ТИНЬКОФФ

Ленивый граф

Или как мы оптимизировали старт приложения

План доклада

- Как работает граф зависимостей
- Зачем его делать ленивым
- Стратегии ленивой инициализации
- Результаты на нашем проекте

Александр Таганов

Тимлид команды маркетинговые коммуникации

a.taganov@tinkoff.ru

Почему ленивая инициализация?

Положение проекта на Q422

Положение проекта на Q422

Рост кодовой базы на 30% в год

Положение проекта на Q422

Рост кодовой базы на 30% в год

Рост корневого компонента

Положение проекта на Q422

Рост кодовой базы на 30% в год

Рост корневого компонента

Рост времени инициализации di модулей

Положение проекта на Q422

Рост кодовой базы на 30% в год

Рост корневого компонента

Рост времени инициализации di модулей

Рост времени сборки и индексации App модуля

Цель

Цель

Бизнес

Ускорить старт приложения

Цель

Бизнес

Ускорить старт приложения

Разработчики

Ускорить время сборки

План рефакторинга

План рефакторинга

AppComponent

Разделить на части и вынести код в модули

План рефакторинга

AppComponent

Разделить на части и вынести код в модули

Миграция на ManualDI

Избавиться от кодогенерации. Ускорить сборку

План рефакторинга

AppComponent

Разделить на части и вынести код в модули.

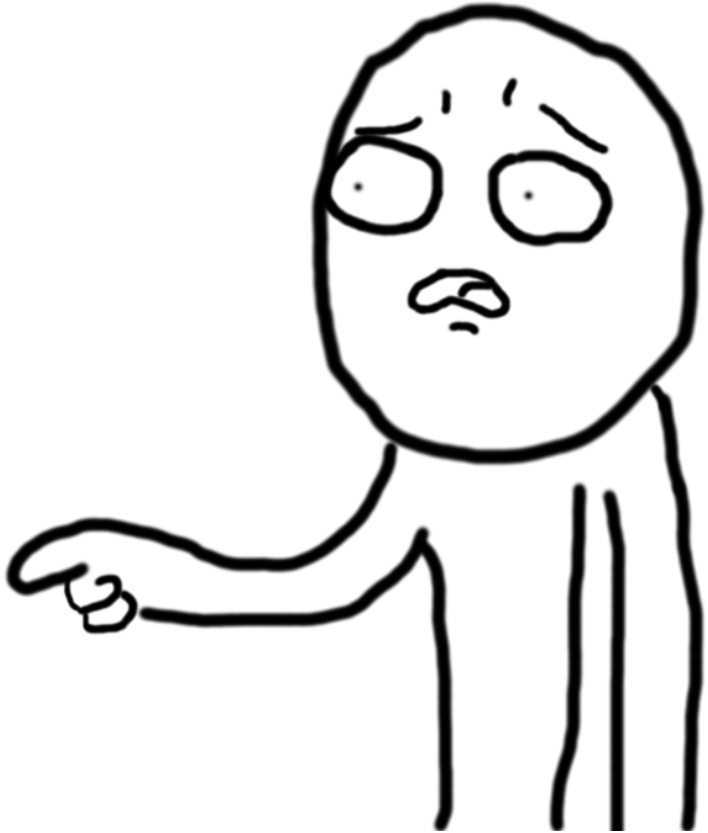
Миграция на ManualDI

Избавиться от кодогенерации. Ускорить сборку

Ленивая инициализация

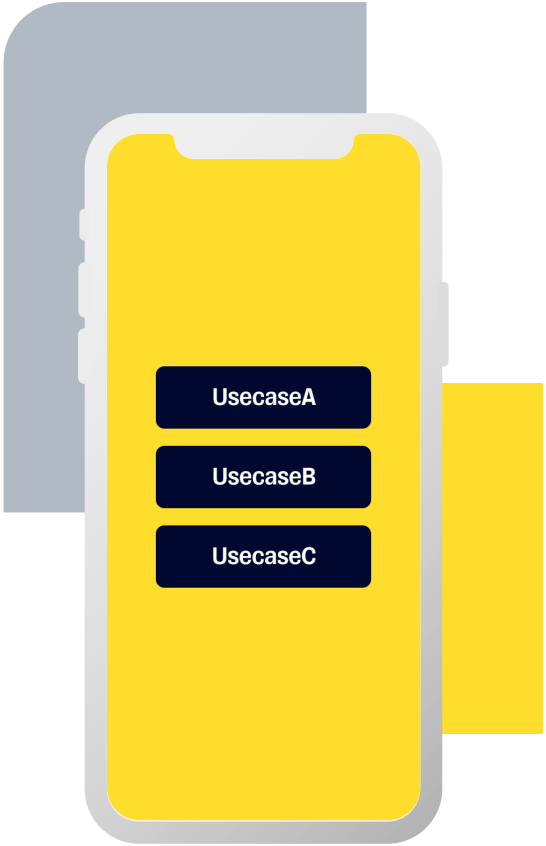
Перевести проблемные компоненты на ленивые рельсы. Ускорить старт приложения

Документация по ManualDI



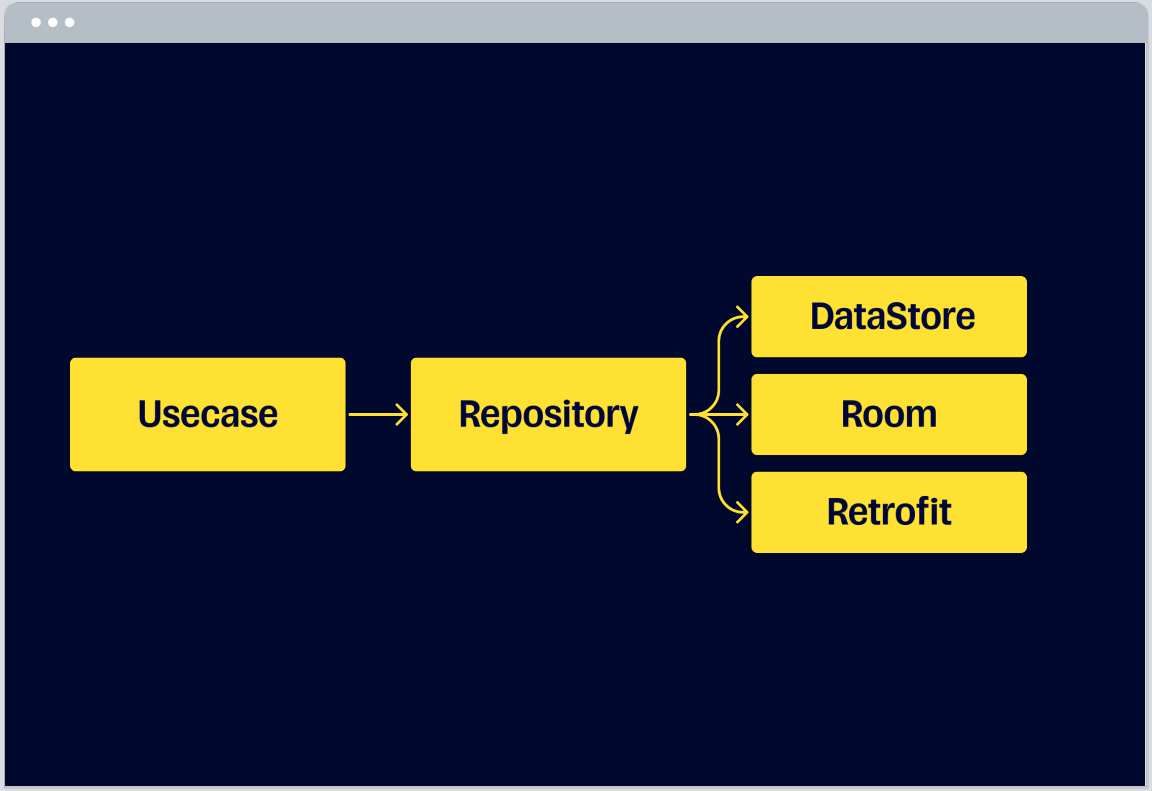

I NEVER ASKED FOR THIS

Как устроен граф зависимостей?

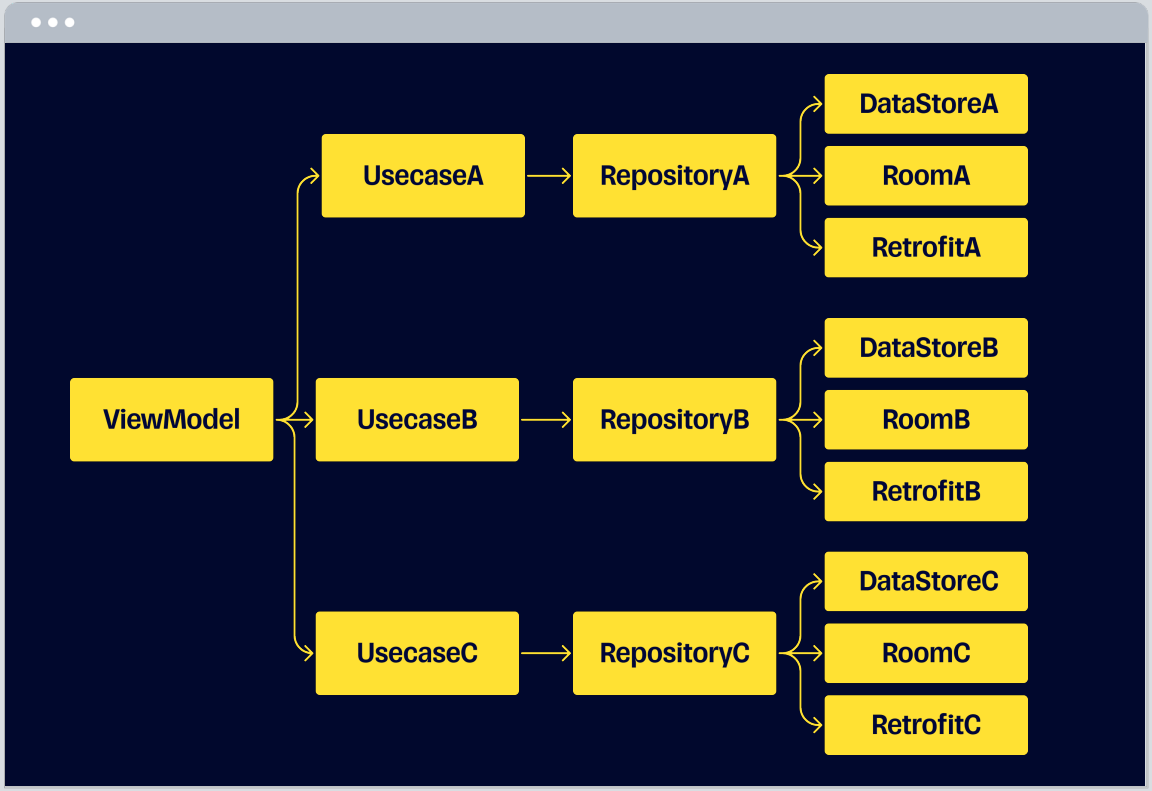


Тестовое приложение

Структура юзкейса



Граф зависимостей



Как проходили замеры?

Как проходили замеры?

Реальный девайс Realme 8

Как проходили замеры?

Реальный девайс Realme 8

Release версия

Как проходили замеры?

Реальный девайс Realme 8

Release версия

20 прогонов

Как проходили замеры?

