ОС: Android
Версия приложения: 15.3.3
Минимальная поддерживаемая ОС: Android 6.0 (API level 23)
Compile и target SDK: Android 13 (API level 33)
Ветка: tfs-mirror

Цель этого отчета — предоставить обзор текущего состояния исходного кода Android-приложения ВСК, выявить потенциальные риски и предложить рекомендации по улучшению качества кода и устранению обнаруженных недостатков.

Аудит проводится с использованием инструментов MobSF и ручного код-ревью. Анализируется стек, архитектура, кодовая база, безопасность, тестируемость кода, качество построения UI, логирование и производительность.

Анализ стека

Стек: Kotlin, Java, RxJava2, Realm, kapt, Parcelize, Firebase Distribution&Analytics&Crashlytics&InAppMessaging, Kover, SonarQube, Paranoid String obfuscator , Material, Play Services, Google Maps, Retrofit2, Okhttp3, Tinkoff Decoro, Fresco, PhotoDraweeView, Lottie, AndroidX lifecycle, Dagger2, Joda-time library, Regula Document Reader, FlexBox, Facebook Stetho ,Timber, Adapter Delegates, MobMetrica, Install referrer, Paging, WorkManager, Sentry, Mockk, Junit, Kotest, MockWebServer, huawei&google mobile services, Chucker, RxRelay, AndroidPdfViewer, Paranoid obfuscator, Android Analytics for Snowplow, PersistentCookieJar, StompProtocolAndroid, ZXing Android Embedded, Pact Consumer, ReLinker, Process Phoenix, RxPermissions, RxBinding.

Здорово, что используется Kotlin DSL для Gradle.

Замечания:

- Не для всех зависимостей вынесены версии в константы, в некоторых указаны явно, что может привести к неконсистентным зависимостям.
- При указании зависимостей просто из списка ищется первый с подходящим именем, это не эффективно.
- Во многих модулях одного и того же типа (например, в api модулях) используются одни и те же зависимости, для удобства их можно было бы сгруппировать и использовать единым bundle.
- Joda-time можно не использовать, так как весь необходимый функционал есть в java.time (JDK 8+)
- Используйте с осторожностью Paranoid String obfuscator, возможны проблемы с производительностью, к тому же проект больше не поддерживается [Is this project dead or not? · Issue #70 · MichaelRocks/paranoid · GitHub]
- AndroidPdfViewer должна быть заменена на более новые или более мощные альтернативы, так как библиотека давно не поддерживается и имеет большое количество обнаруженных проблем [GitHub - DImuthuUpe/AndroidPdfViewer: Android view for displaying PDFs rendered with PdfiumAndroid].

Рекомендации:

- Использовать Gradle Version Catalogs Migrate your build to version catalogs | Android Studio | Android Developers. Подумать об отказе от функции ‘resolveVersion’.
- Удобный плагин, который можно использовать для проверки наличия обновлений библиотек в зависимостях: GitHub - ben-manes/gradle-versions-plugin: Gradle plugin to discover dependency updates.
- Постепенно переехать с RxJava на Coroutines. Помимо улучшения читаемости и поддержки, это облегчит использование многих библиотек, бОльшая часть которых из коробки работает на Coroutines и Flow. Исчезнет потребность и в RxRelay, RxPermissions и RxBinding, что также упростит код и поддержку. Либо, как минимум, переехать на более новую версию RxJava3 для улучшения производительности.
- Переписать модули, которые написаны на Java на Kotlin, опять же для удобства в дальнейшей поддержке. Плюс, скорее всего функционал этих модулей давно не обновлялся, возможно, там могут скрываться места потенциальных багов.
- *Не увидела использования Adapter Delegates в проекте, возможно, есть и другие неиспользуемые подключенные библиотеки, но без поиска и навигации по проекту, это тяжело проверить.
- Можно задуматься о поддержке нововведений Android 14 и поднятии target/compile SDK до 34. Migrate apps to Android 14 | Android Developers
    - Для всех сервисов в манифесте должен быть указан хотя бы один Service type https://developer.android.com/about/versions/14/changes/fgs-types-required
    - Поддержка Java 17
    - https://developer.android.com/about/versions/14/behavior-changes-14#partial-photo-library-access
    - Для Broadcast Receivers, зарегистрированных в коде, необходимо явно определять RECEIVER_EXPORTED или RECEIVER_NOT_EXPORTED
    - Android запрещает приложениям отправлять implicit intents внутренним компонентам приложения следующими способами:
        - implicit intents передаются только экспортированным компонентам. Приложения должны либо использовать explicit intent для доставки в неэкспортированные компоненты, либо пометить компонент как экспортированный.
        - Если приложение создает mutable pending intent, в котором не указан компонент или пакет, система выбрасывает исключение.
- Использовать Material3 вместо Material.
- Fresco можно заменить на Glide или Coil для меньшего потребления памяти.
- Facebook Stetho не обновляется давно и с развитием инструментов отладки, встроенных в Android Studio, его актуальность снижается, к тому же в проекте подключен Chucker, который также может закрыть потребность в данной библиотеке.
- Если вы используете SonarQube, возможно вам не потребуется дополнительный инструмент в виде Kover для анализа покрытия кода тестами, так как SonarQube уже имеет встроенный функционал покрытия.

