Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по производственной практике

Выполнил:

Студент группы 893551

Драгун О.В.

Руководитель от БГУИР:  
Летохо А. С.  
Руководититель от предприятия:  
Савельева О. А.

Минск 2022

Оглавление

[Введение 3](#_Toc103434123)

[Структура предприятия 3](#_Toc103434124)

[Специализация 4](#_Toc103434125)

[Индивидуальное задание 5](#_Toc103434126)

[Сведения об используемых технологиях и проекте 6](#_Toc103434127)

[Проект 6](#_Toc103434128)

[Команда 6](#_Toc103434129)

[Ключевые технологии на стороне android 6](#_Toc103434130)

[Реализация индивидуального задания, часть Б 10](#_Toc103434131)

[Работа с Fingerprint и включение Push Kit в AppGallery Connect 10](#_Toc103434132)

[Написание необходимого кода 12](#_Toc103434133)

[Позитивное тестирование 15](#_Toc103434134)

[Реализация индивидуального задания, часть Г 16](#_Toc103434135)

[Вывод 22](#_Toc103434136)

[Список литературы 23](#_Toc103434137)

[Приложение (Листинг кода главного класса) 24](#_Toc103434138)

# Введение

## Структура предприятия

ScienceSoft – международная IT-компания с офисами в Америке, Европе и Азии. Мы разрабатываем программное обеспечение и занимаемся IT-консалтингом, предлагая клиентам своё видение современных технологических решений. ScienceSoft работает с компаниями из 60 стран, но основная часть наших партнёров и заказчиков находится в США, странах ЕС и Персидского залива.

ScienceSoft более 33 лет занимается разработкой программных решений. На протяжении этого времени компания развивалась вместе с мировой сферой IT, перенимая лучшие технологии и практики и создавая свои собственные.

Наш опыт охватывает различные направления IT, от тестирования и QA до аналитики данных и информационной безопасности. Помимо кастомной разработки продуктов, мы занимаемся консалтингом, предлагая свои экспертные знания в цифровой трансформации бизнеса, формировании IT-стратегии, кибербезопасности. Мы развиваемся и в таких актуальных технологиях, как IoT, AI (Deep Learning), AR и VR. В нашем портфолио – проекты для компаний из медицины, промышленности, банковского сектора, ритейла, сферы телекоммуникаций и индустрии развлечений.

## Специализация

Я специализируюсь на разработке нативных android-приложений.

Нативная разработка — это создание продукта, который пишется на оригинальных языках программирования, созданных специально для выбранной платформы. Например, родными языками для Android являются Java и Kotlin, для iOS - Swift и Objective-C. Нативное приложение будет работать только на “своей” платформе. Кроссплатформенные приложения могут работать сразу на нескольких операционных системах. Для этого используются специализированные кроссплатформенные фреймворки, например Flutter или React-Native.

Нативное приложение всегда будет выглядеть лучше, чем то, что разработали по мультиплатформенной технологии. Дизайн, скорость загрузки, доступ ко всем функциям устройства (камера, геолокация, календарь и так далее), интерфейс – все это будет давать нативной разработке сто очков вперед. Кроссплатформенные приложения в этом плане уступают нативным – работают медленнее, а интерфейс значительно отличается.

Главным достоинством кроссплатформенного подхода является то, что скорость разработки выше, нежели у нативной, а времени и ресурсов затрачивается меньше.

Из обилия кроссплатформенных фреймворков я решил выбрать нативную разработку на kotlin, с которым работаю уже более полутора лет.

## Индивидуальное задание

а) Ознакомиться с проектом, командой и используемыми технологиями

б) В существующее мобильное приложение telecom-оператора необходимо добавить поддержку push-нотификаций для платформы Huawei и магазина приложений App Gallery.

Высокоуровневые требования:

1. Реализация должна работать так же стабильно, как текущая имплементация для google-сервисов и Play Market
2. Написанный код должен быть разделен при помощи сборочных скриптов таким образом, чтобы google-часть никак не конфликтовала с Huawei-частью
3. Пуши должны поддерживать Deep-links, картинки, external-ссылки так же, как текущая имплементация для Play Market и google-сервисов

в) Произвести позитивное тестирование и исправление ошибок

г) Найти плагин для создания boilerplate-классов в архитектуре MVP-C и интегрировать его в проект

# Сведения об используемых технологиях и проекте

## Проект

Заказчик - telecom-оператор. Я непосредственно работаю над android-приложением с 500000+ месячных активных пользователей. Оно позволяет просматривать баланс, активировать тарифы и услуги, смотреть детализацию и пр.

## Команда

За Backend отвечает заказчик. На нашей стороне весь код, касающийся платформы android и ios

## Ключевые технологии на стороне android

**IDE: Android studio.**

Android Studio — интегрированная среда разработки для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O.

Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, — официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, macOS и GNU/Linux. 17 мая 2017, на ежегодной конференции Google I/O, Google анонсировал поддержку языка Kotlin, используемого в Android Studio, как официального языка программирования для платформы Android в дополнение к Java и С++.

**100% кода на Kotlin.**

Kotlin — статически типизированный, объектно-ориентированный язык программирования, работающий поверх Java Virtual Machine и разрабатываемый компанией JetBrains. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM.

Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala. Следствием упрощения по сравнению со Scala стали также более быстрая компиляция и лучшая поддержка языка в IDE. Язык полностью совместим с Java, что позволяет Java-разработчикам постепенно перейти к его использованию; в частности, язык также встраивается Android, что позволяет для существующего android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

Сборка при помощи Gradle.

**Dependency injection – Dagger 2.**

Dagger — это полностью статическая среда внедрения зависимостей во время компиляции для Java, Kotlin и Android. Это адаптация более ранней версии, созданной Square и поддерживаемой Google.

Последний выпуск Dagger: Dagger 2.41.

Dagger стремится решить многие проблемы разработки и производительности, которые мешали решениям, основанным на отражении(reflection).

**Concurrency на Kotlin Coroutines.**

Некоторые API инициируют долго протекающие операции (такие как сетевой ввод-вывод, файловый ввод-вывод, интенсивная обработка на CPU или GPU и др.), которые требуют блокировки вызывающего кода в ожидании завершения операций. Сопрограммы обеспечивают возможность избежать блокировки исполняющегося потока путём использования более дешёвой и управляемой операции: приостановки (suspend) сопрограммы.

Сопрограммы упрощают асинхронное программирование, оставив все осложнения внутри библиотек. Логика программы может быть выражена последовательно в сопрограммах, а базовая библиотека будет её реализовывать асинхронно для нас. Библиотека может обернуть соответствующие части кода пользователя в обратные вызовы (callbacks), подписывающиеся на соответствующие события, и диспетчировать исполнение на различные потоки (или даже на разные машины!). Код при этом останется столь же простой, как если бы исполнялся строго последовательно.

Многие асинхронные механизмы, доступные в других языках программирования, могут быть реализованы в качестве библиотек с помощью сопрограмм Kotlin. Это включает в себя async/await из C# и ECMAScript, channels и select из языка Go, и generators /yield из C# или Python.

**Networking на okhttp.**

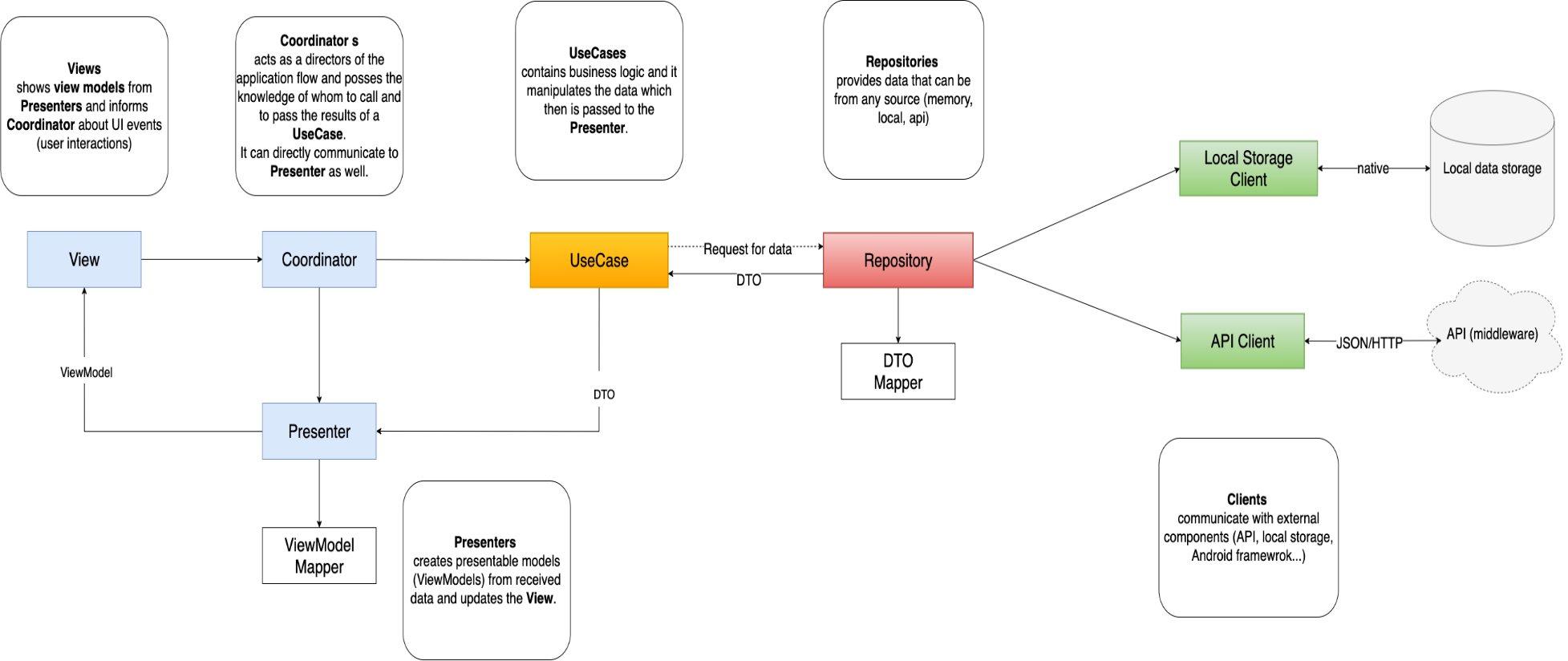
OkHttp — это HTTP-клиент, который по умолчанию эффективен:

