**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра компьютерных технологий и систем**

**РАЗРАБОТКА ИНТЕРАКТИВНОЙ КАРТЫ**

Курсовой проект

Жуковского Павла Сергеевича

Студента 4 курса,

Специальность

«прикладная информатика»

Научный руководитель:

Старший преподаватель кафедры КТС

Кулинкович В. А.

Минск, 2021

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc90212223)

[ГЛАВА 1 Технологии разработки мобильных приложений под операционную систему Android 4](#_Toc90212224)

[1.1 WorkManager 4](#_Toc90212225)

[1.2 Dependency Injection. Koin 6](#_Toc90212226)

[1.3 Retrofit2. Okhttp3 8](#_Toc90212227)

[1.4. Room Database 10](#_Toc90212228)

[1.5. Kotlin Coroutines 13](#_Toc90212229)

[ГЛАВА 2 Технология разработки интерактивной карты Openstreetmap 17](#_Toc90212230)

[2.1 Значение и потенциальное применение OSM 17](#_Toc90212231)

[2.2 Разработка собственного приложения с помощью OSM 19](#_Toc90212232)

[ЗАКЛЮЧЕНИЕ 27](#_Toc90212233)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc90212234)

[ПРИЛОЖЕНИЕ А 29](#_Toc90212235)

# ВВЕДЕНИЕ

Данная курсовая работа посвящена разработке интерактивной карты с различными достопримечательностями БГУ, как научно-исследовательскими, так и учебными. Эта карта содержит в себе информацию о местоположении каждой из достопримечательностей. Такая карта позволит пользователю быстро и удобно понять, где расположена та или иная достопримечательность, а также получить некоторую информацию о ней.

Разработка данной карты безусловно является достаточно актуальной и значимой. Её можно было бы использовать во многих сферах жизни: научно-исследовательской, исторической, туристической и многих других. Более того, проект с разработкой данной карты будет несложно модифицировать и улучшать в будущем, тем самым добавляя в него намного больше возможностей и функционала.

Конкретно этот проект будет отвечать за мобильное приложение с этой интерактивной картой под операционную систему Android. Основная задача мобильного приложения – получить данные о достопримечательностях с веб-сервера и расположить их на интерфейсе пользователя, а также добавить функционал и различные способы взаимодействия с элементами карты. Таким образом, пользователи смогут пользоваться функционалом карты прямо с мобильного устройства, что может быть очень востребовано, например, в туристической сфере. Если говорить про пользователей операционной системы iOS, то в рамках проекта bsu-map для них будет реализовано отдельное мобильное приложение другим разработчиком.

Для создания мобильного приложения с данной интерактивной картой могут понадобиться некоторые технологии Android-разработки, такие как: Kotlin Coroutines, Room Database, Retrofit2, Okhttp3, Koin, WorkManager и многие другие. Однако основополагающей технологией является OpenStreetMap, которая позволяет работать с картами, метками и даже маршрутами. Комбинация вышеуказанных технологий позволит разработать мобильное приложение с интерактивной картой, создание которого планируется в данной курсовой работе.

