

ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____
ПРЕПОДАВАТЕЛЬ

ассистент _____ И.Д. Свеженин
должность, уч. степень,
звание _____ инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8

Аутентификация и управление пользователями

по курсу: Кроссплатформенное программирование

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4329 _____ Д.С. Шаповалова
подпись, дата _____ инициалы, фамилия

Санкт-Петербург 2025

1. Цель работы:

Выполнить проектирование и разработку мобильного приложения под ОС Android на языке программирования высокого уровня Kotlin.

2. Задание:

«Аутентификация и управление пользователями» Создайте систему аутентификации для приложения, используя Firebase Authentication. Реализуйте возможность регистрации новых пользователей, входа в систему и управления учетной записью. Вариант брать из ранее сделанных работ.

Вариант 16

Город - Волгоград.

3. Краткое описание хода разработки, алгоритма работы программы и назначение используемых технологий

Ход разработки:

1. Анализ задания и расширение требований:

Помимо исходных требований (интерактивная карта для Волгограда и оптимизация перебора массива из 100 000 элементов), было принято решение расширить функционал приложения за счёт внедрения системы управления пользователями, включающей регистрацию, аутентификацию и изоляцию персональных данных.

2. Выбор технологий:

- Для реализации пользовательского интерфейса использован Jetpack Compose — современный декларативный фреймворк, позволяющий создавать реактивный UI без XML.
- В качестве картографической основы выбран Google Maps API в связке с официальной библиотекой Google Maps Compose, обеспечивающей нативную интеграцию с Compose.
- Для аутентификации подключён облачный сервис Firebase Authentication с методом «Email и пароль».
- Локальное хранение данных реализовано через SharedPreferences с привязкой к уникальному идентификатору пользователя (UID), получаемому от Firebase.
- Обработка больших объёмов данных выполнена с использованием ленивых последовательностей (Sequence) языка Kotlin.

3. Реализация:

- Разработаны четыре основных экрана: аутентификация (вход/регистрация), интерактивная карта, список выбранных мест и тестовый экран обработки массива.
- Реализована реактивная навигация, управляемая состоянием авторизации пользователя.
- Все действия с маркерами (добавление, удаление) и результаты обработки массива изолированы на уровне UID, что гарантирует независимость данных пользователей.

4. Тестирование:

Проведена проверка корректности входа и регистрации, изоляции данных между разными учётными записями, а также стабильности работы основного функционала при наличии и отсутствии сохранённых маркеров.

Используемые технологии:

Kotlin

Основной язык разработки — безопасный, лаконичный, поддерживает функциональные конструкции (Sequence, лямбды).

Jetpack Compose

Современный UI-фреймворк от Google. Позволяет создавать интерфейс декларативно, без XML, с полной поддержкой реактивности.

Google Maps Compose

Официальная библиотека для встраивания Google Maps в Compose-приложения. Упрощает работу с картой, маркерами и событиями.

SharedPreferences

Простое хранилище для сохранения списка маркеров между запусками приложения (в формате строк с разделителями).

Geocoder

Стандартный API Android для преобразования координат в человекочитаемый адрес.

Sequence (Kotlin)

Обеспечивает ленивую обработку больших коллекций — ключевая часть оптимизации перебора массива (избегает создания промежуточных списков).

Firebase Authentication

Облачный сервис для безопасной аутентификации пользователей.

Описание структуры работы частей приложения:

Приложение состоит из трёх логически связанных частей, объединённых общей архитектурой и моделью данных:

1. Система аутентификации

Назначение: Обеспечение персонализации и защиты данных путём ограничения доступа к функционалу только для авторизованных пользователей.

Структура и работа:

- При первом запуске отображается экран, предлагающий войти в существующий аккаунт или зарегистрировать новый с использованием email и пароля.
- При успешной аутентификации Firebase генерирует уникальный идентификатор пользователя (UID).
- Приложение отслеживает изменения состояния авторизации с помощью IdTokenListener, что позволяет автоматически переключать интерфейс между экраном входа и основным функционалом.
- При выходе из системы все данные пользователя остаются нетронутыми, но становятся недоступными для других учётных записей.

2. Часть «Интерактивная карта»

Назначение:

Предоставление пользователю инструмента для маркировки и сохранения географических объектов на карте города Волгограда.