1. Поддержка HTTP/2 позволяет всем запросам к одному и тому же хосту использовать общий сокет.
2. Пул соединений уменьшает задержку запросов (если HTTP/2 недоступен).
3. Прозрачный GZIP уменьшает размер загрузки.
4. Кэширование ответов полностью избегает сети для повторных запросов.

OkHttp продолжает работать, когда в сети возникают проблемы: он автоматически восстанавливается после распространенных проблем с подключением. Если у вашей службы несколько IP-адресов, OkHttp попытается использовать альтернативные адреса, если первое соединение не удастся. Это необходимо для IPv4+IPv6 и служб, размещенных в резервных центрах обработки данных. OkHttp поддерживает современные функции TLS (TLS 1.3, ALPN, закрепление сертификата). Его можно настроить на откат для широкого подключения.

Использовать OkHttp легко. Его API запроса/ответа разработан с использованием быстрых компоновщиков и неизменяемости. Он поддерживает как синхронные блокирующие вызовы, так и асинхронные вызовы с обратными вызовами.

**Архитектура Приложения - MVP-C**



# Реализация индивидуального задания, часть Б

## Работа с Fingerprint и включение Push Kit в AppGallery Connect

Отпечаток сертификата подписи используется для проверки подлинности приложения. Перед выпуском приложения необходимо создать отпечаток сертификата подписи локально на основе сертификата подписи и настроить его в AppGallery Connect.

Простыми словами Push-service на девайсах с ОС Huawei убеждаются, что разосланные уведомления попадут только к необходимым получателям только внутри нашего приложения. Чтобы нельзя было отправить пуш в чужое приложение на устройстве пользователя.

Прежде чем сделать это, убедимся, что выполнены следующие условия:

Мы создали сертификат подписи приложения.

JDK был установлен на вашем компьютере.

Чтобы сгенерировать отпечаток сертификата подписи, выполняем следующие действия:

1. Открываем интерфейс командной строки (CLI) с помощью команды cmd и запускаем команду cd, чтобы перейти в каталог, где находится keytool.exe.

*cd C:\Program Files\Java\jdk\bin*

1. Запуcкаем *keytool -list -v -keystore <keystore-file>* и отвечаем на запрос. В команде <keystore-file> указывает абсолютный путь к сертификату подписи приложения.

Пример: *keytool -list -v -keystore C:\TestApp.jks*

1. Получите отпечаток сертификата SHA-256.

Изображение выглядит как текст

Автоматически созданное описание

1. В AppGallery Connect вводим полученный фингерпринтИзображение выглядит как текст

   Автоматически созданное описание
2. Скачиваем agconnect-services.json для встраивания в проект

## Написание необходимого кода

Ключевые необходимые вещи:

1. Отправить push notification токен (pnToken) с девайса в момент авторизации в приложение
2. В случае получения по pnToken пуша, распарсить информацию и показать уведомление даже в случае закрытого приложения.
3. Никак не затронуть google-часть

Добавляем во флейвор хуавея в manifest service для приема пушей

<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.nda">  
  
 <application>  
 <meta-data  
 android:name="push\_kit\_auto\_init\_enabled"  
 android:value="true" />  
 <service  
 android:name=".core.notifications.services.HuaweiNotificationService"  
 android:exported="false">  
 <intent-filter>  
 <action android:name="com.huawei.push.action.MESSAGING\_EVENT" />  
 </intent-filter>  
 </service>  
 </application>  
  
</manifest>

Добавляем DI, чтобы dagger включил в граф зависимостей наш новый сервис

import com.nda.core.notifications.services.HuaweiNotificationService  
import dagger.Module  
import dagger.android.ContributesAndroidInjector  
  
@Module  
abstract class ServiceComponentInjector{  
  
 @ContributesAndroidInjector  
 abstract fun notificationService(): HuaweiNotificationService  
}