# 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. рис. 1.1):

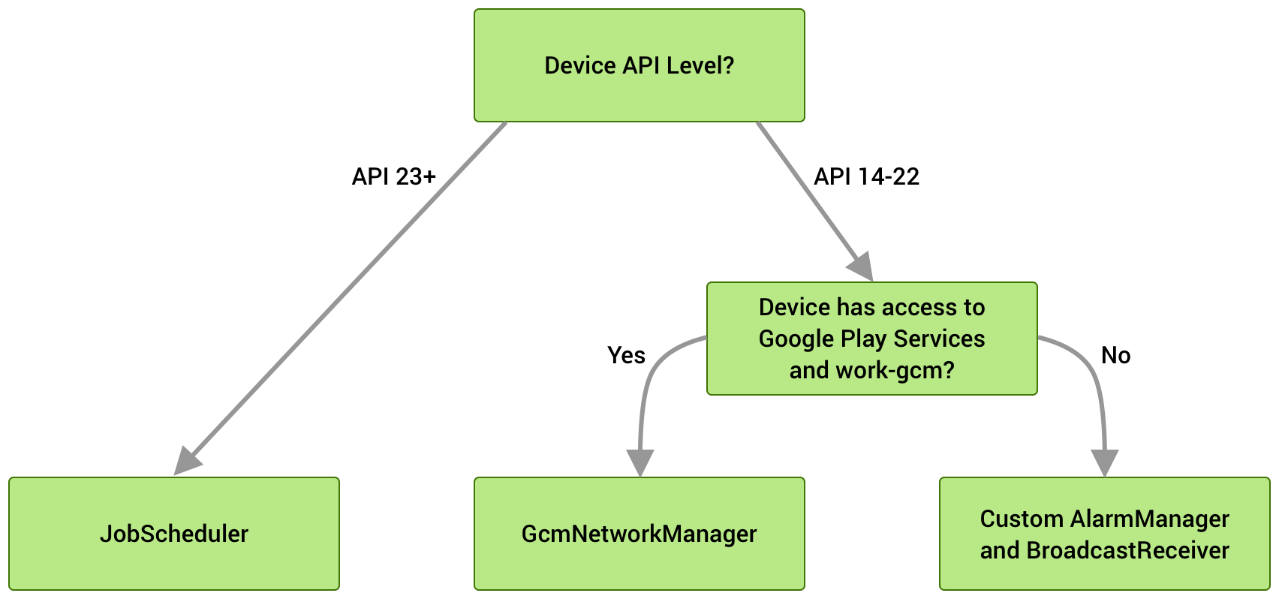


Рисунок 1.1 - Базовая служба диспетчеризации заданий WokManager

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.

## 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.

## 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.**

## 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. рис. 1.2):**

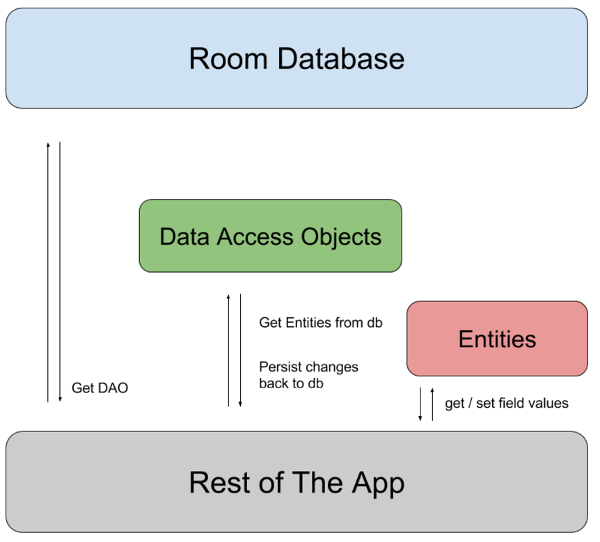


Рисунок 2.2 – Взаимосвязь между компонентами Room

**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.**

## 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. рис. 1.3):

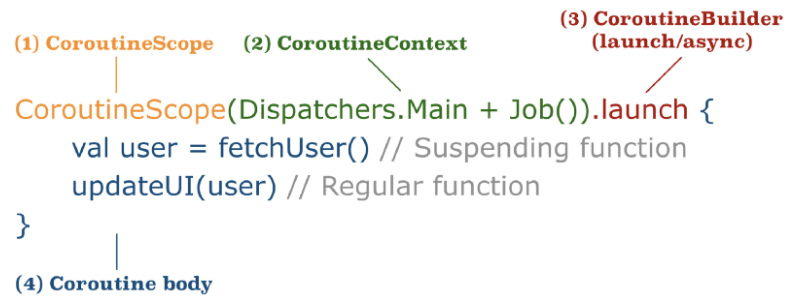


Рисунок 1.3 – Компоненты Kotlin Coroutines

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. рис. 1.4):