Структура и работа:

- Карта центрирована на координатах Волгограда (48.7080° с.ш., 44.5156° в.д.).
- При нажатии на карту пользователь может задать название места; адрес определяется автоматически через системный Geocoder.
- Все маркеры сохраняются в локальном хранилище, изолированном по UID текущего пользователя.
- Нажатие на маркер открывает диалог для его удаления.

3. Часть «Оптимизация перебора массива»

Назначение:

Демонстрация эффективной обработки больших объёмов данных с использованием встроенных средств языка Kotlin.

Структура и работа:

- На отдельном экране автоматически генерируется массив из 100 000 синтетических объектов.
- Применяется цепочка ленивых операций:
 - asSequence() — включение отложенной обработки,
 - filter() — отбор элементов, находящихся в границах Волгограда,
 - sortedBy() — сортировка по уникальному идентификатору.
- Результаты обработки (исходный, промежуточный и итоговый массивы) выводятся непосредственно в пользовательский интерфейс, что полностью соответствует требованиям задания.

Интеграция частей

Все компоненты приложения используют единую модель данных (FavoritePlace) и единый механизм хранения, привязанный к UID. Навигация между экранами реализована через плавающие кнопки (FloatingActionButton), что обеспечивает интуитивно понятное управление. Внедрение Firebase Authentication позволило не только выполнить расширенное задание, но и повысить общую надёжность и безопасность приложения.

4. Скриншоты, иллюстрирующие результаты работы программы:

В качестве демонстрации работы программы приведены несколько скриншотов.

