МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ИВТ
(ИА, ИП, ИВ)

РЕФЕРАТ

по дисциплине
«Визуальное программирование»

по теме:

GPS LOCATION

Студент:
Группа ИА331
К.А. Любимов

Предподаватель:
Р.В. Ахпашев

Новосибирск 2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках отчета рассматривается прилоежние, в котором реализовано **Gps Location**, разработанный на языке Kotlin с использованием Jetpack Compose для описания интерфейса. Программа представляет из себя простой и интуитивно понятный интерфейс на котором можно увидеть ширину и долготу устройства, которое использует приложение, а также приложение записывает полученные данные в формате JSON в память устройства
Приложение демонстрирует работу со следующими компонентами:

- **Jetpack Compose** — для построения UI с использованием современных декларативных подходов;
- **MediaStore API** — для доступа к фалам, хранящимся на устройстве;
- **Fused Location Provider API** — для получение геолокации;
- **Состояния Compose** — для вывода данных и обновления интерфейса.

1 ТЕОРИЯ

В данной главе рассматриваются основные теоретические аспекты, лежащие в основе реализации музыкального плеера на языке программирования Kotlin с использованием современных подходов к разработке пользовательских интерфейсов.

1.1 Jetpack Compose

Jetpack Compose — современный инструмент Google для создания пользовательских интерфейсов на Android. В отличие от традиционного подхода с использованием XML, Jetpack Compose позволяет описывать UI декларативно с помощью кода на Kotlin. Это существенно упрощает реализацию интерактивных компонентов и их обновление при изменении данных.

Ключевые особенности Jetpack Compose:

- **Декларативность**: UI автоматически перестраивается при изменении состояния.
- **Композиторы (Composables)**: функции с аннотацией `@Composable` определяют пользовательский интерфейс.
- **Состояние и реактивность**: при использовании `mutableStateOf` и других типов состояния интерфейс обновляется без ручного вмешательства.
- **Интеграция с ViewModel**: позволяет эффективно управлять состоянием приложения.

Jetpack Compose — мощный инструмент, заменяющий устаревшие методы построения UI, такой как XML и View Binding, и подходит для современных Android-приложений.

1.2 Получение геолокации, и вывод

LocationServices — это компонент Google Play Services, предоставляющий API для работы с геолокацией на Android. Он входит в состав библиотеки com.google.android.gms:play-services-location и предлагает более продви-

нутые и энергоэффективные методы получения данных о местоположении, чем стандартный LocationManager.

- Проверка активности GPS/сети через `LocationManager`
- Использование `FusedLocationProviderClient`
- Получение последней известной локации (`lastLocation`)
- Обновление UI через `mutableStateOf`

1.3 Работа с хранилищем файлов (MediaStore)

Для получения списка файлов, хранящихся на устройстве, используется **MediaStore** — Android API, обеспечивающий доступ к общедоступным медиа-ресурсам.

- Генерация JSON-файла с координатами и временем
- Запись в папку `Downloads`:
  - Для Android 10+: через `MediaStore`
  - Для старых версий: прямой доступ к `File`
- Обновление медиа-базы через `MediaScannerConnection`

Таким образом, обеспечивается сохранение JSON файлов в памяти устройства.

1.4 Принципы взаимодействия компонентов

Приложение запрашивает доступ к геолокации, проверяет активность GPS/сети, получает последние координаты через Google Play Services, сохраняет их в файл (с учётом версии Android) и позволяет вернуться на главный экран. Ошибки и статусы отображаются через уведомления.

Общий цикл работы:

1. Старт активности → Запрос разрешения на геолокацию.
2. Если разрешение есть → Проверка включённого GPS/сети.
3. Если выключено → Уведомление пользователя.
4. Если включено → Запрос последней локации.
5. Успешный запрос → Сохранение координат в файл + обновление UI.

6. Ошибка → Toast с описанием проблемы.
7. Возврат на главный экран → По нажатию кнопки через Intent.