Реальный девайс Realme 8

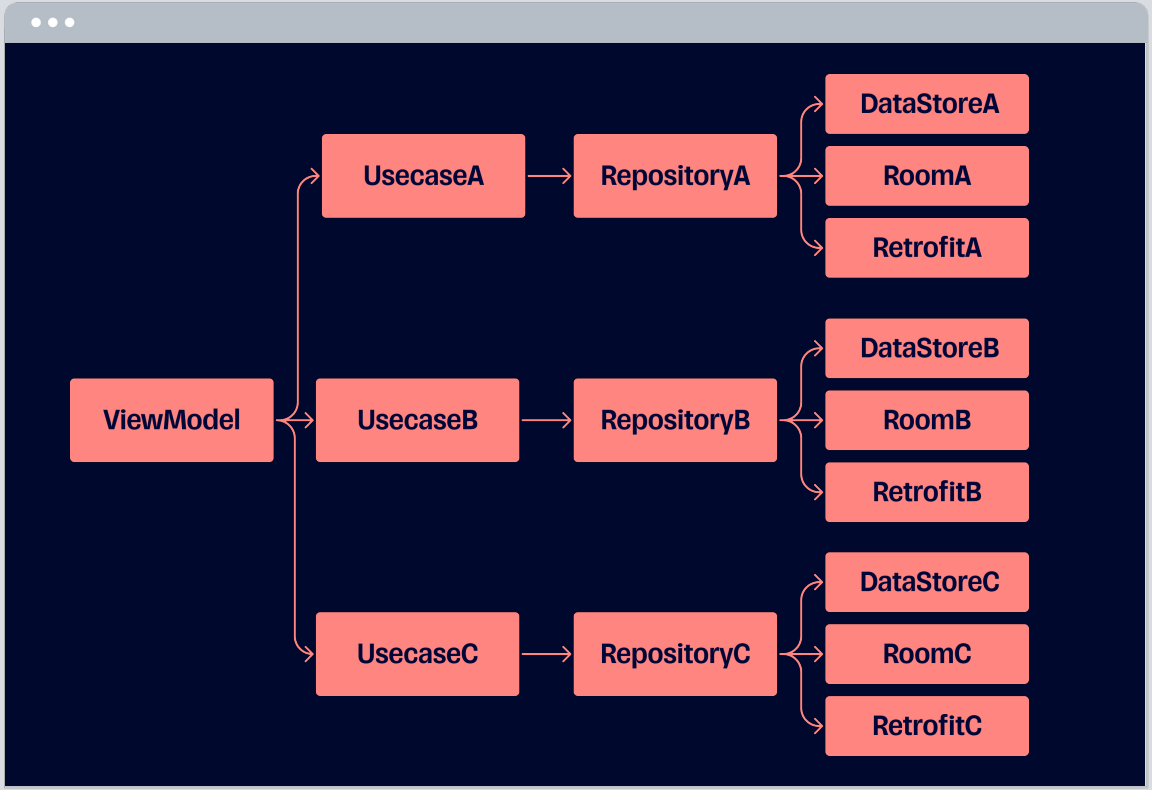
Release версия

20 прогонов

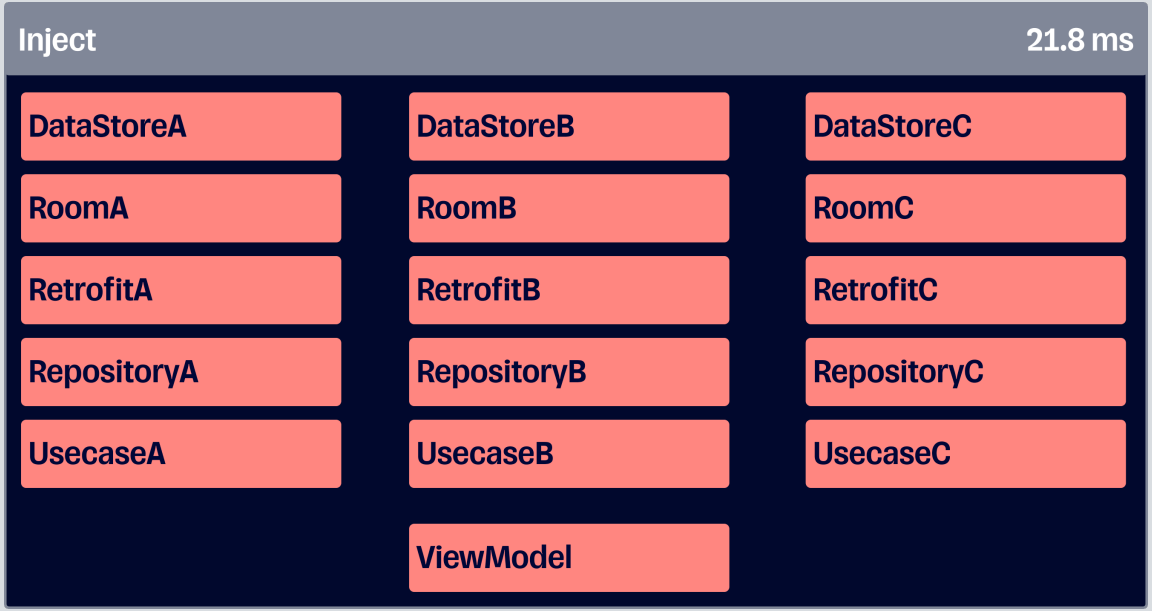
90 персентиль

Синхронная инициализация

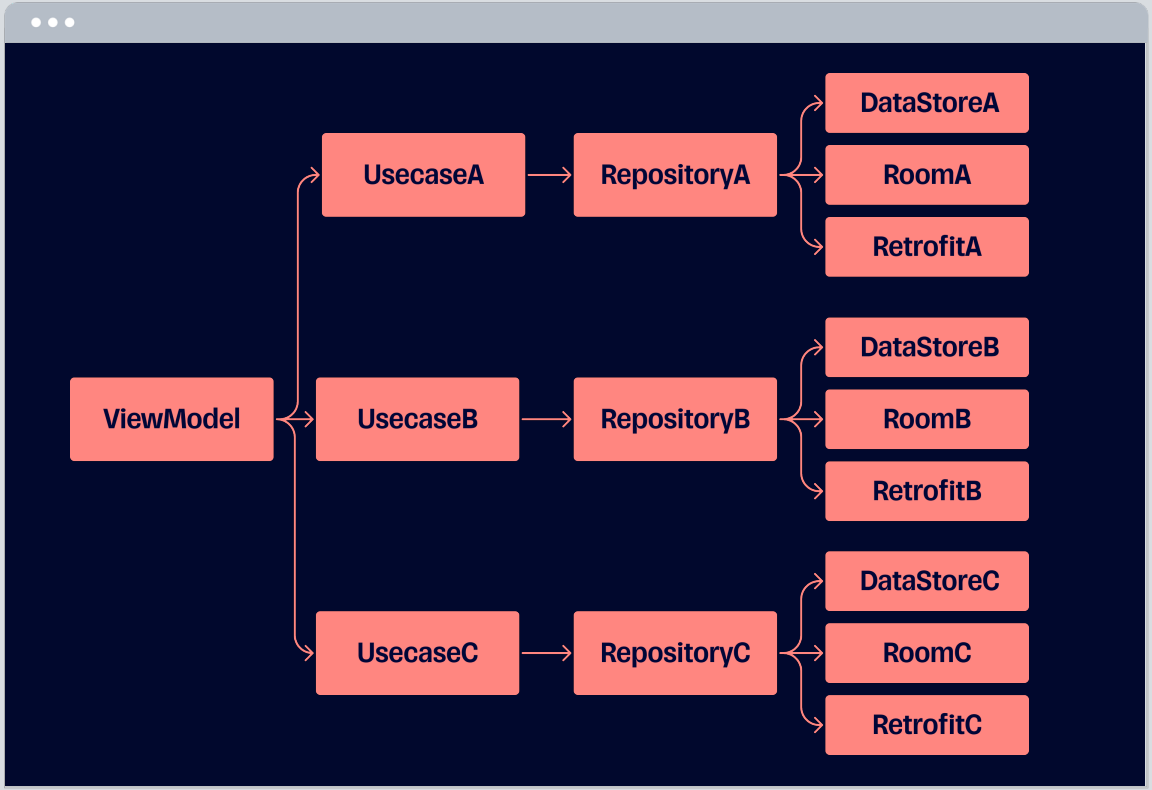
Старт экрана



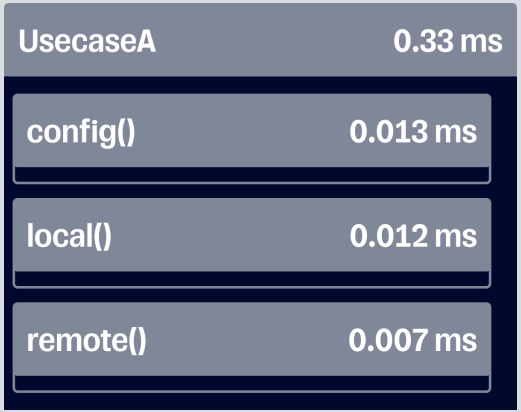
Старт экрана



Клик на кнопку

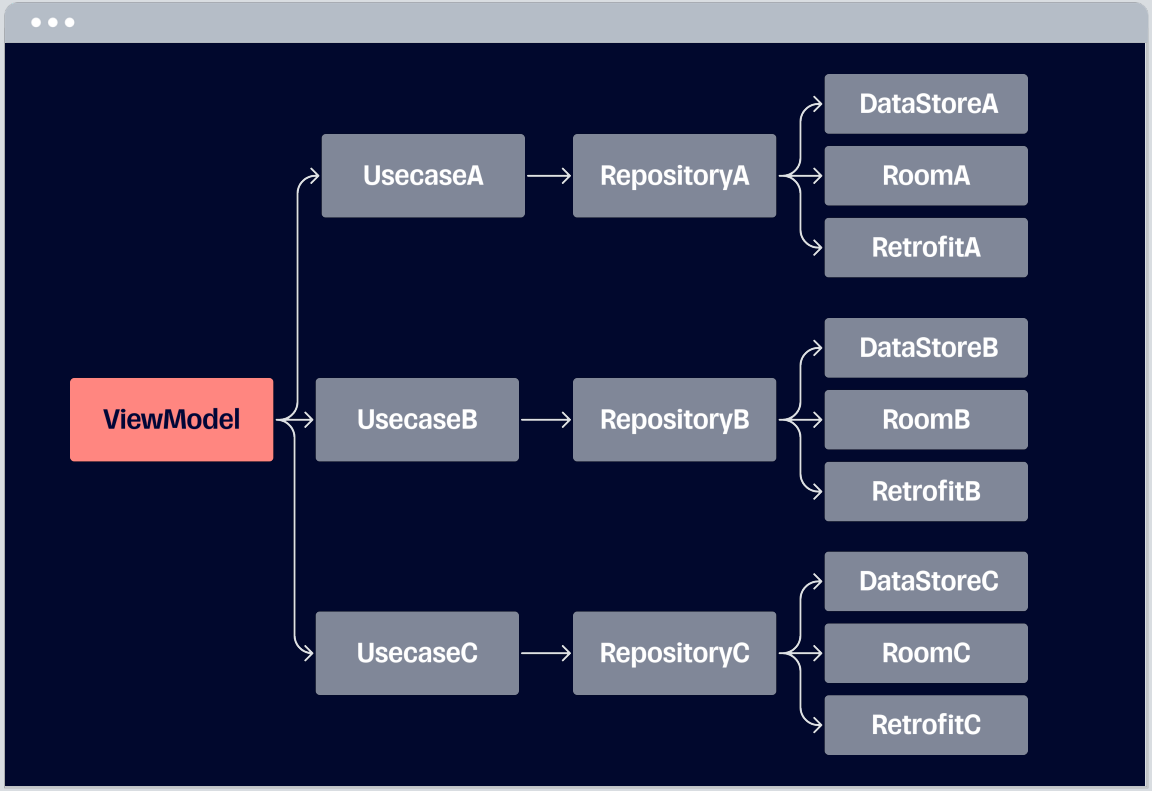


Клик на кнопку

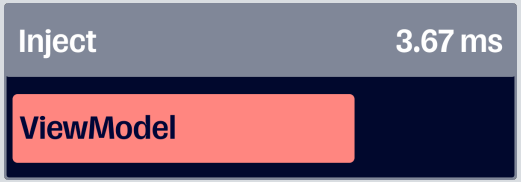


Ленивая ViewModel

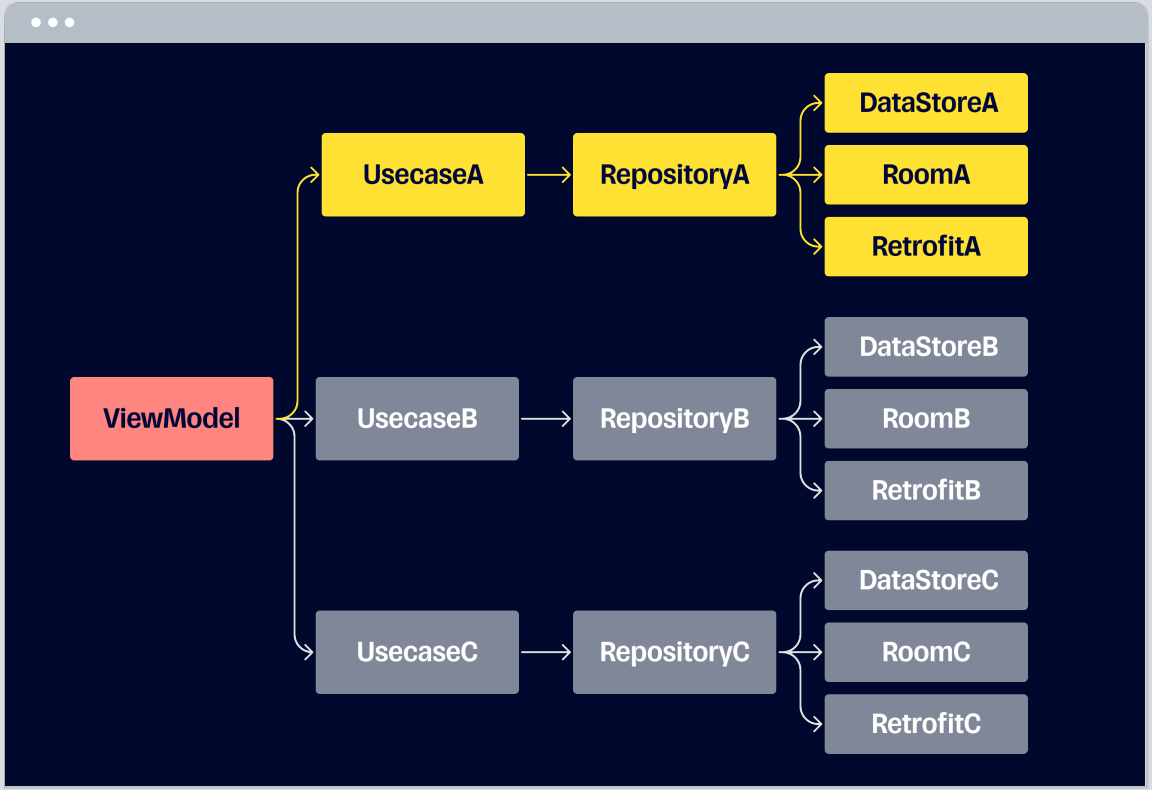
Старт экрана



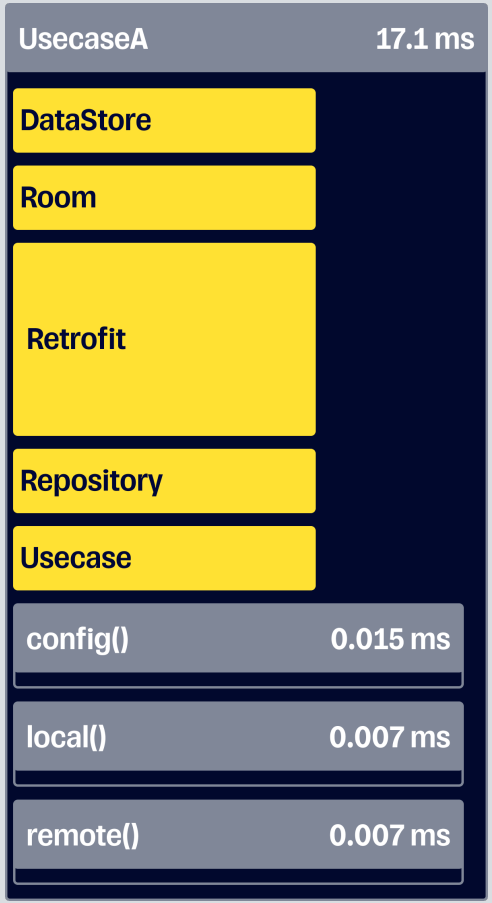
Старт экрана



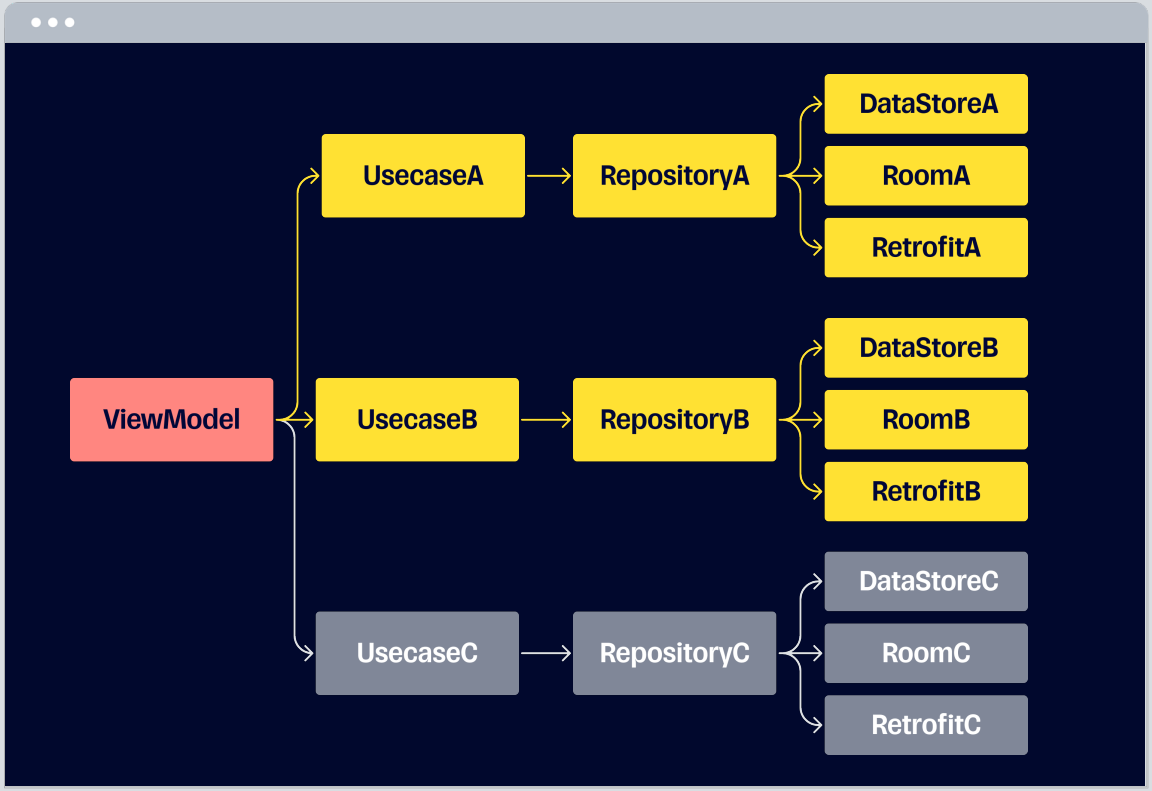
Клик на кнопку A



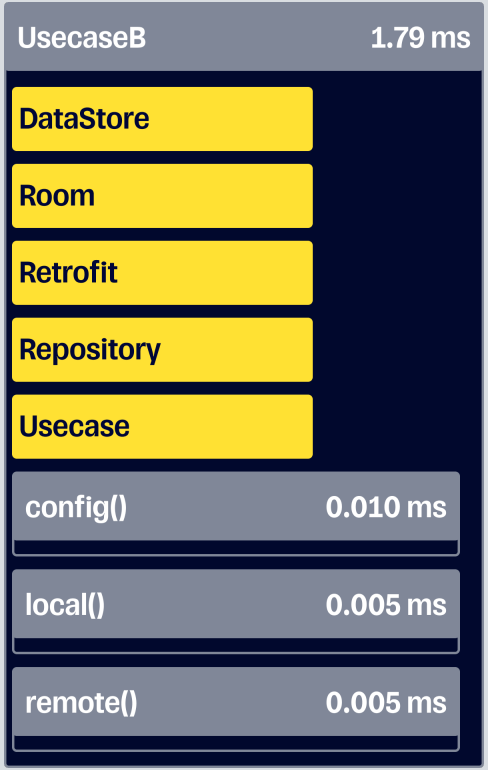
Клик на кнопку A



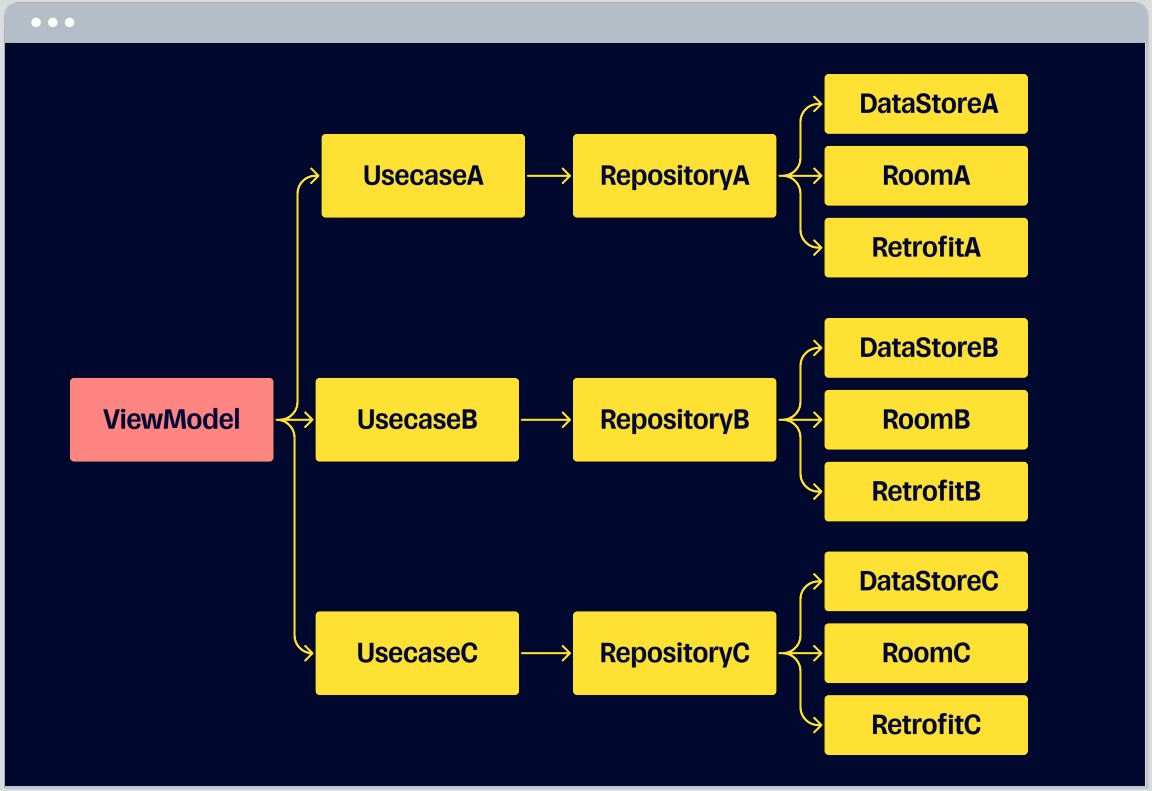
Клик на кнопку B



