**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУСCКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра компьютерных технологий и систем**

**Отчет**

**о прохождении производственной преддипломной практики**

студента 4 курса

Жуковского Павла Сергеевича,

специальность «прикладная информатика»

Научный руководитель:

Старший преподаватель кафедры КТС

Кулинкович В. А.

Руководитель практики от организации:

Специалист по разработке ПО

ООО «Нетко Спорт»

Близнюк А.И.

Минск, 2022

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc98723964)

[ОСНОВНЫЕ ЭТАПЫ ПРОХОЖДЕНИЯ ПРАКТИКИ 4](#_Toc98723965)

[ГЛАВА 1 Технологии разработки мобильных приложений под операционную систему Android 5](#_Toc98723966)

[1.1 WorkManager 5](#_Toc98723967)

[1.2 Dependency Injection. Koin 7](#_Toc98723968)

[1.3 Retrofit2. Okhttp3 10](#_Toc98723969)

[1.4. Room Database 11](#_Toc98723970)

[1.5. Kotlin Coroutines 15](#_Toc98723971)

[ГЛАВА 2 ТехнологиИ разработки интерактивной карты 19](#_Toc98723972)

[2.1 OpenStreetMap 19](#_Toc98723973)

[2.2 Mapbox 19](#_Toc98723974)

[2.3 Сравнительная оценка OpenStreetMap и Mapbox 20](#_Toc98723975)

[ЗАКЛЮЧЕНИЕ 24](#_Toc98723976)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc98723977)

# ВВЕДЕНИЕ

В период с 7 февраля 2022 г. по 3 марта 2022 г. мною была пройдена практика в ООО «Нетко Спорт».

Данная компания собирает выдающихся программистов, дизайнеров, переводчиков и других специалистов для разработки качественного программного обеспечения. Решения, разработанные ООО «Нетко Спорт», широко используются в сфере разработки программного обеспечения для мобильных устройств под операционные системы iOS и Android. Уже долгое время данная компания считается надёжным партнёром во множестве сфер и бизнес-процессов.

Целью преддипломной практики в данной компании была разработка различных мобильных приложений с использованием базы данных, внедрения зависимостей, навигации, стилей и многих других технологий, которые позже оказали мне поддержку в разработке дипломного проекта, связанного с интерактивной картой.

В ходе производственной преддипломной практики от организации руководителем были предложены различные технологии для изучения, которые позже были реализованы в виде мобильного приложения. Данные технологии могут быть затем использованы в приложении дипломного проекта.

В период с 4 марта по 19 марта практика проходила на кафедре КТС (ФПМИ, БГУ). В данный период были тщательно изучены такие технологии разработки интерактивных карт, как OpenStreetMap и Mapbox. Были проанализированы возможности обоих провайдеров как в плане сложности разработки, так и в плане функционала.

Целью данной практики является освоение и закрепление знаний и умений, полученных по всему курсу обучения, проверка возможностей самостоятельной работы в условиях конкретного производства, а также формирование и анализ материалов для выполнения дипломной работы и формирование умения работать с технической и программной документацией.

Среди задач данной практики можно выделить изучение различных технологий Android-разработки, которые понадобятся для реализации приложения (Kotlin, Kotlin Coroutines, WorkManager, Retrofit, Okhttp, Gson, RoomDB, Koin), изучение технологии OpenStreetMap и методов работы с ней, изучение технологии Mapbox и её сравнительная оценка относительно OpenStreetMap.

# ОСНОВНЫЕ ЭТАПЫ ПРОХОЖДЕНИЯ ПРАКТИКИ

*Первый этап* практики (7 февраля – 8 февраля 2022 г.) включал ознакомление меня как практиканта с основными целями и задачами деятельности организации, ее структурой, должностными обязанностями сотрудников организации. В ходе первого этапа был разработали план предстоящей работы, который включал в себя: распределение задач, изучение предложенных технологий и поиск необходимой информации.

*Второй этап* практики (9 февраля - 11 февраля 2022 г.) – ознакомление с технологиями, необходимыми для изучения: Kotlin, Kotlin Coroutines, WorkManager, Retrofit, Okhttp, Gson, RoomDB, Koin.

*Третий этап* практики (12 февраля – 3 марта 2022 г.) – изучение вышеуказанных технологий для разработки мобильных приложений под операционную систему Android и применение их в реальных приложениях.

*Четвертый этап* (4 марта – 12 марта 2022 г.) – изучение и сравнительная оценка таких технологий разработки интерактивных карт, как OpenStreetMap и Mapbox, миграция с одной технологии на другую.

*Пятый этап* (13 марта – 16 марта 2022 г.) – внедрение Mapbox и других технологий, изученных в ходе практики, в дипломный проект.

*Шестой этап* (17 марта – 19 марта 2022 г.) – сбор результатов, подготовка отчёта и презентации.

# ГЛАВА 1 Технологии разработки мобильных приложений под операционную систему Android

## **1.1 WorkManager**

WorkManager – это API, который упрощает планирование надежных асинхронных задач, которые, как ожидается, будут выполняться даже при выходе из приложения или перезапуске устройства. WorkManager API является подходящей и рекомендованной заменой для всех предыдущих API фонового планирования Android, включая FirebaseJobDispatcher, GcmNetworkManager и Job Scheduler . WorkManager включает в себя функции своих предшественников в современном последовательном API, который работает до уровня API 14, но при этом учитывает время автономной работы.

