid Glass» TabBar
//struct RootView: View {
//    @EnvironmentObject var container: AppContainer
//
//    // Текущая вкладка
//    @State private var selection: AppTab = .map
//
//    // При желании можно показать бейджи, как на скрине (пример: на профиле "2")
//    private let badges: [AppTab: Int] = [.profile: 2]
//
//    var body: some View {
//        ZStack(alignment: .bottom) {
//            // Контент – позади, двигается сам по себе.
//            content
//                .transition(.identity)
//
//            // Полупрозрачный «стеклянный» TabBar, закреплённый снизу
//            LiquidTabBar(selection: $selection, badges: badges)
//                .padding(.horizontal, 16)
//                .padding(.bottom, 12)
//                .allowsHitTesting(true)
//        }
//        .ignoresSafeArea(edges: .bottom)
//        .tint(Theme.ColorToken.turquoise)
//    }
//
//    // MARK: - Контент по вкладкам
//    @ViewBuilder
//    private var content: some View {
//        switch selection {
//            case .map:
//                NavigationStack {
//                    MapScreen(
//                        vm: MapViewModel(
//                            service: container.taskService,
//                            searchService: AddressSearchService()
//                        )
//                    )
//                }
//
//            case .route:
//                NavigationStack {
//                    RouteScreen(vm: RouteViewModel(service: container.taskService))
//                }
//
//            case .ads:
//                NavigationStack {
//                    MyAdsScreen()
//                }
//
//            case .chats:
//                NavigationStack {
//                    ChatsScreen(vm: ChatsViewModel(service: container.chatService))
//                }
//
//            case .profile:
//                NavigationStack {
//                    ProfileScreen(vm: ProfileViewModel(service: container.profileService))
//                }
//        }
//    }
//}

//import SwiftUI
//
//////// Главный контейнер приложения с кастомным «стеклянным» TabBar
//struct RootView: View {
//    @EnvironmentObject var container: AppContainer

```

```

//
//    /// Текущая выбранная вкладка
//    @State private var selection: AppTab = .map
//
//    /// Пример бейджа на профиле (красный кружок «2»)
//    private let badges: [AppTab: Int] = [.profile: 2]
//
//    var body: some View {
//        ZStack(alignment: .bottom) {
//            // Контент под TabBar – карта и остальные экраны
//            tabContent
//                .ignoresSafeArea() // фон двигается под таббаром
//
//            // Кастомный «Liquid Glass» TabBar
//            LiquidTabBar(selection: $selection, badges: badges)
//                .padding(.horizontal, 16)
//                .padding(.bottom, 4) // бар чуть выше, не «прилипает» к home-индикатору
//        }
//        .background(Theme.ColorToken.milk) // фон, если вдруг нет карты
//        .tint(Theme.ColorToken.turquoise)
//    }
//
//    // MARK: - Контент для каждой вкладки
//
//    @ViewBuilder
//    private var tabContent: some View {
//        switch selection {
//        case .map:
//            NavigationStack {
//                MapScreen(
//                    vm: .init(
//                        service: container.taskService,
//                        searchService: AddressSearchService()
//                    ), mapMode: .real
//                )
//            }
//
//        case .route:
//            NavigationStack {
//                RouteScreen(vm: .init(service: container.taskService))
//            }
//
//        case .ads:
//            NavigationStack {
//                MyAdsScreen()
//            }
//
//        case .chats:
//            NavigationStack {
//                ChatsScreen(vm: .init(service: container.chatService))
//            }
//
//        case .profile:
//            NavigationStack {
//                ProfileScreen(vm: .init(service: container.profileService))
//            }
//        }
//    }
//}

//import SwiftUI
//
//struct RootView: View {
//    @EnvironmentObject var container: AppContainer
//
//    var body: some View {

```