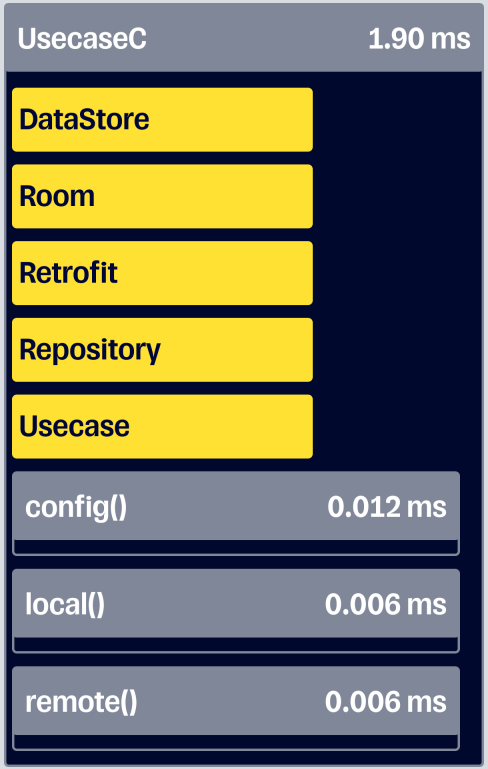
Клик на кнопку B



Клик на кнопку C

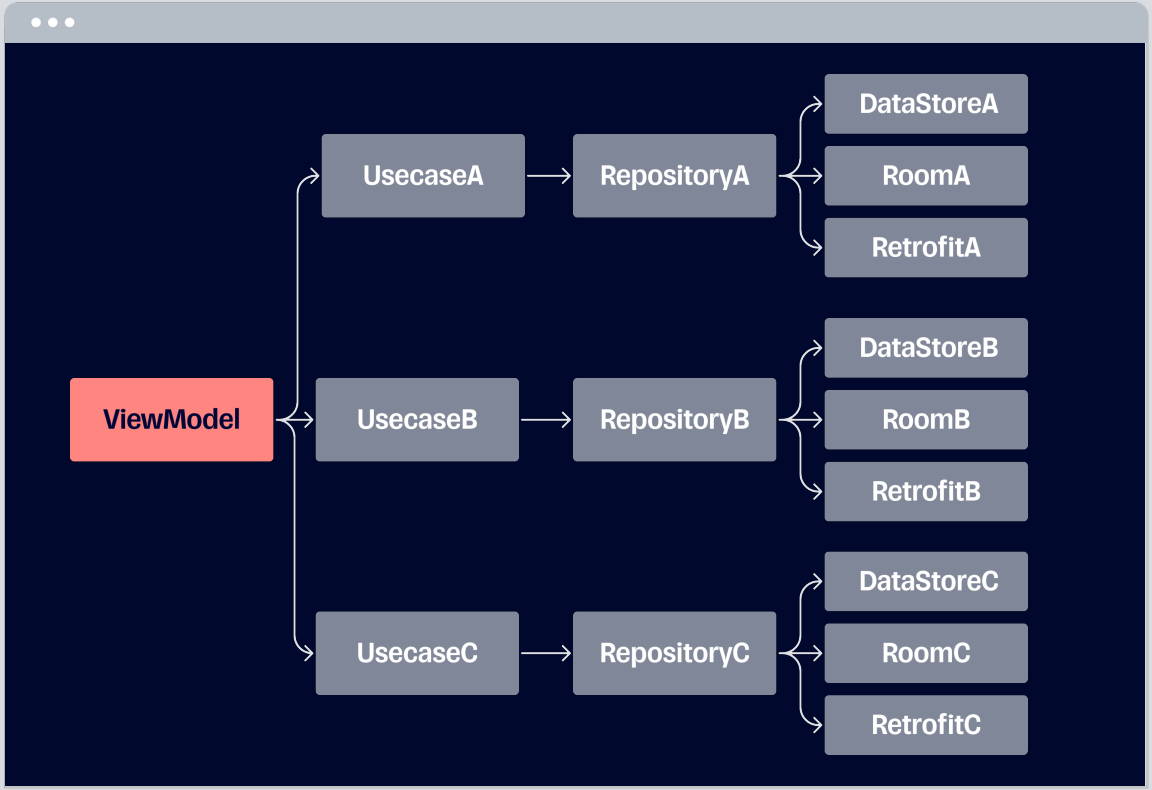


Клик на кнопку C



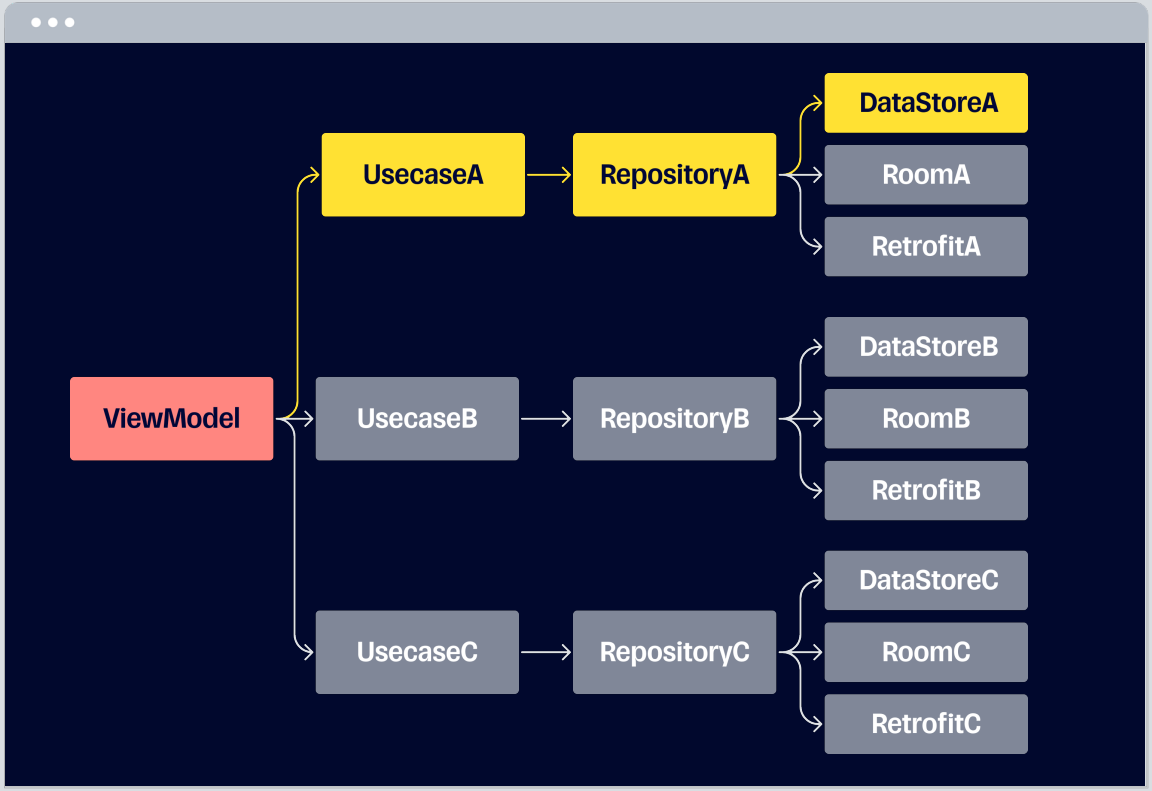
Ленивый граф зависимостей

Старт экрана

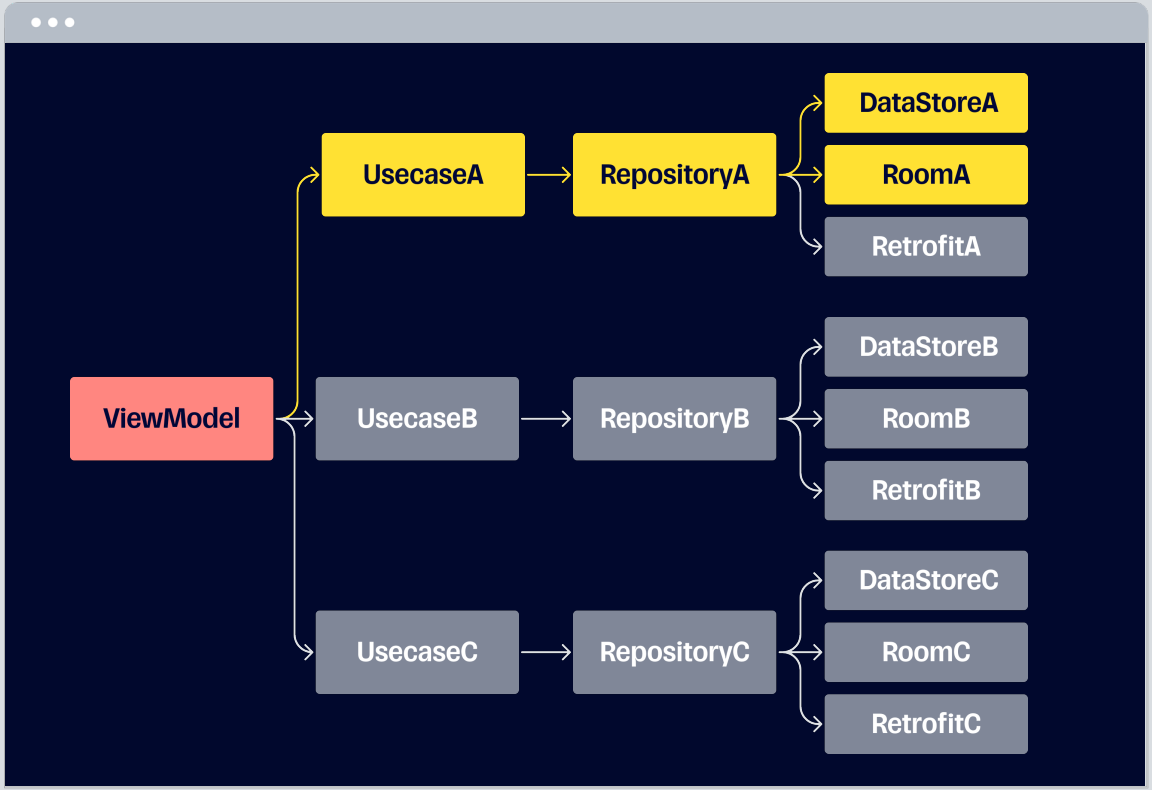


Старт экрана

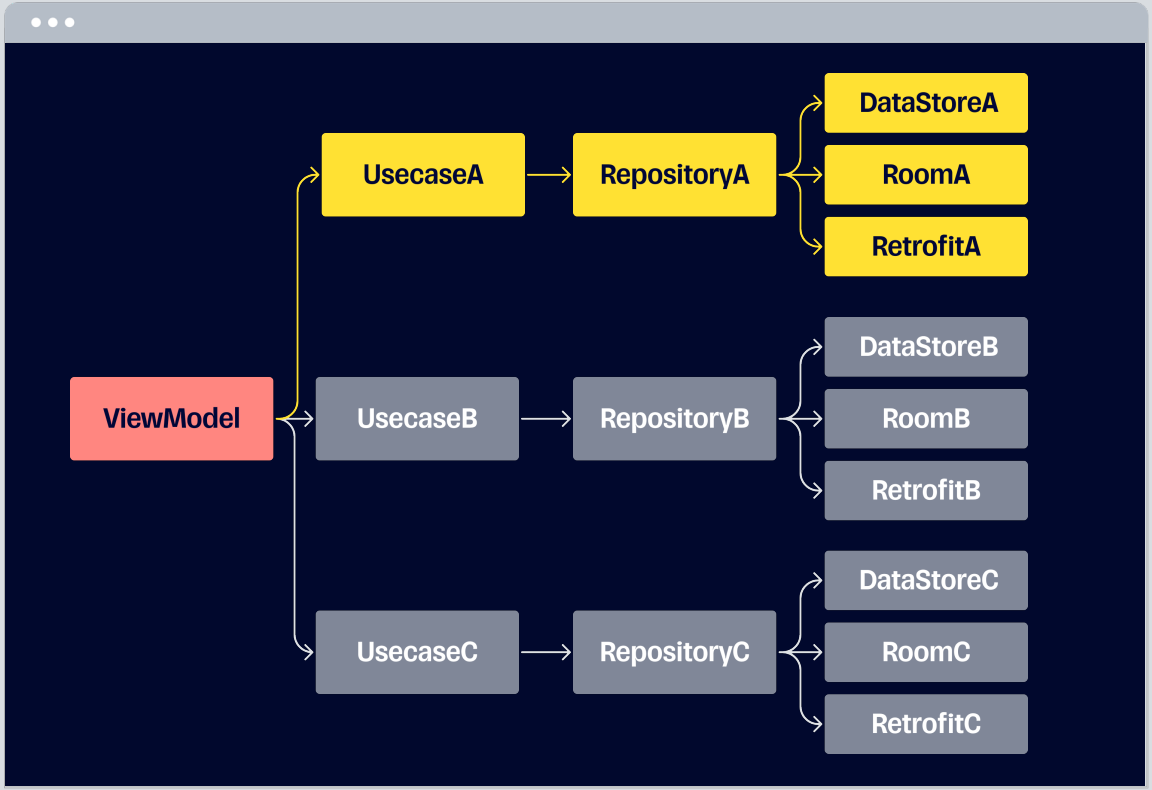
Клик на кнопку A



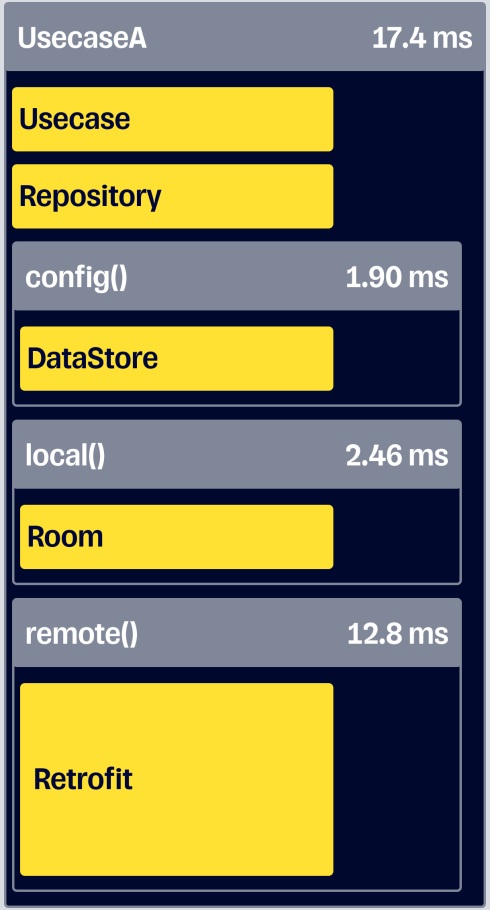
Клик на кнопку A



Клик на кнопку A

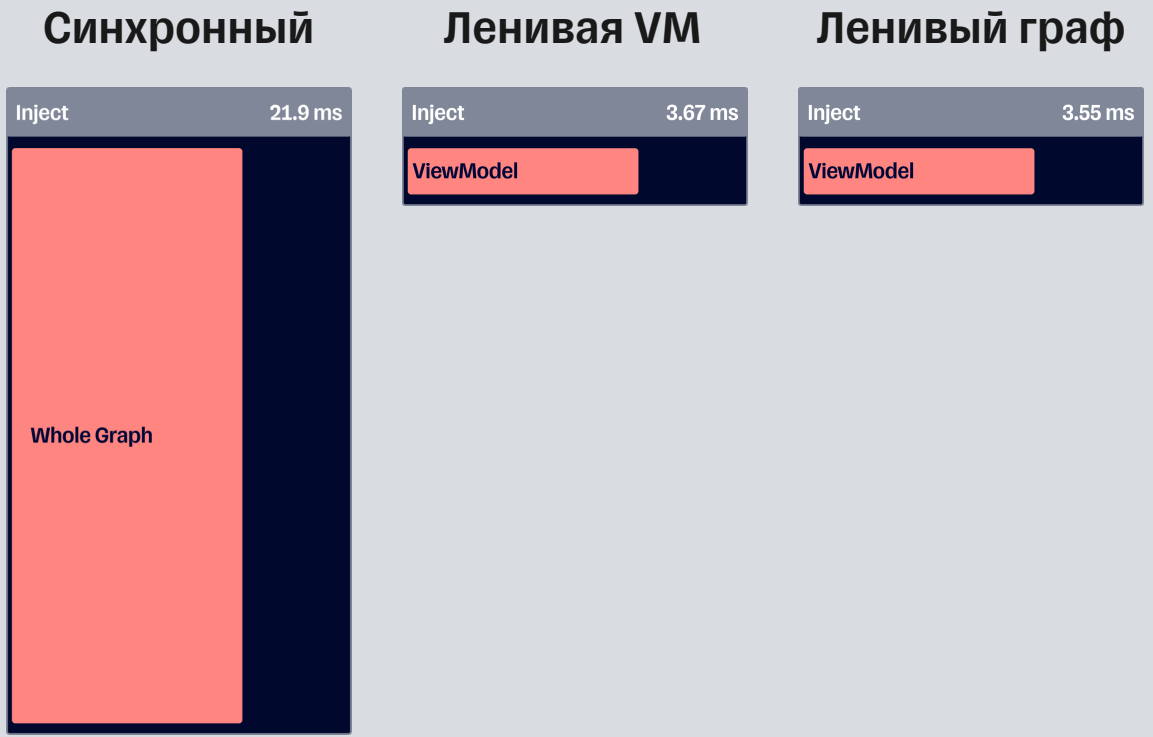


Клик на кнопку A

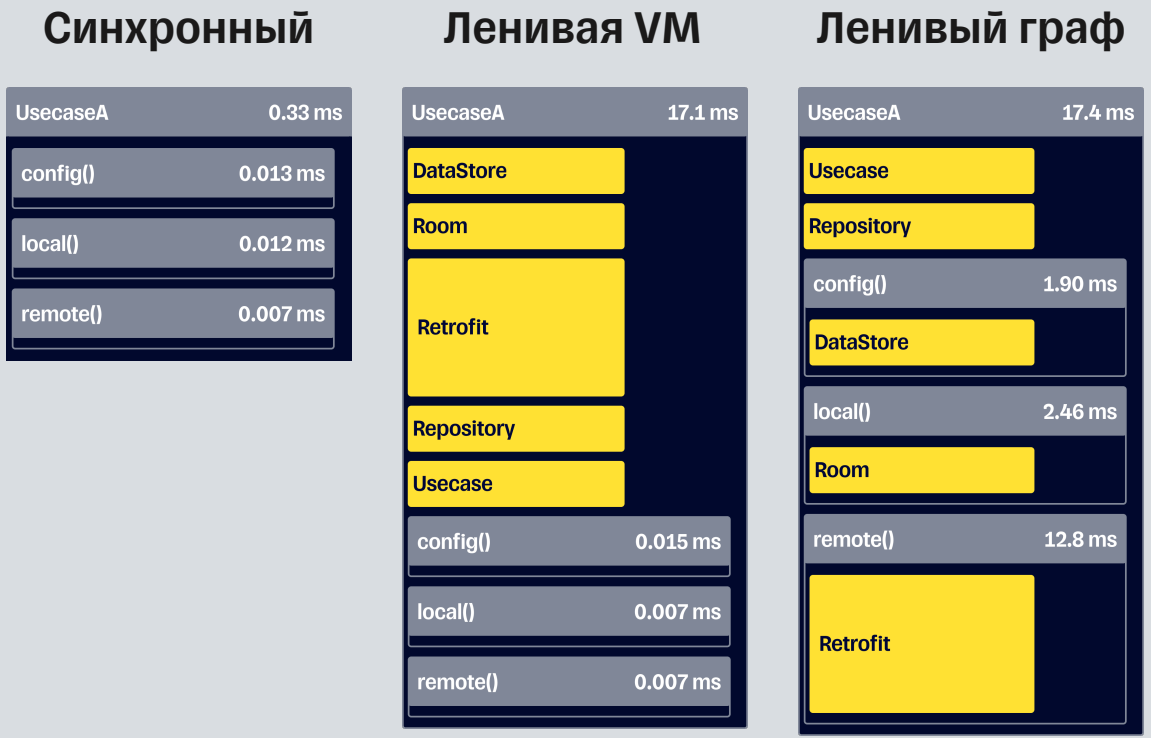


Сравнение

Старт экрана

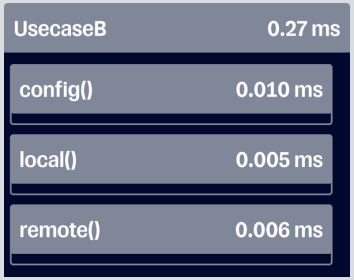


Клик на кнопку A

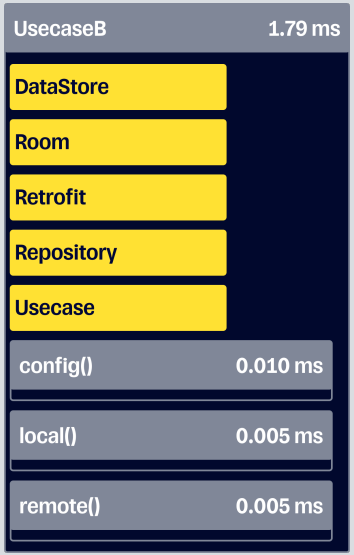


Клик на кнопку B

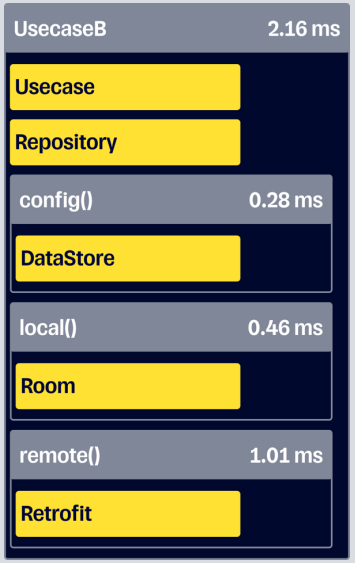