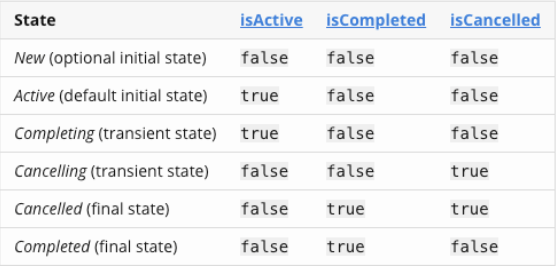


Рисунок 1.4 – Состояния Job

Можно также привести схему потока смены состояний (см. рис. 1.5):

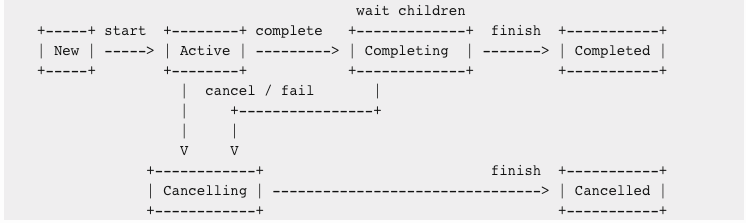


Рисунок 1.5 – Поток смены состояний Job

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 Технология разработки интерактивной карты Openstreetmap

## 2.1 Значение и потенциальное применение OSM

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

Лицензия OpenStreetMap позволяет свободно (или почти свободно) получать доступ ко всем её растровым картам и всем лежащим в их основе пространственным данным. Этот проект направлен на поощрение нового и интересного использования этих данных.

Взглянуть на карту OSM достаточно просто. Создание карты ещё не завершено, но можно заметить, что некоторые места уже прорисованы довольно детально. Отдельные люди и целые коммерческие компании постоянно помогают в улучшении и детализации карт. Данный проект также не является исключением и способствует улучшению OSM.

OSM – это не просто программный проект. Что бы создать карты, технология отрывается от экранов компьютеров пользователей и исследует города и деревни, делаем новую съёмку местности. Это делается коллективными усилиями сообщества (на данный процесс затрачивается очень много времени). Для этого используется wiki-подобное программное обеспечение, которое позволяет совместно редактировать карты. Это значит, что карты все время становятся больше и лучше. Если у вас есть GPS-приемник, вы можете сделать свой вклад в проект, загрузив любые записанные вами треки. Кроме того, мы можем рисовать при помощи спутниковых снимков Bing! Вы можете начать редактировать карту прямо сейчас при помощи онлайн-редактора Potlatch или скачав программы JOSM или Merkaartor.

Множество программных разработок продвигают этот проект в разных направлениях. Как уже упоминалось выше, создаются различные инструменты для редактирования карт. На самом деле OpenStreetMap живёт за счет энтузиазма и программного обеспечения с открытым исходным кодом, начиная от интерфейса карты и вплоть до лежащего в основе протокола доступа к данным (интерфейс веб-сервиса для чтения и записи данных карты). Существует много возможностей для дополнительных проектов, работающих с данными OSM и использующими их, но данной технологии также необходима помощь в исправлении ошибок и добавлении возможностей в её функционал.

Если сравнивать OSM и достаточно популярные Google Maps, то можно выделить достаточно весомое преимущество у первой технологии – открытый исходный код и свободный доступ к растровым картам. Например, при использовании Google Maps приходится выделять намного больше экономических ресурсов на проект, который их использует. В этом плане OSM выигрывает у Google Maps. Это – основная причина, почему для разработки мобильного приложения с интерактивной картой была выбрана именно OSM.

Любые изменения в базе OSM соотносятся с так называемыми наборами изменений (changesets), которые могут быть отменены целиком или частично.

В OSM хранится вся история изменений, потому любой объект или набор объектов может быть возвращён к предыдущему состоянию на любую дату, а текущее состояние данных может быть сверено с предыдущим как по свойствам, так и по геометрии.