Ключевые особенности:

1. Асинхронная обработка данных.
2. Минимальное потребление ресурсов (использование последней известной локации).
3. Реактивное обновление интерфейса (Jetpack Compose).
4. Адаптация под разные версии Android.

2 ПРАКТИКА

В данной главе рассматриваются ключевые аспекты реализации GPS Loacation на языке Kotlin с использованием Jetpack Compose. Основное листингу проекта.

2.1 Среда разработки

Для реализации проекта использовалась следующая среда:

- **Язык программирования**: Kotlin
- **Среда разработки (IDE)**: Android Studio
- **Минимальная версия SDK**: 21 (Android 5.0 Lollipop)
- **Целевая версия SDK**: 33 (Android 13)
- **Библиотеки и технологии:**
  - Jetpack Compose (UI)
  - FusedLocationProviderClient (Google Play Services)
  - LocationManager (проверка статуса GPS/сети)
  - MediaStore и File API (сохранение данных)

2.2 MapPage.kt

Приложение состоит из следующих ключевых компонентов:

- **MapPage.kt** — Основная активность, в которой инициализируется Compose-интерфейс.
- **LocationScreen() (Compose-функция)** — Реализует UI: отображение координат, кнопка возврата.
- **Взаимодействие с локацией:** — FusedLocationProviderClient для получения последней известной локации. LocationManager для проверки активности GPS/сети.

Состояния и данные:

- **Текущие координаты** — широта и долгота (обновляются через mutableStateOf).

– **Статус разрешений** — `ACCESS_FINE_LOCATION` (обрабатывается через `ActivityResultLauncher`).
– **Статус геолокации** — проверка включения GPS/сети.