WorkManager использует базовую службу диспетчеризации заданий на основе следующих критериев (см. рисунок 1.1):

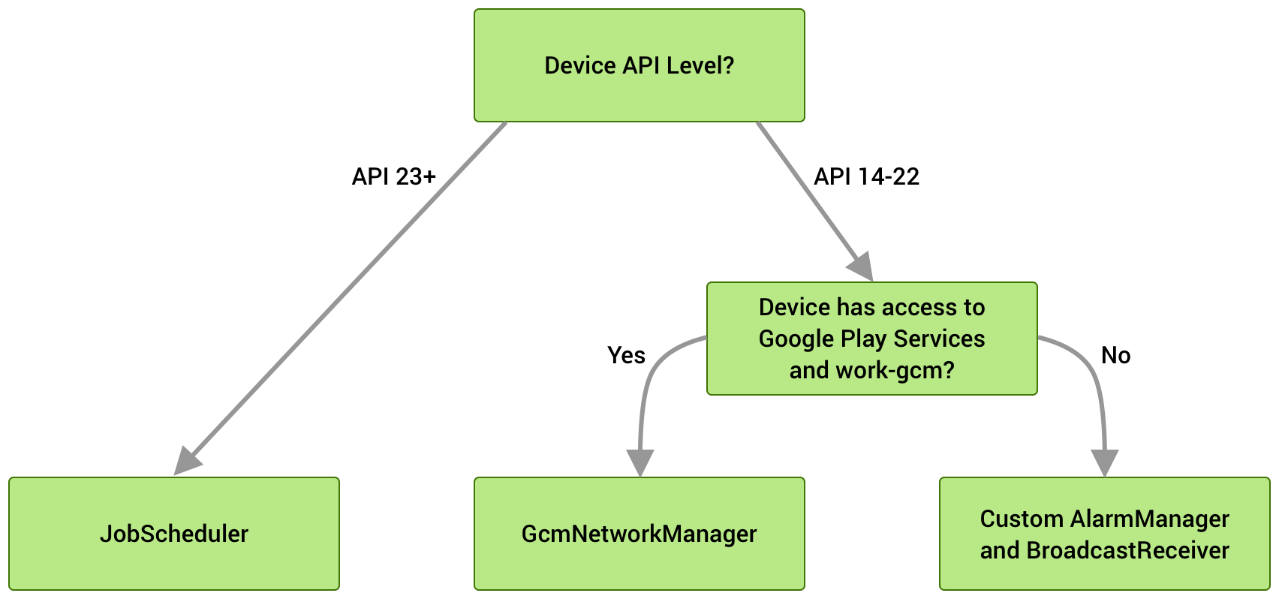


Рисунок 1.1 - Базовая служба диспетчеризации заданий WokManager

Для работы с WorkManager требуется декларативно определить оптимальные условия для работы, используя рабочие ограничения. Например, запускать, только когда устройство подключено к сети Wi-Fi, когда устройство находится в режиме ожидания, или когда на нем достаточно места для хранения.

WorkManager позволяет планировать работу, чтобы выполнить её одноразово или несколько раз с использованием гибких окон планирования. Работа также может быть помечена тегами и названа, что позволяет планировать уникальную заменяемую работу, а также отслеживать или отменять группы работ вместе. Запланированная работа хранится во внутренне управляемой базе данных SQLite, и WorkManager заботится о том, чтобы эта работа сохранялась и переносилась при перезагрузке устройства. Кроме того, WorkManager придерживается функций энергосбережения и передовых методов, таких как режим Doze, поэтому об этом можно не беспокоиться.

Иногда работа не получается. WorkManager предлагает гибкие политики повторных попыток, включая настраиваемую политику экспоненциальной отсрочки.

Для сложной связанной работы можно объединить отдельные рабочие задачи в цепочку с помощью понятного, естественного интерфейса, который позволяет контролировать, какие части выполняются последовательно, а какие – параллельно.

Для каждой рабочей задачи можно определить входные и выходные данные для соответствующей работы. При объединении работы в цепочку WorkManager автоматически передает выходные данные от одной рабочей задачи к другой.

WorkManager интегрируется с RxJava и Coroutines и обеспечивает гибкость в собственных асинхронных интерфейсах.

WorkManager предназначен для работы, которая требует надежности, даже если пользователь переходит за пределы экрана, приложение закрывается или устройство перезагружается. Например:

* Отправка журналов или аналитики в серверные службы
* Периодическая синхронизация данных приложения с сервером

WorkManager не предназначен для фоновой работы в процессе, которая может быть безопасно прекращена, если процесс приложения прекращается, или для работы, требующей немедленного выполнения. Существует руководство по фоновой обработке, которое позволяет узнать, какое решение соответствует тем или иным потребностям.

Данная технология может пригодиться при разработке интерактивной карты. Например, если пользователь мобильного приложения желает, чтобы данные о новых достопримечательностях автоматически поступали на его мобильное устройство, он может включить соответствующую функцию, которая будет загружать эти данные на фоне, даже тогда, когда пользователь не использует приложение.

Данная технология была успешно использована в проекте (см. рисунок 1.2).

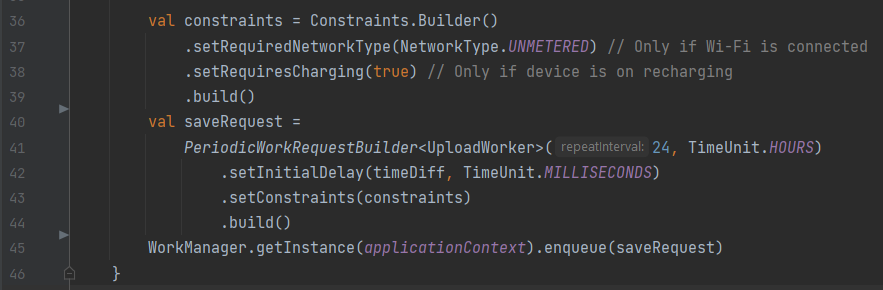


Рисунок 1.2 – Пример использования WorkManager в приложении

## **1.2 Dependency Injection. Koin**

Dependency Injection – это метод программирования, который делает класс независимым от его зависимостей. Это стало возможным благодаря разделению использования объекта и его создания. Многие разработчики Android знакомы с основанными на Java фреймворками внедрения зависимостей, такими как Dagger и Guice.

