

Диплинки на миллион

Как мы перестали сжигать маркетинговый бюджет



Денис Конопелькин
Android-разработчик Циана

Позвольте представиться



Денис Конопелькин

Android-разработчик в команде Платформы Циана

Немного обо мне:

- Интересуюсь системой сборки Gradle и CI
- Внедряю технические метрики и SLO в мобильных приложениях
- Руководжу рабочей группой по BDUI
- Видел, как проведение работ по диплинкам разделило работу коллег на до и после... 😊



Циан – ведущая цифровая платформа операций с недвижимостью



~20 млн

уникальных
пользователей в месяц

2,1 млн

актуальных
объявлений

99%

чистота
базы объявлений

25 лет

экспертизы

15,2 млрд руб.

выручка за 12 месяцев
2025 года выросла
на 16,7%

Топ-15

крупнейших сайтов
недвижимости в мире
по версии Similarweb

1 000+

сотрудников

№1

proptech в инклюзии:
~6 тыс. новых объявлений
в день содержат параметры,
важные для людей на колясках



Приложение Циана – это



>700

Gradle-модулей

>1300

Скриншотных и UI-тестов

~2 млн

Строк кода на Kotlin

25 тыс.

Unit-тестов

18

Android-разработчиков
в команде

300+

Pull Request в месяц





~120

Экранов поддерживает
диплинки

~150 тыс.

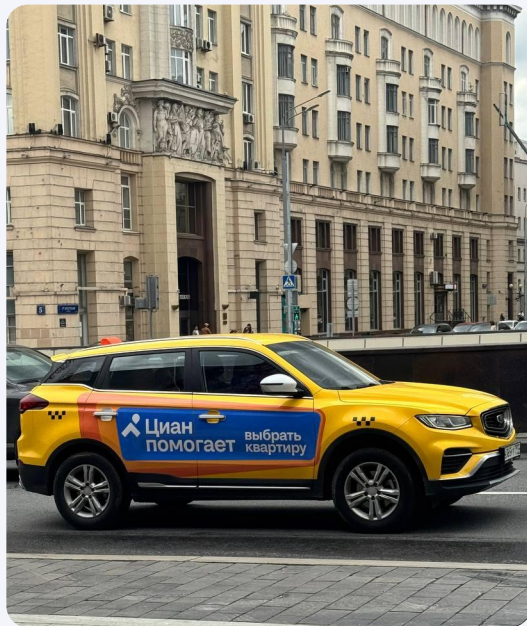
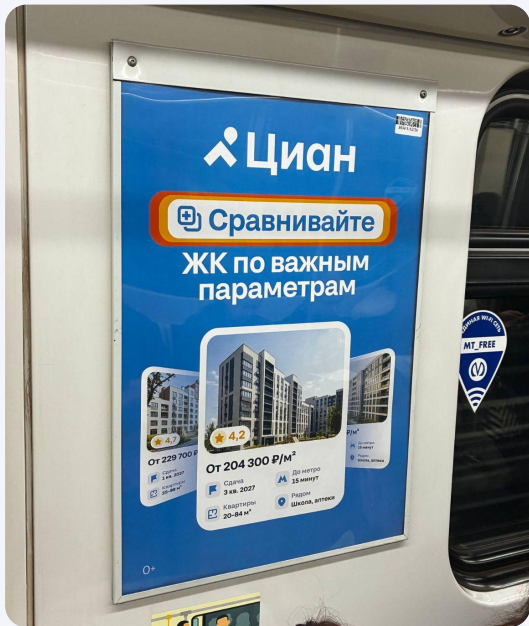
Установок Android-приложения
в месяц приходит
от маркетинговых кампаний

~450 тыс.

Кликов по ретаргетинговым ссылкам
в месяц возвращают пользователей
в Android-приложение



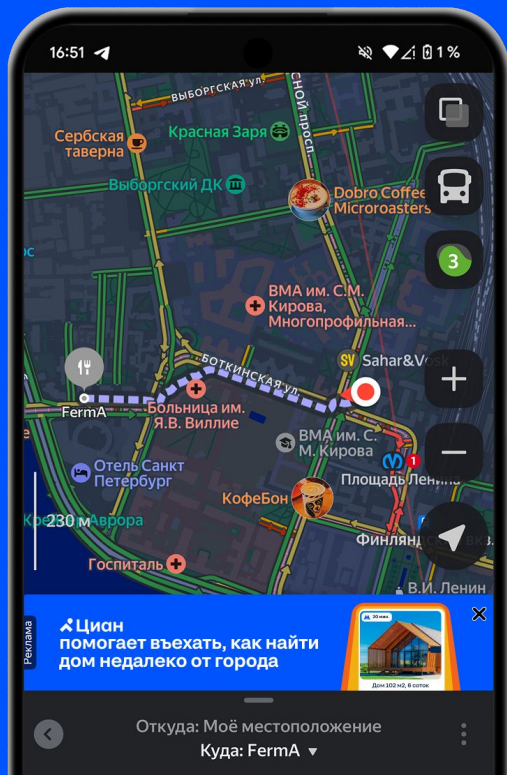
Наверняка вы видели нашу рекламу!



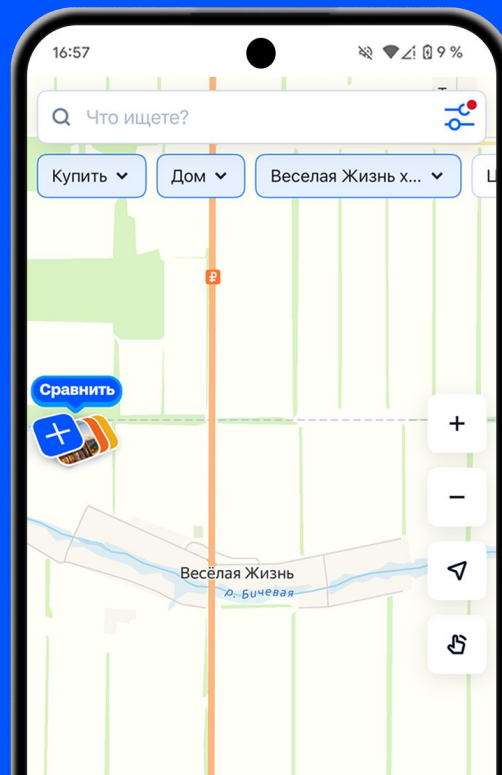
Переход по рекламе может выглядеть так...



Рекламный баннер



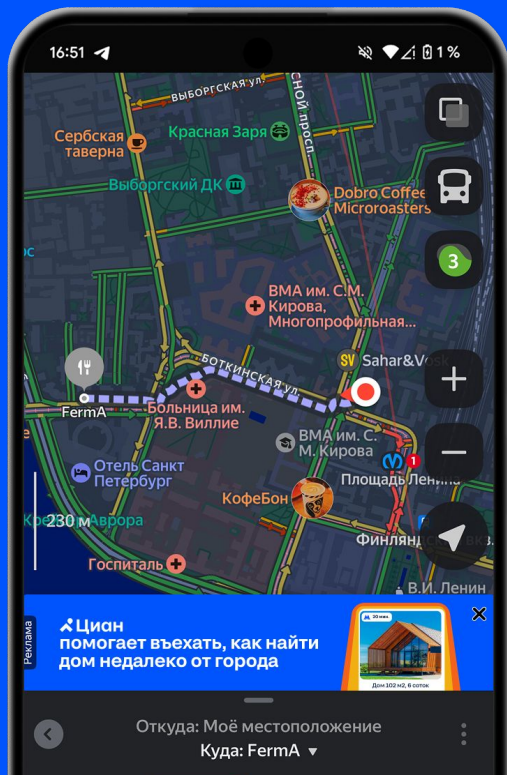
✓ Целевой экран



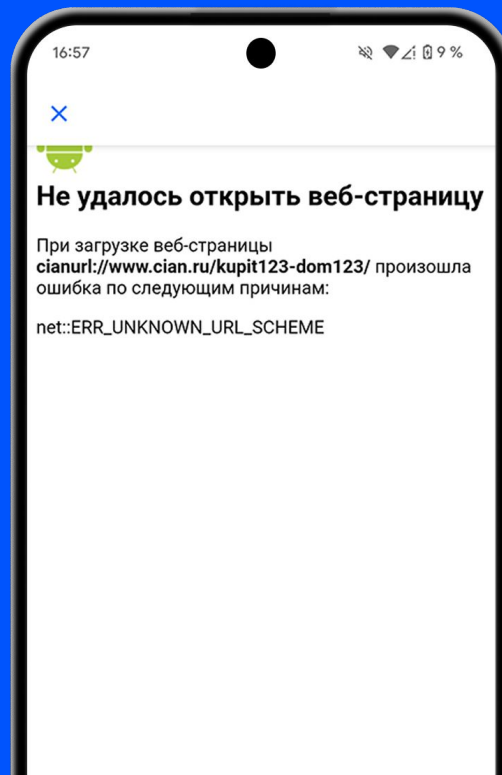
Переход по рекламе может выглядеть так...



Рекламный баннер



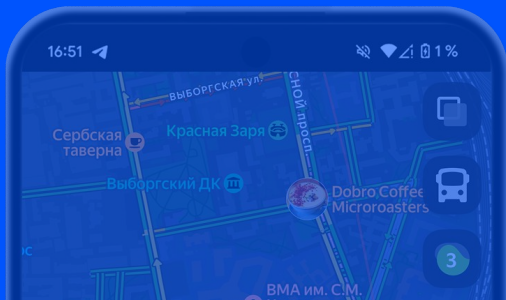
Ошибка



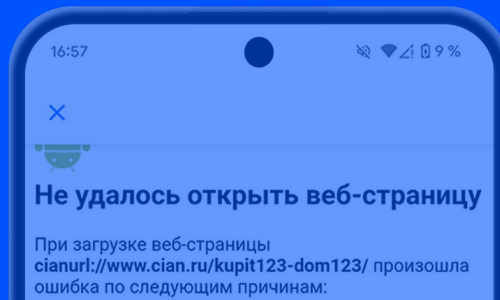
Переход по рекламе может выглядеть так...



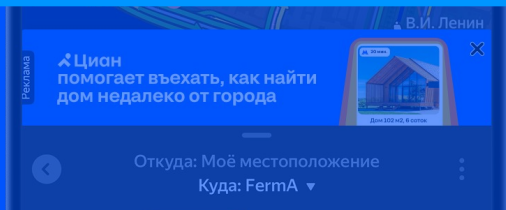
Рекламный баннер



Ошибка



Знаю ли я, сколько денег моя компания сожгла на этой неделе из-за того, что баннер не отработал?



Почему это важно?



~25–30%

выручки Циана уходит в маркетинг*

>1 млн

кликов по нашей рекламе
в месяц

~10%

маркетингового бюджета
приложений сэкономили после
проведения работ

**Каждый клик по рекламе —
оплачен**

**Каждый сломанный клик —
потерян безвозвратно**



Диплинк

ссылка, которая открывает конкретный экран мобильного приложения



Ретаргетинговый диплинк

рекламная ссылка, открывает конкретный экран ранее установленного приложения



Отложенный диплинк (Deferred deeplink)

рекламная ссылка, ведёт на магазин приложений, после установки сразу отображает целевой раздел



О чём сегодня расскажем

- 01 Кто должен нести
ответственность за диплинки
- 02 Анатомия отказов:
почему deeplinks ломаются
- 03 Работающая модель:
процесс, тесты, мониторинг
- 04 Сделаем выводы
и разберем ошибки



Вернёмся

В

прошлое 



Существовали подобные сценарии



Сценарий 1

Агентству могли передать диплинк, который не работал никогда, но его пропустили на тестировании и сжигали маркетинговый бюджет, пока не заметили.



Сценарий 2

Диплинк мог корректно отрабатывать у одних пользователей и не работать у других.



Сценарий 3

Диплинк работал какое-то время, но потом на проде другая команда его сломала.

Мы решили разобраться, откуда взялись такие напасти



Маркетолог

«Хочу запустить маркетинговую кампанию, мне нужно передать
в агентство диплинк на целевой раздел»



Маркетолог

«Хочу запустить маркетинговую кампанию, мне нужно передать
в агентство диплинк на целевой раздел»

Есть похожая кампания с диплинком?