Добавляем Mapper для перевода сервисного сообщения в структуру, понятную приложению, с которой в дальнейшем можно работать

import com.huawei.hms.push.RemoteMessage  
import com.nda.core.notifications.entities.NotificationPayload  
import com.nda.router.DestinationLink  
import javax.inject.Inject  
  
const val *NOTIFICATION\_ID* = "push\_notification\_id"  
const val *TITLE\_KEY* = "title"  
const val *BODY\_KEY* = "body"  
const val *DEEP\_LINK\_KEY* = "deepLink"  
const val *MEDIA\_KEY* = "media"  
const val *BUTTON\_ONE\_KEY* = "button1"  
const val *BUTTON\_TWO\_KEY* = "button2"  
  
class NotificationPayloadMapper @Inject constructor(  
 private val mediaMapper: NotificationMediaMapper,  
 private val buttonMapper: NotificationButtonMapper  
) {  
  
 fun map(remoteMessage: RemoteMessage) = *with*(remoteMessage) **{** NotificationPayload(  
 id = *dataOfMap*[NOTIFICATION\_ID].*orEmpty*(),  
 title = *dataOfMap*[TITLE\_KEY].*orEmpty*(),  
 message = *dataOfMap*[BODY\_KEY].*orEmpty*(),  
 deepLink = DestinationLink(*dataOfMap*[DEEP\_LINK\_KEY].*orEmpty*()),  
 media = mediaMapper.map(*dataOfMap*[MEDIA\_KEY]),  
 firstButton = buttonMapper.map(*dataOfMap*[BUTTON\_ONE\_KEY]),  
 secondButton = buttonMapper.map(*dataOfMap*[BUTTON\_TWO\_KEY])  
 )  
 **}**}

Разделяем текущее грубое получение pnToken, рассчитаное исключительно для Google на два flavora. Таким образом при сборке релизного apk flavor google будет использовать гугл сервисы, а flavor Huawei будет использовать Huawei сервисы. И всё это в одной ветке git.

import com.huawei.hmf.tasks.Tasks  
import com.huawei.hms.aaid.HmsInstanceId  
import com.nda.BaseApp  
import com.nda.BuildConfig  
import timber.log.Timber  
import javax.inject.Inject  
  
class FirebaseInstanceServiceWrapper @Inject constructor() {  
  
 fun currentToken(): String {  
 return try {  
 //Please enter your App\_Id from agconnect-services.json  
 val appId = when (BuildConfig.FLAVOR\_api) {  
 "preprod" -> "nda"  
 "prod" -> "nda"  
 else -> ""//theoretically impossible  
 }  
  
 val token = HmsInstanceId.getInstance(BaseApp.appContext).getToken(appId, "HCM")  
 Timber.d("Token received: $token")  
 token  
 } catch (e: Throwable) {  
 Timber.e(e, "Token receive error: $e")  
 ""  
 }  
 }  
  
 fun currentAppInstanceId(): String = try {  
 val idResult = HmsInstanceId.getInstance(BaseApp.appContext).*aaid* Tasks.await(idResult).id  
 } catch (e: Throwable) {  
 Timber.e(e, "Error while retrieving huawei app instance id")  
 ""  
 }  
}

Наконец пишем сервис, который трасформирует полученную из RemoteMessage NotificationPayload в пуш нотификацию, которая появляется у пользователя.

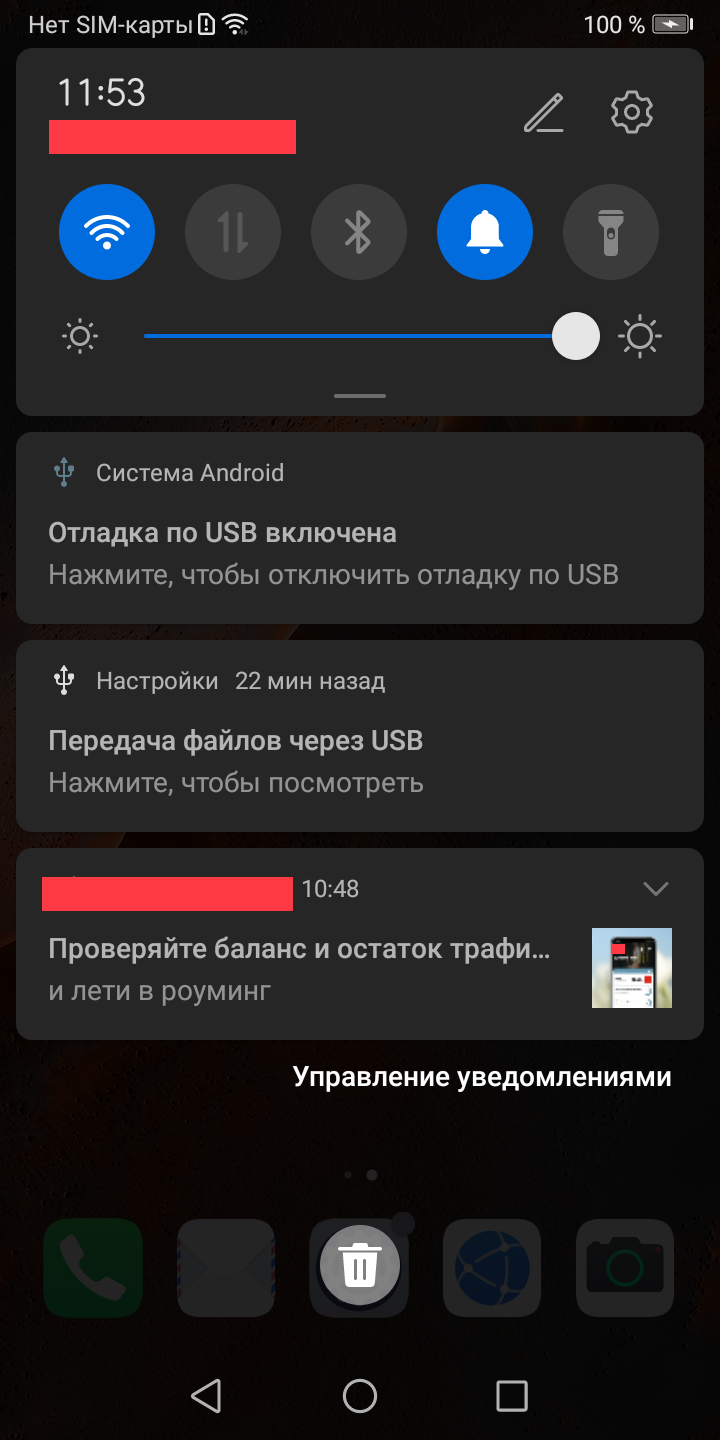
*Код главного класса смотри в приложении*