Существуют сервисы, анализирующие изменения за определённый последний период времени, показывающие характер правок (внесены новые объекты, отредактированы или удалены имеющиеся). Примерами таких сервисов могут быть: OSMCha, WHODIDIT, AChaVi.

Проект OSM существует благодаря участникам, знакомым с местностью, где они живут, а потому вандализм и саботаж, как правило, могут быть замечены, проанализированы и нейтрализованы в сравнительно короткий срок.

Также позволяют выявлять намеренное или случайное нарушение целостности данных так называемые "валидаторы" – сервисы, которые анализируют логическую целостность данных OSM (топологию административных границ и т.п.).

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 OpenStreetMap.

## 2.2 Разработка собственного приложения с помощью OSM

Для работы с OpenStreetMap в Android существует соответствующая библиотека. Для того, чтобы использовать функционал OSM в мобильном приложении, необходимо импортировать её в файл build.gradle:

*// OpenStreetMap version*

*ext.osm\_version = "6.1.10"*

*// OpenStreetMap*

*implementation "org.osmdroid:osmdroid-android:$osm\_version"*

*implementation "org.osmdroid:osmdroid-wms:$osm\_version"*

*implementation "org.osmdroid:osmdroid-mapsforge:$osm\_version"*

*implementation "org.osmdroid:osmdroid-geopackage:$osm\_version"*

Проект с мобильным приложением под операционную систему Android, реализующий в себе интерактивную карту с достопримечательностями БГУ, включает в себя следующий список зависимостей:

1. Kotlin

ext.kotlin\_version = "1.5.31"

1. Build Gradle

ext.build\_gradle\_version = "7.0.3"

1. UnitTest

ext.unit\_test\_version = "4.13.2"

1. Work Manager

ext.work\_version = "2.6.0"

1. Koin version

ext.koin\_version = "3.1.2"

1. RecycleView

ext.recycle\_view\_version = "1.2.1"

1. Gson

ext.gson\_version = "2.8.7"

1. Retrofit2

ext.retrofit2\_version = "2.6.0"

1. Okhttp3

ext.okhttp3\_version = "3.12.0"

1. Room DB

ext.room\_version = "2.3.0"

1. Kotlin Coroutines

ext.kotlinCoroutineVersion = "1.4.3"

1. Layout

ext.constraint\_layout\_version = "2.1.0"

ext.coordinator\_layout\_version = "1.1.0"

ext.lifecycle\_livedata\_ktx\_version = "2.3.1"

ext.cardview\_version = "1.0.0"

ext.material\_version = "1.4.0"

1. ViewModel

ext.lifecycle\_viewmodel\_ktx\_version = "2.3.1"

1. Fragments

ext.fragments\_ktx\_version = "1.3.6"

1. Picasso for Image Loading

ext.picasso\_version = "2.71828"

1. SwipeRefreshLayout

ext.swipe\_refresh\_layout\_version = "1.1.0"

1. Lifecycles only (without ViewModel or LiveData)

ext.lifecycle\_version = "2.4.0-alpha02"

1. OpenStreetMap

ext.osm\_version = "6.1.10"

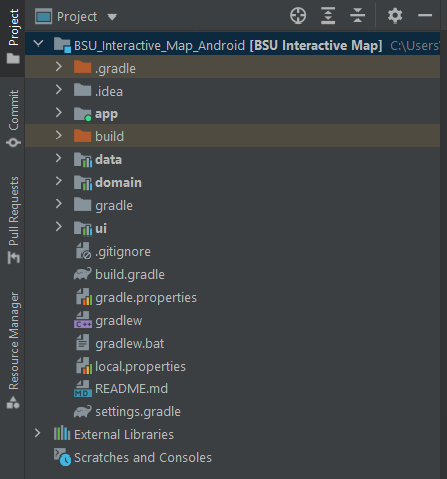
В структуру проекта входят 4 основных модуля (см. рис. 2.1): app, data, domain и ui.