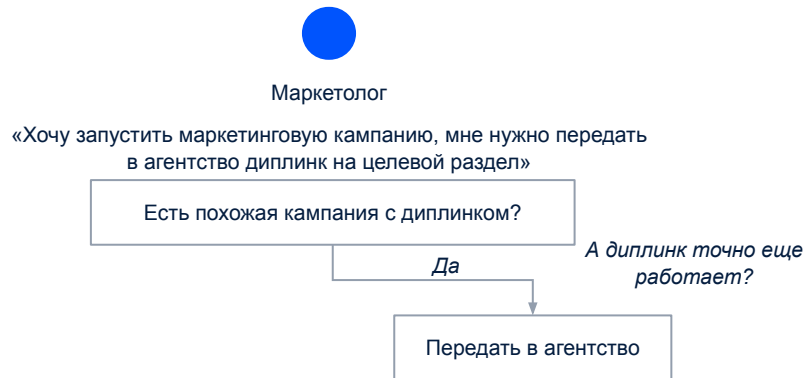
Маркетолог

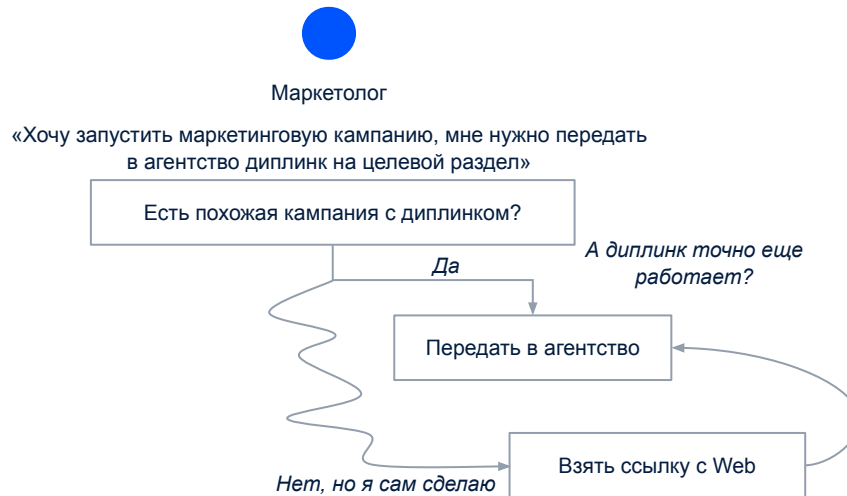
«Хочу запустить маркетинговую кампанию, мне нужно передать
в агентство диплинк на целевой раздел»

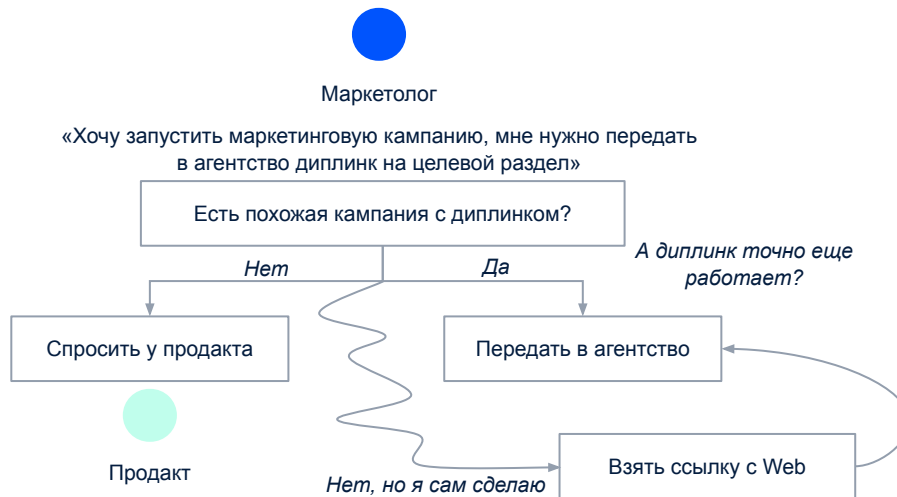
Есть похожая кампания с диплинком?

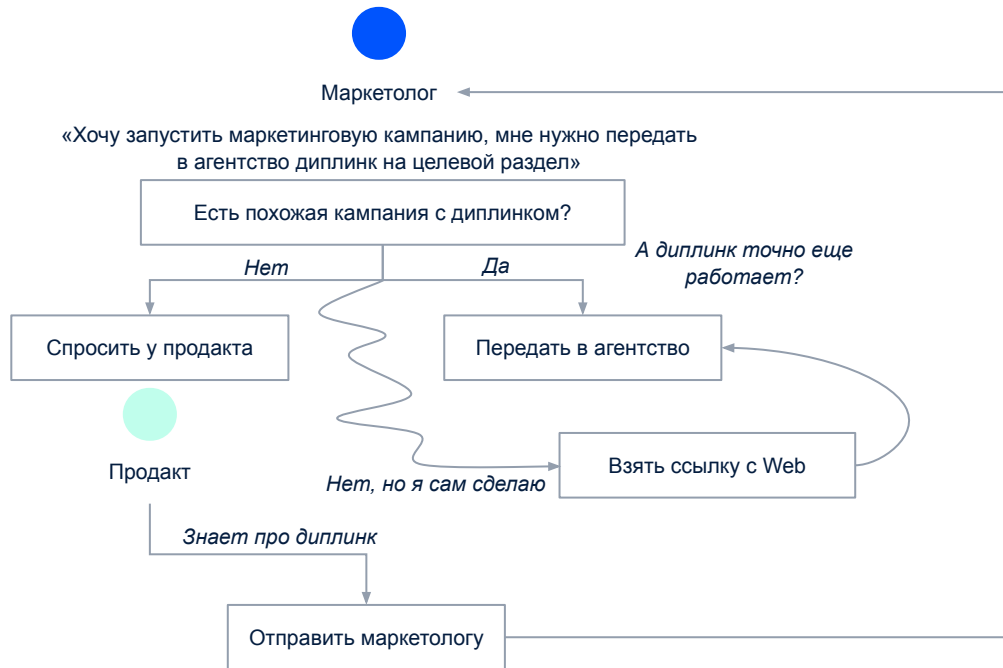
Да

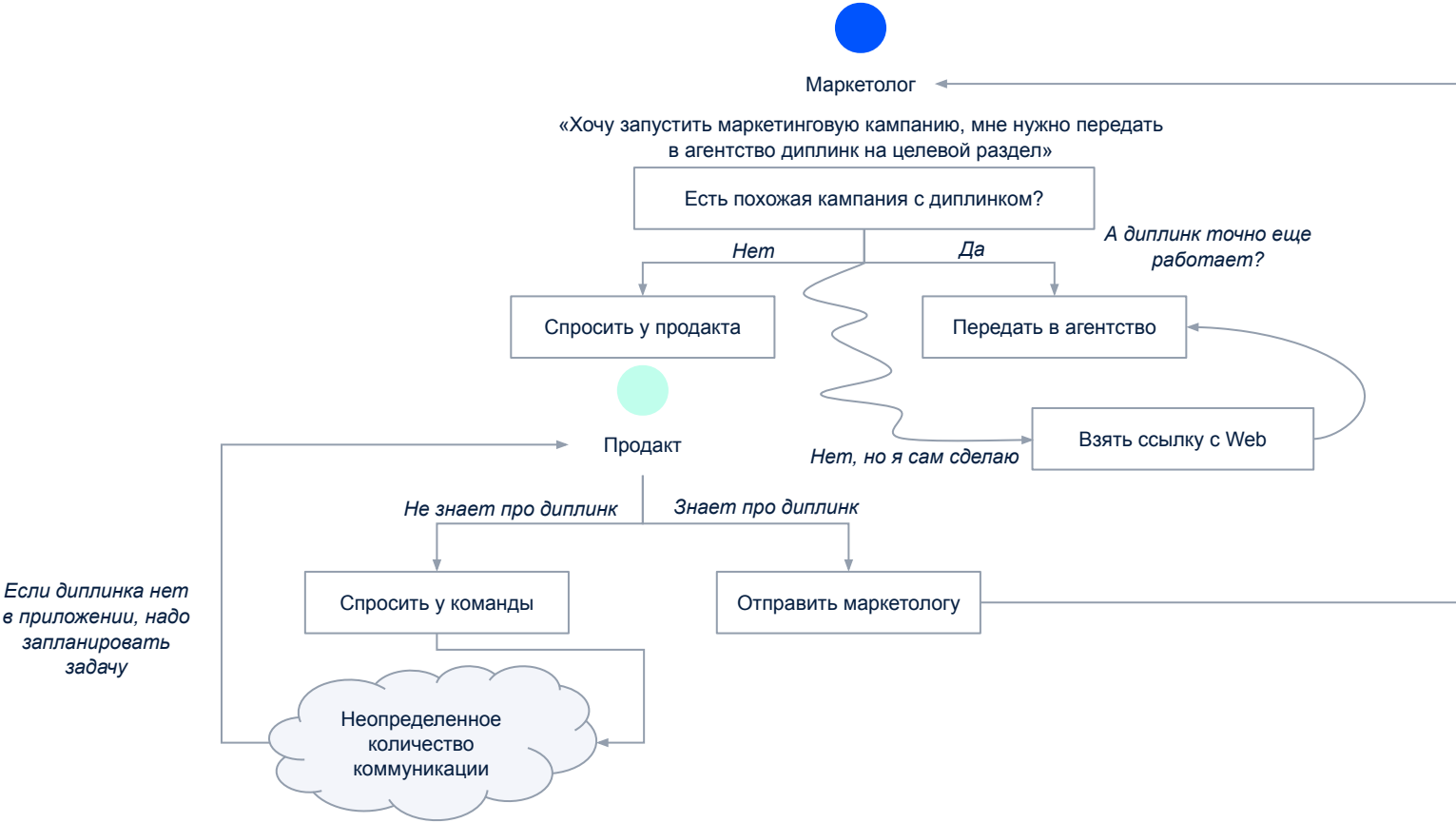
Передать в агентство

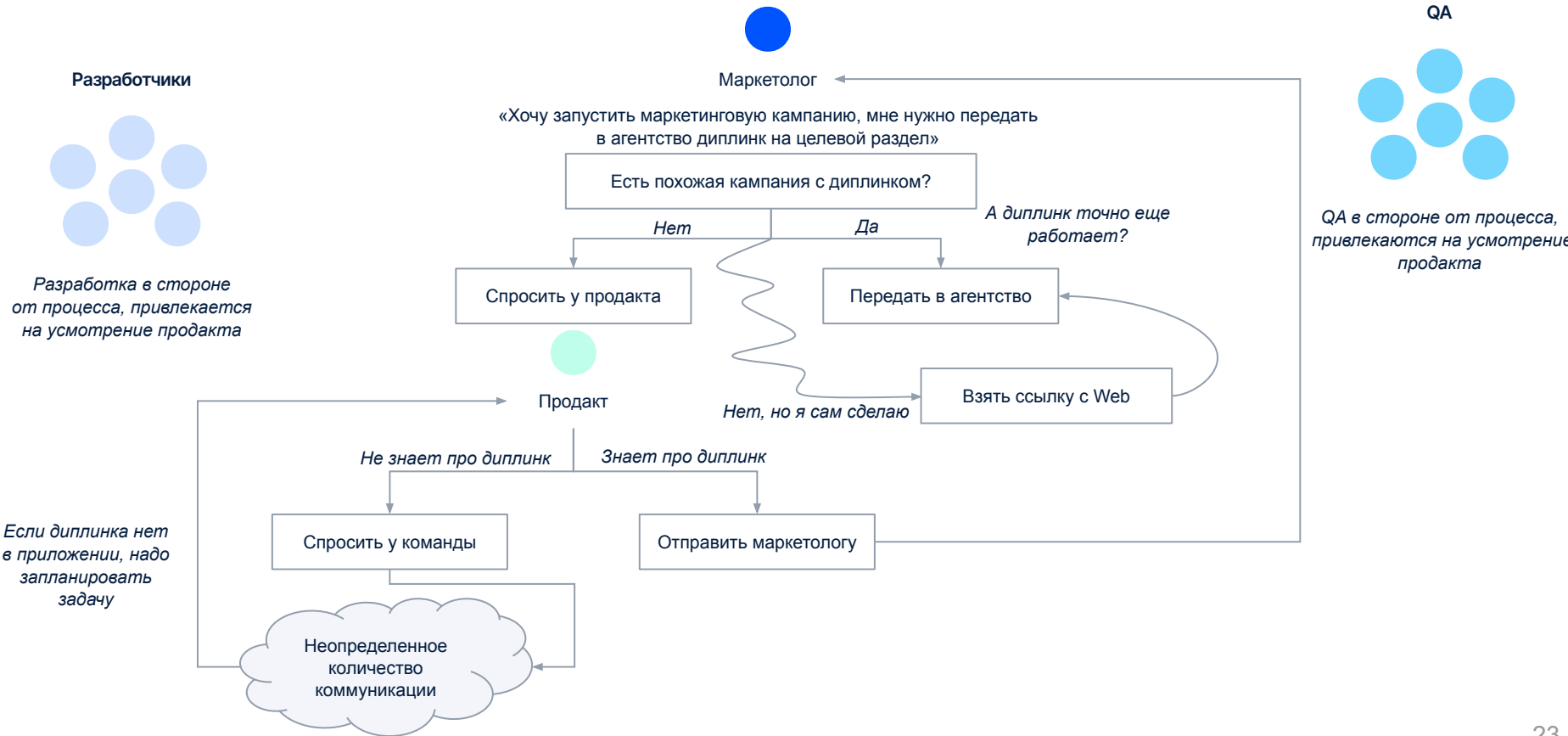


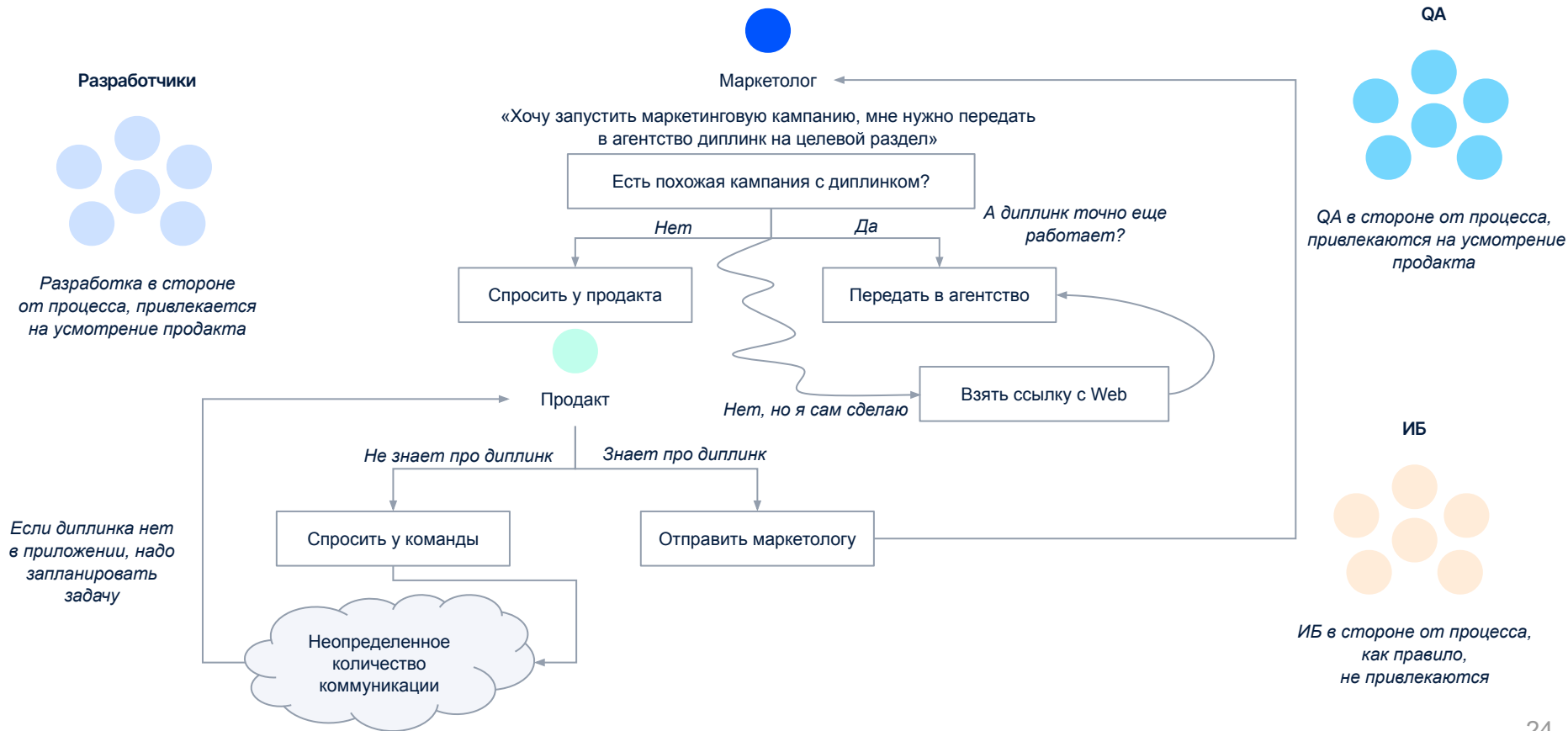














Непрозрачность

Что есть, в каких версиях и как отработает — знает только разработчик. Участие в процессе косвенное, от случая к случаю.