## Позитивное тестирование

Заходим в appGallery Connect, Push Kit  
Считываем дебаггером свой pnToken на Huawei девайсе.  
Заполняем данные для тестовой отправки на один девайс

{ "push\_notification\_id" : 273, "title" : "Проверяйте баланс и остаток трафика", "body" : "и лети в роуминг", "deepLink" : "tagnda://help-topic?topic\_id=22", "media" : { "type" : "image", "url" : "https://NDA:8182/contentcards/1.NDANewScreen\_ProveryayteBalans.jpg" }, "button1" : { "type" : "external", "url" :"https://www. NDA/ru/", "label" : "ext" }, "button2" : { "type" : "webView", "url" :"https://www. NDA/ru/", "label" : "webView" } }

Уведомление попадает на наш девайс. По клику на уведомлении открываем диплинк на внутренний экран приложения, по кнопке ext открываем ссылку системным приложением, по клику на webview открываем свое webview.   
Итоговый вид:

 Изображение выглядит как текст, монитор, электроника, iPod

Автоматически созданное описание

Работает так же, как и в случае google-сервисов. Функциональность флейвора google не затронута. Задача выполнена.

# Реализация индивидуального задания, часть Г

Архитектура нашего приложения подразумевает 3 варианта boilerplate-кода.  
Первый это создание 6 файлов

1. view (activity/fragment)
2. xml layout для view
3. contract или интерфейс, в котором описываются presenter, coordinator и View
4. Presenter
5. Coordinator
6. Module для реализации DI

Первый вариант нужен, когда нам не требуются юзкейсы. Например, какой-то простой экран-заглушка, без обращений к локальным данным или к данным из сети

Второй вариант подразумевает все вышеописанные 6 файлов, плюс юзкейс

1. Usecase

Второй вариант нужен для, допустим, открытия системных опциональных активити, которые необходимо обернуть в try/catch. Лучшее место для этого кода = usecase.

Третий вариант в дополнение к 7 описанным файлам добавляет ещё 2.

1. Repository
2. Интерфейс репозитория

Он нужен, когда мы точно знаем, что создаваемый очередной экран потребует неких сохраненных persistent-данных, либо точно будет ходить в сеть по какому-то новому API.

Найденный мной плагин (подробнее с ним можно ознакомиться по ссылке <https://github.com/gmatyszczak/screen-generator-plugin>) позволяет реализовать все три варианта создания классов. После многих часов настройки интерфейс выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, монитор, черный

Автоматически созданное описание

Где в блоке Category мы можем выбрать набор для создания необходимых 6/7/9 файлов. В блоке custom Variables можем задавать используемые при создании имена переменных, динамически формирующие названия файлов и вставляемые в код. Screen Elements позволяет настраивать кол-во и имена элементов. А в блоке Code Template происходит непосредственное описание внутренностей каждого создаваемого файла.