Однако некоторые фреймворки полностью написаны на Kotlin. Эти фреймворки включают Koin и Kodein. Далее будет отмечаться то, как управлять зависимостями в Android с помощью новой инфраструктуры внедрения зависимостей – Koin.

Фреймворк внедрения зависимостей помогает создавать и управлять зависимостями. Как упоминалось ранее, существует множество основанных на Java фреймворков для внедрения зависимостей Android. Однако с ростом внедрения Kotlin на Android растет спрос на библиотеки, полностью написанные на Kotlin.

Koin – это инфраструктура внедрения зависимостей, которая соответствует этой потребности. Это легкий фреймворк, простой в освоении и не содержащий большого количества шаблонного кода. Эту структуру можно эффективно использовать для управления зависимостями в приложениях под операционную систему Android.

Благодаря мощи языка Kotlin, Koin предоставляет DSL, чтобы описать приложение, вместо того, чтобы аннотировать его или генерировать для него код. Благодаря Kotlin DSL, Koin предлагает интеллектуальный функциональный API для подготовки инъекции зависимостей.

Koin предлагает несколько ключевых слов, позволяющих описать элементы приложения Koin:

* Application DSL, чтобы описать конфигурацию контейнера Koin
* Модуль DSL, чтобы описать компоненты, которые необходимо ввести.

Экземпляр KoinApplication представляет собой конфигурацию экземпляра контейнера Koin. Это позволяет настроить ведение журнала, загрузку свойств и модули.

Чтобы построить новый KoinApplication, используются следующие функции:

* koinApplication {…} – создать KoinApplication конфигурацию контейнера
* startKoin {…} – создать KoinApplication конфигурацию контейнера и зарегистрировать ее в GlobalContext, чтобы разрешить использование GlobalContext API.

Чтобы настроить KoinApplication экземпляр, можно использовать любую из следующих функций:

* logger() – описывает, какой уровень и реализацию Logger использовать (по умолчанию используется EmptyLogger)
* modules() – устанавливает список модулей Koin для загрузки в контейнер
* properties() – загружает свойства HashMap в контейнер Koin
* fileProperties() – загружает свойства из заданного файла в контейнер Koin
* environmentProperties() – загружает свойства из окружения ОС в контейнер Koin
* createEagerInstances() – создаёт нетерпеливые экземпляры (отдельные определения отмечены как createdAtStart)

Можно описать конфигурацию контейнера Koin двумя способами: koinApplication или startKoin функцией.

* koinApplication описывает экземпляр контейнера Koin
* startKoin описывает экземпляр контейнера Koin и регистрирует его в Koin GlobalContext

Зарегистрировав конфигурацию контейнера в GlobalContext, глобальный API может использовать ее напрямую. Любой KoinComponent относится к Koin экземпляру. По умолчанию мы используем экземпляр из GlobalContext.

Модуль Koin – это «пространство» для сбора определений Koin. Он объявляется с помощью module функции.

Объявление одноэлементного компонента означает, что контейнер Koin будет хранить уникальный экземпляр объявленного компонента. Так, мы можем использовать single функцию в модуле, чтобы объявить синглтон.

Single, factory и scoped – это ключевые слова, которые объявляют компоненты через лямбда-выражения. Данное лямбдя-выражение описывает способ создания компонента. Обычно мы создаем экземпляры компонентов через их конструкторов, но можно также использовать любое выражение. Пример объявления компоненты:

*single { Class constructor // Kotlin expression }*

Тип результата лямбда-выражения – это основной тип компонента.

Koin может быть очень удобен при разработке мобильного приложения, так как он позволяет хранить все зависимости проекта в одном файле. Благодаря этой технологии можно избавиться от ряда потенциальных ошибок при компиляции и сборке, что является большим преимуществом.

Внедрение зависимостей получило широкое распространение в реализации проекта (см. рисунок 1.3).