Задержка

Чтобы дать диплинк на целевой экран, нужно завести задачу, дождаться релиза, дождаться раскатки.
80% пользователей получают диплинк через 4-6 недель.



Непредсказуемое качество

Со временем диплинк может сломаться или уводить пользователя на неверный экран.



Низкая безопасность

Новые диплинки заводятся без аппрува ИБ.



Поиск ответственного за диплинк

Кадр из мультсериала «Человек-паук» (Spider-Man)



«Денис, ну какая
техническая сложность?
Диплинк — это просто
ссылка!»

— Скажете мне вы



Техническая сложность



«Deerlink это просто ссылка»

Рекламный креатив

Обертка вендора (Appsflyer)

Обработка редиректа
в магазин приложений (OneLink)

Передача параметров
атрибуции

Resolve ссылки приложением

Навигация



Зона 1

Настройки кампании

Зона 2

Система Android

Зона 3

Приложение



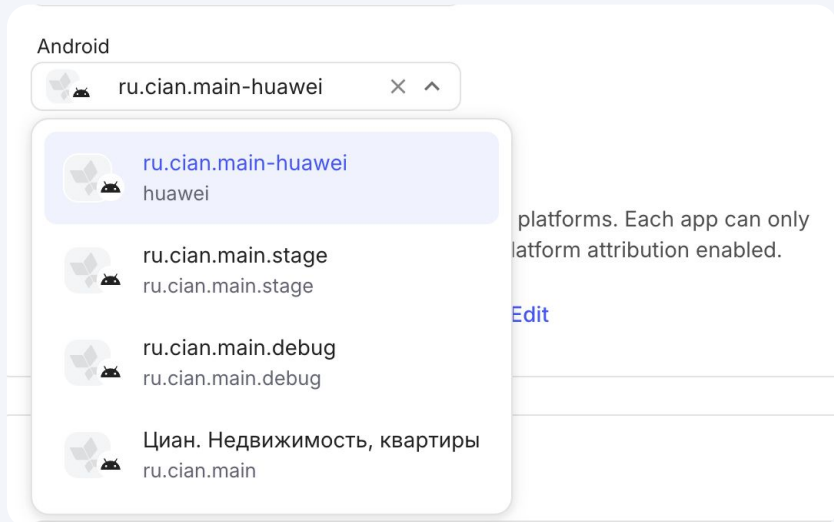
Зона 1 – Настройки кампании

Некорректные данные

Проблема 1:

Неверный packageName в шаблоне кампании

Вместо `ru.cian.main` в кампании для Huawei было указано `ru.cian.main-huawei`, хотя сборки с таким именем у нас никогда не было



Зона 1 – Настройки кампании

OneLink vs Direct AppsFlyer

Проблема 2:

В части iOS-кампаний использовались ссылки для Android и наоборот.

- https://*.onelink.me/ — умная ссылка, которая работает как на iOS, так и на Android
- <https://app.appsflyer.com/ru.cian.main> — ссылка, работающая только на Android

Integration tab

Attribution link tab

Generate and save attribution links and send them to the partner for attributing specific campaigns, ad sets, or single ads - both for UA and retargeting campaigns. You can use either a single-platform link or a OneLink.

[Learn more about attribution link structure and parameters.](#)

To generate an attribution link, follow [these instructions](#).

Note: OneLinks are not supported by VK Ads. Use standard attribution links for retargeting.

Проблема 3.1: При создании ссылки путём подстановки значения в query param не использовалось экранирование.

https://cian.onelink.me/GJyb?c={campaign_name}&<some_params>&deep_link_value=cianurl://spb.cian.ru/cat.php/?deal_type=rent&engine_version=2&offer_type=flat&online_booking=1®ion=1&type=2&<other_params>

Проблема 3.1: При создании ссылки путём подстановки значения в query param не использовалось экранирование.

https://cian.onelink.me/GJyb?c={campaign_name}&<some_params>&deep_link_value=cianurl://spb.cian.ru/cat.php/?deal_type=rent&engine_version=2&offer_type=flat&online_booking=1®ion=1&type=2&<other_params>

Проблема 3.1: При создании ссылки путём подстановки значения в query param не использовалось экранирование.

https://cian.onelink.me/GJyb?c={campaign_name}&<some_params>&deep_link_value=cianurl://spb.cian.ru/cat.php/?deal_type=rent&engine_version=2&offer_type=flat&online_booking=1®ion=1&type=2&<other_params>

Зона 1 – Настройки кампании

Экранирование

Проблема 3.1: При создании ссылки путём подстановки значения в query param не использовалось экранирование.

https://cian.onelink.me/GJyb?c={campaign_name}&<some_params>&deep_link_value=cianurl://spb.cian.ru/cat.php/?deal_type=rent&engine_version=2&offer_type=flat&online_booking=1®ion=1&type=2&<other_params>

Исправленный вариант

https://cian.onelink.me/GJyb?c={campaign_name}&<some_params>&deep_link_value=cianurl%3A%2F%2Fspb.cian.ru%2Fcat.php%2F%3Fdeal_type%3Drent%26engine_version%3D2%26offer_type%3Dflat%26online_booking%3D1%26region%3D1%26type%3D2&<other_params>

Зона 1 – Настройки кампании

Экранирование

Проблема 3.2:

При создании ссылки через web-оснастку кодирования быть не должно

Deep linking & redirection

Deep linking

Specify in-app content for deep linking and deferred deep linking. [Learn more](#)



Note: Deep linking requirements (requires developer)

1. Configure app launch methods in your OneLink template. [Learn more](#)
2. Implement deep link behavior in your apps. [Learn more](#)

Deep link value ?

`cianurl%3A%2F%2Fspb.cian.ru%2Fcat.php%2F` ×

✗ Ошибка



Первичная настройка шаблона кампании должна происходить с **участием технического специалиста**



Важно **донести до маркетинга ценность использования OneLink** вместо Direct Appsflyer



Процесс создания ссылок не всегда очевиден — **стоит подготовить инструкцию**





Зона 1

Настройки кампании

Зона 2

Система Android

Зона 3

Приложение



Зона 2 – Система Android

Магазины приложений

Google Play, AppGallery, GetApps, Galaxy Store, RuStore — у каждого своя механика атрибуции. Необходимо **явно обеспечивать поддержку сторонних магазинов в коде.**

Проблема 4:

Маркетологи включили сторонние магазины, не уведомляя разработчиков.

Android

Redirect to ⓘ
Google Play ▼

^ Redirect to third-party Android app stores
Redirect users to the relevant app store based on their device type. The APK package name ru.cian.main is used for redirection, so make sure the APK name exists in the store before you select it. ⓘ
[Learn more](#)

☒ Huawei AppGallery
☒ Xiaomi Market
☒ Samsung Galaxy Store
☒ Oppo Market
☒ Vivo App Store

Зона 2 – Система Android

Время жизни ссылки

Проблема 5:

Для ретаргетинговых ссылок стоит указывать время их жизни. **Иначе атрибуция не произойдет.** Установка будет выглядеть как обычный organic-install.

Lookback window & Cost parameters

Customize lookback window ? ☒

☒ Days ☐ Hours

 > 7 days

1 5 10 15 20 25 30

Зона 2 – Система Android

Регуляtorика и ограничения трекинга

Проблема 6:

Часть кликов не может быть атрибутирована из-за ограничения рекламных идентификаторов и VPN. Блокировщики рекламы на телефоне или роутере пользователя могут приводить к тому, что домен appsflyer.com будет заблокирован.



Необходимо обеспечить **прозрачную коммуникацию** между маркетингом и разработкой



Принять, что всегда будут клики, которые нельзя атрибутировать, **и это нормально**





Зона 1

Настройки кампании



Зона 2

Система Android

Зона 3

Приложение



Проблема 7: Таймаут 2 секунды (по-умолчанию) на получение ссылки из SDK оказался СЛИШКОМ НИЗКИМ

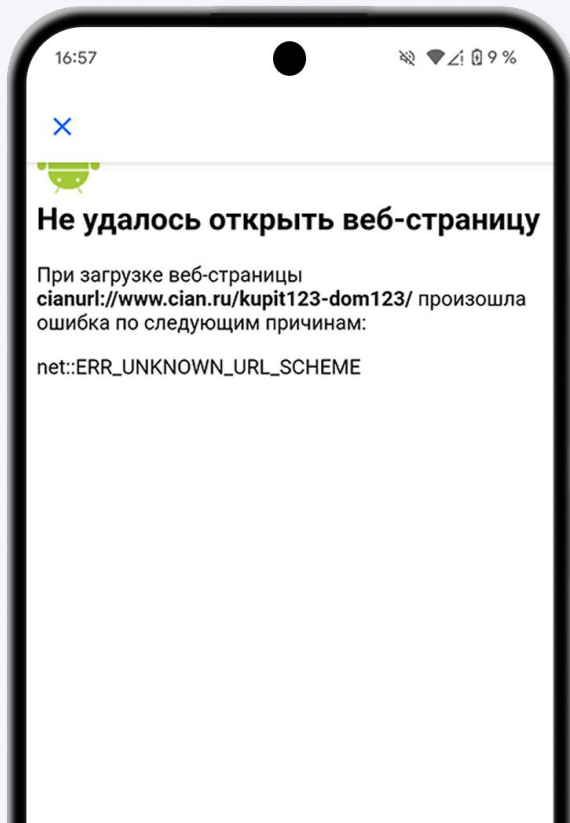
Длительность ответа	p50	p90	p99
	2,26 с	6,03 с	14,6 с

Зона 3 — Приложение

Навигация и обработчики deeplink

Проблема 8:

Некоторые из используемых ссылок со временем перестали работать. Причины разные: закончился или начался продуктовый эксперимент, фича могла быть удалена или переработана. Приложение меняется, и эти изменения могут влиять на диплинки.



Зона 3 — Приложение

Вывод



Единственная зона, которую мы, разработчики, **полностью контролируем кодом.**



Необходимо **обеспечить покрытие** мониторингом и тестами.





Зона 1

Настройки кампании



Зона 2

Система Android



Зона 3

Приложение



И что с этим делать?



Кадр из фильма «Пираты Карибского моря: Сундук мертвеца»

(*Pirates of the Caribbean: Dead Man's Chest*)



Путь А

«Закидать тестами и автоматизацией»: генерация ссылок, скриншотные тесты, авто-каталог

Путь Б

«Сначала навести порядок в процессе, а потом добавлять инструменты».

Почему мы сознательно НЕ выбрали автоматизацию первым шагом



Тесты уже были – и не помогали. Потому что некому было их актуализировать.
Автоматизация без владельца — это просто другой способ сломаться. Сначала нужно:

1

Разобраться,
кто за что отвечает

2

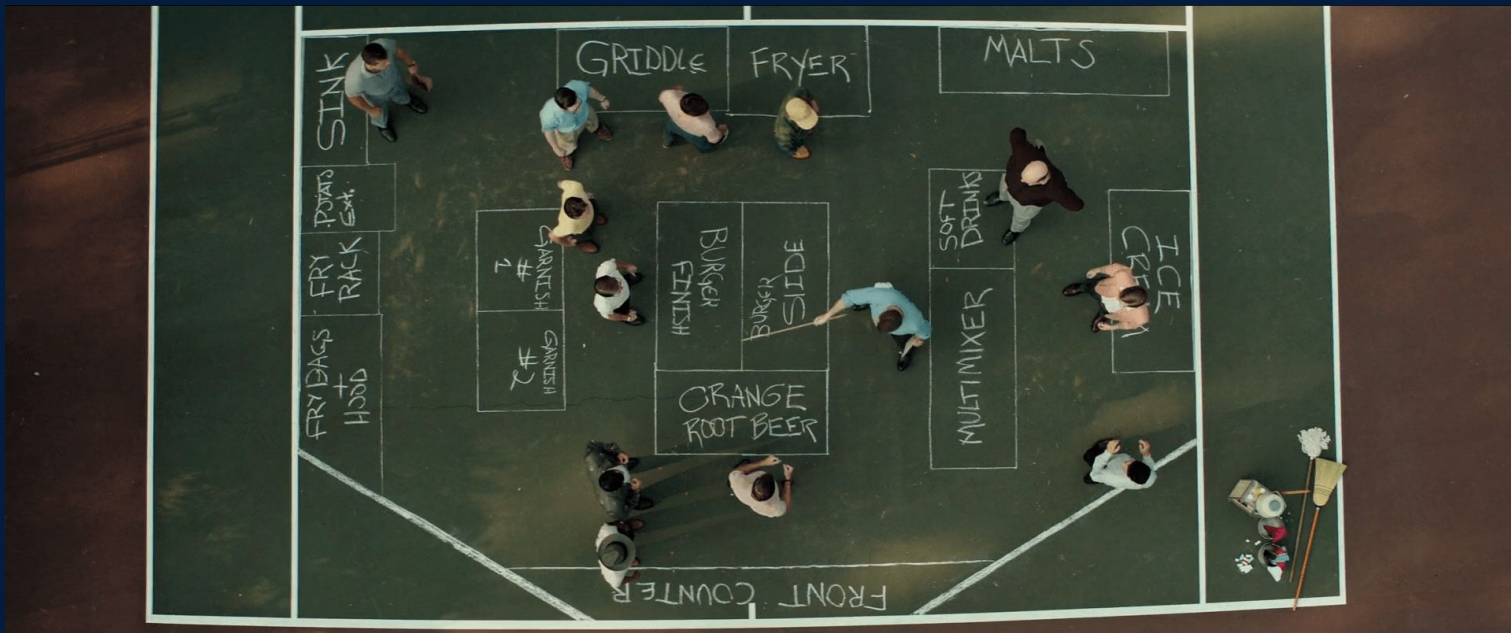
Дешево вскрыть нюансы,
которые не видны сейчас

