

```
// ===== File 1/65: 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 preview = AppContainer(
```

```
        taskService: MockTaskService(),
```

```
        chatService: MockChatService(),
```

```
        profileService: MockProfileService(),
```

```
        announcementService: MockAnnouncementService(),
```

```
        authService: NetworkAuthService()
```

```
    )
```

```
}
```

```
// ===== File 2/65: DelegationApp/App/AppRouter.swift =====
```

```
import SwiftUI
```

```
/// Опциональный "роутер", чтобы можно было использовать как корневой View в превью/в будущем.
```

```
struct AppRouter: View {
```

```
    @EnvironmentObject var container: AppContainer
```

```
    var body: some View {
```

```
        RootView()
```

```
        .environmentObject(container)
```

```
        .environmentObject(container.session)
```

```
    }
```

```
}
```

```
/// Корневой экран приложения: проверка сессии → авторизация → основной таббар.
```

```
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
        }
        // Кастомный TabBar снизу так, чтобы контент не уезжал под него
        .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,
                    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 3/65: DelegationApp/App/DelegationApp.swift =====

```

import SwiftUI
import YandexMapsMobile

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

    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 4/65: DelegationApp/App/YandexMapView.swift =====

```

import SwiftUI
import YandexMapsMobile

/// Обёртка над YMKMapView для использования в SwiftUI.
///
/// Важно: в превью мы не создаём нативную карту вообще,
/// чтобы не падал SwiftUI Preview.
struct YandexMapView: UIViewRepresentable {

    /// Координата центра карты.
    @Binding var centerPoint: YMKPoint?

    final class Coordinator {
        var mapView: YMKMapView?
        var placemark: YMKPlacemarkMapObject?
    }

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

    func makeUIView(context: Context) -> UIView {
        // Контейнер, в который при обычном запуске добавим YMKMapView.
        let container = UIView()
        container.backgroundColor = .clear
    }
}

```

```

//      // В превью – ничего не добавляем, просто пустой UIView.
//      guard !RuntimeEnvironment.isPreview else {
//          return container
//      }

// В обычном запуске инициализируем SDK и карту.
YandexMapConfigurator.configureIfNeeded()

let mapView = YMKMapView(frame: .zero)
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

// Стартовая точка.
let startPoint = centerPoint ?? YMKPoint(
    latitude: 55.751244,
    longitude: 37.618423
)
updateMap(on: mapView!, coordinator: context.coordinator, to: startPoint)

return container
}

func updateUIView(_ uiView: UIView, context: Context) {
    guard
//      !RuntimeEnvironment.isPreview,
        let mapView = context.coordinator.mapView,
        let point = centerPoint
    else { return }

    updateMap(on: mapView, coordinator: context.coordinator, to: point)
}

// MARK: - Internal helpers

private func updateMap(
    on mapView: YMKMapView,
    coordinator: Coordinator,
    to point: YMKPoint
) {
    let map = mapView.mapWindow.map
    let position = YMKCameraPosition(
        target: point,
        zoom: 15,
        azimuth: 0,
        tilt: 0
    )
    let animation = YMKAnimation(type: .smooth, duration: 1.0)
    map.move(with: position, animation: animation, cameraCallback: nil)

    let mapObjects = map.mapObjects
    if let oldPlacemark = coordinator.placemark {
        mapObjects.remove(with: oldPlacemark)
    }
    let placemark = mapObjects.addPlacemark(with: point)
    coordinator.placemark = placemark
}
}

```

```

// ===== File 5/65: 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 6/65: 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 7/65: 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» пузырёк 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 8/65: 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 9/65: 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.. $\max$ , id: \.self) { idx in
            let filled = rating >= Double(idx + 1) - 0.001
            Image(systemName: filled ? "star.fill" : "star")
                .foregroundColor(filled ? Theme.ColorToken.peach : Theme.ColorToken.textSecondary)
        }
    }
}
}

```

```

// ===== File 10/65: 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,
            let url = URL(string: s),
            !s.isEmpty {
            return url
        }

        #if targetEnvironment(simulator)
            return URL(string: "http://127.0.0.1:8000")!
        #else
            // !!! Поменяй на IP твоего Mac (в той же сети Wi-Fi)
            return URL(string: "http://192.168.1.10:8000")!
        #endif
    }()
}

```

```
}
```

```

// ===== File 11/65: 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 12/65: 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
    }
}

// ===== File 13/65: 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 14/65: 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 15/65: 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 16/65: 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 17/65: 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 18/65: 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 19/65: 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 -> AnnouncementDT0
    func myAnnouncements(token: String) async throws -> [AnnouncementDT0]
}
```

```
final class NetworkAnnouncementService: AnnouncementService {
    private let api: APIClient

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

    func createAnnouncement(token: String, request: CreateAnnouncementRequest) async throws -> AnnouncementDT0 {
        try await api.request(.createAnnouncement, body: request, token: token)
    }

    func myAnnouncements(token: String) async throws -> [AnnouncementDT0] {
        try await api.request(.myAnnouncements, token: token)
    }
}
```

```
// ===== File 20/65: 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 21/65: 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 22/65: 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 23/65: 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 24/65: 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 25/65: 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 26/65: 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
        }
    }

    // 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)
}
}