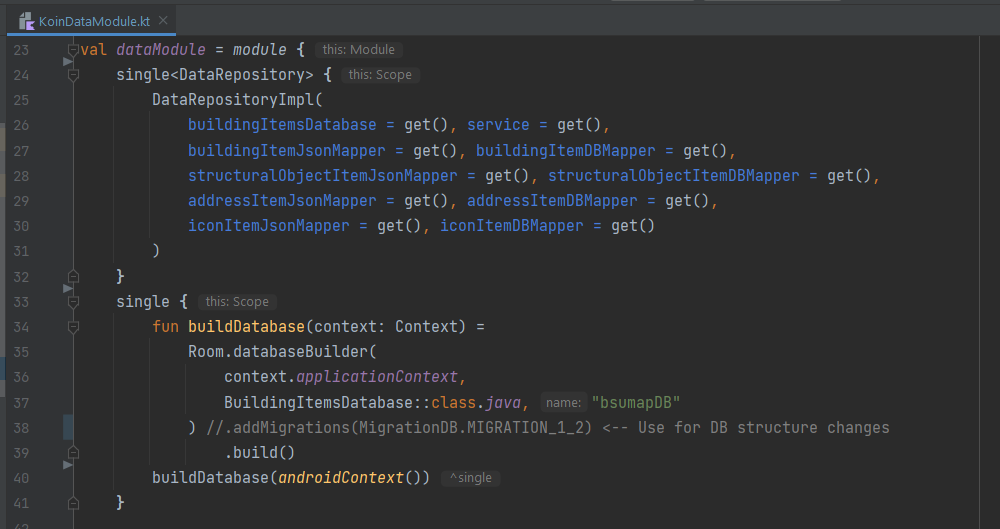


Рисунок 1.3 Пример использования внедрения зависимостей Koin в проекте

## **1.3 Retrofit2. Okhttp3**

Многие сайты имеют собственные API для удобного доступа к своим данным. На данный момент самый распространённый вариант – это JSON. Также могут встречаться данные в виде XML и других форматов. Библиотека Retrofit упрощает взаимодействие с REST API сайта, беря на себя часть рутинной работы. Авторами библиотеки Retrofit являются разработчики из компании "Square", которые написали множество полезных библиотек, например, Picasso, Okhttp, Otto.

Данной библиотекой удобно пользоваться для запроса к различным веб-сервисам с командами GET, POST, PUT, DELETE. Данная технология также позволяет работать в асинхронном режиме, что избавляет от лишнего кода. Стоит отметить, что, при подключении к проекту библиотеки Retrofit2, библиотека Okhttp подключается автоматически, что также очень удобно.

Ниже представлен пример кода, позволяющий создать объект Retrofit:

*Retrofit retrofit = Retrofit.Builder()*

*.baseUrl("https://your.api.url/v2/");*

*.addConverterFactory(ProtoConverterFactory.create())*

*.addConverterFactory(GsonConverterFactory.create())*

*.build();*

В объекте Retrofit мы можем указать ссылку на API веб-сервера, с которым мы работаем. Помимо этого, мы можем указать необходимый нам конвертер данных из существующих, либо использовать собственный конвертер, реализовав интерфейс на основе абстрактного класса Converter.Factory.

Ниже представлен список аннотацией, которые активно используются Retrofit:

* @GET() – GET-запрос для базового адреса. Также можно указать параметры в скобках
* @POST() – POST-запрос для базового адреса. Также можно указать параметры в скобках
* @Path – Переменная для замещения конечной точки, например, username подставится в {username} в адресе конечной точки
* @Query – Задаёт имя ключа запроса со значением параметра
* @Body – Используется в POST-вызовах (из Java-объекта в JSON-строку)
* @Header – Задаёт заголовок со значением параметра
* @Headers – Задаёт все заголовки вместе
* @Multipart – Используется при загрузке файлов или изображений
* @FormUrlEncoded – Используется при использовании пары "имя/значение" в POST-запросах
* @FieldMap – Используется при использовании пары "имя/значение" в POST-запросах
* @Url – Для поддержки динамических адресов

**Так как мобильное приложение вынуждено забирать данные о достопримечательностях с сервера, то ему понадобятся технологии, с помощью которых станет возможным получить доступ к API веб-сервера. Таким образом, библиотеки Retrofit2 и Okhttp3 будут очень востребованы в приложении.**

**Данная технология получила широкое применение в приложении (см. рисунок 1.4).**

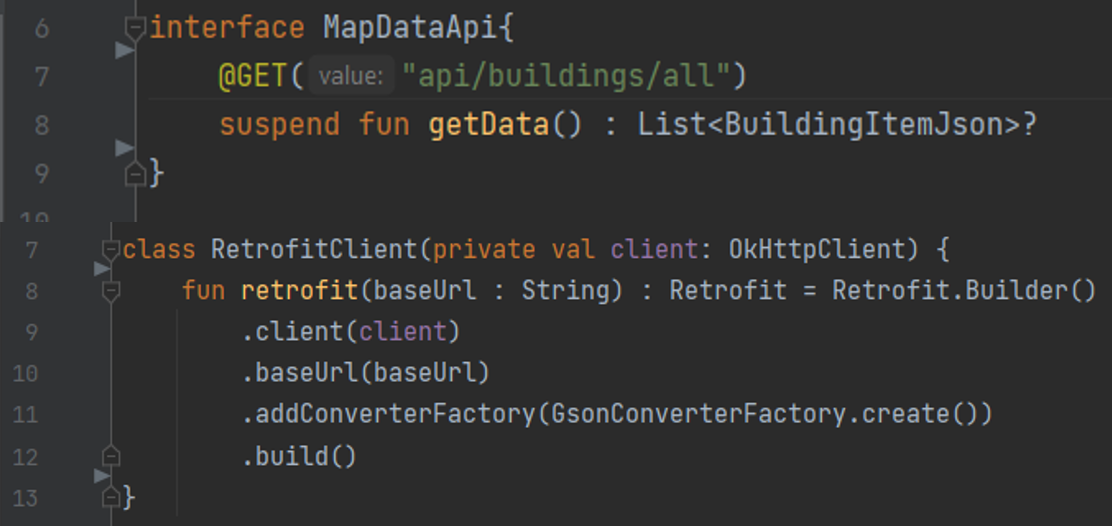
****

Рисунок 1.4 Пример использования технологий Retrofit и Okhttp в приложении

## **1.4. Room Database**

**Приложения, которые обрабатывают нетривиальные объемы структурированных данных, могут значительно выиграть от локального сохранения этих данных. Наиболее распространенный вариант использования - кэширование соответствующих фрагментов данных, чтобы, когда устройство не может получить доступ к сети, пользователь все еще мог просматривать этот контент, находясь в автономном режиме.**