3

Только после этого
автоматизируем



Решение: сначала процесс

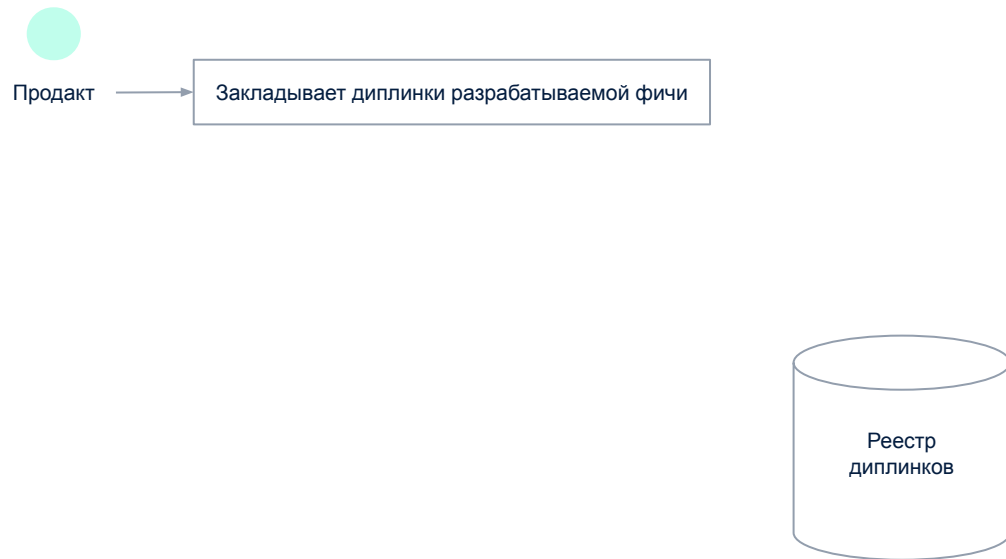


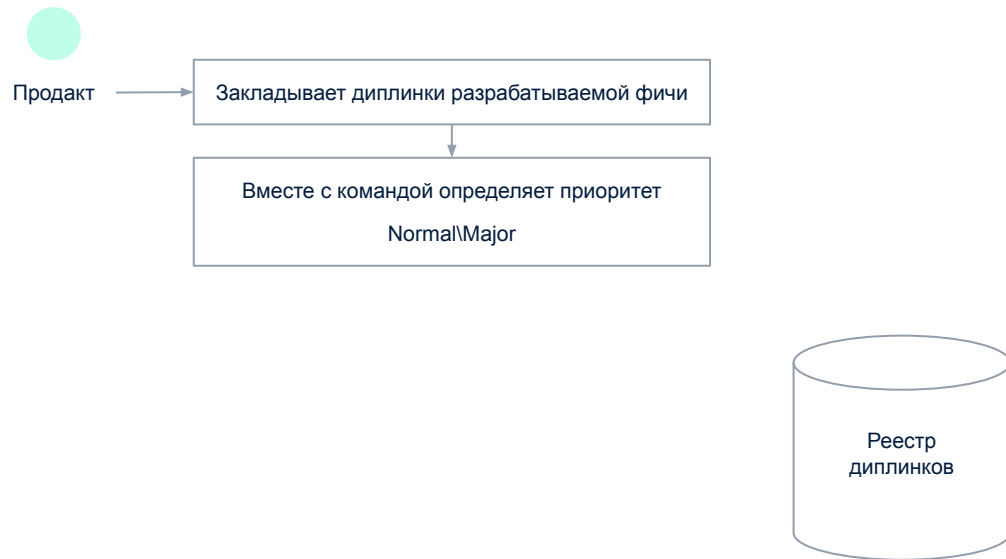
Кадр из фильма «Основатель» (The Founder)