Архитектура

- На UI слое Fragment подписывается на изменения ViewModel для отображения состояния.
- Для DI используется Dagger, корректно выделены компоненты и модули, скоупы жизни, описан подход построения зависимостей в feature_module.md.
- Наличие документации архитектуры в architecture.md радует, однако стоит не забывать ее актуализировать.
- Используется большое количество библиотек, хранящихся в модулях regulaLibs, libs

Модули:

- app
- buildsrc
- checks и checks_android
- etr-camera
- inspectionLib
- mobile_services, mobile_services_google, mobile_services_huawei
- Для каждой feature создано по три модуля: api,impl,stub. Список feature: auth, debug_menu, dms, duu_osago_gu, duu_vehicle, esia , feedback, osago_purchase, partner_discounts, profile, rsa_sdk, support_chat, supposes,telemed_chat, travel
- Core модули test, ui common
- regulaLibs, libs

Навигация:

Использование интерфейса Coordinator для взаимодействия между родительскими и дочерними экранами позволяет четко разделить обязанности и обеспечивает взаимодействие между экранами разного уровня. Использование BehaviorRelay<T> для хранения состояния контейнера обеспечивает реактивный подход к управлению состоянием экрана.

Из описания в architecture.md не до конца ясно, каким образом реализуется механизм смены состояния роутера у родительского экрана, что может привести к ошибкам при реализации. Механизм инициализации в Fragment.afterInject() перед вызовом super может быть потенциально ошибкоопасным и сложно отлаживаемым.

Замечания:

- Использование RxJava ухудшает читаемость и добавляет проблем с регуляцией состояния подписок в зависимости от жизненного цикла (работа с disposables). Использование StateFlow для работы с состоянием экрана улучшило бы читаемость и облегчило управление жизненным циклом данных.
- Не удалось найти где и как добавляется в граф зависимостей Context, тк поиск по проекту не работает. Могут быть проблемы с утечками контекста, стоит убедиться в том, что везде используется Application context для Singleton зависимостей. Также были отмечены случаи, когда Context передавался в класс, но не использовался. Примером может послужить класс EmulatorDetectServiceImpl, в котором Context и ряд других параметров Inject в конструктор, но не используются.

- Раздут app модуль, стоит вынести оттуда функционал в отдельные core-модули. Тяжело в нем ориентировать так как тут большое количество и DI модулей, и базовых сервисов/классов.
- Все модули имеют зависимости на core_ui и core_common, точно ли везде используется функционал из этих модулей?
- Во многих Fragment используется следующий подход к передаче аргументов в ViewModel:

```
Handler(Looper.getMainLooper()).postDelayed({
    viewModel.handleProfileArgs(args)
}, 500)
```

  Вместо этого стоит использовать SavedStateHandle | Android Developers .
- В domain слоях модулей много различных CommandImpl, которые нарушают принцип SingleResponsibility и завязаны на Android зависимости.
- Внутри нескольких Fragment используется private функция, которая напрямую ставит видимость Toolbar в View.Gone. Изменение видимости должно быть в зоне ответственности Activity, так как такие вызовы могут привести к непредвиденному состоянию toolbar. Вместо этого, если возможно, используйте соответствующие callback-и или обсерверы для изменения состояния Activity.
- В файле architecture.md слой данных, ответственный за походы в сеть и работу с хранилищами обозначен как Data, в соответствии с общепринятыми обозначениями. Однако в проекте data это название папок с моделями данных. Вызывает путанницу и усложняет навигацию по проекту.

Рекомендации:

- Можно подумать насчет введения для некоторых модулей единой модели, отображающей состояние экрана, для консистентности и поддержки Unidirectional Data Flow.
- В модуле checks_android находится только манифест, возможно, стоит его ликвидировать.
- Модули etr-camera и inspectionLib написаны на Java, как уже говорилось ранее, стоит их переписать на Kotlin. Это не только приведет к единообразному стилю кода и лучшей поддерживаемости, но и даст доступ ко всем преимуществам Kotin, таким как безопасность по отношению к null и более лаконичный синтаксис.
- Стоит провести ревью библиотек из regulaLibs, libs. Проверить их на актуальность и используемость.
- Изучить альтернативные подходы к навигации, такие как Navigation Component, который предлагает более современный и поддерживаемый механизм навигации.