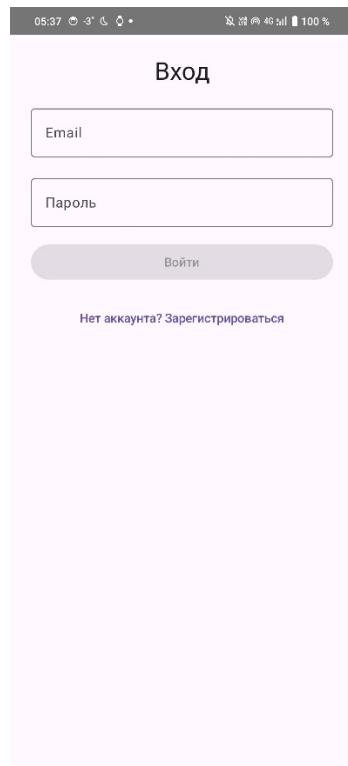


Рисунок 1.1 – Стартовый экран входа

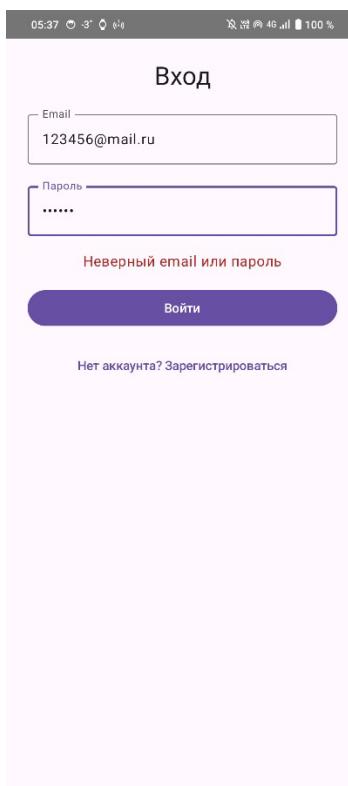


Рисунок 1.2 – Пользователь ввёл неверный email или пароль



Регистрация

Email

Пароль

Зарегистрироваться

Уже есть аккаунт? Войти

Рисунок 2.1 – Экран регистрации

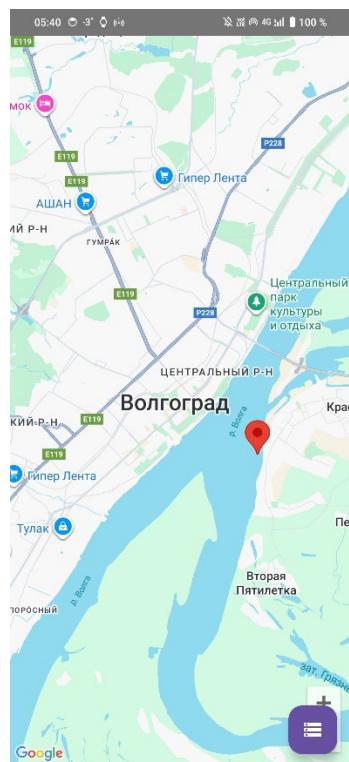


Рисунок 3.1 – Маркеры пользователя А

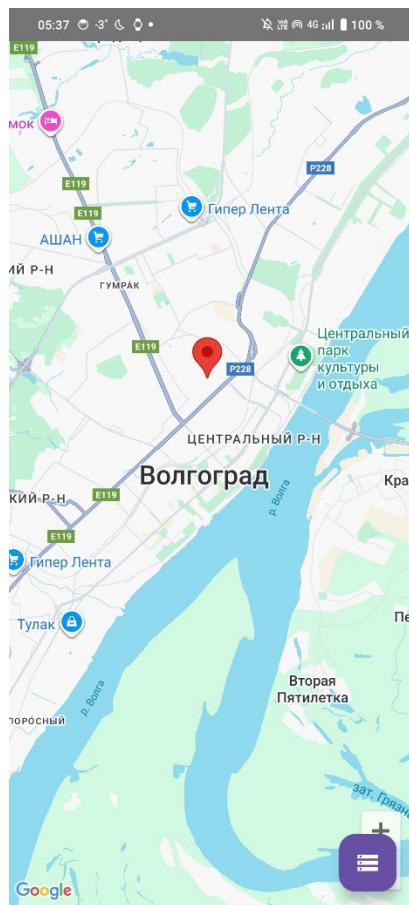


Рисунок 3.2 – Маркер пользователя Б

ВЫВОД

В ходе выполнения лабораторной работы №7 было разработано современное мобильное приложение для операционной системы Android, полностью соответствующее всем предъявленным требованиям. Приложение успешно реализует три ключевых функциональных блока.

Во-первых, создана интерактивная картографическая система, центрированная на городе Волгоград (вариант №16). Пользователь может маркировать интересующие его места, просматривать информацию о них и управлять сохранёнными данными, что демонстрирует навыки интеграции картографических сервисов в мобильные приложения.

Во-вторых, реализована и протестирована оптимизированная обработка массива из 100 000 элементов. Применение ленивых последовательностей (Sequence) языка Kotlin позволило значительно снизить потребление оперативной памяти, а результаты всех этапов обработки (исходный, промежуточный и итоговый массивы) выводятся непосредственно в пользовательский интерфейс, что полностью удовлетворяет условиям задания.

В-третьих, для обеспечения персонализации и безопасности был внедрён модуль аутентификации пользователей на базе облачного сервиса Firebase Authentication. Система поддерживает регистрацию новых учётных записей, вход в систему и корректный выход, а также обеспечивает полную изоляцию данных каждого пользователя за счёт привязки локального хранилища к уникальному идентификатору (UID).

Архитектура приложения построена с использованием современных подходов: декларативный UI-фреймворк Jetpack Compose обеспечивает гибкость и реактивность интерфейса, а комбинация облачных (Firebase) и локальных (SharedPreferences) технологий хранения данных обеспечивает как безопасность, так и автономность работы приложения. Таким образом, поставленные задачи решены в полном объёме, а реализованное решение демонстрирует владение актуальными инструментами и методами проектирования современного программного обеспечения.