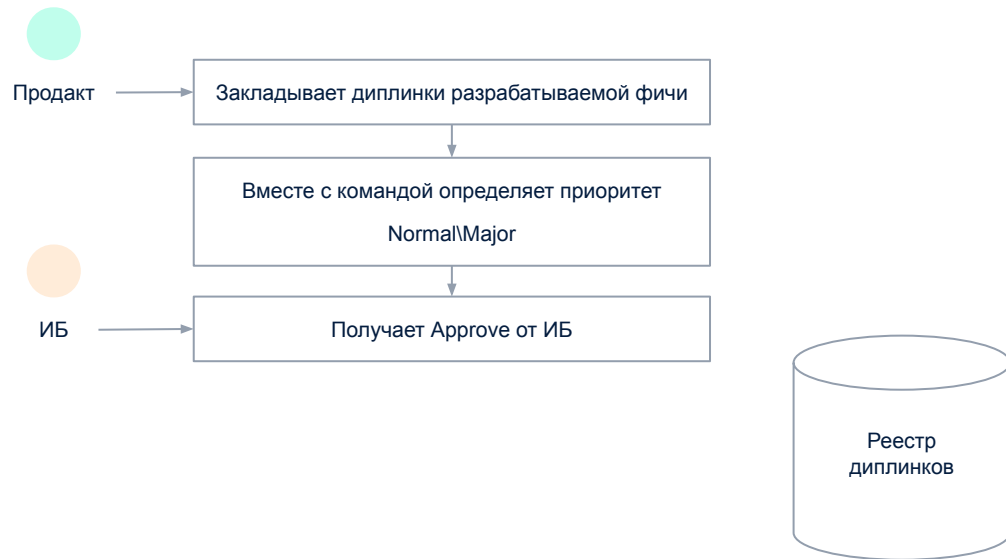


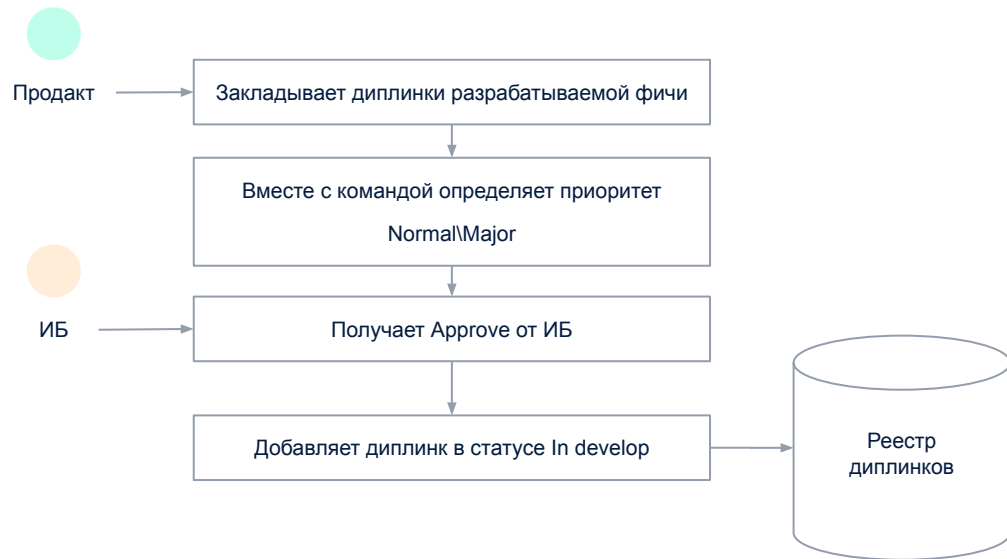
Продакт

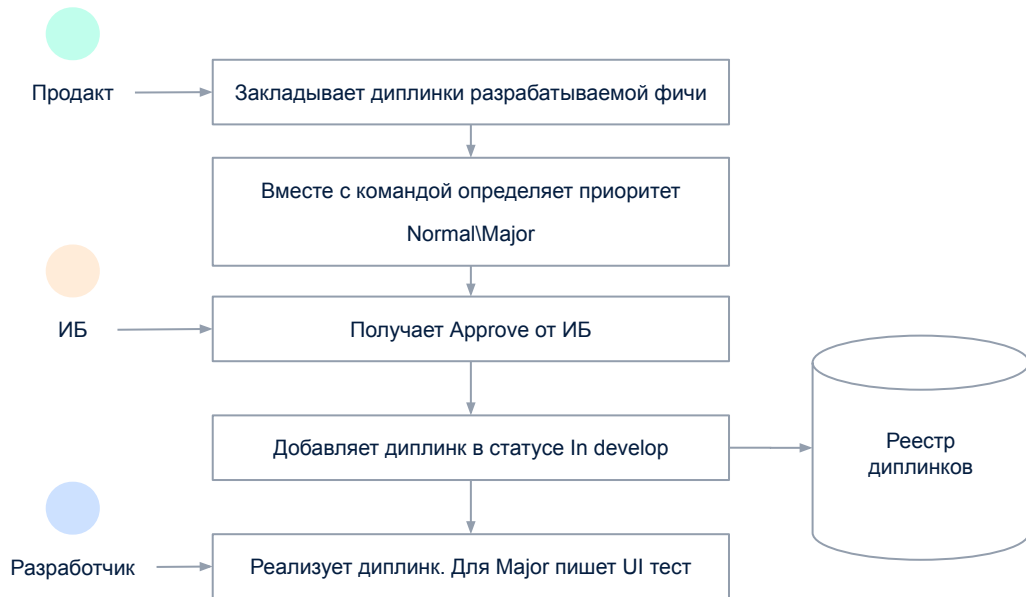


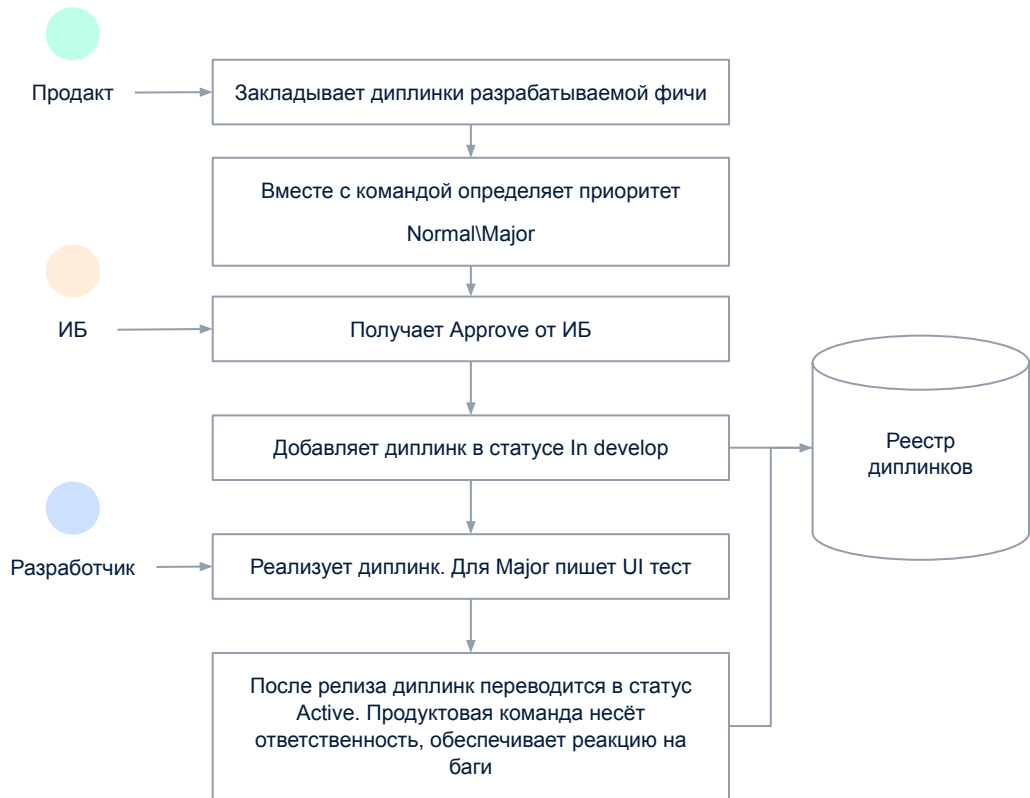


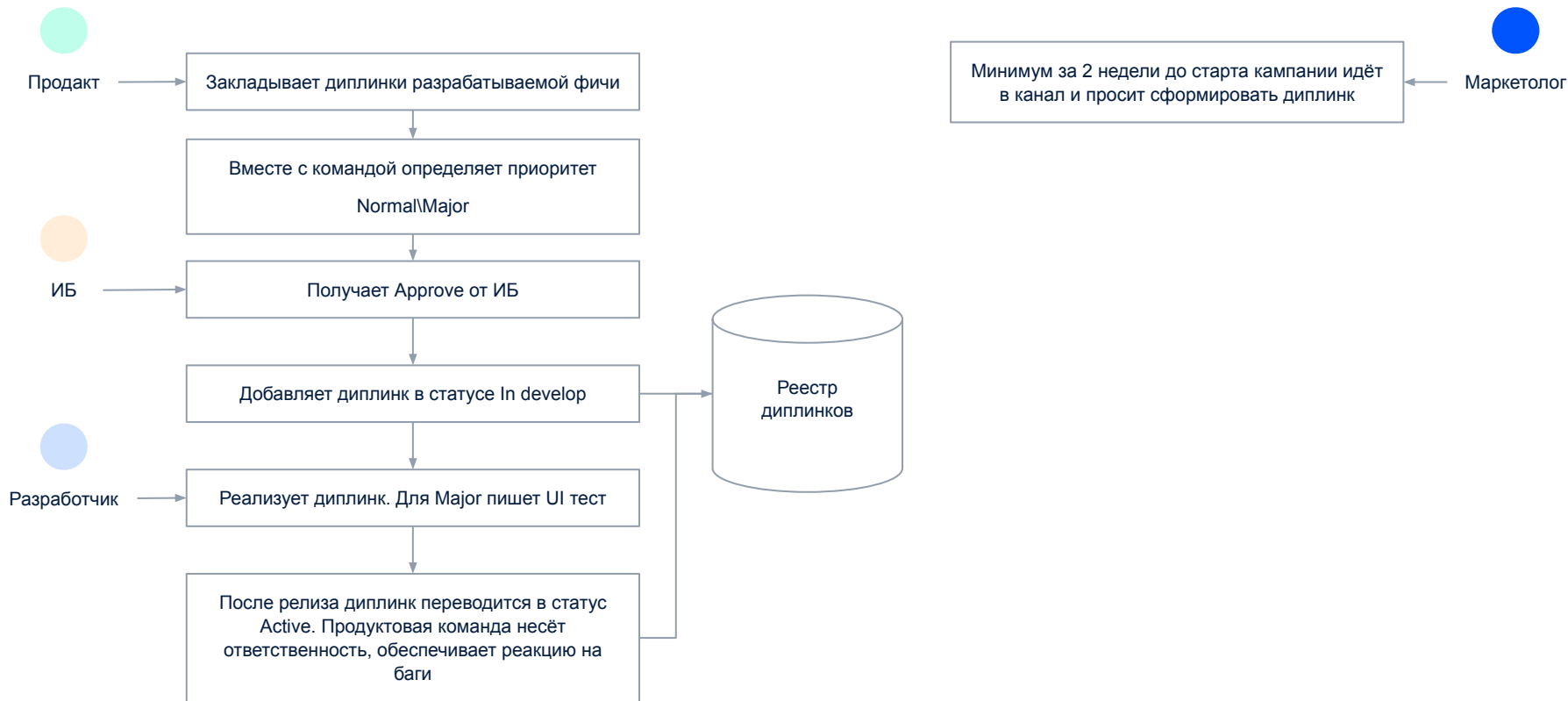


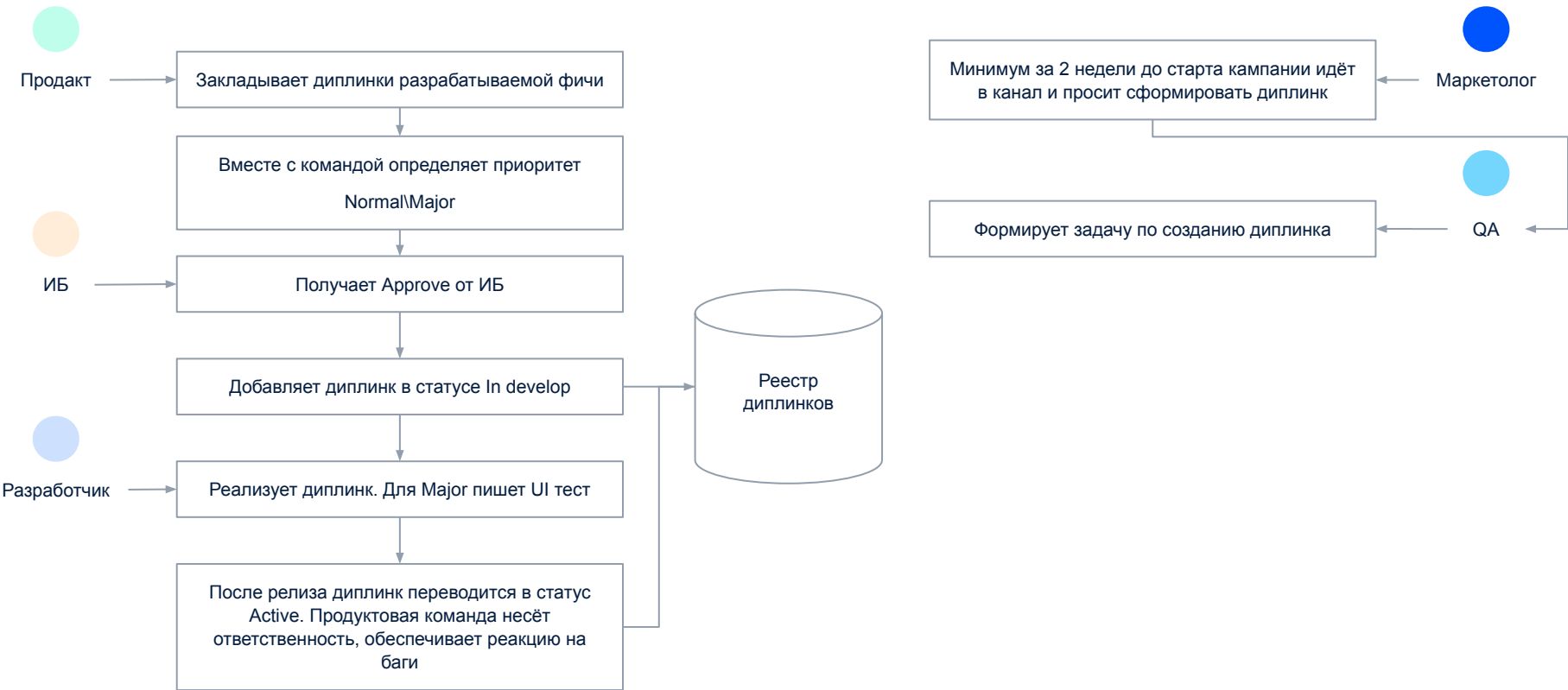


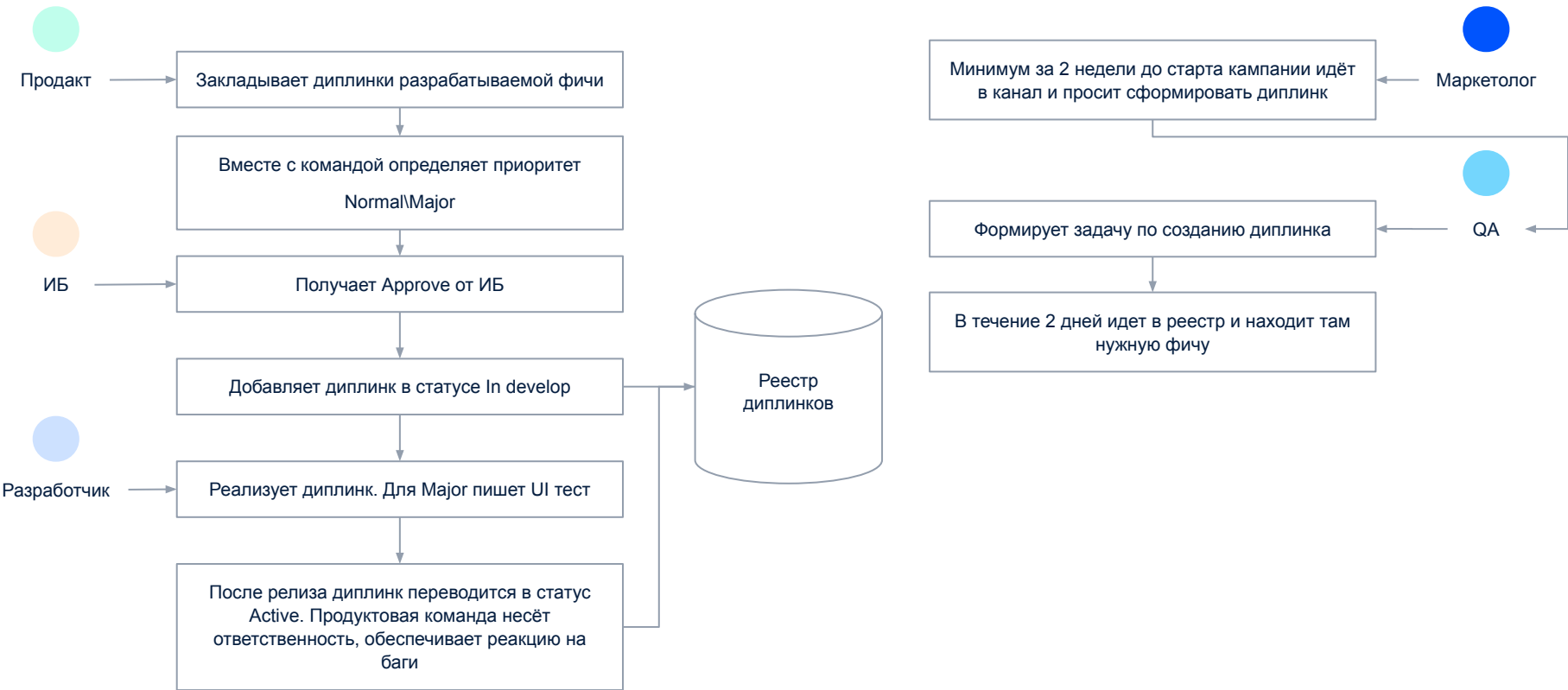


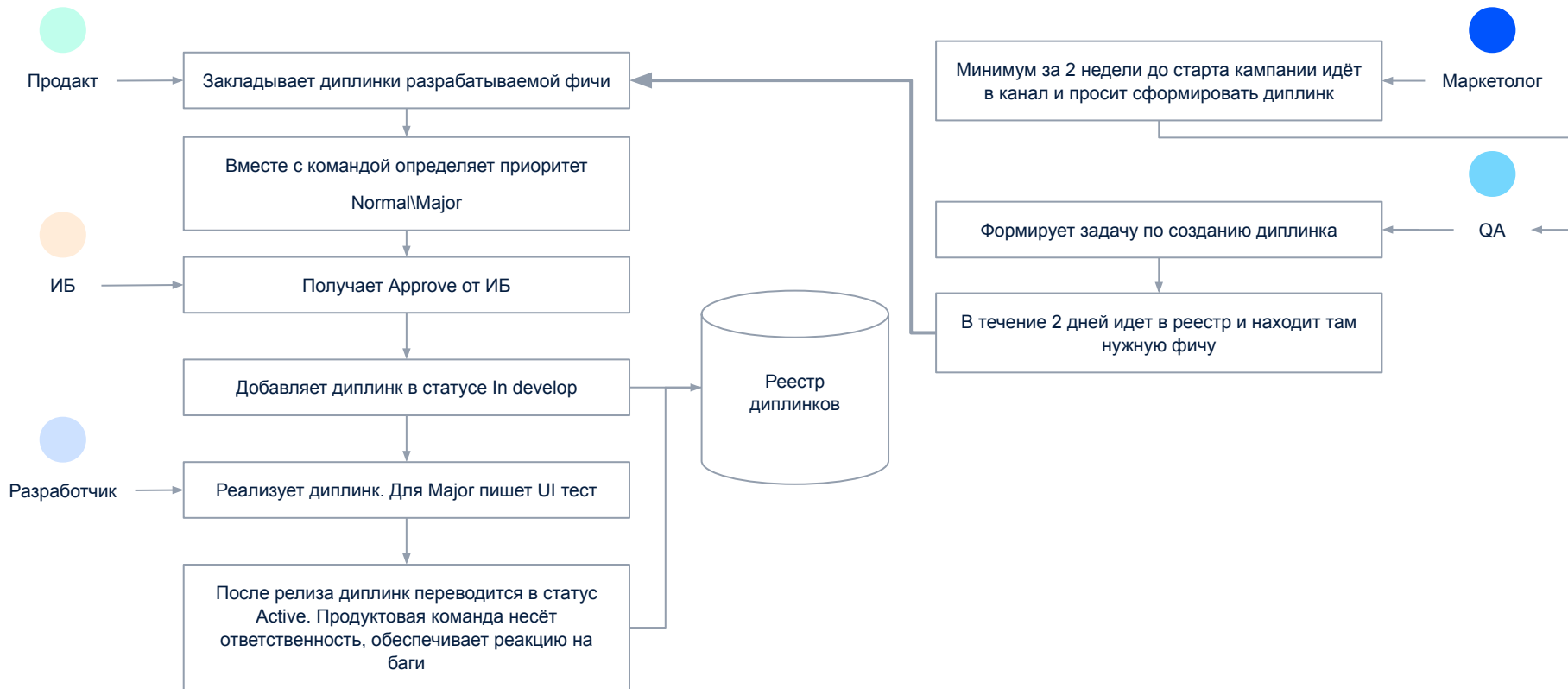


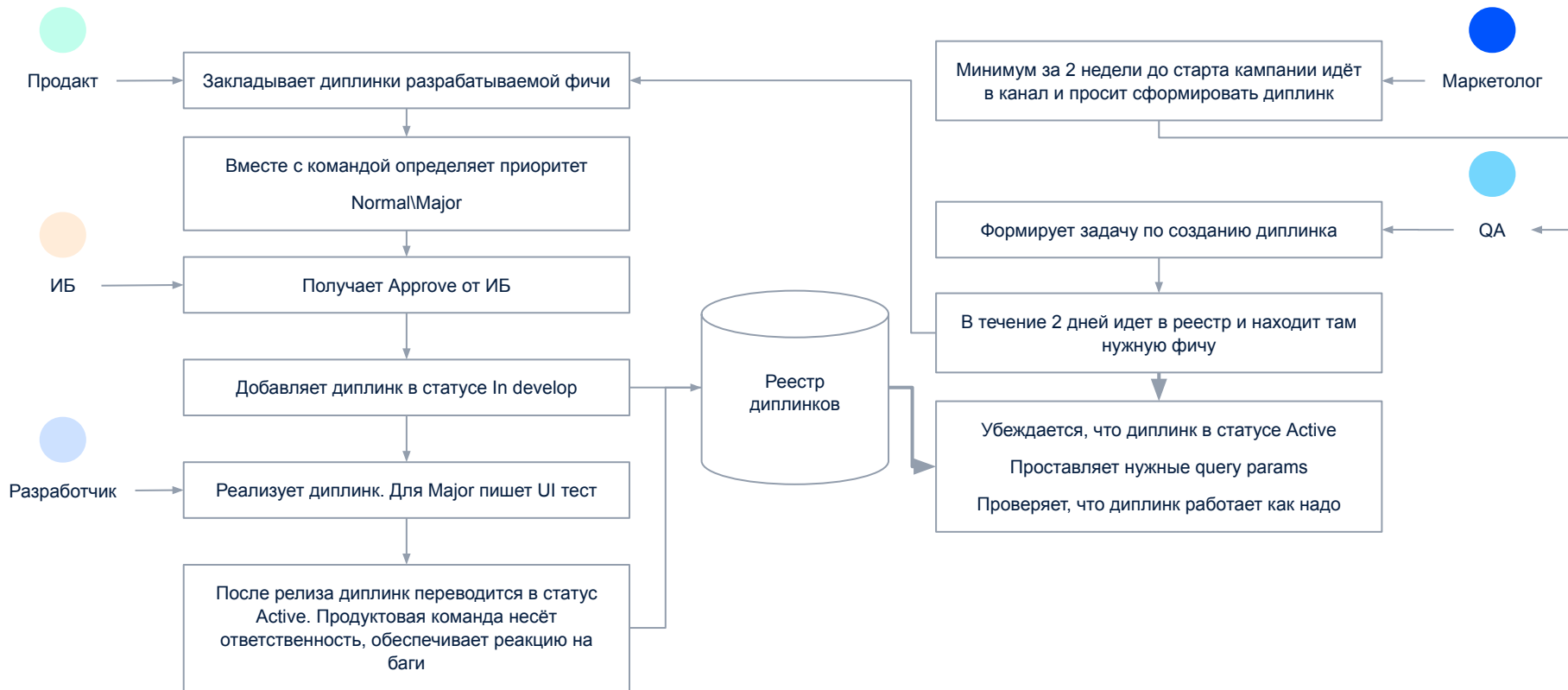


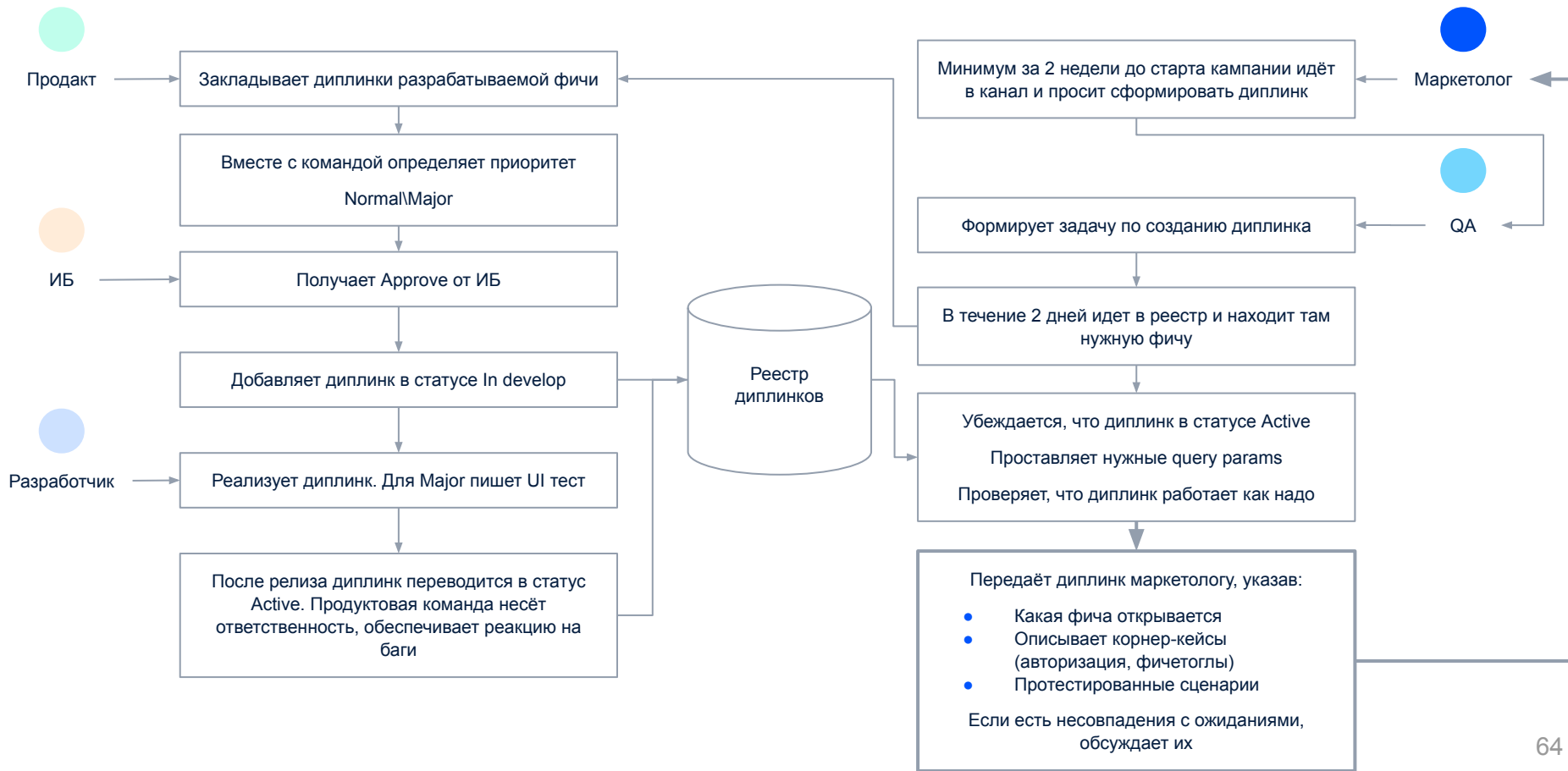












Что может пойти не так?



Диплинк нет в каталоге

QA обсуждает с командой, что
надо добавить. Заводит
задачу в ближайший релиз.



Диплинк не в статусе Active

QA уточняет у команды
причины, если сам не знает.
Принимает решение,
что делать совместно
с командой.



Ссылка сработала не так, как ожидалось

QA заводит баг Major
на продуктовую команду
и уведомляет её об этом.



По каждой проблеме

QA уведомляет маркетолога,
что нужно подождать
и планируемую дату фикса.



Маркетинг / подрядчик

Заказывает ссылку,
не генерирует её сам.



Продуктовая команда

Владелец своих диплинков.

- Управляет ЖЦ;
- Делает багфиксы;
- Определяет приоритет.



Информационная безопасность

Обязательно утверждает
каждый новый диплинк.



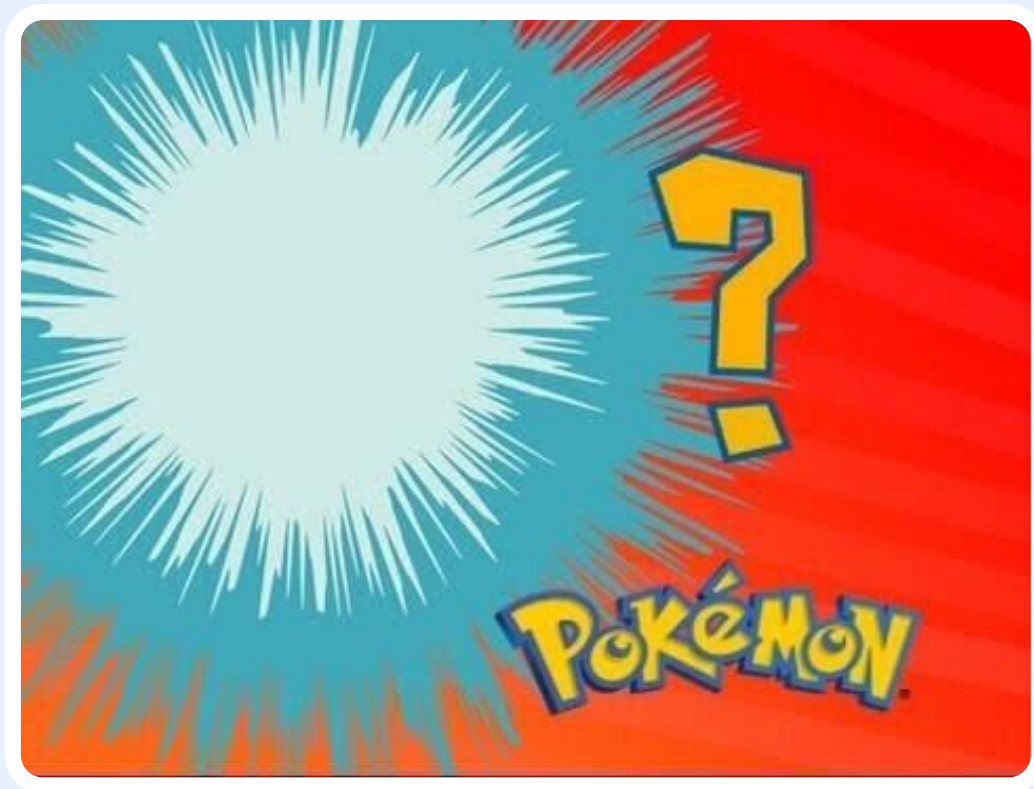
Разработка и QA

- Расширяет реестр по запросу.
- Тестируют, прежде чем передать ссылку заказчику с описанием.



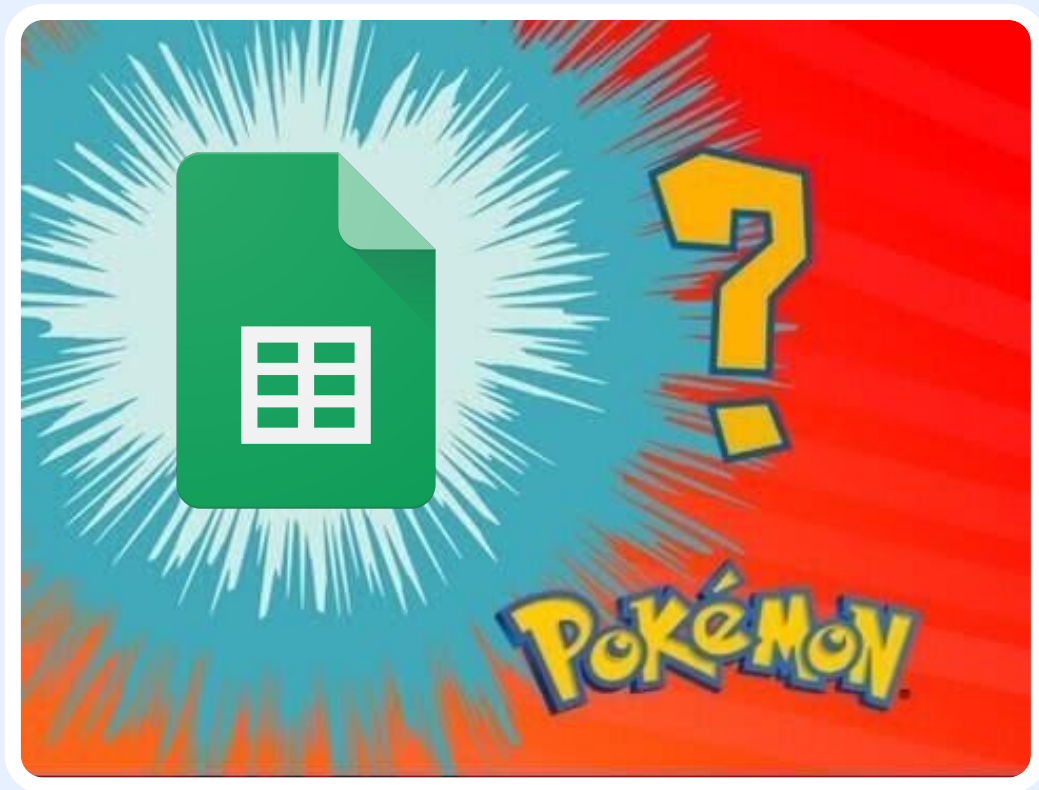
Что такое Deeplink реестр?

- Единый источник правды