Синхронный

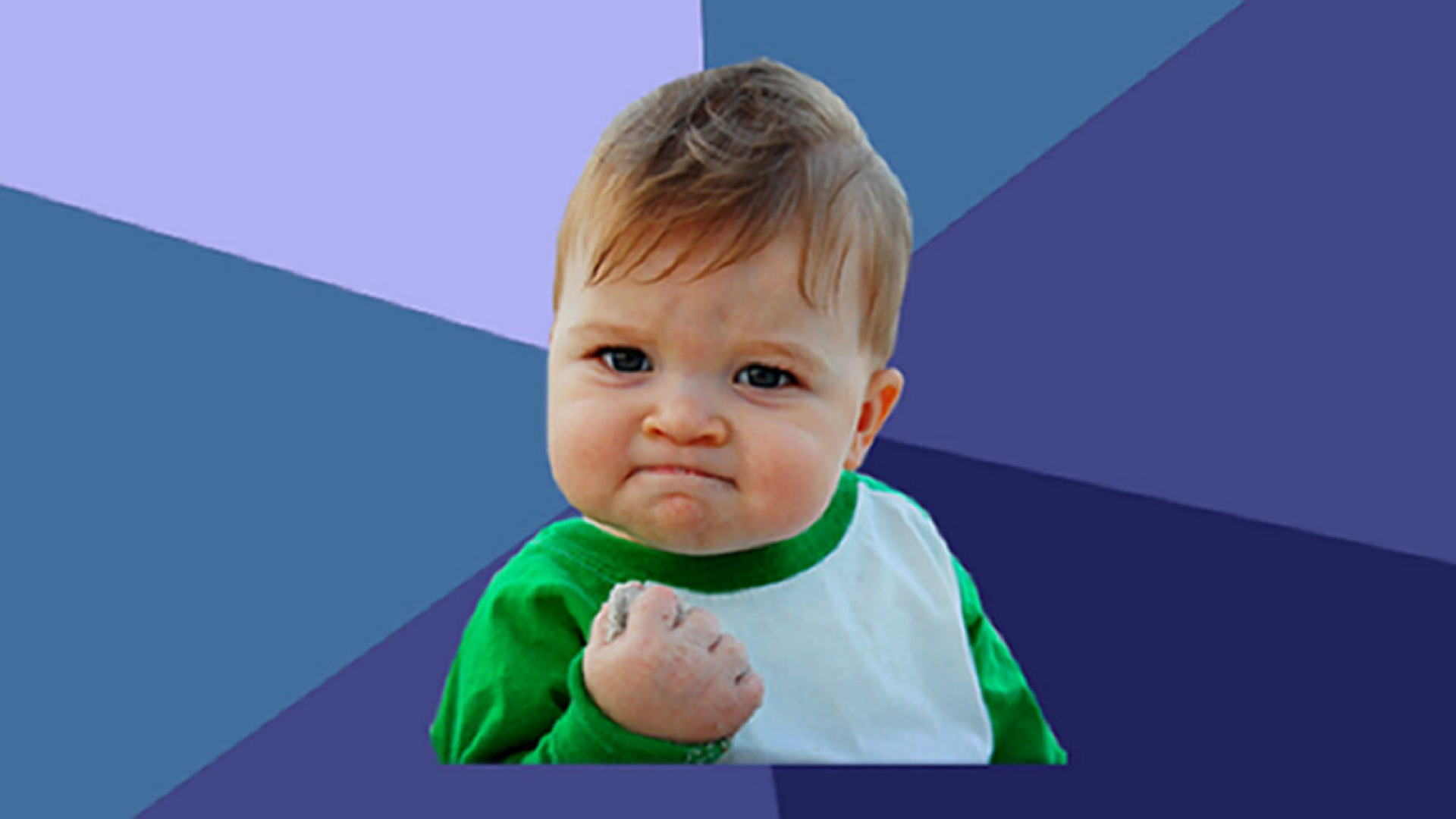


Ленивая VM



Ленивый граф

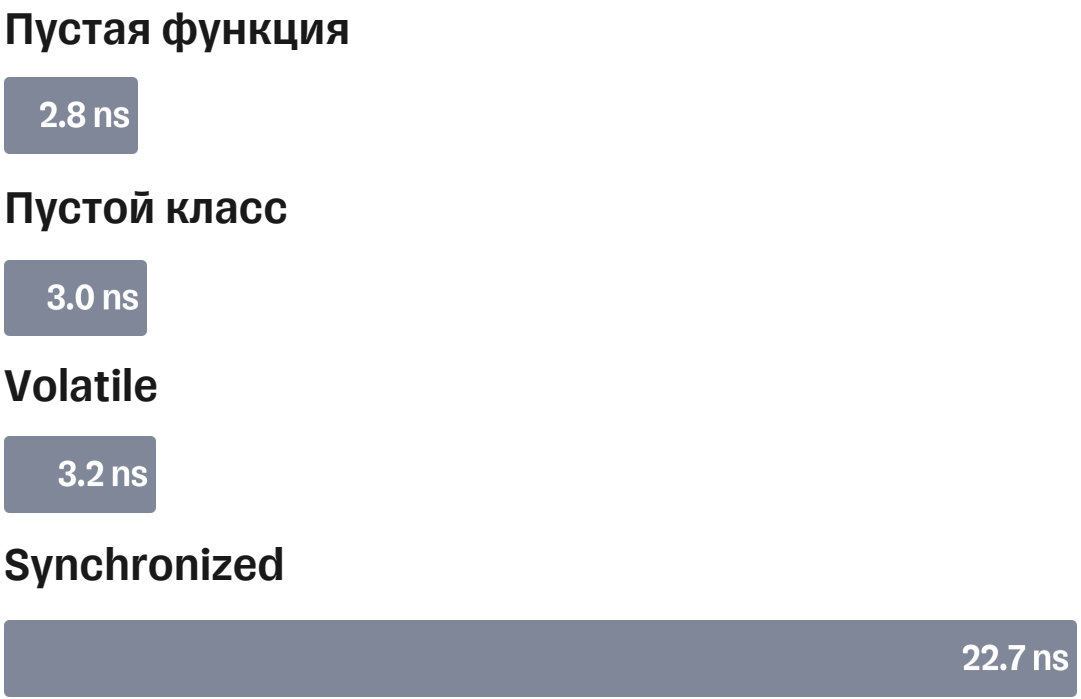




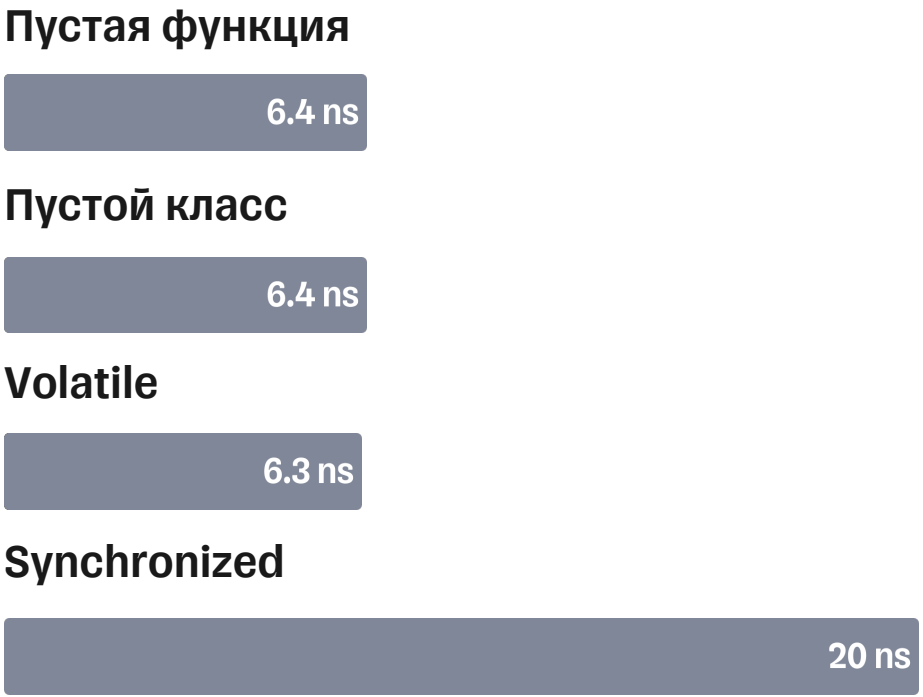
Inject vs Data sources

Что занимает время?

Время выполнения Realme 8



Время выполнения Samsung A7 (2018)



Время выполнения Realme 8

SystemClock.elapsedRealtime()

107 ns

Большой класс – контейнер примитивов

Print

Время выполнения Samsung A7 (2018)

SystemClock.elapsedRealtime()

484 ns

Большой класс – контейнер примитивов

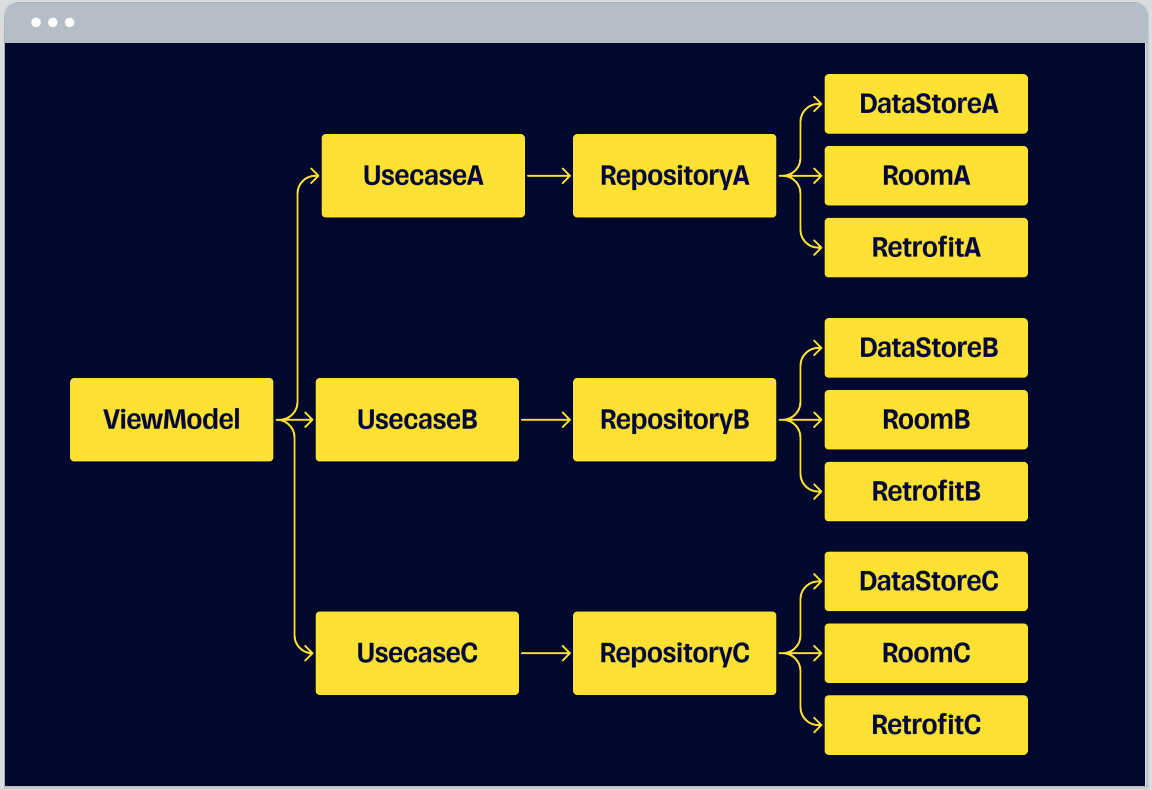
Print

Чем больше объектов и вызовов, тем больше времени

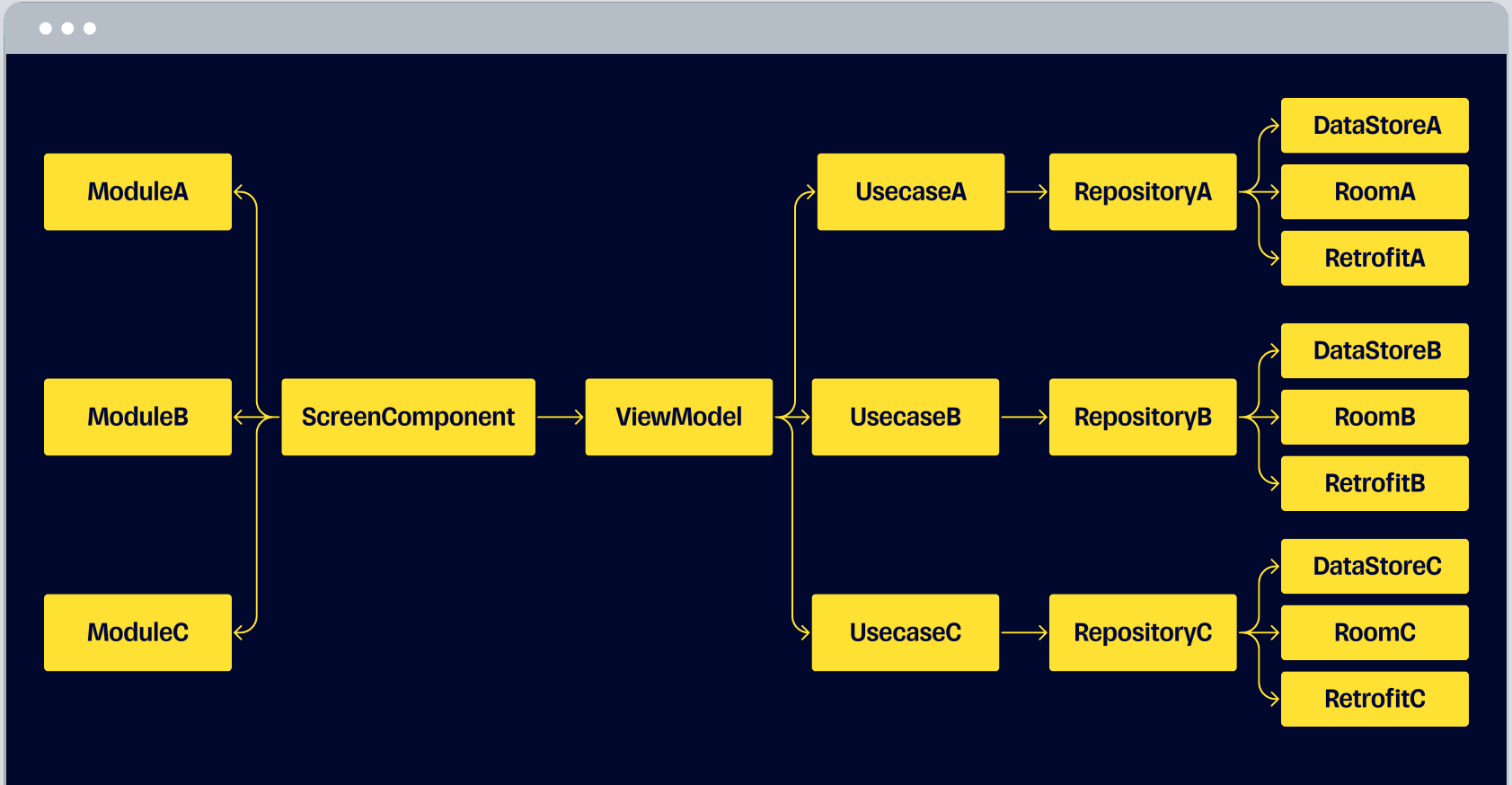
Чем больше объектов и вызовов, тем больше времени

Что скрывается за inject?