Рисунок 2.1 – Структура проекта bsu-map

В модуле app хранится класс WorkManagerApplication, который отвечает за регулярное обновление данных с достопримечательностями в фоновом режиме за счёт технологии WorkManager. На данный момент, приложение обновляет данные для пользователя только раз в сутки и только, если мобильное устройство имеет доступ к сети Wi-Fi, а также подключено к зарядному устройству. В будущем эту настройку можно реализовать так, чтобы для пользователя она была опциональной.

В модуле data хранится, в первую очередь, файл KoinDataModule.kt, который содержит в себе все зависимости этого модуля. Благодаря технологии Koin, мы можем не думать о том, где нам инициализировать переменные или объекты тех или иных классов, а просто использовать метод inject().

В модуле data также хранится класс реализации репозитория – DataRepositoryImpl.kt, в котором хранится объект базы данных Room DB. Помимо этого, есть ряд data-классов для сущностей JSON-данных. Среди них: BuildingItemJson.kt, StructuralItemObjectJson.kt, BuildingItemAddressJson.kt и другие.

Интерфейс MapDataApi, который также находится в модуле data, работает с Retrofit и берёт данные с веб-сервера. Это делается с помощью следующего кода:

*interface MapDataApi{*

*@GET("api/buildings/all")*

*suspend fun getData() : List<BuildingItemJson>?*

*}*

Помимо этого, в data модуле есть класс базы данных (основанный на технологии Room DB), а также классы-сущности этой базы данных. Помимо этих классов-сущностей, существуют также классы, отвечающие за преобразование одних сущностей в другие (BuildingItemDBMapper.kt, AddressItemDBMapper.kt, iconItemDBMapper.kt и другие). В нашем случае, происходит преобразование сначала из JSON формата в формат доменных сущностей, а затем из доменных сущностей в сущности для Room DB, которые он сохраняет в базу данных на основе SQL запросов.

В модуле domain хранятся доменные сущности классов с данными. Этот уровень абстракции находится между JSON форматом и форматом базы данных. Данный уровень нужен для того, чтобы отделить детали реализации друг для друга. Это позволит облегчить модификацию проекта в будущем, если она понадобится.

В модуле ui хранятся все классы, которые отвечают за работу с интерфейсом пользователя. В частности, есть KoinUiModule.kt, который хранит в себе зависимости модуля ui. Помимо этого, есть класс MapViewModel, который работает с данными репозитория и предоставляет их классу фрагмента. Класс фрагмента MapFragment.kt отвечает за развёртывание OSM карты на интерфейсе пользователя, а также за основной его функционал. Есть также класс LoadState.kt, в котором хранятся состояния приложения, что понадобится для отслеживания ошибок.

В модуле интерфейса пользователя также хранятся все ресурсы проекта. Среди них: xml-файлы с векторным изображением иконок для маркеров карты; xml-файлы, описывающие расстановку элементов интерфейса; xml-файлы со строковыми ресурсами и значениями цветов.

MainActivity.kt располагается в ui модуле проекта. Ниже приведён исходный код файла MainActivity.kt, в котором выполняются стартовые команды приложения:

*class MainActivity : AppCompatActivity() {*

*private lateinit var binding: ActivityMainBinding*

*override fun onCreate(savedInstanceState: Bundle?) {*

*super.onCreate(savedInstanceState)*

*binding = ActivityMainBinding.inflate(layoutInflater)*

*val view = binding.root*

*setContentView(view)*

*if (savedInstanceState == null)*

*{*

*supportFragmentManager.beginTransaction()*

*.replace(R.id.fragment\_container, MapFragment())*

*.commit()*

*}*

*}*

*}*

Взаимодействие с OSM реализовано во фрагменте, реализация которого хранится в файле MapFragment.kt (листинг с исходным кодом этого файла приведён в приложении А). Детали реализации описаны в этом фрагменте.