- Не содержит чувствительных данных
- Можно легко дать доступ всем причастным



Что такое Deeplink реестр?

- Единый источник правды
- Не содержит чувствительных данных
- Можно легко дать доступ всем причастным



Что содержит Deerlink реестр?

Внутренний ID

Куда ведёт + скриншот

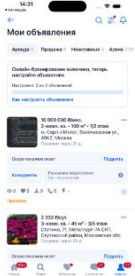
Ответственная команда

Приоритет

Status: In Develop/Active/
Removed/Deprecated/
Broken

Версия App + дата
раскатки **на 80%**

и другое

ID	Статус	Куда ведёт	Скриншот	Домен	Путь ссылки	Параметры ссылки	Примеры ссылок	Команда	Дата последней проверки	Требования	Релиз	Fallback
94	Active	Список Моих Объявлений с заданными фильтрами и открытием оффера по необходимости		my.cian.ru	my-offers	tab:String? - какую табу открыть: rent, sale, archived, inactive, declined offerType:String? - фильтр: flat, suburban, commercial dealType:String? - фильтр: rent, sale sortType:String? - сортировка: updatedAt, movedToArchiveDate, priceAsc, priceDesc, declinedDate, suggestDiscount, publishedDate, completionDate offerId:Int? - какой оффер открыть после открытия списка	https://my.cian.ru/my-offers https://my.cian.ru/my-offers/?offerId=1 https://my.cian.ru/my-offers/?tab=rent&offerType: https://my.cian.ru/my-offers/?tab=rent&offerType:	OCCAMY	06.07.2026	Авторизация. Только для риелторов	Очень давно	на and откоре риелтс открое

Теперь техника

Процесс закрывает 80% боли. Оставшиеся 20% — регрессии.
От них защищает техника.