**Библиотека Room обеспечивает уровень абстракции над SQLite, чтобы обеспечить свободный доступ к базе данных, используя всю мощь SQLite. В частности, Room предоставляет следующие преимущества:**

* **Проверка SQL-запросов во время компиляции.**
* **Удобные аннотации, которые сводят к минимуму повторяющийся и подверженный ошибкам шаблонный код.**
* **Оптимизированные пути миграции базы данных.**

**Благодаря этим преимуществам можно использовать Room вместо непосредственного использования API SQLite.**

**Room состоит из трех основных компонентов:**

1. **Класс базы данных, который содержит саму базу данных и служит основной точкой доступа для базового подключения к постоянным данным приложения.**
2. **Сущности данных, представляющие таблицы в базе данных приложения.**
3. **Объекты доступа к данным (DAO), которые предоставляют методы, которые приложение может использовать для запроса, обновления, вставки и удаления данных в базе данных.**

**Класс базы данных предоставляет приложению экземпляры DAO, связанные с этой базой данных. В свою очередь, приложение может использовать DAO для извлечения данных из базы данных как экземпляров связанных объектов сущностей данных. Приложение также может использовать определенные объекты данных для обновления строк из соответствующих таблиц или для создания новых строк для вставки. Существует тесная взаимосвязь между компонентами Room (см. рисунок 1.5):**

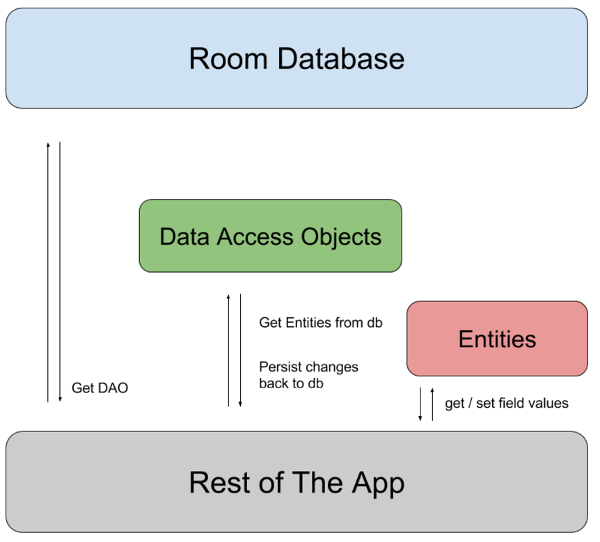


Рисунок 1.5 – Взаимосвязь между компонентами Room

**Следующий код определяет AppDatabaseкласс для хранения базы данных:**

***@Database(entities = [User::class], version = 1)***

***abstract class AppDatabase : RoomDatabase() {***

***abstract fun userDao(): UserDao***

***}***

**AppDatabase определяет конфигурацию базы данных и служит основной точкой доступа приложения к сохраненным данным. Класс базы данных должен удовлетворять следующим условиям:**

* **Класс должен быть аннотирован @Database аннотацией, которая включает в себя entities массив, в котором перечислены все объекты данных, связанные с базой данных.**
* **Класс должен быть расширяемым абстрактным классом RoomDatabase.**
* **Для каждого класса DAO, связанного с базой данных, класс базы данных должен определять абстрактный метод, который не имеет аргументов и возвращает экземпляр класса DAO.**

**Если приложение выполняется в рамках одного процесса, при создании экземпляра AppDatabase объекта нужно следовать шаблону проектирования singleton. Каждый RoomDatabase экземпляр стоит довольно дорого, и редко требуется доступ к нескольким экземплярам в рамках одного процесса. Если же приложение выполняется в нескольких процессах, его следует заключить в enableMultiInstanceInvalidation() вызов построителя базы данных. Таким образом, если есть экземпляр AppDatabase в каждом процессе, то можно сделать недействительным файл общей базы данных в одном процессе, и это признание автоматически распространится на экземпляры AppDatabaseв других процессов.**

**После того, как определён объект данных, DAO и объект базы данных, можно использовать следующий код для создания экземпляра базы данных:**

***val db = Room.databaseBuilder(***

***applicationContext,***

***AppDatabase::class.java, "database-name"***

***).build()***

**Затем можно использовать абстрактные методы из AppDatabase, чтобы получить экземпляр DAO. Например, можно использовать методы из экземпляра DAO для взаимодействия с базой данных:**

***val userDao = db.userDao()***

***val users: List<User> = userDao.getAll()***

**Безусловно, значение создания локальной базы данных для приложения с интерактивной картой очень велико. В случае, если у пользователя приложения не будет доступа к Интернету, он всё равно сможет получить информацию о местоположениях достопримечательностей, если те были добавлены в локальную базу данных ранее.**

**В проекте с интерактивной картой была задействована база данных, использующая технологию RoomDB (см. рисунок 1.6).**

****

Рисунок 1.6 Применение технологии RoomDB для создания базы данных в приложении

## **1.5. Kotlin Coroutines**

**Coroutines – это технология языка Kotlin, позволяющая работать с многопоточными приложениями. Данная технология позволяет писать асинхронный неблокирующий код последовательным образом.**

В Kotlin Coroutines есть специальная функция, которую мы можем объявить с помощью ключевого слова suspend. Приостановка функций может приостановить выполнение сопрограммы, что означает, что она будет ждать, пока приостанавливающие функции не возобновятся. Примером работы сопрограммы может служить следующий код:

*CoroutineScope(Dispatchers.Main + Job()).launch {*

*val user = fetchUser() // A suspending function running in the I/O thread.*

*updateUser(user) // Updates UI in the main thread.*

*}*

*private suspend fun fetchUser(): User = withContext(Dispatchers.IO) {*

*// Fetches the data from server and returns user data.*

*}*

В данном коде можно выделить основополагающие компоненты сопрограммы (см. рисунок 1.7):

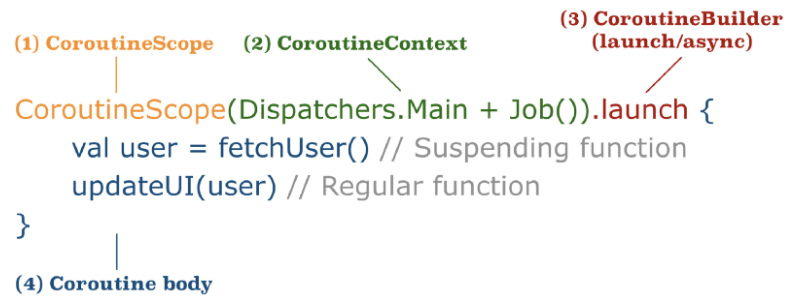


Рисунок 1.7 – Компоненты Kotlin Coroutines

Конструкция CoroutineScope определяет область действия новых сопрограмм. Каждый построитель сопрограмм является расширением CoroutineScope и наследует его coroutineContext для автоматического распространения как элементов контекста, так и отмены.

MainScope же способен создавать основную область для компонентов пользовательского интерфейса. Он работает в основном потоке с SupervisorJob(), что означает, что сбой одного из его дочерних заданий не повлияет на другие.

Существует также GlobalScope. Это область, которая не привязана к какой-либо работе. Она используется для запуска сопрограмм верхнего уровня, которые работают в течение всего времени жизни приложения и не отменяются преждевременно.

Сопрограмма всегда выполняется в некотором контексте, представленном значением типа CoroutineContext. CoroutineContext представляет собой набор элементов для определения политики потоковой передачи, обработчика исключений, управления временем жизни сопрограммы. Мы можем использовать оператор плюса для объединения элементов CoroutineContext.

Есть три наиболее важных контекста Coroutine – Dispatchers, CoroutineExceptionHandler и Job.

Dispatchers определяют, какой поток запускает сопрограмму. Сопрограмма может переключать диспетчеров в любое время с помощью withContext(). Подразделяют несколько разновидностей Dispatchers:

1. Dispatchers.Default – использует общий фоновый пул потоков. По умолчанию максимальное количество потоков, используемых этим диспетчером, равно количеству ядер ЦП, но не менее двух
2. Dispatchers.IO – разделяет потоки с Dispatchers.Default, но количество потоков ограничено kotlinx.coroutines.io.parallelism. По умолчанию используется ограничение в 64 потока или количество ядер (в зависимости от того, что больше)
3. Dispatchers.Main – соответствует основному потоку Android
4. Dispatchers.Unconfined – диспетчер сопрограмм, не ограниченный каким-либо конкретным потоком. Сопрограмма сначала выполняется в текущем потоке и позволяет ей возобновить работу в любом потоке, который используется соответствующей функцией приостановки

CoroutineExceptionHandler отвечает за обработку не перехваченных исключений. Как правило, не перехваченные исключения могут быть только результатом сопрограмм, созданных с помощью запуска строителя. Сопрограмма, созданная с использованием async, всегда перехватывает все свои исключения и представляет их в результирующем объекте Deferred.

Job – управляет временем жизни сопрограммы. У задания есть следующие состояния: isActive, isCompleted, isCancelled. Можно использовать метод Job.isActive, чтобы узнать текущее состояние задания (см. рисунок 1.8):

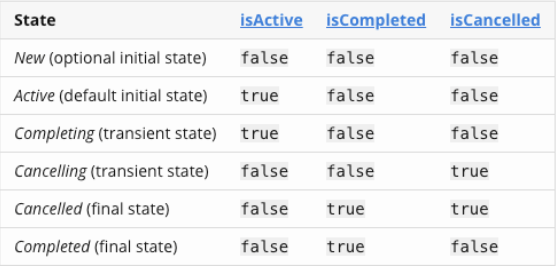


Рисунок 1.8 – Состояния Job

Можно также привести схему потока смены состояний (см. рисунок 1.9):

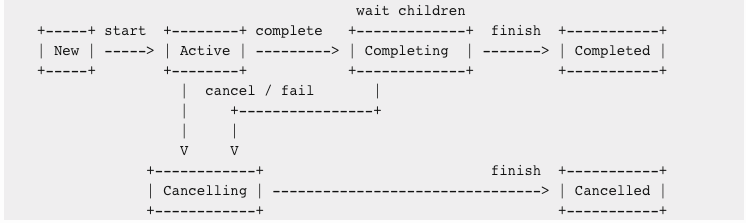


Рисунок 1.9 – Поток смены состояний Job

Задание активно, пока работает сопрограмма.

Если задание не выполнено, не считая исключения, оно отменяется. Задание можно отменить в любое время с помощью функции отмены, которая заставляет его немедленно перейти в состояние отмены.

Задание отменяется, когда завершается выполнение своей работы.

Родительское задание в состоянии завершения или отмены ожидает завершения всех своих дочерних задач перед завершением. Стоит обратить внимание, что состояние завершения является внутренним для задания. Для стороннего наблюдателя завершающее задание все еще активно, в то время как внутренне оно ожидает своих дочерних элементов.