```

//      if container.session.isAuthenticated {
//          MainTabView()
//      } else {
//          AuthScreen()
//      }
//  }
//}
//
//private struct MainTabView: View {
//    @EnvironmentObject var container: AppContainer
//    @State private var selectedTab = 0
//
//    var body: some View {
//        TabView(selection: $selectedTab) {
//            NavigationStack {
//                MapScreen(vm: .init(service: container.taskService))
//            }
//            .tabItem { Label("Капра", systemImage: "map") }
//            .tag(0)
//
//            NavigationStack {
//                RouteScreen(vm: .init(service: container.taskService))
//            }
//            .tabItem { Label("Маршрут", systemImage: "point.topleft.down.curvedto.point.bottomright.up") }
//            .tag(1)
//
//            NavigationStack {
//                ChatsScreen(vm: .init(service: container.chatService))
//            }
//            .tabItem { Label("Чат", systemImage: "bubble.left.and.bubble.right") }
//            .tag(2)
//
//            NavigationStack {
//                ProfileScreen(vm: .init(service: container.profileService))
//                .toolbar {
//                    Button("Logout") { container.session.logout() }
//                }
//            }
//            .tabItem { Label("Профиль", systemImage: "person.circle") }
//            .tag(3)
//        }
//        .tint(Theme.ColorToken.turquoise)
//    }
//}

```

```

// ===== File 62/70: DelegationApp/Features/Route/ViewModel/RouteViewModel.swift =====
import Foundation

```

```

final class RouteViewModel: ObservableObject {
    @Published var pointA: String = "Пушкинская площадь"
    @Published var pointB: String = "Станция МЦК Площадь Гагарина"
    @Published var time: String = "17:00"
    @Published var tasks: [TaskItem] = []

    private let service: TaskService
    init(service: TaskService) {
        self.service = service
        self.tasks = service.loadRouteTasks()
    }
}

```

```

// ===== File 63/70: DelegationApp/Features/Untitled.swift =====
//import SwiftUI
//

```

```
//extension Color {
//    static func hex(_ hex: String) -> Color {
//        let hex = hex.trimmingCharacters(in: CharacterSet.alphanumerics.inverted)
//        var int: UInt64 = 0; Scanner(string: hex).scanHexInt64(&int)
//        let a, r, g, b: UInt64
//        switch hex.count {
//        case 3: (a,r,g,b) = (255, (int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) * 17)
//        case 6: (a,r,g,b) = (255, int >> 16, int >> 8 & 0xFF, int & 0xFF)
//        case 8: (a,r,g,b) = (int >> 24, int >> 16 & 0xFF, int >> 8 & 0xFF, int & 0xFF)
//        default:(a,r,g,b) = (255,0,0,0)
//        }
//        return Color(.sRGB,
//            red: Double(r)/255, green: Double(g)/255,
//            blue: Double(b)/255, opacity: Double(a)/255)
//    }
//}
```

```
// ===== File 64/70: DelegationApp/Others/YandexMapConfigurator.swift =====
import YandexMapsMobile
```

```
/// Централизованная настройка Yandex MapKit.
enum YandexMapConfigurator {
    private static var isConfigured = false

    static func configureIfNeeded() {
        // В SwiftUI Preview вообще не инициализируем SDK.
        if RuntimeEnvironment.isPreview { return }
        guard !isConfigured else { return }

        // сюда твой реальный ключ
        YMKMapKit.setApiKey("df3f9145-2080-42b7-9b91-b879c34236bb")
        YMKMapKit.sharedInstance()
        isConfigured = true
    }
}
```

```
// ===== File 65/70: DelegationApp/RootView.swift =====
//import SwiftUI
//
//struct RootView: View {
//    /// DI-контейнер с сервисами
//    @EnvironmentObject var container: AppContainer
//    @State private var selected = 0
//
//    var body: some View {
//        TabView(selection: $selected) {
//
//            // ? Вкладка КАРТА
//            NavigationStack {
//                MapScreen(vm: MapViewModel(service: container.taskService))
//            }
//            .tabItem {
//                Label("Карта", systemImage: "map")
//            }
//            .tag(0)
//
//            // ? Вкладка МАРШРУТ
//            NavigationStack {
//                RouteScreen(vm: RouteViewModel(service: container.taskService))
//            }
//            .tabItem {
//                Label("Маршрут", systemImage: "point.topleft.down.curvedto.point.bottomright.up")
//            }
//            .tag(1)
//        }
//    }
//}
```