На данный момент приложение позволяет пользователю просматривать местоположение достопримечательностей на карте (см. рис. 2.2.), а также менять масштаб карты и взаимодействовать с маркерами. Пользователь также имеет возможность кликнуть на какой-либо маркер, чтобы получить некоторую информацию о нём (см. рис. 2.3). Помимо этого, можно получить дополнительную информацию о выбранной достопримечательности, перейдя в соответствующий раздел с помощью кнопки в диалоговом окне маркера.



Рисунок 2.2 – Демонстрация приложения bsu-map

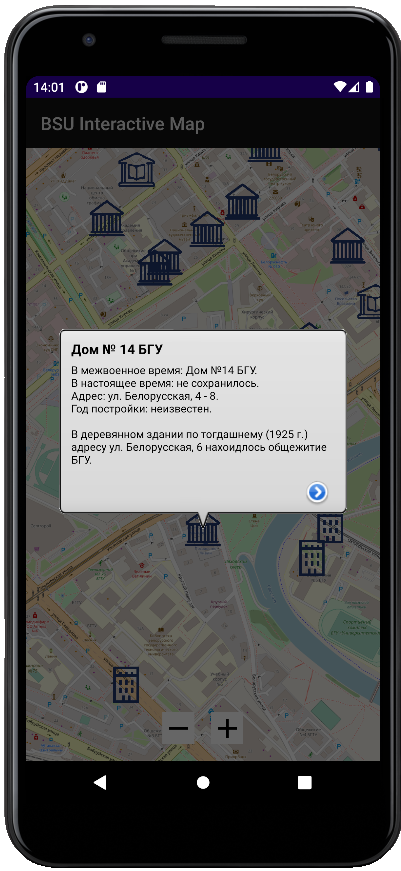


Рисунок 2.3 – Демонстрация приложения bsu-map (взаимодействие с маркером)

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы удалось реализовать мобильное приложение под операционную систему Android, которое работает с картами OpenStreetMap и отображает информацию о достопримечательностях БГУ.

Данное приложение включает в себя ряд технологий, используемых Android-разработчиками, включая OpenStreetMap. 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 возможность построения маршрутов для пользователя, чтобы облегчить ознакомление с достопримечательностями. Помимо маршрутов, также планируется добавление нескольких слоёв для отображения достопримечательностей в зависимости от временной шкалы (по годам). Данная функция может быть очень полезна тем, кто желает изучить историю БГУ с помощью данного приложения. Также, в приложении есть потенциал к добавлению режима просмотра достопримечательностей в 3D формате.

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: 14.11.2021
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: 23.11.2021

# ПРИЛОЖЕНИЕ А

**Листинг файла MapFragment.kt**

package com.example.ui.fragments

import android.os.Bundle

import android.preference.PreferenceManager

import android.util.Log

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import androidx.core.content.res.ResourcesCompat

import androidx.fragment.app.Fragment

import androidx.lifecycle.Lifecycle

import androidx.lifecycle.lifecycleScope

import androidx.lifecycle.repeatOnLifecycle

import com.example.domain.BuildingItem

import com.example.ui.R

import com.example.ui.databinding.MapBinding

import com.example.ui.viewModels.LoadState

import com.example.ui.viewModels.MapViewModel

import com.google.android.material.snackbar.BaseTransientBottomBar

import com.google.android.material.snackbar.Snackbar

import kotlinx.coroutines.flow.collect

import kotlinx.coroutines.launch

import org.koin.core.component.KoinComponent

import org.koin.core.component.inject

import org.osmdroid.config.Configuration

import org.osmdroid.tileprovider.tilesource.TileSourceFactory

import org.osmdroid.util.GeoPoint

import org.osmdroid.views.CustomZoomButtonsController

import org.osmdroid.views.MapView

import org.osmdroid.views.overlay.OverlayItem

class MapFragment : Fragment(), KoinComponent {

private val viewModel: MapViewModel by inject()

private var \_binding: MapBinding? = null

private val binding get() = \_binding!!

private lateinit var map: MapView

override fun onCreateView(

inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?