**Безусловно, значимость Kotlin Coroutines очень велика и для мобильного приложения с интерактивной картой. Например, для отслеживания состояния приложения необходимо запустить отдельный поток, который будет регистрировать результаты работы тех или иных функций, после чего выводить для пользователя на интерфейс соответствующее сообщение. Отдельные потоки также могут понадобиться за тем, чтобы обновлять локальную базу данных с информацией о достопримечательностях в фоновом режиме, пока пользователь использует приложение.**

# ГЛАВА 2 ТехнологиИ разработки интерактивной карты

## **2.1 OpenStreetMap**

OpenStreetMap – это карта всего мира, которую может редактировать каждый. Данная карта создается практически с чистого листа по GPS-трекам и распространяется под свободной лицензией.

Данная технология уже была подробно рассмотрена в рамках курсовой работы, поэтому в данный момент рассмотрим более детально другую технологию – Mapbox. Mapbox базируется на картах OSM, но при этом является более функциональным и перспективным инструментом.

## **2.2 Mapbox**

Mapbox – американская технология пользовательских онлайн-карт для веб-сайтов и приложений, таких как Foursquare, Lonely Planet, Financial Times, The Weather Channel, Instacart Inc. и Snapchat. С 2010 года он быстро расширил нишу пользовательских карт в ответ на ограниченный выбор, предлагаемый поставщиками карт, такими как Google Maps.

Mapbox является создателем или значительным участником некоторых картографических библиотек и приложений с открытым исходным кодом, включая библиотеку JavaScript Mapbox GL-JS (с открытым исходным кодом до версии 2), спецификацию MBTiles, картографическую среду разработки TileMill, библиотеку JavaScript Leaflet, а также язык оформления карт и парсер CartoCSS.

Настройка карты для некоммерческих клиентов была предложена уже в 2010 году. Ранняя работа над инструментами OpenStreetMap, включая редактор iD, финансировалась за счет гранта в размере 575 000 долларов США от Knight Foundation.

11 июля 2016 г. MapQuest прекратил работу с открытым API тайлов, а такие пользователи, как GNOME Maps, были переведены на временно бесплатный уровень сервера тайлов Mapbox[10] при рассмотрении альтернатив.

В октябре 2017 года SoftBank инвестировал 164 миллиона долларов в Mapbox Inc. вместе с другими существующими инвесторами, включая венчурные компании Foundry Group, DFJ Growth, DBL Partners и Thrive Capital. В ноябре 2017 года Mapbox приобрела белорусский стартап Mapdata, специализирующийся на нейронных сетях.]В январе 2018 года Mapbox приобрела команду разработчиков механизма маршрутизации с открытым исходным кодом Valhalla.

В декабре 2020 года Mapbox выпустила вторую версию своей библиотеки JavaScript для онлайн-отображения карт, Mapbox GL JS. Ранее открытый исходный код под лицензией BSD, новая версия перешла на проприетарное лицензирование. Это привело к форку открытого исходного кода MapLibre GL и инициированию проекта MapLibre.

Таким образом, данные Mapbox берутся из открытых источников данных, таких как OpenStreetMap и NASA, а также из приобретенных проприетарных источников данных, таких как DigitalGlobe. Технология основана на Node.js, Mapnik, GDAL и Leaflet.

Mapbox использует анонимные данные телеметрии, такие как Strava и RunKeeper, для выявления вероятных отсутствующих данных в OpenStreetMap с помощью автоматических методов, а затем вручную применяет исправления или сообщает о проблеме участникам OSM.

В функционал мобильного приложения с интерактивной картой достопримечательностей БГУ входит расстановка объектов на карте, размещение различной информации об этих объектах, а также, в перспективе, возможность построения маршрутов между этими объектами. Помимо того, в проекте планируется организовать работу со слоями на карте. Например, отдельный слой карты для исторических достопримечательностей и отдельный слой карты для современных достопримечательностей. Всё это можно реализовать при помощи Mapbox.

## **2.3 Сравнительная оценка OpenStreetMap и Mapbox**

Для того, чтобы определить, какая из двух вышеупомянутых технологий разработки интерактивной карты больше подходит для поставленных задач, необходимо было провести их сравнительную оценку.

Преимущества и недостатки каждой из технологий приведены в таблице ниже (см. таблицу 2.1).

|  |  |  |
| --- | --- | --- |
| Технология | Преимущества | Недостатки |
| OpenStreetMap | * API OpenStreetMap является бесплатным * Карта с открытым исходным кодом | * Требует создания дополнительных сервисов * Ограниченное количество запросов |
| Mapbox | * Индивидуальность и гибкость * Быстрое время загрузки и отличная производительность * Автономный режим в API * Подход с открытым исходным кодом * Стандартизированная обработка данных | * Плохое покрытие карты в некоторых регионах * Достаточно долгий процесс изучения Mapbox API разработчиками |

Таблица 2.1 Сравнительная оценка технологий OpenStreetMap и Mapbox

На рисунках ниже представлены способы подключений технологий OpenStreetMap (см. рисунок 2.1) и Mapbox (см. рисунок 2.2).

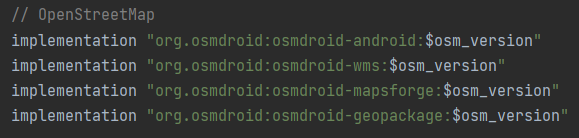


Рисунок 2.1 Пример подключения технологии OpenStreetMap в проект

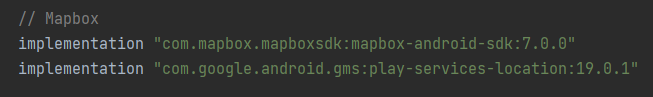


Рисунок 2.2 Пример подключения технологии Mapbox в проект

В результате удалось провести успешную миграцию с технологии OpenStreetMap (см. рисунок 2.3) на технологию Mapbox (см. рисунок 2.4).