```

//
//      // ? Вкладка ЧАТЫ
//      NavigationStack {
//          ChatsScreen(vm: container.chatService)
//      }
//      .tabItem {
//          Label("Чаты", systemImage: "bubble.left.and.bubble.right")
//      }
//      .tag(2)
//
//      // ? Вкладка ПРОФИЛЬ
//      NavigationStack {
//          ProfileScreen(vm: ProfileViewModel(service: container.profileService))
//      }
//      .tabItem {
//          Label("Профиль", systemImage: "person")
//      }
//      .tag(3)
//      }
//      .tint(Theme.ColorToken.turquoise)
//      .background(Theme.ColorToken.milk)
//  }
//}
////
////#Preview {
////    RootView()
////    .environmentObject(AppContainer.preview)
////}

```

```

// ===== File 66/70: iCuno test/ContentView.swift =====
//
// ContentView.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```

// ===== File 67/70: iCuno test/iCuno_testApp.swift =====
//
// iCuno_testApp.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```

//import SwiftUI
//
//@main
//struct iCuno_testApp: App {
//    var body: some Scene {
//        WindowGroup {
//            ContentView()
//        }
//    }
//}

```

```

// ===== File 68/70: iCuno testTests/iCuno_testTests.swift =====
//

```

```

// iCuno_testTests.swift
// iCuno testTests
//
// Created by maftuna murtazaeva on 07.11.2025.
//

import XCTest
@testable import iCuno_test

final class iCuno_testTests: XCTestCase {

    override func setUpWithError() throws {
        // Put setup code here. This method is called before the invocation of each test method in the class.
    }

    override func tearDownWithError() throws {
        // Put teardown code here. This method is called after the invocation of each test method in the
class.
    }

    func testExample() throws {
        // This is an example of a functional test case.
        // Use XCTAssert and related functions to verify your tests produce the correct results.
        // Any test you write for XCTest can be annotated as throws and async.
        // Mark your test throws to produce an unexpected failure when your test encounters an uncaught error.
        // Mark your test async to allow awaiting for asynchronous code to complete. Check the results with
assertions afterwards.
    }

    func testPerformanceExample() throws {
        // This is an example of a performance test case.
        self.measure {
            // Put the code you want to measure the time of here.
        }
    }

}

```

```

// ===== File 69/70: iCuno testUITests/iCuno_testUITests.swift =====
//
// iCuno_testUITests.swift
// iCuno testUITests
//
// Created by maftuna murtazaeva on 07.11.2025.
//

import XCTest

final class iCuno_testUITests: XCTestCase {

    override func setUpWithError() throws {
        // Put setup code here. This method is called before the invocation of each test method in the class.

        // In UI tests it is usually best to stop immediately when a failure occurs.
        continueAfterFailure = false

        // In UI tests it's important to set the initial state - such as interface orientation - required for
your tests before they run. The setUp method is a good place to do this.
    }

    override func tearDownWithError() throws {
        // Put teardown code here. This method is called after the invocation of each test method in the
class.
    }

    @MainActor

```

```

func testExample() throws {
    // UI tests must launch the application that they test.
    let app = XCUIApplication()
    app.launch()

    // Use XCTAssert and related functions to verify your tests produce the correct results.
}

@MainActor
func testLaunchPerformance() throws {
    // This measures how long it takes to launch your application.
    measure(metrics: [XCTApplicationLaunchMetric()]) {
        XCUIApplication().launch()
    }
}
}

```

```

// ===== File 70/70: iCuno testUITests/iCuno_testUITestsLaunchTests.swift =====
//
// iCuno_testUITestsLaunchTests.swift
// iCuno testUITests
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```

import XCTest

```

```

final class iCuno_testUITestsLaunchTests: XCTestCase {

    override class var runsForEachTargetApplicationUIConfiguration: Bool {
        true
    }

    override func setUpWithError() throws {
        continueAfterFailure = false
    }

    @MainActor
    func testLaunch() throws {
        let app = XCUIApplication()
        app.launch()

        // Insert steps here to perform after app launch but before taking a screenshot,
        // such as logging into a test account or navigating somewhere in the app

        let attachment = XCTAttachment(screenshot: app.screenshot())
        attachment.name = "Launch Screen"
        attachment.lifetime = .keepAlways
        add(attachment)
    }
}

```