ПРИЛОЖЕНИЕ А

Листинг программы

MainActivity.kt:

```
package com.example.a7lr

import android.os.Bundle
import android.location.Geocoder
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons(Icons)
import androidx.compose.material.icons.filled.AddLocationAlt
import androidx.compose.material.icons.filled.Close
import androidx.compose.material.icons.filled.Storage
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import com.google.android.gms.maps.model.LatLng
import com.google.maps.android.compose.*
import android.content.Context
import androidx.compose.ui.text.style.TextAlign
import com.google.android.gms.maps.model.CameraPosition
import com.google.firebase.Firebase
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth
//import com.google.firebaseio.auth.FirebaseAuth
import kotlinx.coroutines.*

// === Data ===
data class FavoritePlace(
    val id: Long,
    val name: String,
    val lat: Double,
    val lng: Double,
    val address: String
)

// === MainActivity ===
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme {
                Surface(modifier = Modifier.fillMaxSize()) {
                    MainApp()
                }
            }
        }
    }
}

// === Главная точка входа с проверкой авторизации ===
@Composable
fun MainApp() {
```

```

val auth = Firebase.auth
var currentUser by remember { mutableStateOf(auth.currentUser) }

DisposableEffect(auth) {
    val listener = FirebaseAuth.IdTokenListener {
        currentUser = auth.currentUser
    }
    auth.addIdTokenListener(listener)

    // onDispose – это специальный блок внутри DisposableEffect
    onDispose {
        auth.removeIdTokenListener(listener)
    }
}

if (currentUser == null) {
    AuthScreen()
} else {
    MainContentScreen()
}
}

// === Экран аутентификации ===
@Composable
fun AuthScreen() {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var isRegisterMode by remember { mutableStateOf(false) }
    var error by remember { mutableStateOf<String?>(null) }

    val auth = Firebase.auth

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(24.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            if (isRegisterMode) "Регистрация" else "Вход",
            style = MaterialTheme.typography.headlineMedium,
            textAlign = TextAlign.Center,
            modifier = Modifier.fillMaxWidth()
        )

        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier.fillMaxWidth()
        )

        OutlinedTextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Пароль") },
            modifier = Modifier.fillMaxWidth(),
            visualTransformation =
                androidx.compose.ui.text.input.PasswordVisualTransformation()
        )

        error?.let { Text(it, color = MaterialTheme.colorScheme.error) }
    }
}

```

```

        Button(
            onClick = {
                error = null
                if (isRegisterMode) {
                    auth.createUserWithEmailAndPassword(email, password)
                        .addOnCompleteListener { task ->
                            if (!task.isSuccessful) {
                                error = task.exception?.message ?: "Ошибка
регистрации"
                            }
                        }
                } else {
                    auth.signInWithEmailAndPassword(email, password)
                        .addOnCompleteListener { task ->
                            if (!task.isSuccessful) {
                                error = "Неверный email или пароль"
                            }
                        }
                }
            },
            modifier = Modifier.fillMaxWidth(),
            enabled = email.isNotBlank() && password.length >= 6
        ) {
            Text(if (isRegisterMode) "Зарегистрироваться" else "Войти")
        }

        TextButton(onClick = { isRegisterMode = !isRegisterMode }) {
            Text(
                if (isRegisterMode) "Уже есть аккаунт? Войти"
                else "Нет аккаунта? Зарегистрироваться"
            )
        }
    }
}

// === Основной контент (доступен только авторизованным) ===
@Composable
fun MainContentScreen() {
    var screen by remember { mutableStateOf("map") }
    when (screen) {
        "map" -> MapScreen { screen = "favorites" }
        "favorites" -> FavoritesScreen(
            onArrayClick = { screen = "array" },
            onSignOut = {
                Firebase.auth.signOut()
                screen = "map" // после выхода вернёмся к карте, но MainApp
перекинет на Auth
            }
        )
        "array" -> ArrayTestScreen { screen = "map" }
    }
}

// === Вспомогательная функция: получить хранилище для текущего пользователя
===
fun getMarkersPrefs(context: Context): android.content.SharedPreferences {
    val uid = Firebase.auth.currentUser?.uid ?: "guest"
    return context.getSharedPreferences("markers_$uid", Context.MODE_PRIVATE)
}

// === Карта ===
@Composable