Интеграционные тесты
на Normal

UI тесты на Major

Мониторинг





Major-deeplink

- Открывают критические бизнес-сценарии
- ~30 deeplink'ов
- Проверяются при помощи E2E UI-тестов



Normal-deeplink

- Открывают менее приоритетные экраны
- ~120 deeplink'ов
- Проверяются при помощи интеграционных тестов

Распределением занимаются **продуктовые команды**

Один диплинк = два класса

- **Qualifier** – примеры URL + условие срабатывания.
- **Factory** – сам обработчик, строит Intent-стек.

```
class FavoritesDeepLinkIntentFactoryQualifier
@Inject constructor() : DeepLinkIntentFactoryQualifier(
    testUrls = setOf(
        "https://www.cian.ru/favorites/",
        "https://www.cian.ru/rent/flat/favorites/",
    ),
    condition = {
        type {
            pathSegmentPrefixPredicate("/favorites")
        }
    }
)
```

```
class FavoritesDeepLinkIntentFactory
@Inject constructor(
    private val dashBoardStarter: ActivityStarter,
) : DeepLinkIntentFactory {
    override fun handleData(
        context: Context, data: DeepLinkData, tag: String,
    ): DeepLinkStack = /* строим Intent */
}
```

Один диплинк = два класса

- Qualifier – примеры URL + условие срабатывания.
- **Factory** – сам обработчик, строит Intent-стек.

```
class FavoritesDeepLinkIntentFactoryQualifier
@Inject constructor() : DeepLinkIntentFactoryQualifier(
    testUrls = setOf(
        "https://www.cian.ru/favorites/",
        "https://www.cian.ru/rent/flat/favorites/",
    ),
    condition = {
        type {
            pathSegmentPrefixPredicate("/favorites")
        }
    }
)
```

```
class FavoritesDeepLinkIntentFactory
@Inject constructor(
    private val dashboardStarter: ActivityStarter,
) : DeepLinkIntentFactory {
    override fun handleData(
        context: Context, data: DeepLinkData, tag:
String,
    ): DeepLinkStack = /* строим Intent */
    }
```

```
@Module class DeepLinkIntentFactoriesModule
```

```
@Provides
```

```
fun provideFactoriesMap(...): LinkedHashMap<DeepLinkIntentFactoryQualifier, DeepLinkIntentFactory> {
```

```
    FitstDeepLinkIntentFactoryQualifier
```

```
    FitstDeepLinkIntentFactory
```

```
    SecondDeepLinkIntentFactoryQualifier
```

```
    SecondDeepLinkIntentFactory
```

```
    ThirdDeepLinkIntentFactoryQualifier
```

```
    ThirdDeepLinkIntentFactory
```

```
}
```

```
@Module class DeepLinkIntentFactoriesModule
```

```
@Provides
```

```
fun provideFactoriesMap(...): LinkedHashMap<DeepLinkIntentFactoryQualifier, DeepLinkIntentFactory> {
```

```
    FirstDeepLinkIntentFactoryQualifier
```

```
    FirstDeepLinkIntentFactory
```

```
    SecondDeepLinkIntentFactoryQualifier
```

```
    SecondDeepLinkIntentFactory
```

```
    ThirdDeepLinkIntentFactoryQualifier
```

```
    ThirdDeepLinkIntentFactory
```

```
}
```

```
@Module class DeepLinkIntentFactoriesModule
```

```
@Provides
```

```
fun provideFactoriesMap(...): LinkedHashMap<DeepLinkIntentFactoryQualifier, DeepLinkIntentFactory> {
```

```
    FitstDeepLinkIntentFactoryQualifier
```

```
    FitstDeepLinkIntentFactory
```

```
    SecondDeepLinkIntentFactoryQualifier
```

```
    SecondDeepLinkIntentFactory
```

```
    ThirdDeepLinkIntentFactoryQualifier
```

```
    ThirdDeepLinkIntentFactory
```

```
}
```

Получение Factory

- **Через FactoryProvider**
- Ключ – DeepLinkData
- Проверяет соответствие DeepLinkData условиям из Qualifier

```
interface DeepLinkIntentFactoryProvider {

    fun getFactory(data: DeepLinkData):
        Pair<String, DeepLinkIntentFactory>

    @TestOnly
    fun getFactories(): Map<DeepLinkIntentFactoryQualifier,
        DeepLinkIntentFactory>
}

data class DeepLinkData(
    val url: String,
    val intentExtras: Bundle?
) {
    val scheme: String
    val host: String
    val path: String?
    val pathSegments: List<String>
    val queryParams: Map<String, String>
    val uri by lazy { Uri.parse(url) }
    ...
}
```

Получение Factory

- Через FactoryProvider
- **Ключ – DeepLinkData**
- Проверяет соответствие DeepLinkData условиям из Qualifier

```
interface DeepLinkIntentFactoryProvider {

    fun getFactory(data: DeepLinkData): Pair<String, DeepLinkIntentFactory>

    @TestOnly
    fun getFactories(): Map<DeepLinkIntentFactoryQualifier, DeepLinkIntentFactory>
}

data class DeepLinkData(
    val url: String,
    val intentExtras: Bundle?
) {
    val scheme: String
    val host: String
    val path: String?
    val pathSegments: List<String>
    val queryParams: Map<String, String>
    val uri by lazy { Uri.parse(url) }
    ...
}
```

Получение Factory

- Через FactoryProvider
- Ключ – DeepLinkData
- Проверяет соответствие DeepLinkData **условиям из Qualifier**

```
interface DeepLinkIntentFactoryProvider {  
  
    fun getFactory(data: DeepLinkData):  
        Pair<String, DeepLinkIntentFactory>  
  
    @TestOnly  
    fun getFactories(): Map<DeepLinkIntentFactoryQualifier, DeepLinkIntentFactory>  
}  
  
data class DeepLinkData(  
    val url: String,  
    val intentExtras: Bundle?  
) {  
    val scheme: String  
    val host: String  
    val path: String?  
    val pathSegments: List<String>  
    val queryParams: Map<String, String>  
    val uri by lazy { Uri.parse(url) }  
    ...  
}
```

1

Поднимаем реальный DI-граф

Запускаем DashboardActivity → Dagger собирает LinkedHashMap со всеми фабриками.

2

Проверяем уникальность URL

Собираем все testUrls из всех Qualifier в Set, сравниваем размеры Set и Map. Если где-то один и тот же URL прикреплен к двум разным фабрикам — тест падает.

3

Проверяем соответствие URL → Factory

Для каждого testUrl запрашиваем у провайдера фабрику и сравниваем с той, к чьему Qualifier этот URL принадлежит.

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashBoardActivity.instance(it, DashBoardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()
    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }

    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashboardActivity.instance(it, DashboardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()
    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }

    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashBoardActivity.instance(it, DashBoardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }

    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashBoardActivity.instance(it, DashBoardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()
    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }
    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashBoardActivity.instance(it, DashBoardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()
    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }
    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashBoardActivity.instance(it, DashBoardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()
    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }

    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashBoardActivity.instance(it, DashBoardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()
    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }

    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```

Интеграционный тест на диплинки

- Запускаем главный экран
- Проверяем, что url в зарегистрированных DeeplinkQualifier не повторяются
- Проверяем соответствие URL → Factory

```

step("Запускаем главный экран") {
    TestRouter.startScreen {
        DashBoardActivity.instance(it, DashBoardInitParams())
    }
}

step("Проверяем, что урлы в зарегистрированных DeeplinkQualifier не повторяются") {
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()
    val qualifiers = factoryProvider.getFactories().keys
    val urlList = qualifiers.map { it.testUrls.toList() }.flatten()

    val urlSet = urlList.toSet()

    assert(urlList.size == urlSet.size)
}

step("Проверяем, что получаем корректные фабрики по урлам") {
    val urlFactoryMap = mutableMapOf<String, DeepLinkIntentFactory>()
    val factoryProvider = CoreDiDependencies.get().dependenciesProvider
        .provide<DeepLinksAppDependencies>()
        .provideDeepLinkIntentFactoryProvider()

    factoryProvider.getFactories().forEach {
        it.key.testUrls.forEach { url ->
            urlFactoryMap[url] = it.value
        }
    }

    urlFactoryMap.forEach {
        try {
            val data = DeepLinkData(it.key, null)
            val actualFactory = factoryProvider.getFactory(data).second
            assert(actualFactory::class == it.value::class)
        } catch (throwable: Throwable) {
            val message = "Wrong factory ${it.value} for link ${it.key} "
            throw AssertionError(message, throwable)
        }
    }
}

```



- Проверяет, что **приложение реагирует на Deeplink** и condition возвращает правильную фабрику
- Проверяет, что **один URL не ведет на 2 фабрики**, исключает коллизии
- Не нужно отдельно добавлять диплинки, тест автоматом подхватывает все фабрики
- **Очень** быстро выполняется



- **Не проверяет**, что фабрика открывает целевой экран

Теперь техника



Интеграционные тесты
на Normal

UI тесты на Major

Мониторинг



Принципы для E2E тестов

Как обеспечить корректное состояние

- Ходят на реальный бэкенд в реальный БД.
- Состояние создаётся на лету через тестовые ручки, доступные только во внутреннем контуре.
- В коде существует статический флаг `BuildConfig.isAutotest()`, по которому можно настроить специальные условия для тестов. Например, не отправлять аналитику.
- Передача специальных сервисных заголовков позволяет бэкенде знать, что обращение идет из теста.

E2E UI-тест

- Создаём пользователя
- Авторизуемся
- Отправляем Intent с Uri диплинк в систему через ActivityTestRule
- Проверяем, что открылся необходимый экран

```
private const val DEEPLINK_FAVORITES = "https://www.cian.ru/favorites/"

class FavoritesDeeplinkUiTest {
    ...
    @get:Rule
    val rule = ActivityTestRule(DeepLinkActivity::class.java, true, false)

    fun openFavoritesByDeeplink() {
        lateinit var user: UserData
        kaspessoRule.before {
            user = CreateUser().createAgentAccount()
        }.after {
            // nothing
        }.run {
            step("Авторизуемся полученным пользователем") {
                scenario(
                    AuthorizationScenario(user)
                )
            }
            step("Открываем линк $DEEPLINK_FAVORITES") {
                val intent = Intent(Intent.ACTION_VIEW, Uri.parse(DEEPLINK_FAVORITES))
                rule.launchActivity(intent)
            }
            step("Проверяем, что открылся экран Избранного") {
                flakySafely(timeoutMs = 15 * 1000L) {
                    composeTestRule
                        .onNodeWithTag(FavoritesV2ComposeTags.screen)
                        .assertIsDisplayed()
                }
            }
        }
    }
}
```

Создание пользователя

- Заполняем данные пользователя
- Отправляем запрос в тестовую ручку
- Получаем ответ

```
fun createAgentAccount(agentType: String = "Specialist", money: Int = 3000):
UserData {
    val randomEmail = RandomizerUtil().getRandomEmail()
    val randomPassword = RandomizerUtil().getRandomPassword()
    val randomPhone = RandomizerUtil().getRandomPhone()

    val userBody = CreateTestUserModel(
        email = createEmail(randomEmail),
        password = randomPassword,
        phone = randomPhone,
        subAccountCount = 1,
        agentType = agentType,
        isProfileApproved = true,
        phoneConfirmed = true,
        money = money
    )

    val body = CreateTestUserRequest(
        user = userBody
    )
    val response = TestApiService.instance.createTestUser(body).blockingGet()

    return UserData(
        userId = response.createdUser.userId.toString(),
        password = response.createdUser.password,
        email = response.createdUser.email,
```

Создание пользователя

- Заполняем данные пользователя
- Отправляем запрос в тестовую ручку
- Получаем ответ

```
fun createAgentAccount(agentType: String = "Specialist", money: Int = 3000):
UserData {
    val randomEmail = RandomizerUtil().getRandomEmail()
    val randomPassword = RandomizerUtil().getRandomPassword()
    val randomPhone = RandomizerUtil().getRandomPhone()

    val userBody = CreateTestUserModel(
        email = createEmail(randomEmail),
        password = randomPassword,
        phone = randomPhone,
        subAccountCount = 1,
        agentType = agentType,
        isProfileApproved = true,
        phoneConfirmed = true,
        money = money
    )

    val body = CreateTestUserRequest(
        user = userBody
    )

    val response =
TestApiService.instance.createTestUser(body).blockingGet()

    return UserData(
        userId = response.createdUser.userId.toString(),
        password = response.createdUser.password,
        email = response.createdUser.email,
        phone = response.phone
    )
}
```

Создание пользователя

- Заполняем данные пользователя
- Отправляем запрос в тестовую ручку
- Получаем ответ

```
fun createAgentAccount(agentType: String = "Specialist", money: Int = 3000):
UserData {
    val randomEmail = RandomizerUtil().getRandomEmail()
    val randomPassword = RandomizerUtil().getRandomPassword()
    val randomPhone = RandomizerUtil().getRandomPhone()

    val userBody = CreateTestUserModel(
        email = createEmail(randomEmail),
        password = randomPassword,
        phone = randomPhone,
        subAccountCount = 1,
        agentType = agentType,
        isProfileApproved = true,
        phoneConfirmed = true,
        money = money
    )

    val body = CreateTestUserRequest(
        user = userBody
    )
    val response = TestApiService.instance.createTestUser(body).blockingGet()

    return UserData(
        userId = response.createdUser.userId.toString(),
        password = response.createdUser.password,
        email = response.createdUser.email,
        phone = response.phone
    )
}
```

E2E UI-тест

- Создаём пользователя
- Авторизуемся
- Отправляем Intent с Uri диплинк в систему через ActivityTestRule
- Проверяем, что открылся необходимый экран

```
private const val DEEPLINK_FAVORITES = "https://www.cian.ru/favorites/"

class FavoritesDeeplinkUiTest {
    ...
    @get:Rule
    val rule = ActivityTestRule(DeepLinkActivity::class.java, true, false)

    fun openFavoritesByDeeplink() {
        lateinit var user: UserData
        kaspessoRule.before {
            user = CreateUser().createAgentAccount()
        }.after {
            // nothing
        }.run {
            step("Авторизуемся полученным пользователем") {
                scenario(
                    AuthorizationScenario(user)
                )
            }
            step("Открываем линк $DEEPLINK_FAVORITES") {
                val intent = Intent(Intent.ACTION_VIEW, Uri.parse(DEEPLINK_FAVORITES))
                rule.launchActivity(intent)
            }
            step("Проверяем, что открылся экран Избранного") {
                flakySafely(timeoutMs = 15 * 1000L) {
                    composeTestRule
                        .onNodeWithTag(FavoritesV2ComposeTags.screen)
                        .assertIsDisplayed()
                }
            }
        }
    }
}
```