Листинг 2.1 — MapPage.kt

```
package com.example.mycal.activities

import android.Manifest
import android.content.ContentValues
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.location.LocationManager
import android.os.Bundle
import android.os.Environment
import android.provider.MediaStore
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.compose.
   rememberLauncherForActivityResult
import androidx.activity.enableEdgeToEdge
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.layout.*
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import androidx.core.content.ContextCompat
import com.example.mycal.MainPage
import com.example.mycal.ui.theme.MycalTheme
import com.example.mycal.ui.theme.Rose
```

```
import com.example.mycal.ui.theme.Russian_Violete
import com.google.android.gms.location.LocationServices
import org.json.JSONObject

class LocationActivity : ComponentActivity() {

    @OptIn(ExperimentalMaterial3Api::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        setContent {
            MycalTheme {
                Surface(modifier = Modifier.fillMaxSize(), color
                   = Russian_Violete) {
                    LocationScreen()
                }
            }
        }
    }

    @Composable
    fun LocationScreen() {
        val context = LocalContext.current

        var latText by remember { mutableStateOf("-") }
        var lonText by remember { mutableStateOf("-") }
        var hasPermission by remember { mutableStateOf(false) }

        val permissionLauncher =
           rememberLauncherForActivityResult(
            contract = ActivityResultContracts.RequestPermission
               ()
        ) { granted ->
            hasPermission = granted
            if (!granted) {
                Toast.makeText(
                    context,
                    "                          ",
                    Toast.LENGTH_SHORT
                ).show()
            }
```

```
        }

        LaunchedEffect(Unit) {
            permissionLauncher.launch(Manifest.permission.
               ACCESS_FINE_LOCATION)
        }

        LaunchedEffect(hasPermission) {
            if (hasPermission) {
                val lm = context.getSystemService(Context.
                   LOCATION_SERVICE) as LocationManager
                val enabled = lm.isProviderEnabled(
                   LocationManager.GPS_PROVIDER)
                        || lm.isProviderEnabled(LocationManager.
                           NETWORK_PROVIDER)
                if (!enabled) {
                    Toast.makeText(
                        context,
                        "                    ,      ",
                        Toast.LENGTH_LONG
                    ).show()
                } else {
                    val permissionGranted = ContextCompat.
                       checkSelfPermission(
                        context,
                        Manifest.permission.ACCESS_FINE_LOCATION
                    ) == PackageManager.PERMISSION_GRANTED

                    if (permissionGranted) {
                        val client = LocationServices.
                           getFusedLocationProviderClient(context
                           )
                        client.lastLocation
                            .addOnSuccessListener { location ->
                                if (location != null) {
                                    latText = location.latitude.
                                       toString()
                                    lonText = location.longitude.
                                       toString()
                                    saveCoordsToDownloads(context
                                       , location.latitude,
                                       location.longitude)
```

```
                            } else {
                                Toast.makeText(
                                    context,
                                    "                ",
                                    Toast.LENGTH_SHORT
                                ).show()
                            }
                        }
                        .addOnFailureListener {
                            Toast.makeText(
                                context,
                                "                ",
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                } else {
                    Toast.makeText(
                        context,
                        "                    ",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            }
        }
    }

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(24.dp),
        verticalArrangement = Arrangement.SpaceBetween,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Column(
            horizontalAlignment = Alignment.
               CenterHorizontally
        ) {
            Text(
                "          ",
                style = MaterialTheme.typography.
                   headlineMedium,
                color = Color.White
```

```
            )
            Spacer(Modifier.height(16.dp))
            Text("    : $latText", style = MaterialTheme.
               typography.bodyLarge, color = Color.White)
            Text("    : $lonText", style = MaterialTheme.
               typography.bodyLarge, color = Color.White)
        }

        Button(
            onClick = {
                context.startActivity(Intent(context,
                   MainPage::class.java))
            },
            colors = ButtonDefaults.buttonColors(Rose),
            modifier = Modifier
                .fillMaxWidth()
                .height(56.dp)
        ) {
            Text("Back to Main", color = Color.White)
        }
    }
}

private fun saveCoordsToDownloads(context: Context, lat:
   Double, lon: Double) {
    val json = JSONObject().apply {
        put("latitude", lat)
        put("longitude", lon)
        put("timestamp", System.currentTimeMillis())
    }
    val filename = "coords_${System.currentTimeMillis()}.json
       "
    val bytes = json.toString(2).toByteArray()

    val values = ContentValues().apply {
        put(MediaStore.Downloads.DISPLAY_NAME, filename)
        put(MediaStore.Downloads.MIME_TYPE, "application/json
           ")
        put(MediaStore.Downloads.RELATIVE_PATH, Environment.
           DIRECTORY_DOWNLOADS)
    }
    val uri = context.contentResolver.insert(
```

```
                MediaStore.Downloads.EXTERNAL_CONTENT_URI, values
            )
            if (uri != null) {
                context.contentResolver.openOutputStream(uri).use {
                    it?.write(bytes) }
                Toast.makeText(
                    context,
                    "              Downloads/$filename",
                    Toast.LENGTH_LONG
                ).show()
            } else {
                Toast.makeText(context, "                Downloads",
                    Toast.LENGTH_SHORT).show()
            }
        }
}
```

Интерфейс реализован с помощью Jetpack Compose. В нём отображаются:

- **Основной контент**
  - Заголовок ”Текущие координаты”(белый цвет, крупный шрифт)
  - Поле ”Широта: —”(белый цвет)
  - Поле ”Долгота: —”(белый цвет)
- **Кнопка внизу экрана**
  - Текст ”Back to Main”(белый цвет)
  - Розовый фон (цвет Rose)
  - Занимает всю ширину экрана
  - Высота 56dp

Общий принцип работы:

- **Инициализация и запрос разрешений:**
  - При старте активити автоматически запускается запрос разрешения `ACCESS_FINE_LOCATION`
  - Используется ActivityResultLauncher для обработки ответа пользователя

  - Состояние hasPermission отслеживает наличие/отсутствие разрешения

- **Проверка доступности геолокации:**
  - После получения разрешения проверяется статус GPS/сетевых провайдеров
  - Если геолокация отключена - показывается Toast с просьбой включить
  - При успешной проверке переходит к получению координат