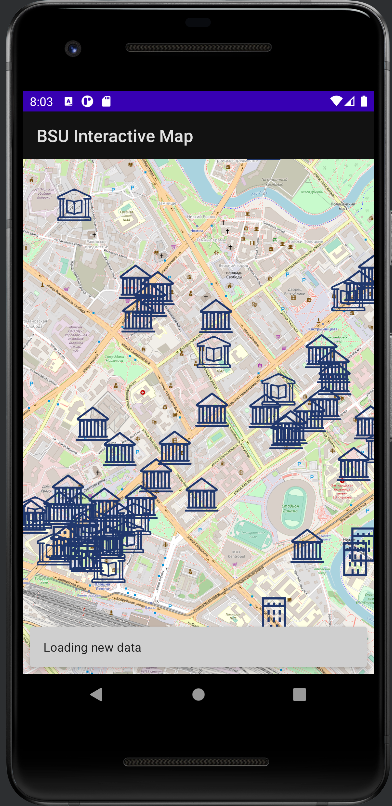


Рисунок 2.3 Пример запуска приложения с помощью технологии OpenStreetMap

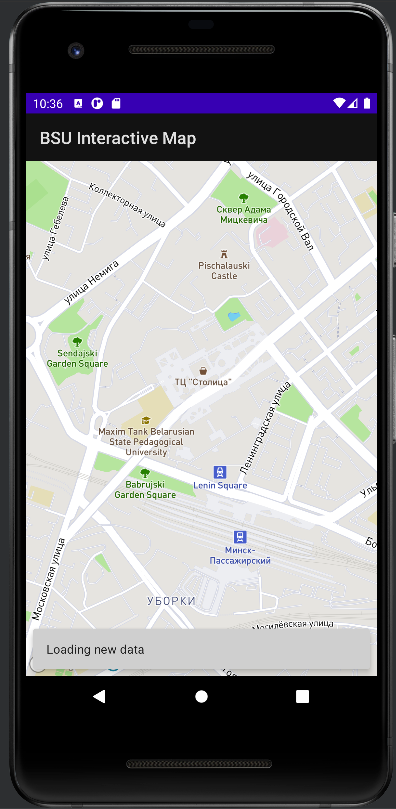


Рисунок 2.4 Пример запуска приложения с помощью технологии Mapbox

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы удалось изучить множество различных технологий разработки мобильных приложений под операционную систему Android (Kotlin, Kotlin Coroutines, WorkManager, Retrofit, Okhttp, Gson, RoomDB, Koin), а также совершить сравнительную оценку технологий разработки интерактивных карт OpenStreetMap и Mapbox, после чего внедрить изученные технологии в дипломный проект, связанный с разработкой интерактивной карты.

Опыт, полученный в ходе производственной преддипломной практики на кафедре, поможет в дальнейшей разработке дипломного проекта, который базируется на разработке интерактивной карты.

Данный проект включает в себя ряд технологий, используемых Android-разработчиками, включая Mapbox. В частности, в приложении происходит совместная работа с API, базой данных, внедрением зависимостей, картами и другими библиотеками.

С помощью текущей версии приложения пользователь может узнать информацию о местоположении различных достопримечательностей (как исторических, так и современных), а также узнать информацию о них при помощи клика по маркеру на карте.

Стоит отметить, что хранение данных в приложении уже реализовано на разных уровнях абстракции. В приложении есть как доменные сущности, так и сущности базы данных. Это позволит облегчить модификацию приложения в будущем, так как происходит разделение структуры сущностей от деталей их реализации.

За счёт использования локальной базы данных, можно кешировать данные на мобильном устройстве пользователя, благодаря чему улучшается производительность приложения. Помимо этого, происходит регулярное обновление данных в фоновом режиме.

В будущем планируется добавить в проект режимов экскурсий по достопримечательностям, а также экскурсия в дополненной реальности во внутреннем дворике БГУ. С помощью функционала технологии Mapbox данные модификации могут быть реализованы.

Работа над данным проектом продолжится в ближайшее время. В будущем будет реализовано больше возможностей для взаимодействия с приложением.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Android for Developers [Электронный ресурс] / Режим доступа: <https://developer.android.com/> – Дата доступа: 09.02.2022
2. Android Programming: The Big Nerd Ranch Guide (Big Nerd Ranch Guides) 3rd Edition : учеб.-метод. пособие / Bill Phillips, Chris Stewart, Kristin Marsicano, 2017
3. Head First Android Development: A Brain-Friendly Guide 2nd Edition : учеб.-метод. пособие / Dawn Griffiths, David Griffiths, 2017
4. Professional Android 4th Edition : учеб.-метод. пособие / Reto Meier, Ian Lake, 2018
5. Android Programming: Pushing the Limits 1st Edition, Kindle Edition / учеб.-метод. пособие / Erik Hellman, 2013
6. The Busy Coder's Guide to Advanced Android Development 0002- Edition / учеб.-метод. пособие / Mr. Mark L Murphy, 2011
7. Android Design Patterns / учеб.-метод.пособие / Greg Nudelman, 2013
8. Android – OpenStreetMap Wiki [Электронный ресурс] / Режим доступа: <https://wiki.openstreetmap.org/wiki/Android> – Дата доступа: 07.03.2022
9. Mapbox Wikipedia [Электронный ресурс] / Режим доступа: <https://en.wikipedia.org/wiki/Mapbox> – Дата доступа: 12.03.2022