```

```

fun MapScreen(onFavoriteClick: () -> Unit) {
    val context = LocalContext.current
    val prefs = getMarkersPrefs(context) // ← теперь с UID
    var markers by remember { mutableStateOf(loadMarkers(prefs)) }
    var dialogState by remember {
        mutableStateOf<DialogState>(DialogState.Closed)
    }

    val cameraPositionState = rememberCameraPositionState {
        position = CameraPosition.fromLatLngZoom(LatLng(48.7080, 44.5156),
12f)
    }

    Box(modifier = Modifier.fillMaxSize()) {
        GoogleMap(
            modifier = Modifier.fillMaxSize(),
            cameraPositionState = cameraPositionState,
            onMapClick = { latLng ->
                dialogState = DialogState.NameInput(latLng, "")
            }
        ) {
            markers.forEach { m ->
                Marker(
                    state = MarkerState(LatLng(m.lat, m.lng)),
                    title = m.name,
                    snippet = m.address,
                    onClick = {
                        dialogState = DialogState.DeleteConfirm(m)
                        true
                    }
                )
            }
        }
    }

    FloatingActionButton(
        onClick = onFavoriteClick,
        containerColor = MaterialTheme.colorScheme.primary,
        modifier = Modifier
            .align(Alignment.BottomEnd)
            .padding(16.dp)
    ) {
        Icon(Icons.Default.Storage, "Избранное")
    }

    when (val state = dialogState) {
        is DialogState.NameInput -> {
            NameInputDialog(
                latLng = state.LatLng,
                initialName = state.name,
                onConfirm = { name ->
                    val address = getAddress(context,
state.LatLng.latitude, state.LatLng.longitude)
                    val id = System.currentTimeMillis()
                    val place = FavoritePlace(id, name,
state.LatLng.latitude, state.LatLng.longitude, address)
                    saveMarker(place, prefs)
                    markers = markers + place
                    dialogState = DialogState.Closed
                },
                onDismiss = { dialogState = DialogState.Closed }
            )
        }
        is DialogState.DeleteConfirm -> {
            DeleteDialog(

```

```

        place = state.place,
        onConfirm = {
            removeMarker(state.place.id, prefs)
            markers = markers.filter { it.id != state.place.id }
            dialogState = DialogState.Closed
        },
        onDismiss = { dialogState = DialogState.Closed }
    )
}
DialogState.Closed -> {}
}

}

// === Избранное (с кнопкой выхода) ===
@Composable
fun FavoritesScreen(onArrayClick: () -> Unit, onSignOut: () -> Unit) {
    val context = LocalContext.current
    val prefs = getMarkersPrefs(context)
    var markers by remember { mutableStateOf(loadMarkers(prefs)) }

    Box(modifier = Modifier.fillMaxSize()) {
        LazyColumn(
            modifier = Modifier.fillMaxSize(),
            contentPadding = PaddingValues(16.dp),
            verticalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            items(markers, key = { it.id }) { place ->
                FavoriteItem(place) {
                    removeMarker(place.id, prefs)
                    markers = loadMarkers(prefs)
                }
            }
        }

        Column(
            modifier = Modifier
                .align(Alignment.BottomEnd)
                .padding(16.dp)
        ) {
            FloatingActionButton(
                onClick = onArrayClick,
                containerColor = MaterialTheme.colorScheme.primary,
                modifier = Modifier.size(56.dp)
            ) {
                Icon(Icons.Default.AddLocationAlt, "Текст")
            }

            Spacer(modifier = Modifier.height(16.dp))

            FloatingActionButton(
                onClick = onSignOut,
                containerColor =
MaterialTheme.colorScheme.secondaryContainer,
                modifier = Modifier.size(56.dp)
            ) {
                Icon(Icons.Default.Close, "Выйти")
            }
        }
    }
}

// === Остальной код без изменений (FavoritesScreen, ArrayTestScreen,

```

```

утилиты...) ===

@Composable
fun FavoriteItem(place: FavoritePlace, onDelete: () -> Unit) {
    Card(modifier = Modifier.fillMaxWidth()) {
        Column(modifier = Modifier.padding(12.dp)) {
            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                Text(place.name, style =
MaterialTheme.typography.titleMedium)
                IconButton(onClick = onDelete) {
                    Icon(Icons.Default.Close, "Удалить")
                }
            }
            Text("${place.lat}, ${place.lng}\n${place.address}", style =
MaterialTheme.typography.bodySmall)
        }
    }
}

@Composable
fun ArrayTestScreen(onBack: () -> Unit) {
    val context = LocalContext.current
    val prefs = getMarkersPrefs(context)
    var original by remember { mutableStateOf<List<String>>(emptyList()) }
    var intermediate by remember { mutableStateOf<List<String>>(emptyList()) }
    var final by remember { mutableStateOf<List<String>>(emptyList()) }

    LaunchedEffect(Unit) {
        val places = if (prefs.contains("places")) {
            loadMarkers(prefs)
        } else {
            (1..100000).map { i ->
                FavoritePlace(
                    id = i.toLong(),
                    name = "Место $i",
                    lat = 48.7 + (kotlin.random.Random.nextDouble() - 0.5) *
0.4,
                    lng = 44.5 + (kotlin.random.Random.nextDouble() - 0.5) *
0.4,
                    address = "Адрес $i"
                )
            }
        }

        original = places.take(5).map { it.name }

        val filtered = places.asSequence()
            .filter { p -> p.lat in 48.5..49.0 && p.lng in 44.0..45.0 }
            .toList()
        intermediate = filtered.take(5).map { it.name }

        val sorted = filtered.asSequence()
            .sortedBy { it.id }
            .toList()
        final = sorted.take(5).map { it.name }
    }

    Box(modifier = Modifier.fillMaxSize()) {
        Column(