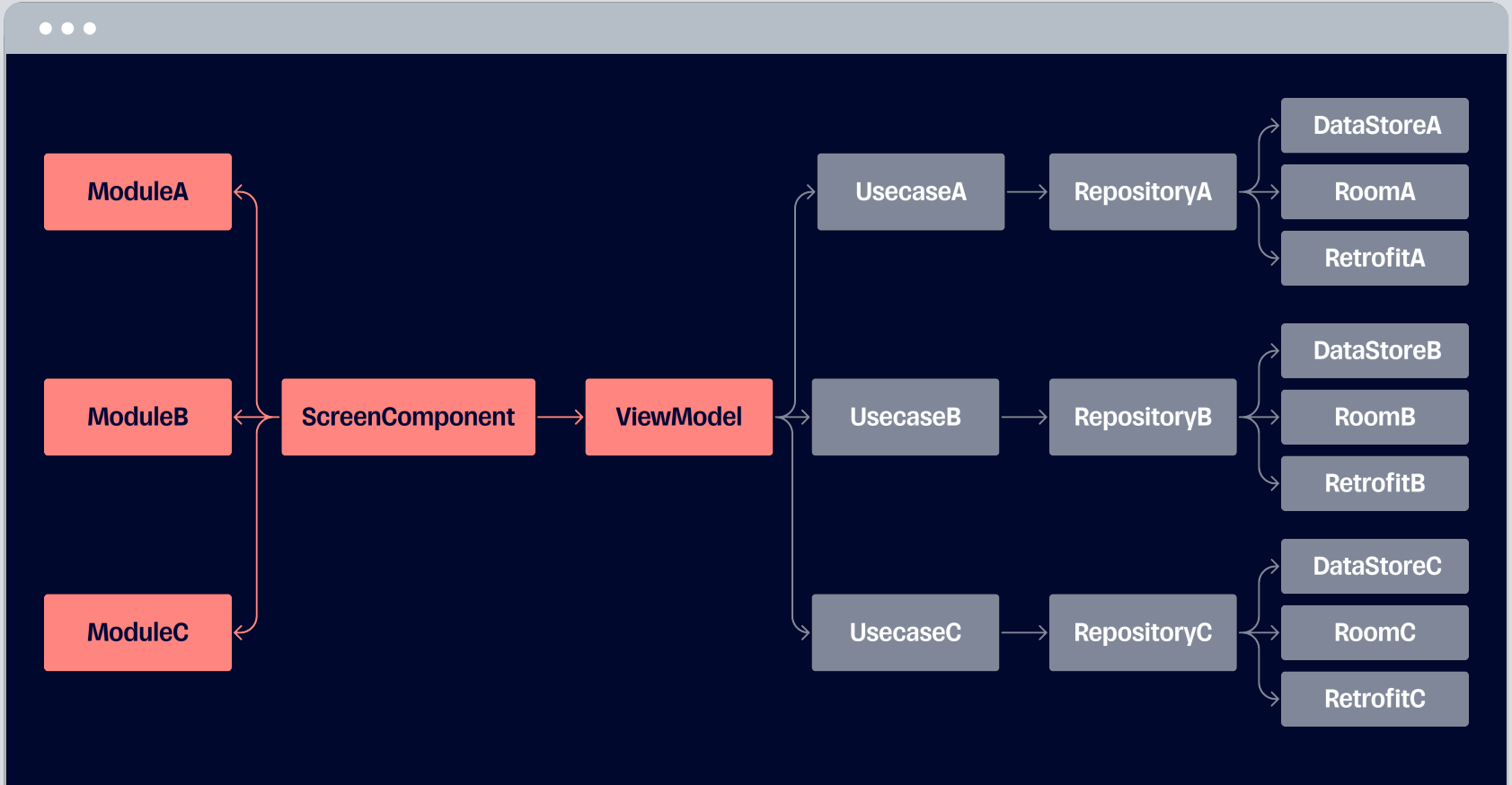
Граф зависимостей



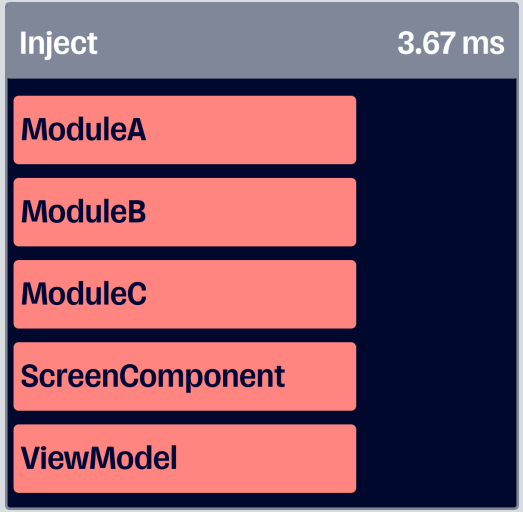
Полный граф


ModuleA
ModuleB
ModuleC
ScreenComponent
ViewModel
UsecaseA
UsecaseB
UsecaseC
RepositoryA
RepositoryB
RepositoryC
DataStoreA
RoomA
RetrofitA
DataStoreB
RoomB
RetrofitB
DataStoreC
RoomC
RetrofitC

Старт экрана

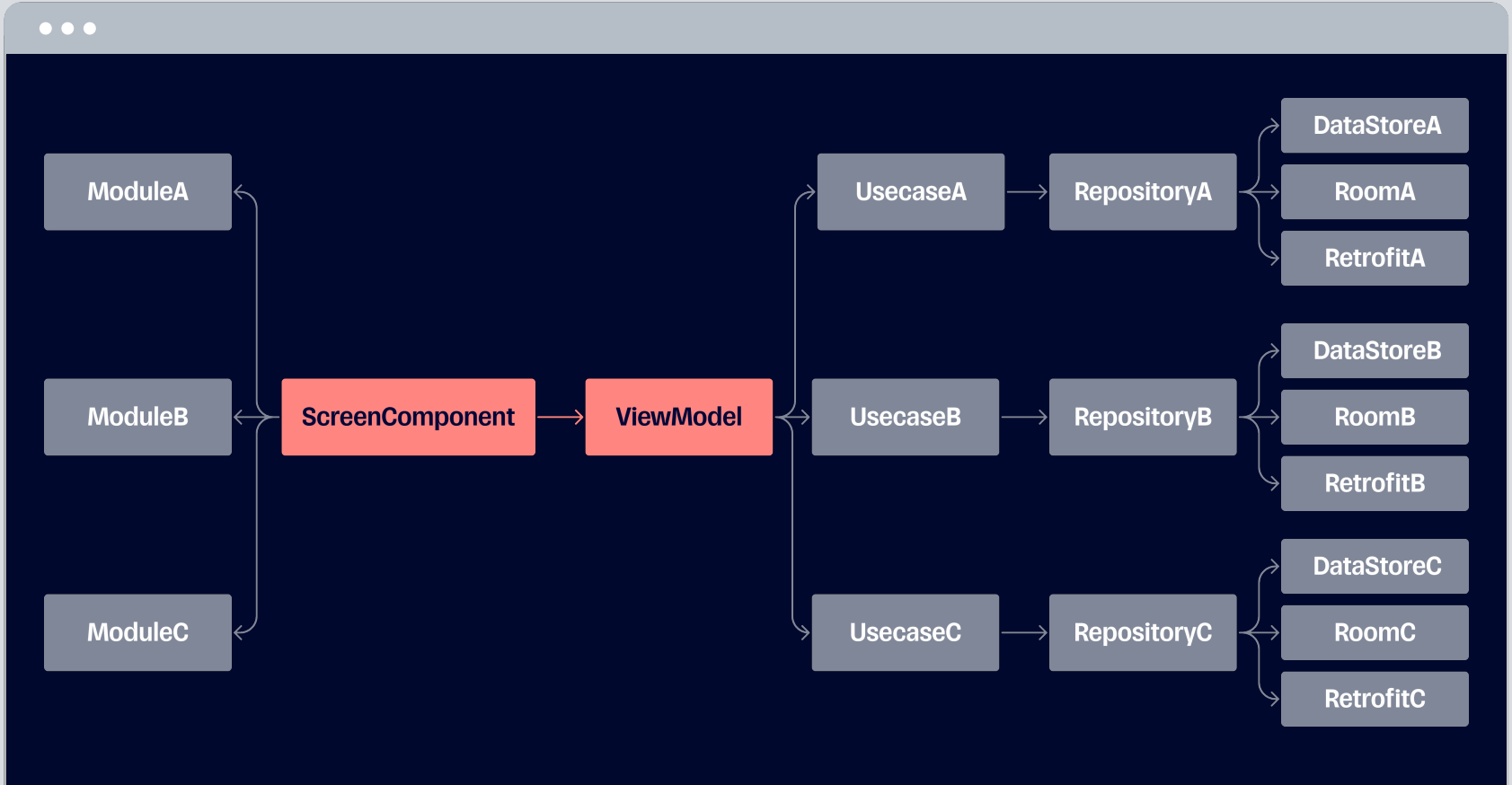


Старт экрана

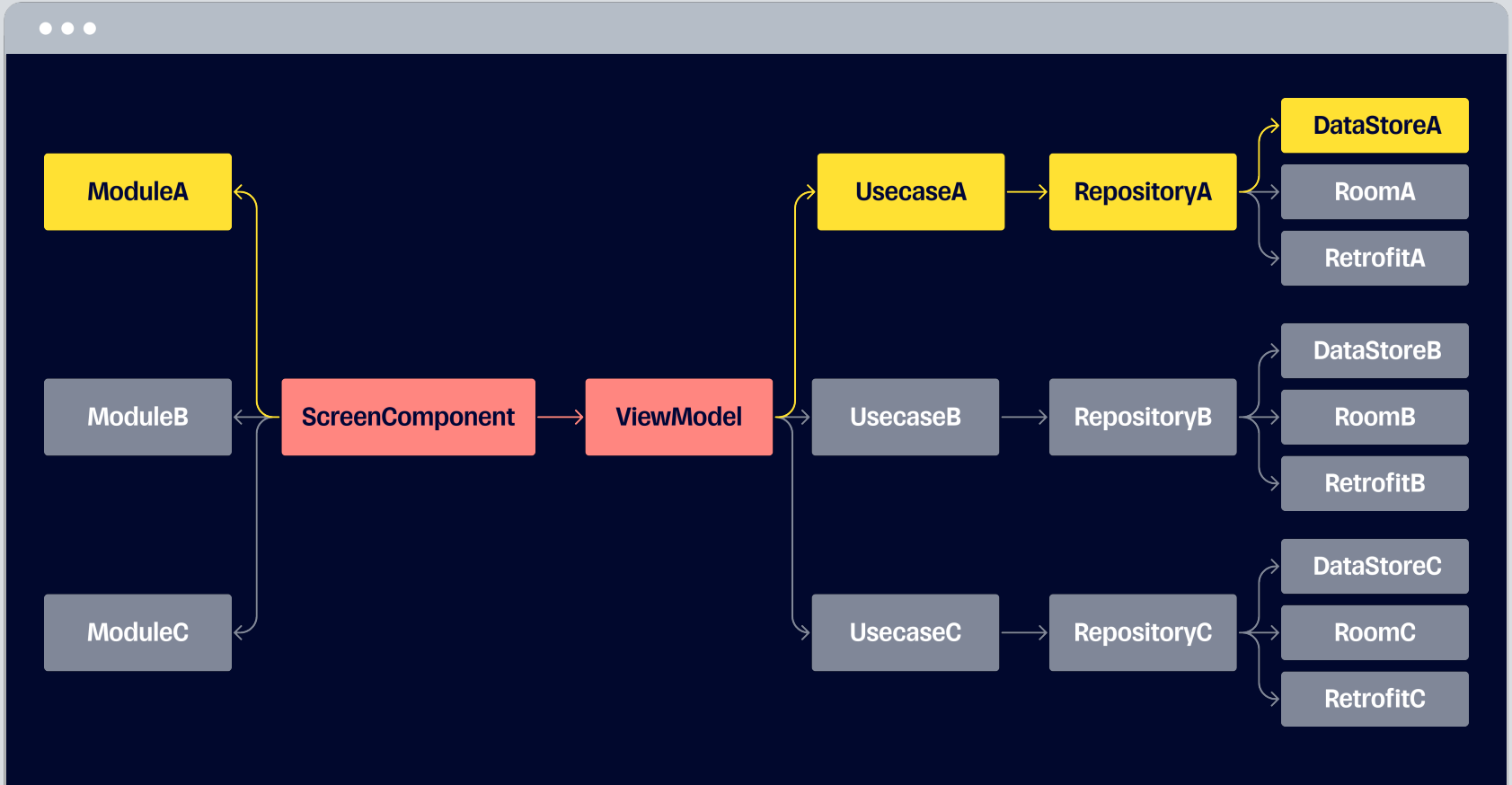


Какая цель?

Цель


ModuleA
ModuleB
ModuleC
ScreenComponent
ViewModel
UsecaseA
UsecaseB
UsecaseC
RepositoryA
RepositoryB
RepositoryC
DataStoreA
RoomA
RetrofitA
DataStoreB
RoomB
RetrofitB
DataStoreC
RoomC
RetrofitC

Цель


ModuleA
ModuleB
ModuleC
ScreenComponent
ViewModel
UsecaseA
UsecaseB
UsecaseC
RepositoryA
RepositoryB
RepositoryC
DataStoreA
RoomA
RetrofitA
DataStoreB
RoomB
RetrofitB
DataStoreC
RoomC
RetrofitC

Как реализовать?

Subcomponents

Subcomponents

Легкое расширения графа

Subcomponents

- Легкое расширения графа
- Возможна ленивость внутри компонента

Subcomponents

- Легкое расширения графа
- Возможна ленивость внутри компонента
- Полная целостность графа

Subcomponents

- Требуются приседания для ленивости внутри компонента

Subcomponents

- ✗ Требуются приседания для ленивости внутри компонента
- ✗ Нужно объявлять subcomponents в корне графа

Передавать компонент в dependencies

Передавать компонент в dependencies

Легкое расширения графа

Передавать компонент в dependencies

- Легкое расширения графа
- Не нужно объявлять subcomponents

Передавать компонент в dependencies

✕ Теряется полный контроль целостности графа

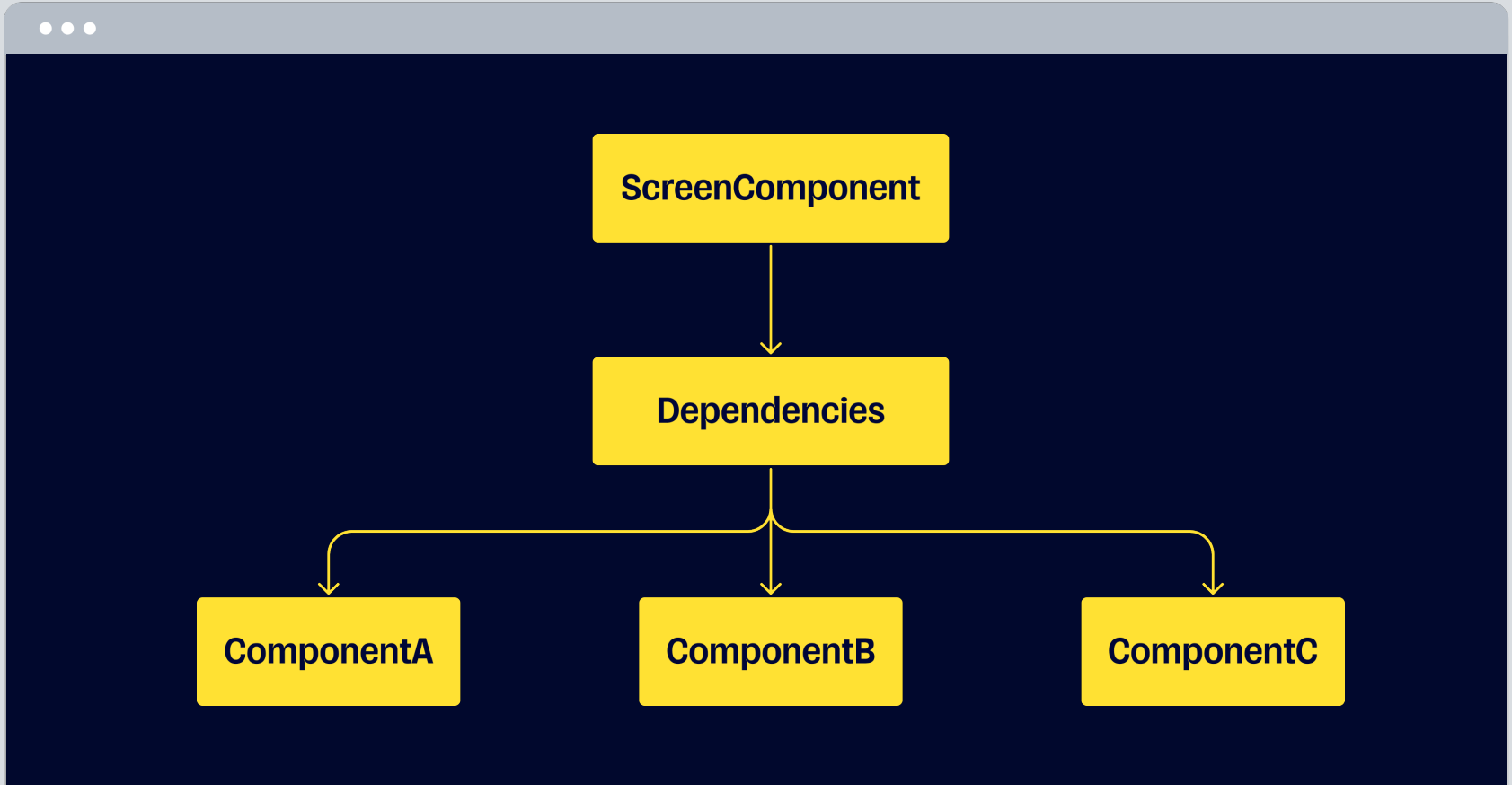
Передавать компонент в dependencies

- ✗ Теряется полный контроль целостности графа
- ✗ Нельзя добиться ленивости внутри компонента

Как мы сделали в Тинькофф?

Разделяем граф на части

Разделяем граф на части



Component Holder

```
interface BaseComponentHolder<Component : DIComponent> {

    fun get(): Component

    fun set(component: Component)

    fun clear()
}
```

Dependencies

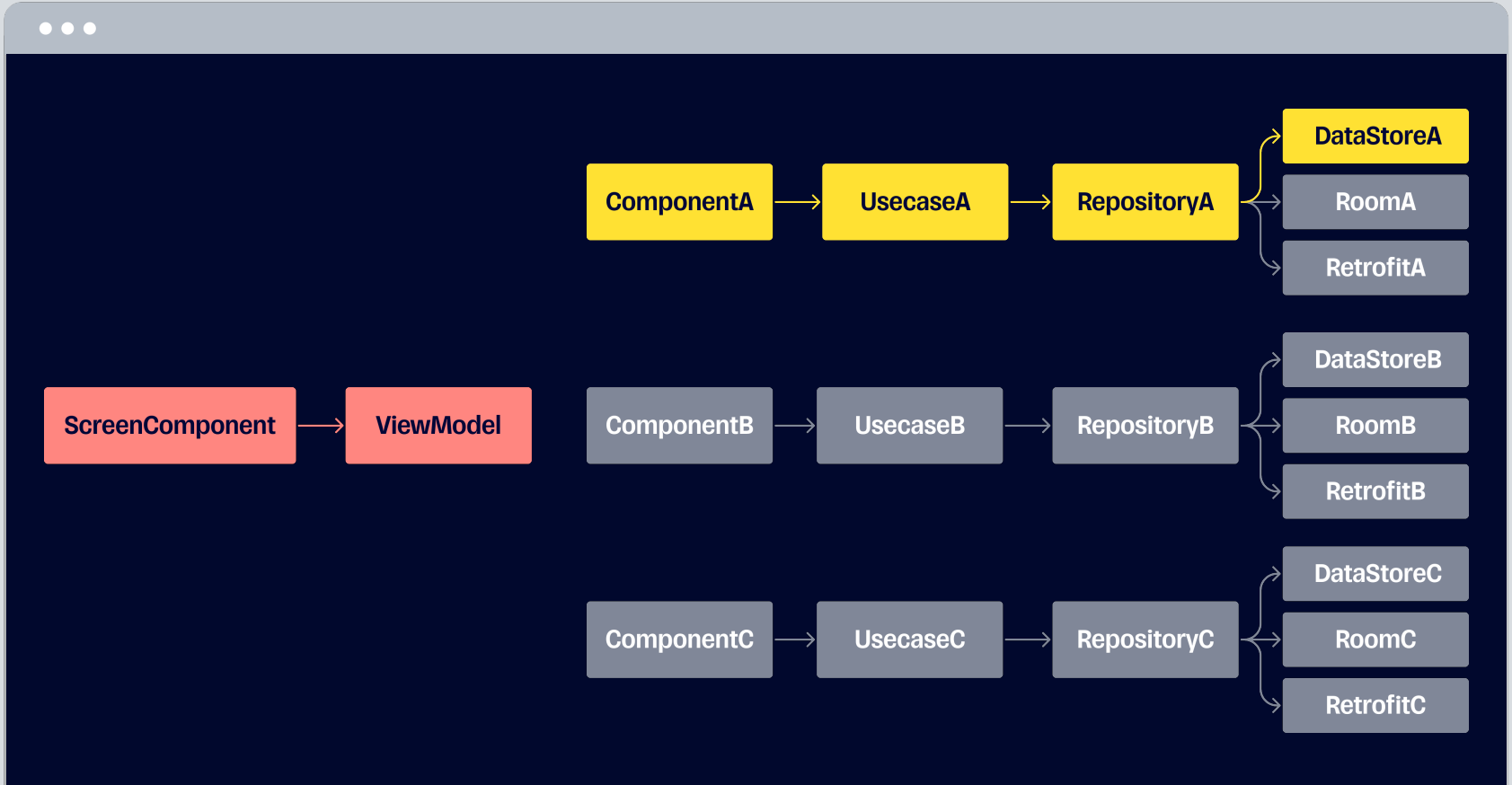
```
interface LazyComponentsDependencies {

    val usecaseA: UsecaseA
        get() = FeatureAComponentHolder.get().usecaseA
    val usecaseB: UsecaseB
        get() = FeatureBComponentHolder.get().usecaseB
    val usecaseC: UsecaseC
        get() = FeatureCComponentHolder.get().usecaseC

    class Impl : LazyComponentsDependencies
}
```