E2E UI-тест

- Создаём пользователя
- **Авторизуемся**
- Отправляем Intent с Uri диплинк в систему через ActivityTestRule
- Проверяем, что открылся необходимый экран

```
private const val DEEPLINK_FAVORITES = "https://www.cian.ru/favorites/"

class FavoritesDeeplinkUiTest {
    ...
    @get:Rule
    val rule = ActivityTestRule(DeepLinkActivity::class.java, true, false)

    fun openFavoritesByDeeplink() {
        lateinit var user: UserData
        kaspressoRule.before {
            user = CreateUser().createAgentAccount()
        }.after {
            // nothing
        }.run {
            step("Авторизуемся полученным пользователем") {
                scenario(
                    AuthorizationScenario(user)
                )
            }
            step("Открываем линк $DEEPLINK_FAVORITES") {
                val intent = Intent(Intent.ACTION_VIEW, Uri.parse(DEEPLINK_FAVORITES))
                rule.launchActivity(intent)
            }
            step("Проверяем, что открылся экран Избранного") {
                flakySafely(timeoutMs = 15 * 1000L) {
                    composeTestRule
                        .onNodeWithTag(FavoritesV2ComposeTags.screen)
                        .assertIsDisplayed()
                }
            }
        }
    }
}
```

Авторизация

- Заполняем данные для открытия экрана авторизации
- Отправляем роутеру команду на открытие экрана

```
class AuthorizationScenario<ScenarioData>(user: UserData) :
    BaseScenario<ScenarioData>() {

    override val steps: TestContext<ScenarioData>().() -> Unit = {

        step("Вводим данные созданного пользователя и авторизуемся") {
            TestRouter.startScreen {
                val initParams = AuthScreenInitParams(
                    startFragmentType = StartFragmentType.LOGIN,
                    isAutoLogin = true,
                    email = user.userId,
                    password = user.password,
                    openFrom = AuthorizationScenario::class.java.name
                )
                MainAuthorizationActivity.instance(it, initParams)
            }
        }
    }
}
```

Авторизация

- Заполняем данные для открытия экрана авторизации
- Отправляем роутеру команду на открытие экрана

```
class AuthorizationScenario<ScenarioData>(user: UserData) :
    BaseScenario<ScenarioData>() {

    override val steps: TestContext<ScenarioData>.<() -> Unit = {

        step("Вводим данные созданного пользователя и авторизуемся") {
            TestRouter.startScreen {
                val initParams = AuthScreenInitParams(
                    startFragmentType = StartFragmentType.LOGIN,
                    isAutoLogin = true,
                    email = user.userId,
                    password = user.password,
                    openFrom =
                AuthorizationScenario::class.java.name
            )
            MainAuthorizationActivity.instance(it, initParams)
        }
    }
}
```

Е2Е UI-тест

- Создаём пользователя
- **Авторизуемся**
- Отправляем Intent с Uri диплинк в систему через ActivityTestRule
- Проверяем, что открылся необходимый экран

```
private const val DEEPLINK_FAVORITES = "https://www.cian.ru/favorites/"

class FavoritesDeeplinkUiTest {
    ...
    @get:Rule
    val rule = ActivityTestRule(DeepLinkActivity::class.java, true, false)

    fun openFavoritesByDeeplink() {
        lateinit var user: UserData
        kaspressoRule.before {
            user = CreateUser().createAgentAccount()
        }.after {
            // nothing
        }.run {
            step("Авторизуемся полученным пользователем") {
                scenario(
                    AuthorizationScenario(user)
                )
            }
            step("Открываем линк $DEEPLINK_FAVORITES") {
                val intent = Intent(Intent.ACTION_VIEW, Uri.parse(DEEPLINK_FAVORITES))
                rule.launchActivity(intent)
            }
            step("Проверяем, что открылся экран Избранного") {
                flakySafely(timeoutMs = 15 * 1000L) {
                    composeTestRule
                        .onNodeWithTag(FavoritesV2ComposeTags.screen)
                        .assertIsDisplayed()
                }
            }
        }
    }
}
```

E2E UI-тест

- Создаём пользователя
- Авторизуемся
- Отправляем Intent с Uri диплинк в систему через **ActivityTestRule**
- Проверяем, что открылся необходимый экран

```
private const val DEEPLINK_FAVORITES = "https://www.cian.ru/favorites/"

class FavoritesDeeplinkUiTest {
    ...
    @get:Rule
    val rule = ActivityTestRule(DeepLinkActivity::class.java, true, false)

    fun openFavoritesByDeeplink() {
        lateinit var user: UserData
        kaspressoRule.before {
            user = CreateUser().createAgentAccount()
        }.after {
            // nothing
        }.run {
            step("Авторизуемся полученным пользователем") {
                scenario(
                    AuthorizationScenario(user)
                )
            }

            step("Открываем линк $DEEPLINK_FAVORITES") {
                val intent =
                    Intent(Intent.ACTION_VIEW, Uri.parse(DEEPLINK_FAVORITES))
                rule.launchActivity(intent)
            }

            step("Проверяем, что открылся экран Избранного") {
                flakySafely(timeoutMs = 15 * 1000L) {
                    composeTestRule
                        .onNodeWithTag(FavoritesV2ComposeTags.screen)
                        .assertIsDisplayed()
                }
            }
        }
    }
}
```

E2E UI-тест

- Создаём пользователя
- Авторизуемся
- Отправляем Intent с Uri диплинк в систему через ActivityTestRule
- Проверяем, что открылся необходимый экран

```
private const val DEEPLINK_FAVORITES = "https://www.cian.ru/favorites/"

class FavoritesDeeplinkUiTest {
    ...
    @get:Rule
    val rule = ActivityTestRule(DeepLinkActivity::class.java, true, false)

    fun openFavoritesByDeeplink() {
        lateinit var user: UserData
        kaspressoRule.before {
            user = CreateUser().createAgentAccount()
        }.after {
            // nothing
        }.run {
            step("Авторизуемся полученным пользователем") {
                scenario(
                    AuthorizationScenario(user)
                )
            }
            step("Открываем линк $DEEPLINK_FAVORITES") {
                val intent = Intent(Intent.ACTION_VIEW, Uri.parse(DEEPLINK_FAVORITES))
                rule.launchActivity(intent)
            }
            step("Проверяем, что открылся экран Избранного") {
                flakySafely(timeoutMs = 15 * 1000L) {
                    composeTestRule
                        .onNodeWithTag(FavoritesV2ComposeTags.screen)
                        .assertIsDisplayed()
                }
            }
        }
    }
}
```



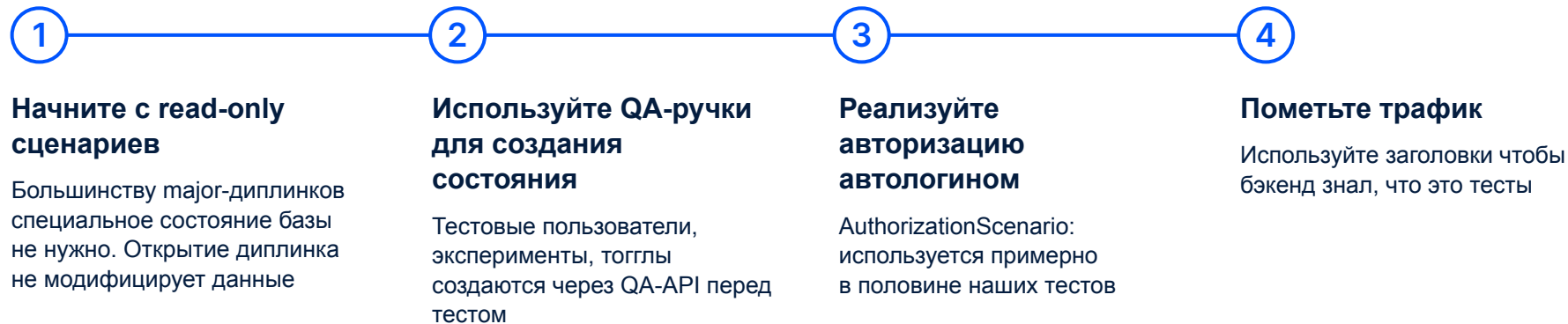
- Проверяет полный путь пользователя от отправки intent в систему до открытия целевого экрана
- Работает на **реальном** окружении



- Требуется писать отдельно на каждый major-deerlink
- Долго выполняется (30 тестов ~ 20 минут)
- Может флакать

С чего начать внедрение тестов

Если бы у нас их не было



Теперь техника



Интеграционные тесты
на Normal



UI тесты на Major

Мониторинг

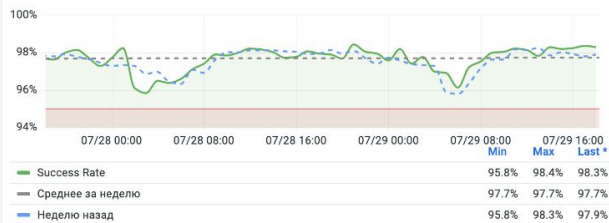


AppsFlyer (Deeplinks)

Success Rate (Получения DeeplinkResult) ▾



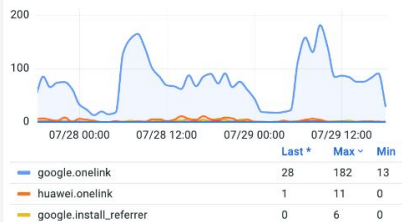
Success Rate (Получения DeeplinkResult)



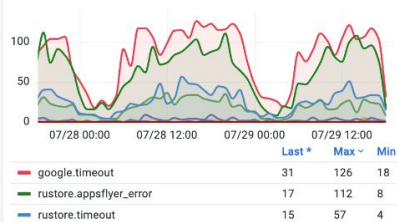
Трафик (Получения DeeplinkResult)



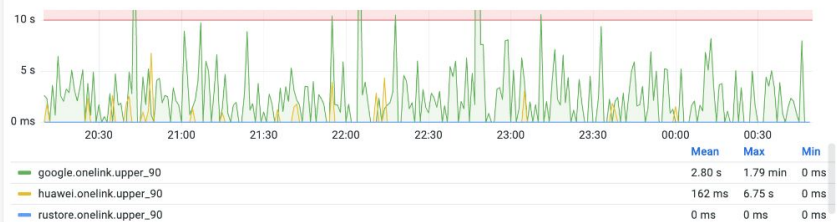
Успешно обработанные отложенные Deeplinks по сторам



Ошибки отложенных Deeplinks по сторам



Appsflyer response time (отложенные диплинки)



Дашборд в Grafana



Техметрики

- Процент успешных инициализаций AF*
- Процент успешно полученных Conversion Data
- Время ответа AF* в 90 перцентиле
- Процент распаршенных диплинков
- Процент диплинков, для которых найдены обработчики
- Процент открытых навигацией целевых экранов
- ...



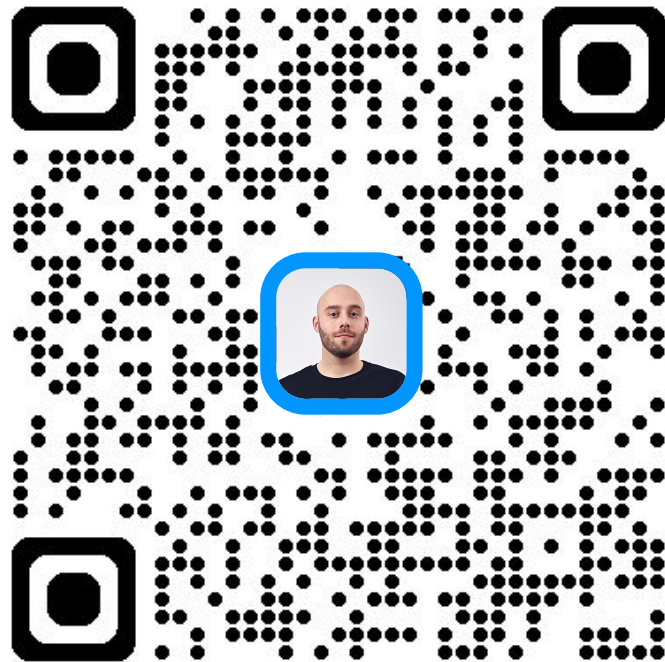
SLO-сценарии

- Процент успешных получений диплинка
- Величина потока успешных получений диплинка день-ко-дню, неделя-к-неделе

Подробнее про SLO и метрики в Циан

Читайте на нашем Хабре

Как следить за состоянием мобильного приложения?
<https://habr.com/ru/companies/cian/articles/843822/>





90% → 98%

Успешно обработанных ретаргетинговых deeplink'ов



Качественные изменения



Маркетинг

Перестал бояться
запускать кампании
на новые фичи.
Получил предсказуемый
интерфейс.



Разработчики

Перестали быть
единственным
носителем знания.



QA

Получили чёткие
критерии и реестр как
рабочий инструмент.



Информационная безопасность

Участвуют в процессе,
а не наблюдают
со стороны.



Чему мы научились



Сначала ответственность, потом инструменты

Без владельца автоматизация — это просто новый способ сломаться.

Тестирование диплинков ≠
Тестирование экранов

Это отдельная дисциплина с собственным стеком отказов.

Ручная проверка перед кампанией – самая дешёвая и эффективная защита

Дороже автоматике, но окупается на первой же спасённой кампании.

Мы узнаём о поломке раньше, чем теряем деньги

SLO с алертами заменяет «маркетинг прибежал в Slack» и «пользователь обратился в поддержку».

Каждый клик по рекламе — оплачен

Каждый сломанный клик — потерян безвозвратно!



**В начале доклада я спросил вас,
сколько маркетингового бюджета
сжигаете**

В начале доклада я спросил вас, сколько маркетингового бюджета сжигаете

«Откройте рекламную кампанию, кликните по ссылке, поставьте
приложение из стор, посмотрите, куда вас приведёт»

В начале доклада я спросил вас, сколько маркетингового бюджета сжигаете

«Откройте рекламную кампанию, кликните по ссылке, поставьте приложение из стор, посмотрите, куда вас приведёт»

Если что-то пошло не так – вам стоит задаться этим вопросом :)

Спасибо за внимание

Телеграм:
[@dkonopelkin](https://t.me/dkonopelkin)



Почта: d.konopelkin@cian.ru