Анализ исходного кода

Зафиксирован code style, есть кастомные правила для линта. Инструкция по установке настроек есть в файле architecture.md. Используется Sonarqube.

Замечания:

- В коде встречаются override функций, которые после вызывают super реализацию. Возникает вопрос, зачем тогда пишется override. Пример:

  ```
  public override fun showAsDialog(view:View){
          super.showAsDialog(view)
  }
  ```
- Стоит выносить все константы (числовые, строковые) в const переменные, это облегчит их переиспользование и изменение, к тому же корректный нейминг еще и облегчит понимание предназначения константы.

Рекомендации:

- Своевременно анализировать результаты отчетов Sonarqube и исправлять замечания.
- Добавить codeStyle check как этап CI/CD, настроить pre-commit, pre-push git hooks для локальной проверки соответствия codestyle перед отправкой изменений в репозиторий. Это позволит экономить ресурсы runnerов.
- Стоит использовать библиотеку LeakCanary для обнаружения и устранения утечек памяти.

Анализ безопасности

Анализ безопасности был проведен с помощью MobSF.

Permissions:

- android.permission.ACCESS_FINE_LOCATION - Точно ли необходима информация о точном местоположении пользователя, возможно, android.permission.ACCESS_COARSE_LOCATION будет достаточно.
- android.permission.CAMERA и android.permission.READ_MEDIA_IMAGES - Возможно, достаточно было воспользоваться системным пикером с помощью ContractsApi.
- android.permission.READ_CALENDAR, android.permission.READ_CONTACTS, android.permission.RECORD_AUDIO и android.permission.WRITE_CALENDAR - Не совсем понятно, какой функционал обеспечивают эти разрешения. Если функциональность можно реализовать без непосредственного доступа к данным, исследуйте возможность использования Intent или другого механизма, который не требует прямого доступа и разрешения.
- android.permission.READ_EXTERNAL_STORAGE и android.permission.WRITE_EXTERNAL_STORAGE - Стоит задуматься, действительно ли нужен доступ. С Android 10 (API уровня 29) Scoped Storage обеспечивает доступ к части внешнего хранилища и избавляет от необходимости запроса этих разрешений.

Network Security:

- Использование Clear text для всех доменов

- WebView игнорирует ошибки SSL-сертификата и принимает любой SSL-сертификат. Это делает приложение уязвимым для MITM-атак. ru/vsk/insurance/presentation/stage/web/helpers/WebViewHelperImpl.java

Manifest analysis:

- Application Data can be Backed up[android:allowBackup=true]. Этот флаг позволяет любому пользователю создавать резервные копии данных приложения через adb. Он позволяет пользователям, включившим отладку по USB, копировать данные приложения с устройства.

Замечания:

- В соответствии с новой политикой Google, необходимо предоставить пользователю возможность Удалять свой аккаунт внутри приложения : **Understanding Google Play's app account deletion requirements - Play Console Help**

Рекомендации:

- Если приложению не нужно знать точное местоположение (например, для навигации), то можно использовать ACCESS_COARSE_LOCATION для определения приблизительного положения, что может быть менее инвазивно для конечного пользователя.
- Если необходимость в камере ограничивается выбором изображений, использование системного пикера изображений может исключить необходимость в разрешениях CAMERA и READ_MEDIA_IMAGES. Это также уменьшит проблемы с конфиденциальностью.
- Настройте Network Security Config в вашем manifest-файле и укажите, что приложение должно использовать только зашифрованные соединения (HTTPS).
- Настройте ваш WebView, чтобы он не игнорировал ошибки SSL, и добавьте надлежащую обработку ошибок SSL, чтобы предотвратить MITM-атаки.
- Установите `android:allowBackup="false"`, чтобы предотвратить создание резервных копий данных приложения через ADB, если это не соответствует вашей политике безопасности. Примечание: это также может негативно сказаться на пользовательском опыте, так как пользователи не смогут восстанавливать свои данные при смене устройства.

Тестируемость кода

Для тестирования используются следующие инструменты:

- **Mockk:** Библиотека мокирования для Kotlin, позволяет создавать моки и шпионы для unit тестирования.
- **Junit:** Одна из основных библиотек для unit-тестирования в Java и Kotlin.
- **Kotest**: Библиотека для тестирования на Kotlin, предоставляет множество стилей написания тестов и интегрируется с Junit.