Старт экрана

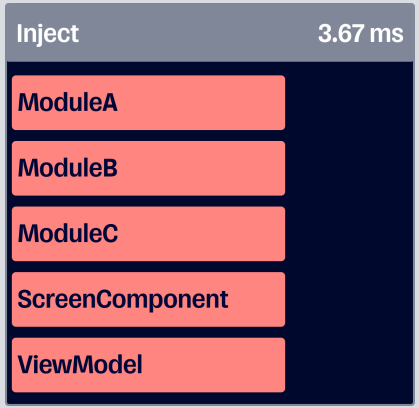
Клик на кнопку



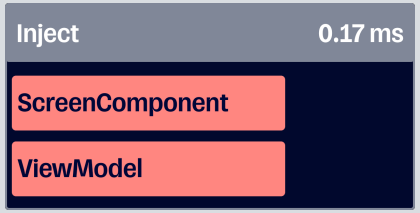
Сравнение

Старт

Ленивый граф

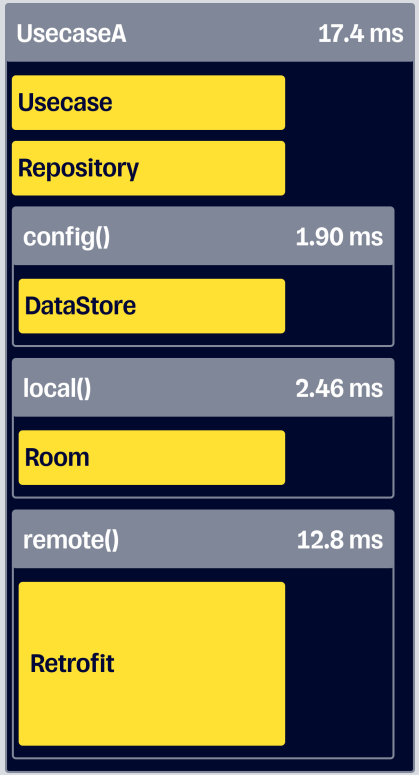


Ленивые зависимости

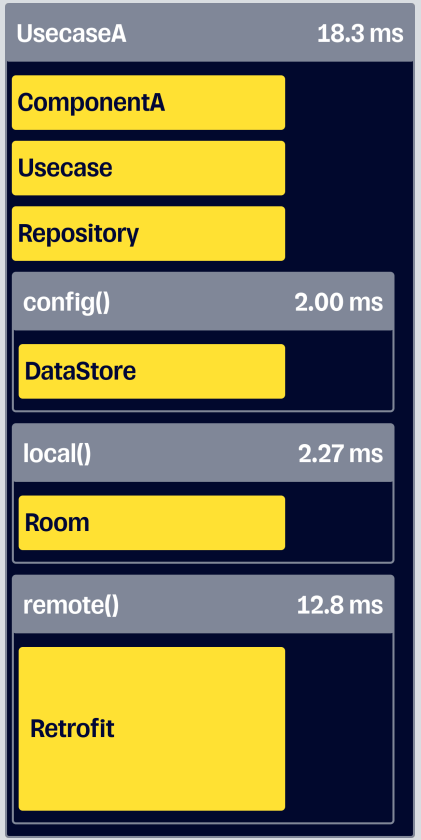


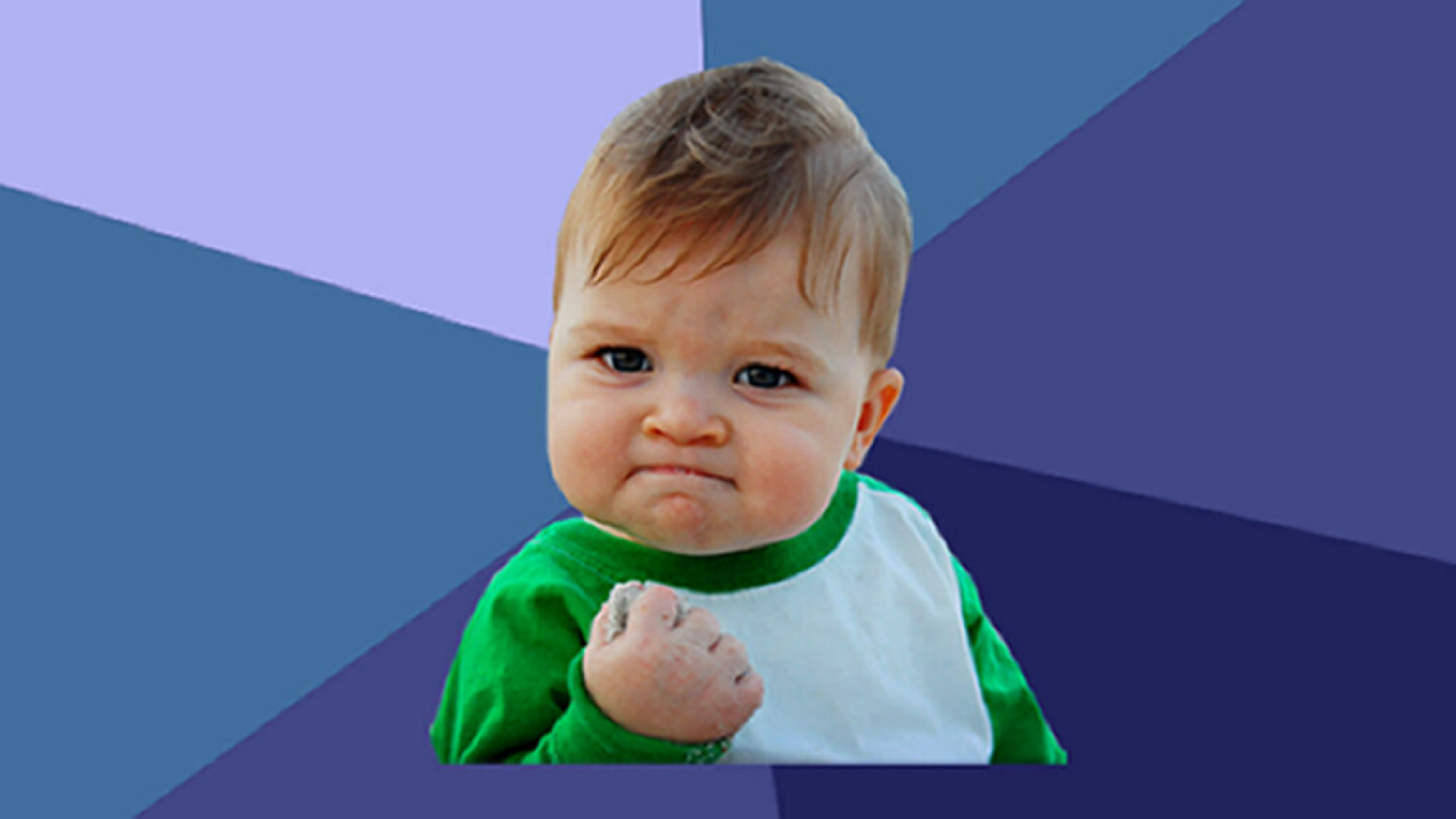
Клик на кнопку

Ленивый граф



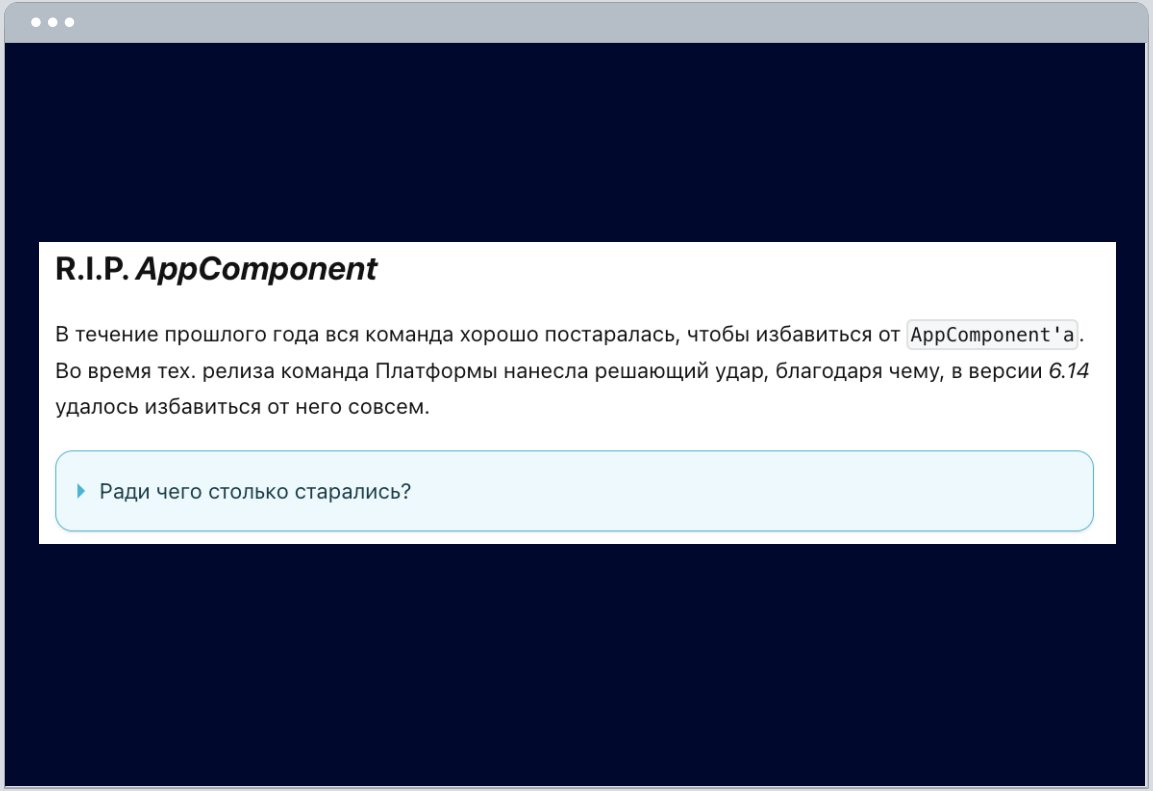
Ленивые зависимости





Где реальные результаты?

Отчет 23Q1


R.I.P. AppComponent
В течение прошлого года вся команда хорошо постаралась, чтобы избавиться от AppComponent'a. Во время тех. релиза команда Платформы нанесла решающий удар, благодаря чему, в версии 6.14 удалось избавиться от него совсем.
Ради чего столько старались?

Отчет 23Q1

Отчет 23Q1

Время сборки

Сократили с 2m 4.002s до 57.189s

Отчет 23Q1

Время сборки

Сократили с 2m 4.002s до 57.189s

Время инициализации

Сократили на 23.85%

Перевод компонентов на ленивый подход

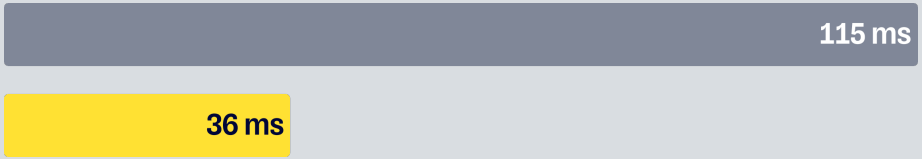
Перевод компонентов на ленивый подход

Компонент SSL (47 мс → 4мс)

Перевод компонентов на ленивый подход

Компонент SSL (47 мс → 4мс)

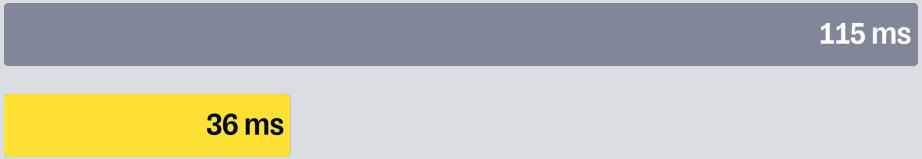
Компонент баз данных (115 мс → 36мс)



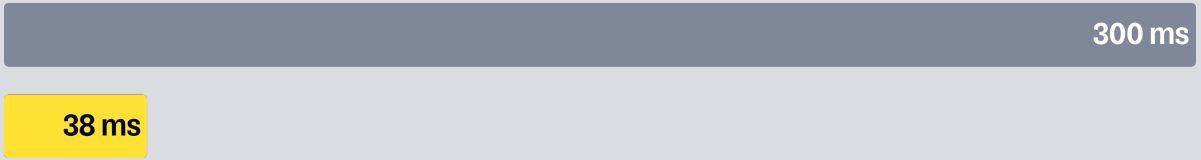
Перевод компонентов на ленивый подход

Компонент SSL (47 мс → 4мс)

Компонент баз данных (115 мс → 36мс)

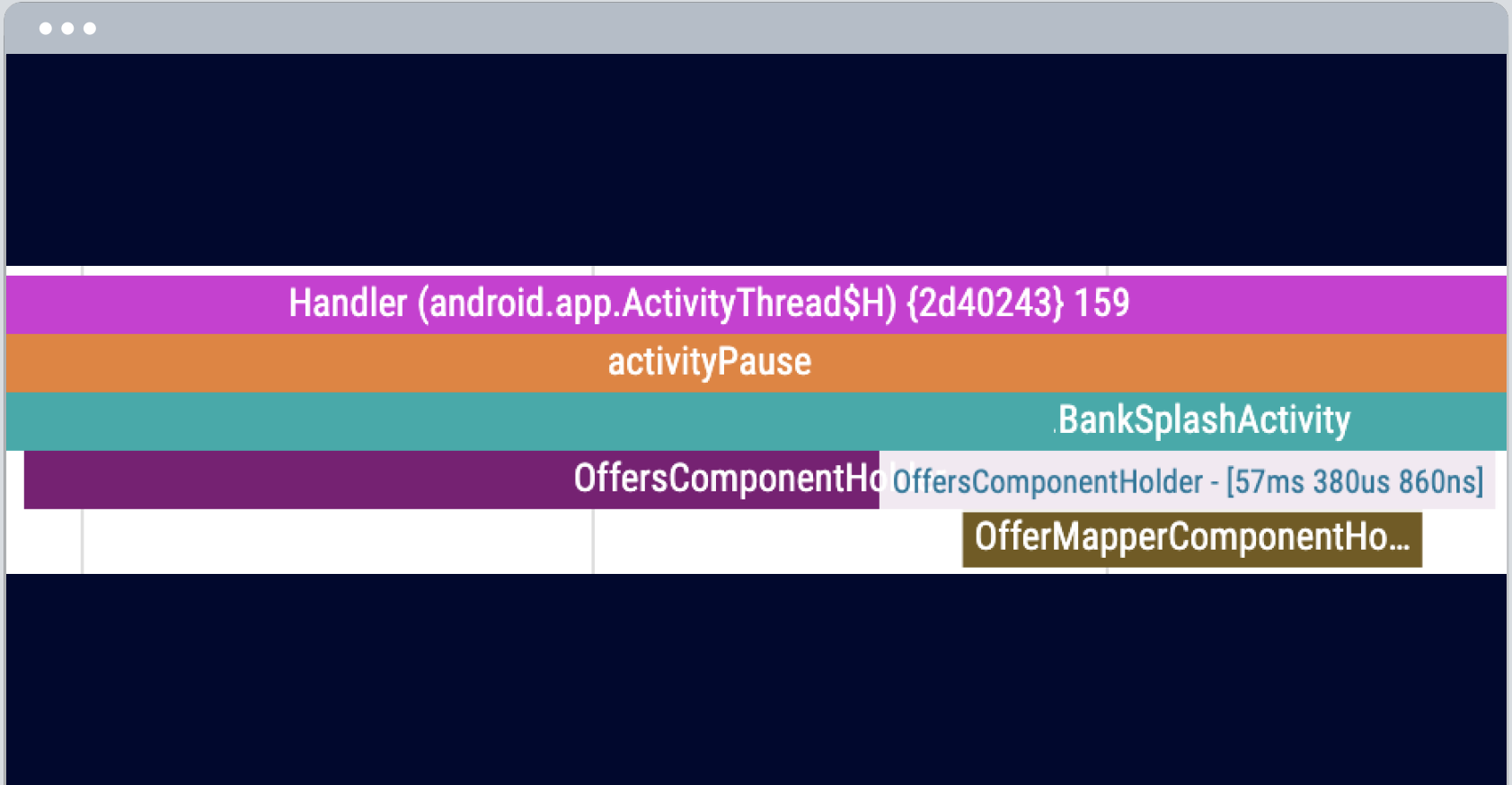


Компонент конфигурации (300 мс → 38мс)