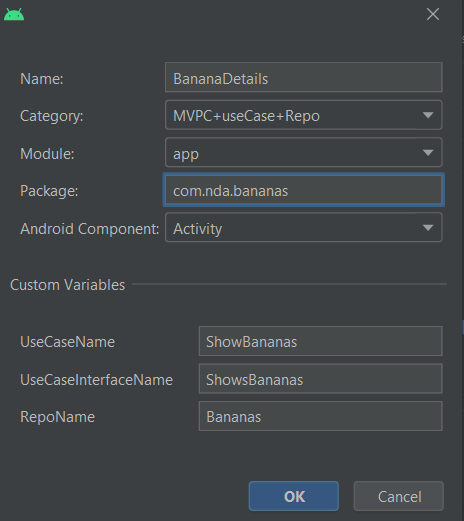
Создание очередного типового экрана выглядит следующим образом:

В интерфейс android-studio по ПКМ добавляется новая категория screen

Изображение выглядит как текст

Автоматически созданное описание

По клику на которую мы видим следующее окно



В нем мы можем ввести переменную Name и кастомные переменные UseCaseName, UseCaseInterfaceName, RepoName. Выбрать пакет и модуль, где будет создан код. Выбрать категорию (6, 8 или 9 файлов).

Результат работы настроенного плагина в максимальной конфигурации (9 файлов) будет следующий:

1. view (activity/fragment)

class BananaDetailsActivity : BaseActivity(), BananaDetailsContract.View {  
 override val binding: BananaDetailsActivityBinding by *viewBinding*()  
  
 @Inject lateinit var coordinator: BananaDetailsContract.Coordinator  
  
 companion object : BaseActivityCompanion(BananaDetailsActivity::class.*java*)  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 coordinator.bind(this)  
 }  
  
 override fun onDestroy() {  
 coordinator.unbind(this)  
 super.onDestroy()  
 }  
}  
  
//*TODO: add activity to manifest*<activity  
android:name="com.nda.bananas.view.BananaDetailsActivity"  
android:screenOrientation="portrait"/>  
  
//*TODO: register activity in AppComponentInjector*@PerActivity  
@ContributesAndroidInjector(modules = [BananaDetailsModule::class])  
abstract fun bindBananaDetailsActivity(): BananaDetailsActivity

1. xml layout для view

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
</androidx.constraintlayout.widget.ConstraintLayout>

1. contract или интерфейс, в котором описываются presenter, coordinator и View

interface BananaDetailsContract {  
  
 interface View: NavigationContract.View  
  
 interface Presenter : NavigationContract.Presenter<View>, ShowBananas.ShowsBananas  
  
 interface Coordinator : NavigationContract.Coordinator<View>  
}

1. Presenter

class BananaDetailsPresenter @Inject constructor(  
) : NavigationPresenter<View>(), Presenter

1. Coordinator

class BananaDetailsCoordinator @Inject constructor(  
 presenter: Presenter,  
 tracker: Tracker,  
 private val usecaseShowBananas: ShowBananas  
) : NavigationCoordinator<View, Presenter>(presenter, tracker), Coordinator {  
 override fun interactionWith(interaction: MoleculeInteraction) {  
 *TODO*("Not yet implemented")  
 }  
}

1. Module для реализации DI

@Module  
class BananaDetailsModule {  
  
 @Provides  
 @PerActivity  
 fun coordinator(coordinator: BananaDetailsCoordinator): BananaDetailsContract.Coordinator =  
 coordinator  
  
 @Provides  
 @PerActivity  
 fun presenter(presenter: BananaDetailsPresenter): BananaDetailsContract.Presenter =  
 presenter  
   
 @Provides  
 @PerActivity  
 fun repository(repository: BananasRepositoryImpl): BananasRepository = repository  
}

1. Usecase

class ShowBananas @Inject constructor(  
 backgroundContext: BackgroundContext,  
 private val repoBananas: BananasRepository  
) : CoroutineRunner by BackgroundRunner(backgroundContext) {  
 suspend operator fun invoke(presenter: ShowsBananas) = launch(  
 **{** *TODO*("Execution not implemented!") **}**,  
 **{** *TODO*("Error handling not implemented!") **}** )  
  
 interface ShowsBananas{  
 }  
}

1. Repository

class BananasRepositoryImpl @Inject constructor(  
 private val networkService: NetworkService  
) : BananasRepository {  
   
}

1. Интерфейс репозитория

interface BananasRepository {  
   
}

В дереве проекта это выглядит следующим образом:

Изображение выглядит как текст

Автоматически созданное описание

# Вывод

В ходе производственной практики была изучена специфика предметной области, изучены документация по работе с сервисами Huawei, архитектура конкретного большого проекта.

Было успешно выполнено индивидуальное задание, состоящее в интеграции Push-нотификаций для Huawei AppGallery. Написанный код активно используется компанией-заказчиком, доставляя пуши в альтернативный Play Store магазин приложений.

Также был интегрирован плагин для создания boilerplate-кода, который по итогу экономит около полутора часов каждому программисту на проекте каждый раз, когда требуется создать корсет очередного экрана.

В ходе практики были получены навыки работы по специальности. Был повышен уровень владения вышеизложенными технологиями разработки программного обеспечения, благодаря интересному и познавательному индивидуальному заданию, а также благодаря хорошему и ответственному руководителю практики.