- **MockWebServer**: Утилита от Square, которая позволяет тестировать HTTP клиенты создавая фиктивный HTTP сервер.
- **Kover**: Инструмент для измерения покрытия кода тестами на Kotlin.
- **Jacoco:** Данный инструмент предоставляет данные о том, какие части кода были и не были выполнены при запуске тестов, что позволяет оценить эффективность и полноту тестового сценария. Эти данные помогают разработчикам выявить слабо протестированные области и повысить степень доверия к коду.

Все инструменты актуальны и являются достаточно популярными. Однако использование Kover может быть избыточным, как говорилось ранее.

Рекомендации:

- Уделить больше внимания покрытию тестами ViewModel.
- Следить за покрытием кода с помощью Sonarqube и Jacoco.

Общее качество построения UI

Используется XML верстка. Не удалось оценить состояние UIKit и его компонентов из-за проблем с доступами.

Был также проведен анализ Accessibility приложения и выявлены следующие проблемы:

- Проблемы со scale текста:
    - Большое количество textView с фиксированными размерами (или внутри ViewGroup с фиксированными размерами) и scalable текстом внутри.
    - Не используется sp для размеров текста.

- Отсутствие лейблов для screen readers (описание элемента, которое бы могло быть озвучено для людей с нарушением зрения)

- Размер нажатия:
    - Есть view с размерами меньше 48dp, которые являются кликабельными

Замечания:

- В нескольких ViewHolder была замечена установка clickListener внутри функции bind(). Метод onBindViewHolder вызывается каждый раз при bind view с представлением, получается, что каждый раз добавляется новый listener. Вместо этого стоит добавлять clickListener в onCreateViewHolder.
- Большое количество пустых файлов для ресурсов, которые стоит ликвидировать.
- Есть повторяющиеся ресурсы в нескольких модулях, в частности цвета, которые должны находится в core модулях или UiKit. В модулях должны находится ресурсы специфичные только для конкретных экранов модуля.

- Переведите все размеры текста в единицы sp (scale-independent pixels), чтобы обеспечить корректное масштабирование текста согласно настройкам доступности устройства.

Рекомендации:

- Зафиксировать единый нейминг id view, сейчас нет общего подхода: где-то tv_discount_product, где то simpleDraweeView.
- Использовать dimens для всех отступов и размеров. Сейчас только в некоторых местах используются dimens. Это важно для обеспечения консистентности интерфейса, упрощения масштабирования под различные экраны, улучшения поддерживаемости приложения и повышения эффективности разработки.
- Стоит зафиксировать в core_ui alias для различных ресурсов, это позволит избежать использования полного qualifiera в других модулях и повысит читаемость кода и удобство разработки. Например:

  resourses.getString(ru.vsk.insuarance.core.ui.R.string.error_bottom_sheet_title)

  ⇓

  resources.getString(CoreUiString.error_bottom_sheet_title)

- Избегайте использования фиксированных размеров для ViewGroup, которые содержат текст, обеспечьте изменение размеров при масштабировании текста.
- Добавьте описания (android:contentDescription) к интерактивным элементам, таким как кнопки и пользовательские элементы управления, которые могут быть прочитаны вспомогательными технологиями (например, TalkBack)
- Увеличьте размеры всех кликабельных элементов, чтобы они имели минимальные размеры 48dp x 48dp, как рекомендовано в руководстве по написанию приложений.

Оценка логирования в проекте

Самописного решения для логирования замечено не было, однако используются следующие инструменты:

- **Timber:** Удобная обертка для логирования в Android, облегчает создание и форматирование логов.
- **MobMetrica** (Yandex Metrica): Сервис аналитики от Яндекса, позволяющий отслеживать события, сессии и другие данные использования приложениями.

Рекомендации:

- Проверьте, что события и пользовательские атрибуты, которые вы отслеживаете с помощью Yandex Metrica, соответствуют вашим аналитическим и бизнес-потребностям.
- Обеспечьте, что логи и события аналитики не содержат личные данные пользователей без их разрешения и соответствуют политикам конфиденциальности и GDPR.
- Настройте разные уровни логирования (например, DEBUG, INFO, WARN, ERROR), чтобы иметь возможность фильтровать важность сообщений в логах.

Анализ производительности

Нет возможности провести данные работы из-за отсутствия доступов.

Аудит производительности направлен на выявление следующих аспектов:

- **Время запуска приложения.** Измерение времени, необходимого для запуска перехода к главному экрану. -> **Gradle Doctor (runningcode.github.io)**
- **Эффективность работы с памятью**. Анализ использования памяти приложением и выявление утечек памяти. -> **LeakCanary**
- Оптимизация работы с батареей. Оценка влияния приложения на продолжительность работы устройства от одной зарядки. Inspect energy use with Energy Profiler | Android Studio | Android Developers Android Developers Blog: How to effectively A/B test power consumption for your Android app's features (googleblog.com)