- **Получение местоположения:**
  - Используется FusedLocationProviderClient для доступа к API локации
  - Запрашивается последнее известное местоположение (lastLocation)
  - Реализованы коллбеки:
    * onSuccessListener - обработка успешного получения координат
    * onFailureListener - обработка ошибок получения

- **Обновление UI и сохранение данных:**
  - Полученные координаты обновляют состояние latText/lonText
  - Вызывается saveCoordsToDownloads() для сохранения в файл:
    * Создается JSON-объект с координатами и временной меткой
    * Используется MediaStore API для сохранения в папку Downloads
    * Имя файла генерируется по шаблону `coords_<timestamp>.json`

- **Обратная связь с пользователем:**
  - Toast-уведомления о статусе операций
  - Визуальное отображение координат в реальном времени
  - Обработка ошибок (отсутствие разрешения, проблемы с записью файла)

- **Навигация:**

  - Кнопка ”Back to Main”запускает MainPage через явный Intent
  - Сохранение данных происходит асинхронно, не блокируя UI

- **Ключевые технологии:**
  - Jetpack Compose для UI
  - Location Services API (Fused Location Provider)
  - Система разрешений Android (Runtime Permissions)
  - ContentResolver для работы с MediaStore
  - JSON для сериализации данных
  - Асинхронная обработка через Listeners

- **Поток данных:**
  - Запрос разрешения → Проверка GPS → Получение локации → Обновление UI → Сохранение в файл → Возможность возврата на главный экран

Особенность: Все операции с геолокацией и файловой системой выполняются асинхронно, что предотвращает блокировку основного потока UI.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы было разработано Android-приложение для получения и сохранения геолокационных данных с использованием современных инструментов и технологий. Я опирался на репозиторий преподавателя [1], на видеоролики из ютуба [2] и сайт Metonit [3]. В качестве фреймворка для пользовательского интерфейса использован **Jetpack Compose**, что позволило создать гибкий и адаптивный интерфейс с декларативным подходом.

Основной функционал приложения включает:

- Автоматический запрос разрешений на доступ к геолокации через **Runtime Permissions**.
- Отображение текущих координат (широты и долготы) в реальном времени.
- Сохранение данных в формате **JSON** в системную папку `Downloads` через **MediaStore API**.
- Интеграцию с **FusedLocationProviderClient** для точного определения местоположения.

При разработке были применены лучшие практики проектирования:

- Асинхронная обработка операций с использованием коллбеков и **Listeners**.
- Реализация устойчивости к изменениям конфигурации устройства.
- Организация кода с учётом принципов расширяемости и поддерживаемости.

Полученный результат демонстрирует корректную работу всех компонентов: успешное получение координат, их визуализацию в интерфейсе, сохранение в файл с уникальным именем, а также обработку ошибок при отсутствии разрешений или отключённой геолокации. Пользовательский интерфейс обеспечивает интуитивное взаимодействие, а система уведомлений через **Toast** информирует о статусе операций.

Разработка позволила получить практические навыки:

– Работы с геолокацией в Android, включая взаимодействие с **Google Play Services**.
– Создания адаптивных интерфейсов на базе **Jetpack Compose**.
– Управления файловой системой через **ContentResolver** и **MediaStore**.

Перспективы развития приложения включают интеграцию с картографическими сервисами (например, **Google Maps**), реализацию фонового отслеживания локации, а также добавление синхронизации данных с облачными хранилищами. Данная работа подтверждает возможность создания устойчивых и масштабируемых решений в области мобильной разработки на платформе Android.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Ахпашев Р.* Репозиторий преподавателя с материалами по практическим работам. — 2025. — URL: https://github.com/sibsutisTelecomDep/blog (дата обр. 20.05.2025).
2. Видеохостинг YouTube. — 2025. — URL: https://youtube.com (дата обр. 20.05.2025) ; Материалы по Kotlin.
3. Руководство по языку Kotlin. — 2025. — URL: https://metanit.com/kotlin/tutorial/ (дата обр. 20.05.2025).