```

```

// ===== File 27/65: 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
    ) {
        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
        defer { 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)
    }
}

// ===== File 28/65: 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

    // Без ведущего "/" – безопасно для appendingPathComponent
    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"
        }
    }
}

var method: HTTPMethod {

```

```

        switch self {
        case .register, .login, .createAnnouncement:
            return .POST
        case .me, .myAnnouncements:
            return .GET
        }
    }

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

// ===== File 29/65: 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 30/65: 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()
        }

        let result = try await group.next()!
        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)
    }
}

```



```

//
// TaskService.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

import Foundation

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

// ===== File 32/65: 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 33/65: DelegationApp/Core/Utils/Extentions/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 34/65: 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 func 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 35/65: DelegationApp/Features/Ads/AdView/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 36/65: 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: (AnnouncementDTO) -> Void

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

    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 {
            let created = await vm.submit(draft: draft)
            if let created {
                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)
}
}

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 37/65: 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

    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) {
    goAudience = true
}
}
.navigationBarBackButtonHidden(true)
.toolbar { backToolbar }
.navigationDestination(isPresented: $goAudience) {
    AdAudienceScreen(
        draft: draft,
        accent: accent,
        onFinish: onFinish
    )
}
}

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

// ===== File 38/65: 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 39/65: 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,
        onCreate: @escaping (AnnouncementDTO) -> Void
    ) {
        _vm = StateObject(wrappedValue: CreateAdFlowViewModel(service: service, session: session))
        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 40/65: 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 showValidationAlert: Bool = false
    @State private var validationText: String = ""

    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 и видео (позже)",
    subtitle: "Пока без загрузки. Оставлено как точка расширения.",
    accent: accent
) {
    Text("Подключишь PhotosPicker + upload в Worker/Service.")
        .font(.system(size: 14, weight: .medium))
        .foregroundColor(.secondary)
}

}
.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) {}
}

```



```

        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 и видео (позже)",
    subtitle: "Пока без загрузки. Оставлено как точка расширения.",
    accent: accent
) {
    Text("Подключишь PhotosPicker + upload в Worker/Service.")
        .font(.system(size: 14, weight: .medium))
        .foregroundColor(.secondary)
}
}
.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: "Продолжить", accent: accent) {
        let res = draft.validateForCurrentStep()
        if res.ok {
            goContact = true
        } else {
            validationText = res.message
            showValidationAlert = true
        }
    }
}
}
}

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 42/65: 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 43/65: DelegationApp/Features/Ads/Model/CreateAdDraft.swift =====
//
// CreateAdDraft.swift
// iCuno test
//
// Created by maftuna murtazaeva on 18.02.2026.
//

import Foundation

@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 startDate: Date = .now
@Published var hasEndTime: Bool = false
@Published var endDate: Date = .now

@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 = ""

// 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

// MARK: - Validation
func validateForCurrentStep() -> (ok: Bool, message: String) {
    guard let category else { return (false, "Выберите категорию") }
    guard !title.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty else {
        return (false, "Заполните название объявления")
    }
    switch category {
    case .delivery:
        if pickupAddress.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty {

```

```

        return (false, "Укажите адрес забора")
    }
    if dropoffAddress.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty {
        return (false, "Укажите адрес доставки")
    }
    case .help:
        if helpAddress.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty {
            return (false, "Укажите адрес")
        }
    }
    return (true, "")
}

// MARK: - Mapping to request
func toCreateRequest() -> CreateAnnouncementRequest {
    let iso = ISO8601DateFormatter()

    var data: [String: JSONValue] = [:]

    data["category"] = .string(category?.rawValue ?? "")
    data["budget"] = budget.isEmpty ? .null : .string(budget)

    data["contact_name"] = contactName.isEmpty ? .null : .string(contactName)
    data["contact_phone"] = contactPhone.isEmpty ? .null : .string(contactPhone)
    data["contact_method"] = .string(contactMethod.rawValue)

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

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

    if let category {
        switch category {
        case .delivery:
            data["pickup_address"] = .string(pickupAddress)
            data["dropoff_address"] = .string(dropoffAddress)
            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"] = cargoLength.isEmpty ? .null : .string(cargoLength)
            data["cargo_width"] = cargoWidth.isEmpty ? .null : .string(cargoWidth)
            data["cargo_height"] = cargoHeight.isEmpty ? .null : .string(cargoHeight)

            data["floor"] = floor.isEmpty ? .null : .string(floor)
            data["has_elevator"] = .bool(hasElevator)
            data["need_loader"] = .bool(needLoader)

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

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