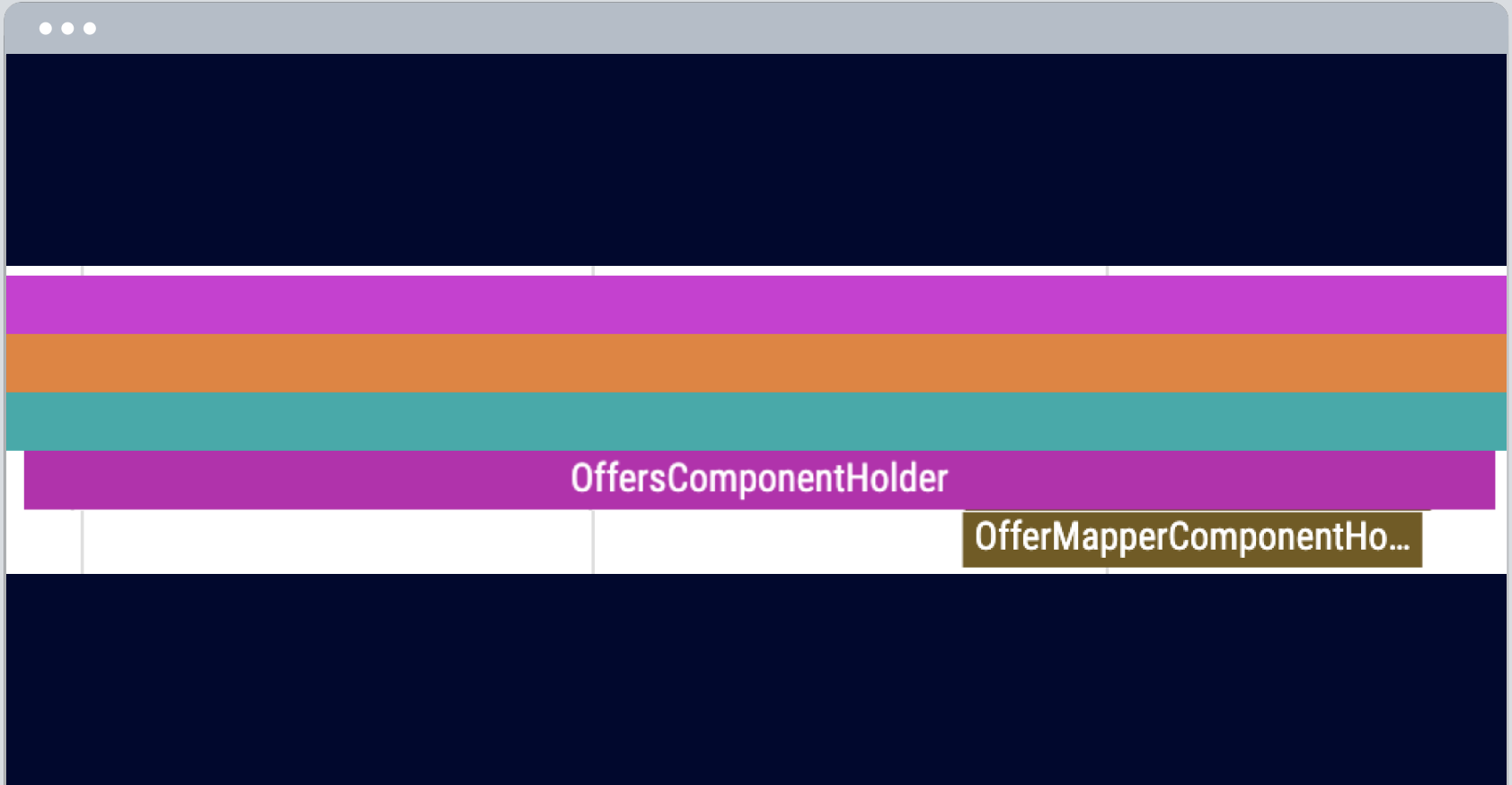


OffersComponent до рефакторинга

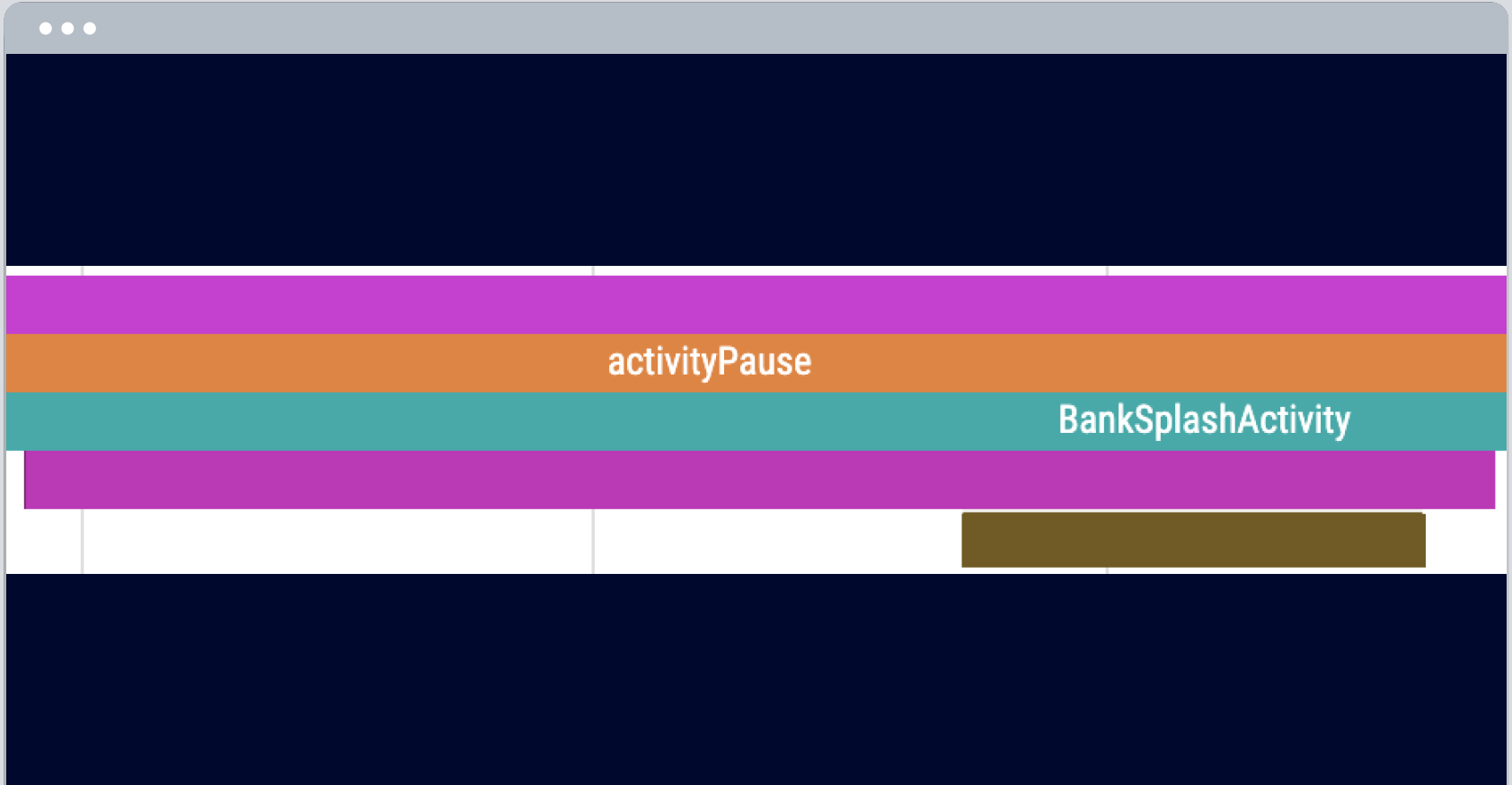
OffersComponent до рефакторинга



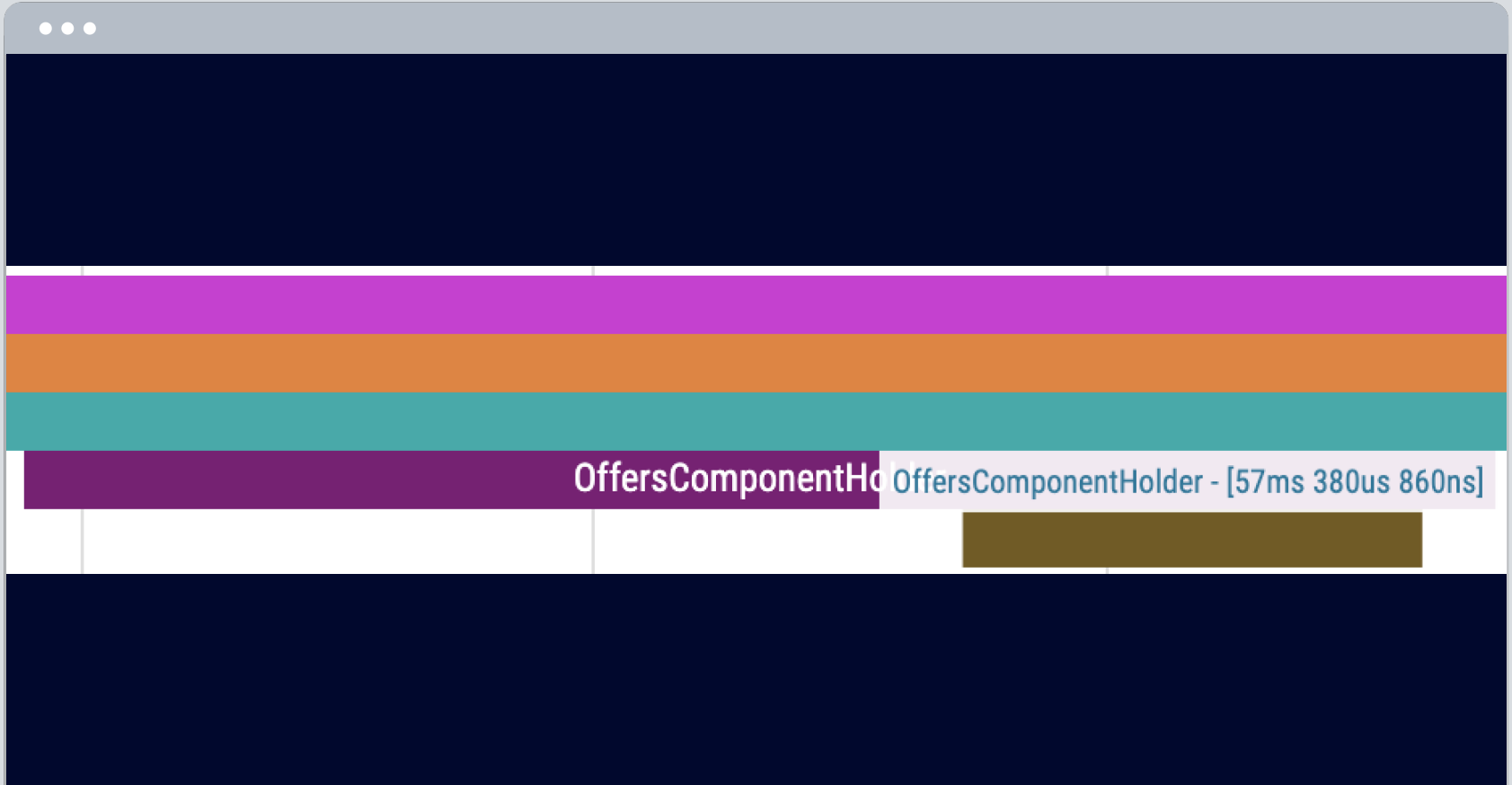
OffersComponent до рефакторинга



OffersComponent до рефакторинга

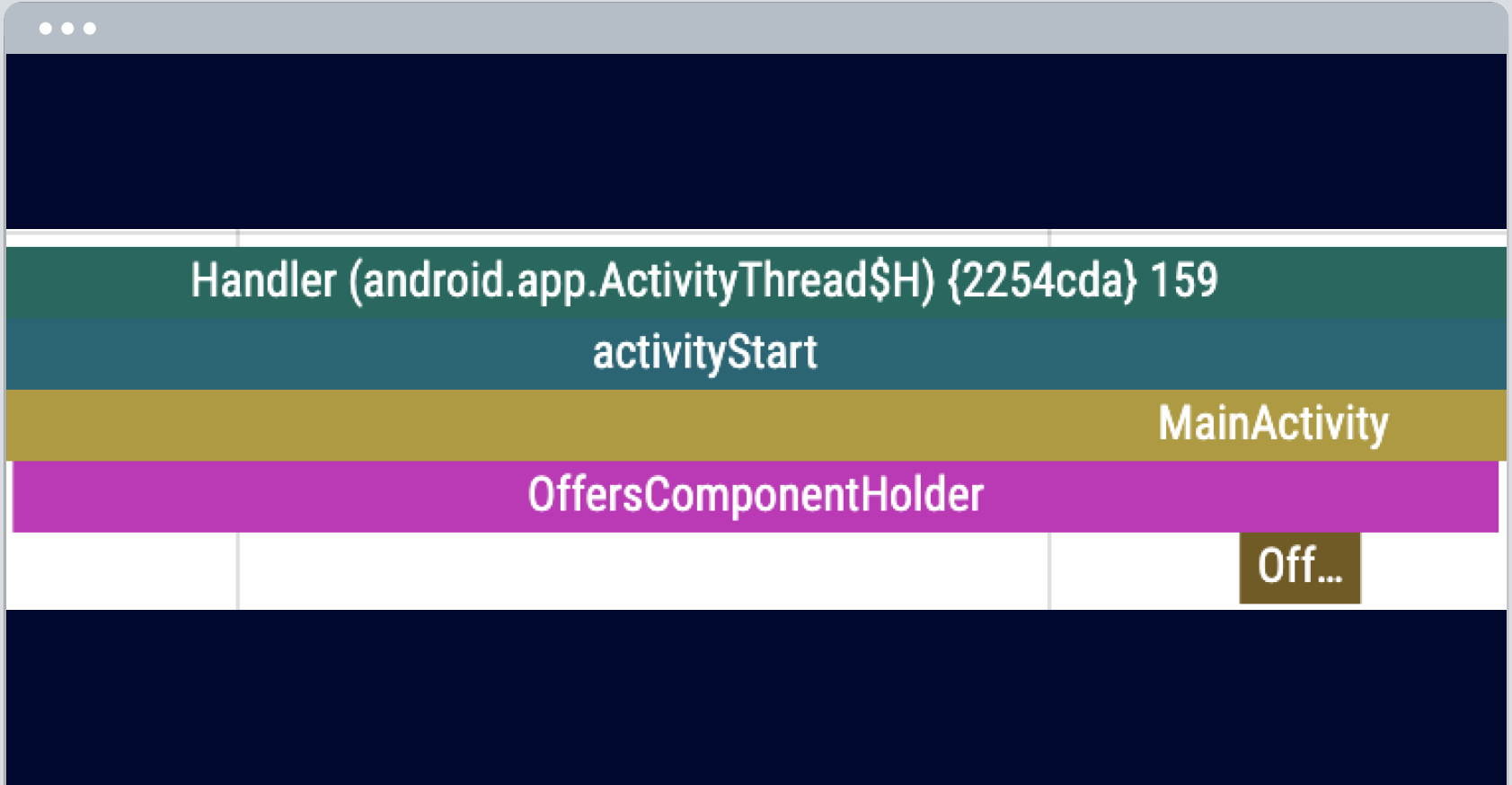


OffersComponent до рефакторинга

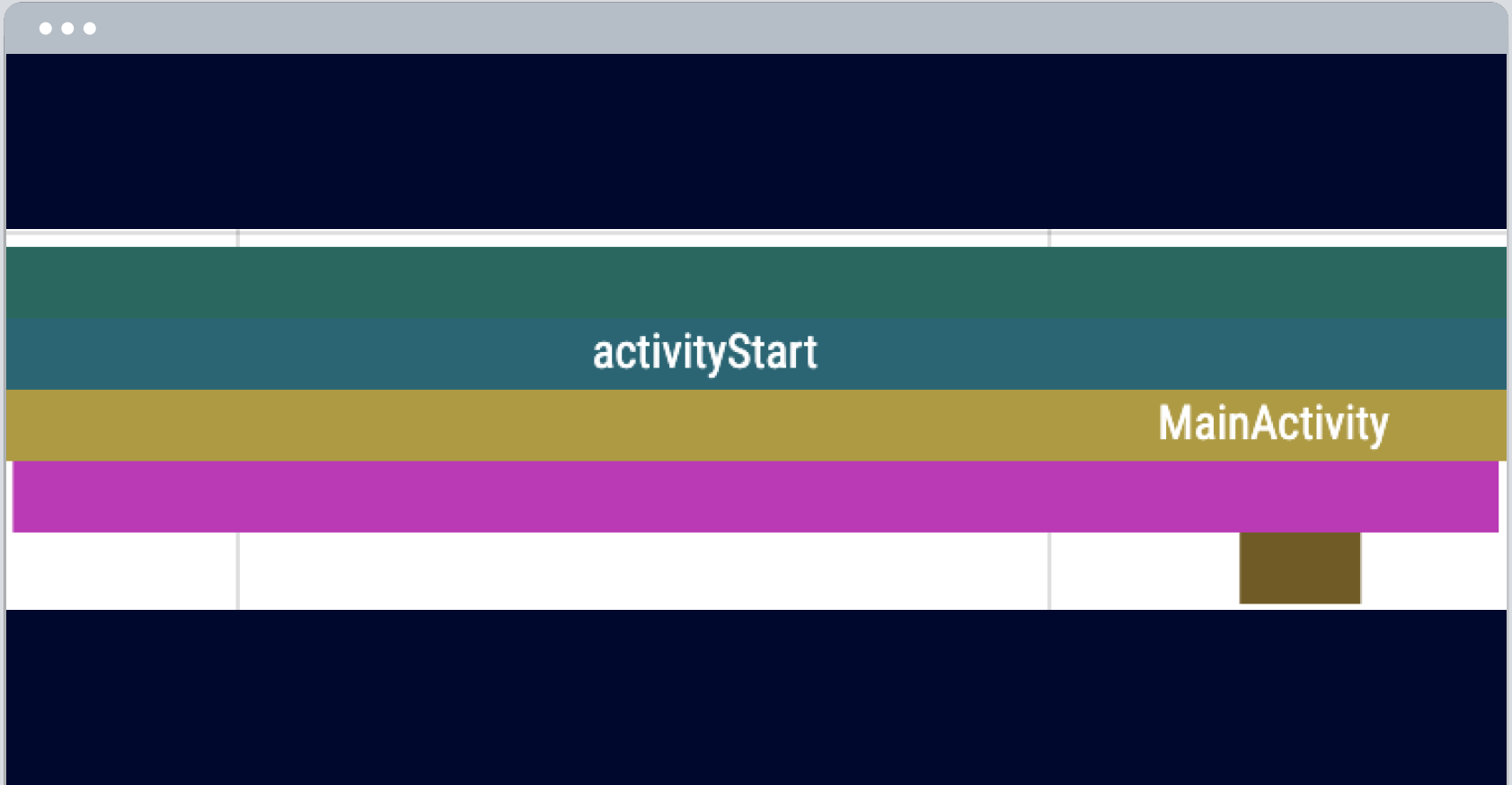


OffersComponent после рефакторинга

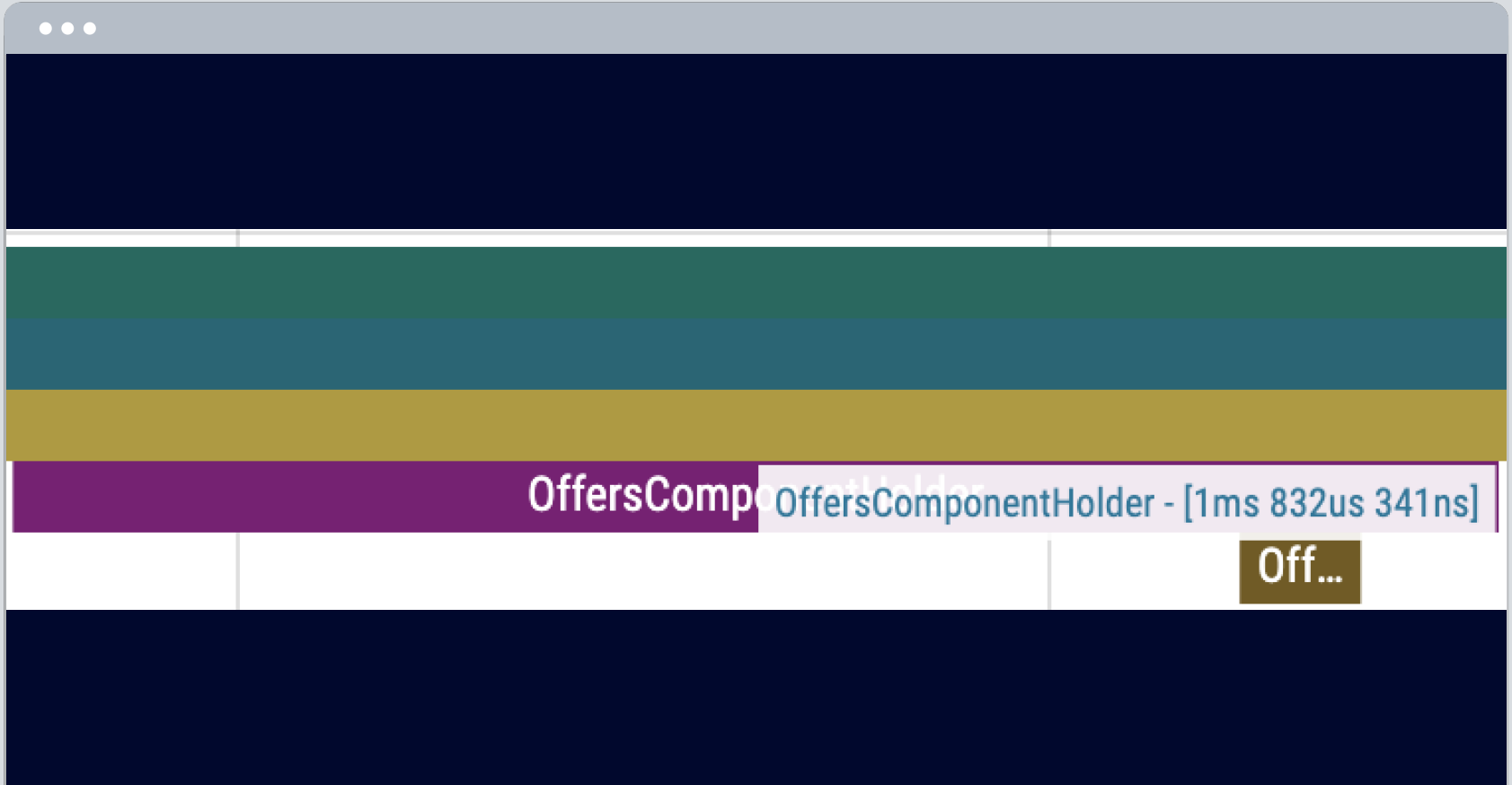
OffersComponent после рефакторинга



OffersComponent после рефакторинга



OffersComponent после рефакторинга



Теория соотносится с практикой

Есть ли последствия?

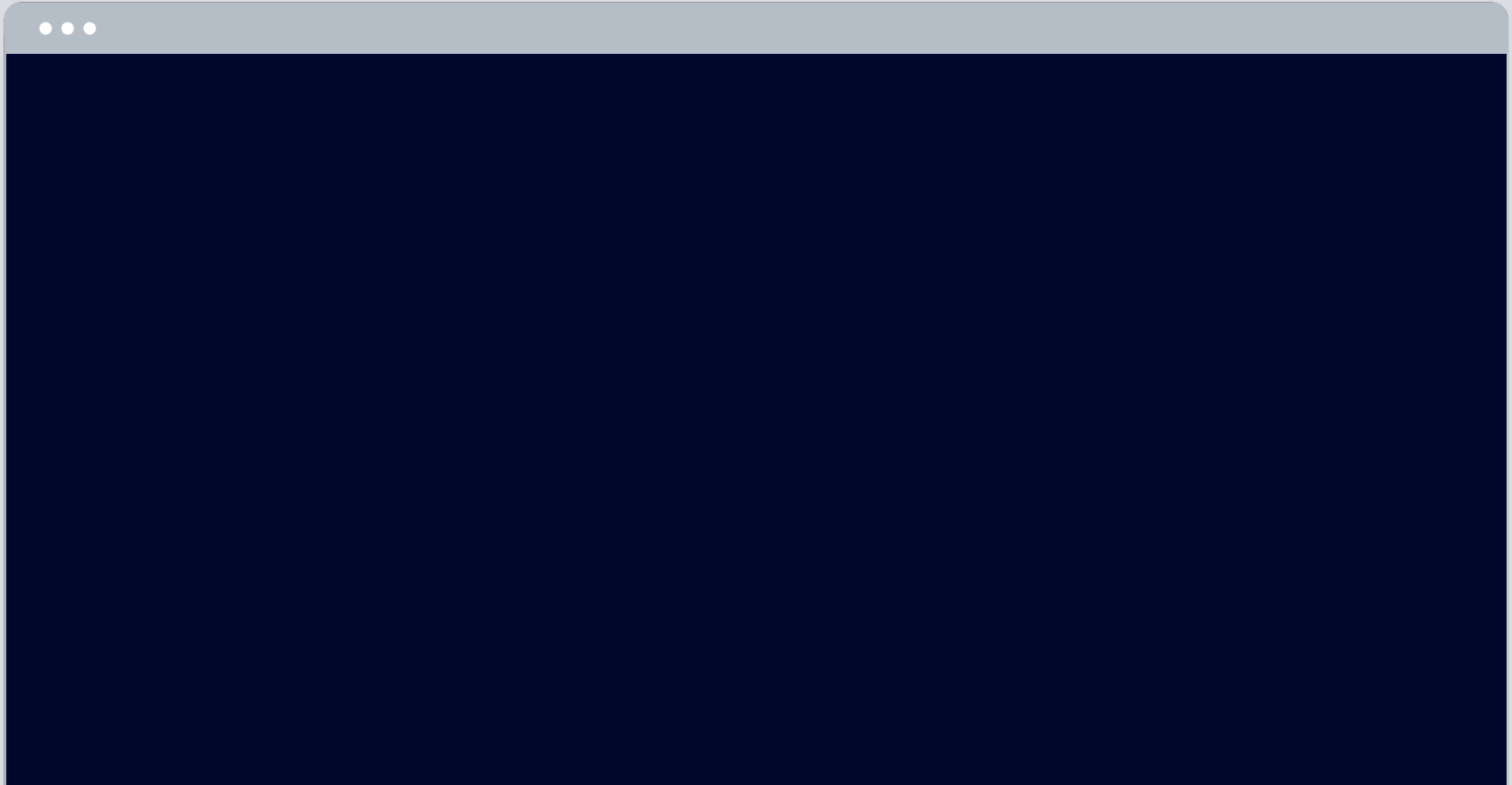
Менее удобный синтаксис

```
internal class ExampleViewModelLazy @Inject constructor(
    private val usecaseLazyA: Lazy<UsecaseA>,
    private val usecaseLazyB: Lazy<UsecaseB>,
    private val usecaseLazyC: Lazy<UsecaseC>,
) : BaseExampleViewModel() {

    override val usecaseA: UsecaseA
        get() = usecaseLazyA.get()
    override val usecaseB: UsecaseB
        get() = usecaseLazyB.get()
    override val usecaseC: UsecaseC
        get() = usecaseLazyC.get()
```

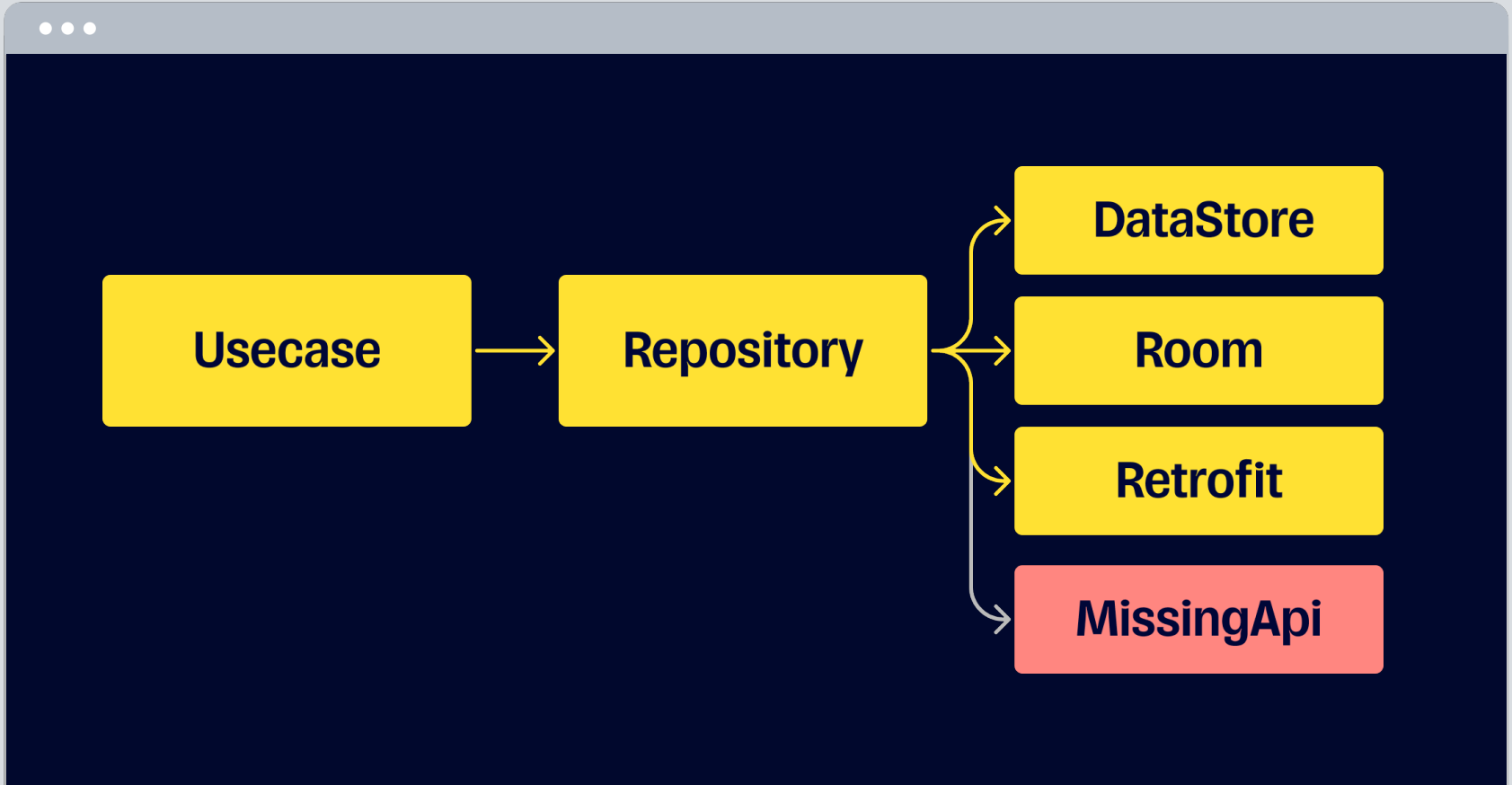
Понять и простить

Иногда нужен главный поток



Инициализировать заранее

Потеря целостности графа



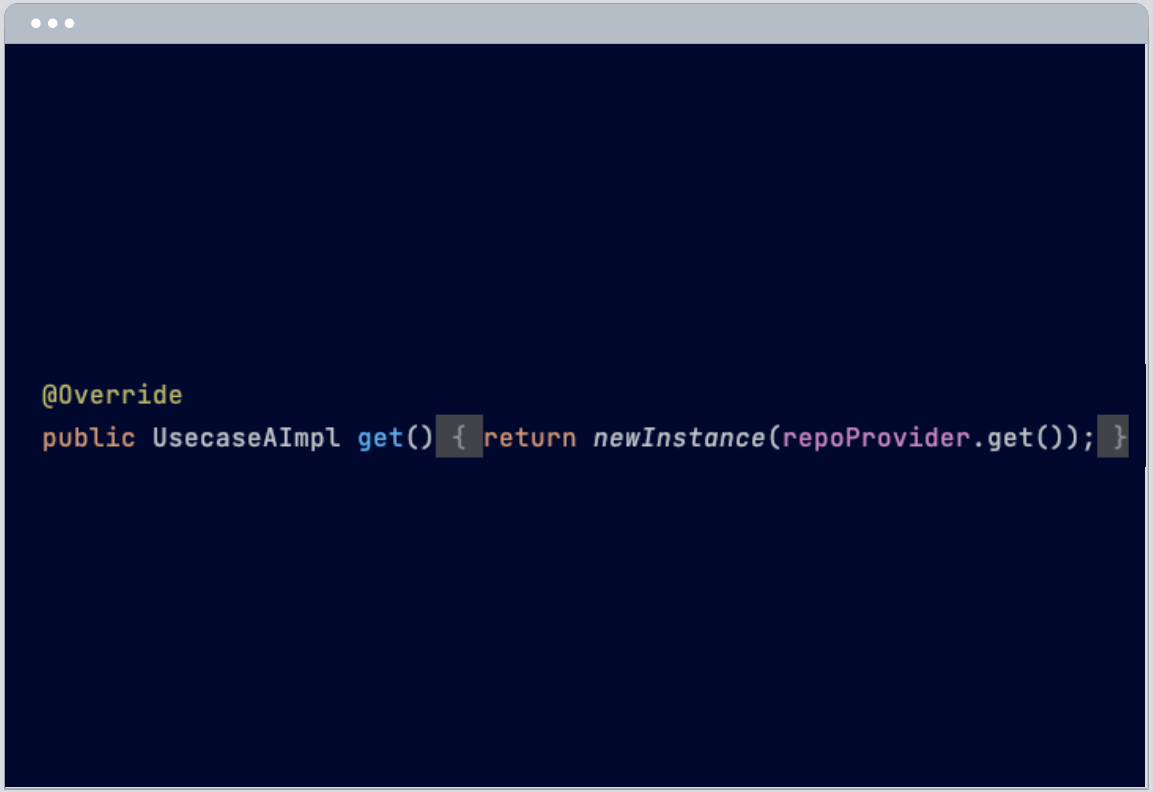
Пишем тесты

```
@RunWith(AndroidJUnit4::class)
@Config(application = BaseApp::class)