# Список литературы

1. <https://www.scnsoft.by/>
2. <https://developer.huawei.com/consumer/en/doc/development/HMSCore-Guides/android-config-agc-0000001050170137>
3. <https://ru.wikipedia.org/wiki/Kotlin>
4. <https://dagger.dev/>
5. <https://square.github.io/okhttp/>
6. <https://developer.android.com/>
7. <https://github.com/gmatyszczak/screen-generator-plugin>
8. Kotlin in action <https://www.amazon.com/Kotlin-Action-Dmitry-Jemerov/dp/1617293296>
9. Atomic Kotlin <https://www.atomickotlin.com/atomickotlin/>

# Приложение (Листинг кода главного класса)

import android.app.Notification  
import android.app.NotificationChannel  
import android.app.NotificationManager  
import android.app.PendingIntent  
import android.content.Context  
import android.os.Build  
import androidx.core.app.NotificationCompat  
import androidx.core.app.TaskStackBuilder  
import com.bumptech.glide.Glide  
import com.bumptech.glide.RequestBuilder  
import com.bumptech.glide.request.target.Target  
import com.huawei.hms.push.HmsMessageService  
import com.huawei.hms.push.RemoteMessage  
import com.nda.ui.core.Link  
import com.nda.ui.core.LinkType  
import com.nda.R  
import com.nda.core.coroutinecontext.BackgroundContext  
import com.nda.core.notifications.entities.NotificationMedia  
import com.nda.core.notifications.entities.NotificationParams  
import com.nda.core.notifications.entities.NotificationPayload  
import com.nda.core.notifications.mappers.NotificationPayloadMapper  
import com.nda.core.notifications.repository.NotificationEnabled  
import com.nda.core.preferences.PreferenceProvider  
import com.nda.login.usecases.Authenticate  
import com.nda.router.DestinationLink  
import com.nda.router.ExternalRouter  
import com.nda.splash.view.SplashActivity  
import com.nda.translations.Dictionary  
import com.nda.user.repository.UserRepository  
import com.nda.web.view.WebViewRouter  
import dagger.android.AndroidInjection  
import kotlinx.coroutines.CoroutineExceptionHandler  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.coroutineScope  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
import timber.log.Timber  
import javax.inject.Inject  
import kotlin.coroutines.CoroutineContext  
  
  
class HuaweiNotificationService : HmsMessageService() , CoroutineScope {  
  
 @Inject lateinit var notificationPayloadMapper: NotificationPayloadMapper  
 @Inject lateinit var userRepository: UserRepository  
 @Inject lateinit var preferenceProvider: PreferenceProvider  
 @Inject lateinit var backgroundContext: BackgroundContext  
 @Inject lateinit var dictionary: Dictionary  
 @Inject lateinit var authenticate: Authenticate  
  
 override val coroutineContext: CoroutineContext  
 get() = backgroundContext.get() + *CoroutineExceptionHandler* **{** \_, throwable **->** Timber.d(throwable)  
 **}** private val randomRange = 0..Int.MAX\_VALUE  
  
 override fun onCreate() {  
 super.onCreate()  
 AndroidInjection.inject(this)  
 }  
  
 */\*\*  
 \* This method is used to receive downstream data messages.  
 \* This method callback must be completed in 10 seconds. Otherwise, you need to start a new Job for callback processing.  
 \*  
 \** ***@param*** *message RemoteMessage  
 \*/* override fun onMessageReceived(remoteMessage: RemoteMessage?) {  
 Timber.d("Push received: ${remoteMessage?.*data*}")  
 if (shouldSendNotification()) {  
 remoteMessage?.*let* **{** val payload = notificationPayloadMapper.map(remoteMessage)  
 */\*\*  
 \* The only reason to launch new coroutine is the glide get()  
 \* method which should be called off the main thread. Before this  
 \* image was like in 1/10 notifications  
 \*/  
 launch* **{** sendNotification(payload)  
 **}  
 }** }  
 }  
  
 override fun onNewToken(token: String) {  
 Timber.d("New FCM token created: $token")  
 if (preferenceProvider[NotificationEnabled]!!) {  
 RegisterDeviceService.enqueueWork(*baseContext*)  
 }  
 }  
  
 private fun shouldSendNotification(): Boolean {  
 val notificationEnabled = preferenceProvider[NotificationEnabled] ?: false  
 val isLoggedIn = userRepository.current() != null  
 Timber.d("User logged: $isLoggedIn")  
 return notificationEnabled //&& isLoggedIn // we should show some type of notifications for non-auth-users (ex: sim-activation)  
 }  
  
 private suspend fun sendNotification(notificationPayload: NotificationPayload) {  
 val deepLinkIntent = SplashActivity.createNotificationIntent(  
 this,  
 NotificationParams(  
 id = notificationPayload.id,  
 title = notificationPayload.title,  
 destination = notificationPayload.deepLink,  
 linkType = LinkType.DeepLink,  
 button = NotificationParams.Button.Main  
 )  
 )  
  