): View {

\_binding = MapBinding.inflate(inflater, container, false)

return binding.root

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

val ctx = view.context

Configuration.getInstance().load(ctx, PreferenceManager.getDefaultSharedPreferences(ctx))

map = view.findViewById(R.id.map)

map.zoomController.setVisibility(CustomZoomButtonsController.Visibility.ALWAYS)

map.setTileSource(TileSourceFactory.MAPNIK)

map.setMultiTouchControls(true)

val pointBelarus = GeoPoint(53.894,27.547)

map.controller.setCenter(pointBelarus)

map.controller.setZoom(12.0)

var dataList: List<BuildingItem> = emptyList()

lifecycleScope.launch {

repeatOnLifecycle(Lifecycle.State.STARTED) {

viewModel.dataFlow.collect {

dataList = it

setMarkers(dataList)

Log.d("JSON DATA", "Json data, received by MapViewModel:\n$dataList")

}

}

}

val timeLength = if (dataList.isEmpty()) // If not data in general (even in database)

Snackbar.LENGTH\_INDEFINITE

else { // If no NEW data, but we have OLD data in database

Snackbar.LENGTH\_LONG

}

lifecycleScope.launch {

repeatOnLifecycle(Lifecycle.State.STARTED) {

viewModel.state.collect {

when (it) {

LoadState.LOADING -> {

createSnackbar(resources.getString(R.string.loading),

requireContext().getColor(R.color.black))

}

LoadState.SUCCESS -> {

createSnackbar(resources.getString(R.string.loadingSuccess),

requireContext().getColor(R.color.black))

setMarkers(dataList)

}

LoadState.INTERNET\_ERROR -> {

createSnackbarWithReload(timeLength,

resources.getString(R.string.errorNetwork))

}

LoadState.UNKNOWN\_ERROR -> {

createSnackbarWithReload(timeLength,

resources.getString(R.string.errorUnknownNoData))

}

LoadState.EMPTY\_DATA\_ERROR -> {

createSnackbarWithReload(timeLength,

resources.getString(R.string.errorNoNewsOnAPI))

}

else -> createSnackbar(resources.getString(R.string.launching),

requireContext().getColor(R.color.black))

}

}

}

}

if (savedInstanceState == null && viewModel.state.value == LoadState.IDLE) {

viewModel.loadData()

}

}

override fun onDestroyView() {

super.onDestroyView()

\_binding = null

}

private fun setMarkers(dataList: List<BuildingItem>) {

for (item in dataList) {

val marker = ResourcesCompat.getDrawable(resources, resources

.getIdentifier("ic\_" + item.markerPath?.substringBefore('.'),

"drawable", activity?.packageName), null)

val overlayItemWindow = MapItemizedOverlay(

marker, requireContext()

)

val overlayItem = OverlayItem(

item.name, item.address?.description, GeoPoint(

item.address!!.latitude!!.toDouble(), item.address!!.longitude!!.toDouble())

)

overlayItemWindow.addOverlay(overlayItem)

map.overlays.add(overlayItemWindow)

}

}

private fun createSnackbar(message: String, color: Int) {

var timeLength = BaseTransientBottomBar.LENGTH\_LONG

if (message == getString(R.string.loading))

timeLength = BaseTransientBottomBar.LENGTH\_INDEFINITE

val snackbar = Snackbar.make(

binding.mapContainer,

message,

timeLength

)

snackbar.setTextColor(color)

snackbar.show()

}

private fun createSnackbarWithReload(snackbarTimeLength: Int, messageError: String) {

val snackbar = Snackbar.make(

binding.mapContainer,

messageError,

snackbarTimeLength

)

snackbar.setTextColor(requireContext().getColor(R.color.black))

snackbar.setActionTextColor(requireContext().getColor(R.color.black))

snackbar.setAction(R.string.reload) {

viewModel.loadData()

}

snackbar.show()

}

override fun onResume() {

super.onResume()

map.onResume()

}

override fun onPause() {

super.onPause()

map.onPause()

}

}