```

```
// ===== File 44/65: 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?

    private let service: AnnouncementService
    private let session: SessionStore

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

    func submit(draft: CreateAdDraft) async -> AnnouncementDTO? {
        guard let token = session.token else {
            errorText = "Нет токена сессии"
            return nil
        }

        isSubmitting = true
        defer { isSubmitting = false }

        do {
            let req = draft.toCreateRequest()
            let created = try await service.createAnnouncement(token: token, request: req)
            return created
        } catch {
            errorText = error.localizedDescription
            return nil
        }
    }
}
```

```
// ===== File 45/65: 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: [AnnouncementDTO] = []

    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 46/65: 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 47/65: 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:
                            .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)
.navigationBarTitle("Сообщения")
}
}

// #Preview {
//     ChatsScreen(
//         vm: ChatsViewModel(service: MockChatService())
//     )
// }

// ===== File 48/65: 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 49/65: DelegationApp/Features/Chats.swift =====
//
// Chats.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

// ===== File 50/65: DelegationApp/Features/Map/MapScreen/MapScreen.swift =====
import SwiftUI

// MARK: - Экран "Карта"

struct MapScreen: View {
    @StateObject private var vm: MapViewModel
    @State private var showCreate = false

    /// Режим отображения карты (настоящая карта / плейсхолдер).
    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) {
            // Задний слой – Яндекс-карта на весь экран
            mapArea
                .background(Color.green)
                .cornerRadius(15)

            // Верхний слой – поиск + ошибка + чипсы
            VStack(spacing: 5) {
                // небольшой отступ от статус-бара
                Spacer().frame(height: 50)

                searchBar
                    .background(Color.red)
                    .background(Color.clear)
                    .cornerRadius(15)
                errorLabel
                    .background(Color.clear)
                    .cornerRadius(15)
                chipsRow
                    .background(Color.clear)
                    .cornerRadius(15)

                Spacer()
            }
            .padding(.horizontal, 16)
            .padding(.top, 8)
            .ignoresSafeArea()
        }
    }

    // MARK: - Сабвью

    /// Поисковая строка.
    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)
    // карточка на «стеклянном» фоне поверх карты
    .background(.ultraThinMaterial)
    .clipShape(RoundedRectangle(cornerRadius: 16, style: .continuous))
    .softCardShadow()
}

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

/// Горизонтальный список фильтров-чипсов.
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)
                            }
                        }
                    )
                )
            }
        }
        .padding(0)
    }
    // важное изменение: НЕТ .background(Color.green)
    // фон прозрачный → чипсы "висят" над картой
}

/// Слой с картой.
private var mapArea: some View {
    MapCanvasView(centerPoint: $vm.centerPoint, mode: mapMode)
        .ignoresSafeArea(edges: .top) // карта под всей версткой и под системными бары
}
}

```

```

// ===== File 51/65: DelegationApp/Features/Map/MapViewModel/MapViewModel.swift =====
//
// MapScreen.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```

import SwiftUI
import YandexMapsMobile
import Foundation

```

```
// MARK: - Режимы отображения карты

/// Режим отображения карты.
/// - `.real` — настоящая карта Яндекса.
/// - `.placeholder` — заглушка (для превью / когда карту рендерить нельзя).
enum MapDisplayMode {
    case real
    case placeholder
}

/// Централизованная конфигурация карты.
///
/// Здесь одна точка, где ты решаешь, что именно показывается:
/// настоящая карта или заглушка.
enum MapDisplayConfig {

    /// Основная функция, которая решает, в каком режиме показывать карту.
    ///
    /// Можно поменять реализацию, добавить флаги DEBUG/RELEASE,
    /// удалённые конфиги и т.д.
    static func defaultMode() -> MapDisplayMode {
        // Пример:
        // В DEBUG можно держать заглушку, чтобы карта не мешала верстать UI.
        // В RELEASE — реальная карта.
        #if DEBUG
            return .real
        #else
            return .real
        #endif
    }
}

// MARK: - Холст карты

/// Вью, которая отвечает ТОЛЬКО за "холст карты":
/// она выбирает — показать реальную карту или заглушку.
///
/// Важно: логика поиска/маршрутов/заданий работает независимо от этого выбора,
/// потому что она живёт в `MapViewModel` и сервисах.
/// Обёртка над картой/заглушкой.
/// Она не знает ни про фильтры, ни про поиск — только про то, ЧТО рисовать.
struct MapCanvasView: View {

    /// Точка, на которую центрируется карта.
    @Binding var centerPoint: YMKPoint?

    /// Текущий режим отображения (реальная карта / заглушка).
    let mode: MapDisplayMode

    var body: some View {
        Group {
            switch mode {
            case .real:
                // Живая карта Яндекса.
                YandexMapView(centerPoint: $centerPoint)

            case .placeholder:
                // Заглушка для превью / работы над UI.
                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)
            }
        )
    }
}
}
}
}