 val contentIntent = TaskStackBuilder.create(this)  
 .addNextIntent(deepLinkIntent)  
 .getPendingIntent(randomRange.*random*(), PendingIntent.*FLAG\_UPDATE\_CURRENT*)  
  
 val channelId = getString(R.string.*app\_name*)  
 val notificationBuilder = NotificationCompat.Builder(this, channelId)  
 .setAutoCancel(true)  
 .setDefaults(Notification.*DEFAULT\_SOUND* or Notification.*DEFAULT\_VIBRATE*)  
 .setContentTitle(notificationPayload.title)  
 .setContentText(notificationPayload.message)  
 .setSmallIcon(R.drawable.*ic\_notification\_icon*)  
 .setStyle(NotificationCompat.BigTextStyle().bigText(notificationPayload.message))  
 .setContentIntent(contentIntent)  
 .setPictureIfAvailable(notificationPayload.media)  
  
 val notificationManager =  
 getSystemService(Context.*NOTIFICATION\_SERVICE*) as NotificationManager  
 notificationManager.*createChannel*(channelId, getString(R.string.*app\_name*))  
  
 notificationPayload.firstButton?.*let* **{** notificationBuilder.*addButtonActionFor*(  
 id = notificationPayload.id,  
 link = **it**,  
 buttonType = NotificationParams.Button.First  
 )  
 **}** notificationPayload.secondButton?.*let* **{** notificationBuilder.*addButtonActionFor*(  
 id = notificationPayload.id,  
 link = **it**,  
 buttonType = NotificationParams.Button.Second  
 )  
 **}** Timber.d("Sending push to manager: $notificationPayload")  
 notificationManager.notify(randomRange.*random*(), notificationBuilder.build())  
 }  
  
 private fun NotificationCompat.Builder.addButtonActionFor(  
 id: String,  
 link: Link,  
 buttonType: NotificationParams.Button  
 ) =  
 addAction(0, dictionary.getString(link.title), intentFor(id, link, buttonType))  
  
 private fun intentFor(  
 id: String,  
 link: Link,  
 buttonType: NotificationParams.Button  
 ): PendingIntent? {  
 val destinationLink = when (link.type) {  
 is LinkType.DeepLink -> DestinationLink(url = link.destination)  
 is LinkType.WebView -> WebViewRouter.createDestinationLink(link.destination)  
 is LinkType.External -> ExternalRouter.createDestinationLink(link.destination)  
 else -> null  
 }  
  
 return if (destinationLink != null) {  
 val params = NotificationParams(  
 id = id,  
 title = link.title,  
 destination = destinationLink,  
 linkType = link.type,  
 button = buttonType  
 )  
 val intent = SplashActivity.createNotificationIntent(this, params)  
 TaskStackBuilder.create(this)  
 .addNextIntent(intent)  
 .getPendingIntent(randomRange.*random*(), PendingIntent.*FLAG\_UPDATE\_CURRENT*)  
 } else {  
 null  
 }  
 }  
  
 private suspend fun NotificationCompat.Builder.setPictureIfAvailable(  
 media: NotificationMedia?  
 ): NotificationCompat.Builder {  
  
 if (media is NotificationMedia.Image && !media.url.*isNullOrBlank*()) {  
 val pictureBuilder = Glide.with(this@HuaweiNotificationService)  
 .asBitmap()  
 .load(media.url)  
  
 val bigPicture = pictureBuilder.getOrNull()  
 val thumbPicture = pictureBuilder.getOrNull(LARGE\_ICON\_WIDTH, LARGE\_ICON\_HEIGHT)  
  
 setStyle(NotificationCompat.BigPictureStyle())  
 setLargeIcon(thumbPicture)  
 setStyle((NotificationCompat.BigPictureStyle().bigPicture(bigPicture)))  
 }  
 return this  
 }  
  
 private suspend fun <T> RequestBuilder<T>.getOrNull(  
 width: Int = Target.*SIZE\_ORIGINAL*,  
 height: Int = Target.*SIZE\_ORIGINAL* ): T? = coroutineScope **{** withContext(coroutineContext)**{** try {  
 val target = submit(width, height)  
 val result = target.get()  
 Glide.with(this@HuaweiNotificationService).clear(target)  
 result  
 } catch (e: Exception) {  
 Timber.d(e.toString())  
 null  
 }  
 **}  
  
 }** private fun NotificationManager.createChannel(id: String, name: CharSequence) {  
 if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*O*) {  
 val channel = NotificationChannel(id, name, NotificationManager.*IMPORTANCE\_DEFAULT*)  
 createNotificationChannel(channel)  
 }  
 }  
  
 companion object {  
 private const val LARGE\_ICON\_WIDTH = 200  
 private const val LARGE\_ICON\_HEIGHT = 200  
 }  
}