```

```

        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
            .verticalScroll(rememberScrollState()),
            verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        Text("Тест массива (100 000 элементов)", style =
MaterialTheme.typography.headlineSmall)
        Section("Исходный массив (первые 5)", original)
        Section("Промежуточный (только Волгоград)", intermediate)
        Section("Итоговый (отсортирован по ID)", final)
    }

    FloatingActionButton(
        onClick = onBack,
        containerColor = MaterialTheme.colorScheme.primary,
        modifier = Modifier
            .align(Alignment.BottomEnd)
            .padding(16.dp)
    ) {
        Icon(Icons.Default.AddLocationAlt, "Назад")
    }
}

@Composable
fun Section(title: String, items: List<String>) {
    Column {
        Text(title, style = MaterialTheme.typography.titleMedium, modifier =
Modifier.padding(vertical = 4.dp))
        Divider()
        items.forEach { item ->
            Text("• $item", style = MaterialTheme.typography.bodyMedium)
        }
    }
}

sealed class DialogState {
    object Closed : DialogState()
    data class NameInput(val latLng: LatLng, val name: String) : DialogState()
    data class DeleteConfirm(val place: FavoritePlace) : DialogState()
}

@Composable
fun NameInputDialog(latLng: LatLng, initialName: String, onConfirm: (String) -> Unit, onDismiss: () -> Unit) {
    var name by remember { mutableStateOf(initialName) }
    AlertDialog(
        onDismissRequest = onDismiss,
        title = { Text("Название места") },
        text = {
            TextField(value = name, onValueChange = { name = it },
placeholder = { Text("Без названия") })
        },
        confirmButton = {
            TextButton(onClick = { onConfirm(name.ifBlank { "Без названия" }) })
        },
        dismissButton = { TextButton(onClick = onDismiss) { Text("Отмена") } }
    )
}

```

```

@Composable
fun DeleteDialog(place: FavoritePlace, onConfirm: () -> Unit, onDismiss: () -> Unit) {
    AlertDialog(
        onDismissRequest = onDismiss,
        title = { Text(place.name) },
        text = { Text("Удалить место?\n${place.address}") },
        confirmButton = { TextButton(onClick = onConfirm) { Text("Удалить") } },
        dismissButton = { TextButton(onClick = onDismiss) { Text("Отмена") } }
    )
}

fun loadMarkersprefs: android.content.SharedPreferences): List<FavoritePlace> {
    return prefs.getStringSet("places", emptySet())?.mapNotNull { s ->
        val p = s.split("|")
        if (p.size == 5) FavoritePlace(p[0].toLong(), p[1], p[2].toDouble(),
            p[3].toDouble(), p[4]) else null
    } ?: emptyList()
}

fun saveMarker(place: FavoritePlace, prefs: android.content.SharedPreferences) {
    val set = prefs.getStringSet("places", mutableSetOf()!!).toMutableSet()

    set.add("${place.id}|${place.name}|${place.lat}|${place.lng}|${place.address}")
    prefs.edit().putStringSet("places", set).apply()
}

fun removeMarker(id: Long, prefs: android.content.SharedPreferences) {
    val set = prefs.getStringSet("places", mutableSetOf()!!).toMutableSet()
    set.removeIf { it.startsWith("$id|") }
    prefs.edit().putStringSet("places", set).apply()
}

fun getAddress(context: Context, lat: Double, lng: Double): String {
    return try {
        Geocoder(context).getFromLocation(lat, lng,
        1)?.firstOrNull()?.getAddressLine(0) ?: "Адрес не найден"
    } catch (e: Exception) {
        "Ошибка"
    }
}

```