```

/// Простая заглушка вместо карты.

```

struct MapPlaceholderView: View {
    var body: some View {
        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)
                }
            )
    }
}

```

// MARK: - ViewModel карты

/// ViewModel для экрана карты: фильтры, задачи, поиск адреса, центр карты.

```

final class MapViewModel: ObservableObject {

    // MARK: - Фильтры (чипы)

    @Published var chips: [String] = [
        "Купить", "Доставить", "Забрать",
        "Помочь", "Перенести", "Другое"
    ]

    @Published var selected: Set<String> = []

    // MARK: - Задачи рядом

    @Published var tasks: [TaskItem] = []

    // MARK: - Поиск и карта

    /// Текст в поле поиска адреса.
    @Published var searchText: String = ""

    /// Текущая точка, на которую центрируется карта.
    @Published var centerPoint: YMKPoint?

    /// Сообщение об ошибке (например, "Ничего не найдено").
    @Published var errorMessage: String?

    private let service: TaskService
    private let searchService: AddressSearchService

    init(
        service: TaskService,
        searchService: AddressSearchService = AddressSearchService()
    ) {
        self.service = service
        self.searchService = searchService

        // Загружаем задачи поблизости (как и раньше).
    }
}

```

```

self.tasks = service.loadNearbyTasks()

// Стартовая точка карты – Москва (можешь поменять на Самарканд).
self.centerPoint = YMKPoint(
    latitude: 55.751244,
    longitude: 37.618423
)
}

// MARK: - Логика фильтров

func toggle(_ chip: String) {
    if selected.contains(chip) {
        selected.remove(chip)
    } else {
        selected.insert(chip)
    }
}

// MARK: - Поиск адреса

/// Выполнить поиск по адресу и сдвинуть карту.
///
/// Важно: этот код работает даже тогда, когда на UI показывается заглушка.
/// Просто не будет рендериться сама карта, но `centerPoint` обновится,
/// и при включении настоящей карты ты сразу увидишь правильную точку.
func performSearch() {
    let query = searchText.trimmingCharacters(in: .whitespacesAndNewlines)
    guard !query.isEmpty else {
        // Пустой запрос – просто сбрасываем ошибку.
        errorMessage = nil
        return
    }

    searchService.searchAddress(query) { [weak self] point in
        DispatchQueue.main.async {
            guard let self else { return }

            if let point {
                // Успех: центрируем карту в этой точке.
                self.centerPoint = point
                self.errorMessage = nil
            } else {
                // Ничего не нашли.
                self.errorMessage = "Ничего не найдено"
            }
        }
    }
}

}

}

// ===== File 52/65: 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("О себе")
//         Text("Напиши пару слов о себе, чтобы заинтересовать собеседника")
//             .font(.footnote)
//         TextField("О 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)
//     }
// }

```



```

private var header: some View {
    VStack(alignment: .leading, spacing: 12) {
        HStack(alignment: .center, spacing: 14) {
            Circle().fill(Theme.ColorToken.milk).frame(width: 56, height: 56)
                .overlay(Image(systemName: "person.fill").font(.system(size:
26))).foregroundStyle(Theme.ColorToken.turquoise))

            VStack(alignment: .leading, spacing: 6) {
                Text(vm.profile.name).font(.system(size: 20, weight: .semibold))
                Text(vm.profile.phone).foregroundStyle(Theme.ColorToken.textSecondary)
                    .font(.system(size: 14))
            }
            Spacer()
            Text("ID")
                .font(.system(size: 13, weight: .bold))
                .padding(.vertical, 6).padding(.horizontal, 10)
                .background(RoundedRectangle(cornerRadius: 10).fill(Theme.ColorToken.peach.opacity(0.3)))
        }

        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)).foregroundColor(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 54/65: 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 55/65: DelegationApp/Features/Route/View/RouteScreen.swift =====
import SwiftUI

enum PreviewData {
    static let container = AppContainer.preview

    static let chatsVM = ChatsViewModel(service: MockChatService())
    static let mapVM = MapViewModel(service: MockTaskService())
    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 56/65: 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, движется 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.isAuthorized {
//            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("Карта", 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 57/65: 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 58/65: 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 59/65: 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 60/65: 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 61/65: iCuno test/ContentView.swift =====
//
// ContentView.swift
// iCuno test
//
// Created by maftuna murtazaeva on 07.11.2025.
//

```

```

// ===== File 62/65: 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 63/65: 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 64/65: 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 65/65: 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)
    }
}

```