internal class CheckInitializedHoldersTest : BaseHoldersTest() {
    @Test
    fun checkThatHoldersInitialized() {
        val notInitialized = findNotInitializedHolders()
        assertTrue(
            message: "Found not initialized holders $notInitialized",
            notInitialized.isEmpty(),
        )
    }
}
```

Можно ли еще что-то сделать?

Интерфейсы ЛИ

Provider


@Override
public UsecaseAImpl get() { return newInstance(repoProvider.get()); }

Dagger.Lazy

```java
@Override
public T get() {
  Object result = instance;
  if (result == UNINITIALIZED) {
    synchronized (this) {
      result = instance;
      if (result == UNINITIALIZED) {
        result = provider.get();
        instance = reentrantCheck(instance, result);
        /* Null out the reference to the provider. We are never going to need it again, so we
         * can make it eligible for GC. */
        provider = null;
      }
    }
  }
  return (T) result;
}
```

Dagger.Lazy

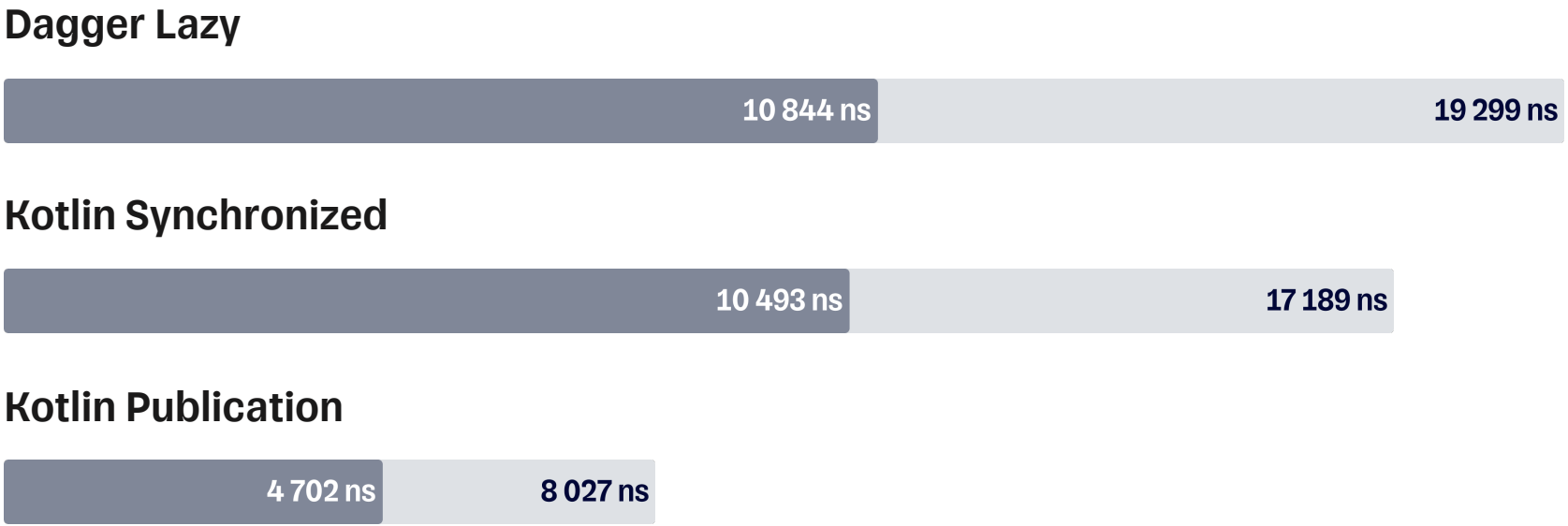
```
synchronized (this) {
  result = instance;
  if (result == UNINITIALIZED) {
    result = provider.get();
    instance = reentrantCheck(instance, result);
    /* Null out the reference to the provider. We
     * can make it eligible for GC. */
    provider = null;
  }
}
```

Dagger.Lazy

```
if (result == UNINITIALIZED) {
  result = provider.get();
  instance = reentrantCheck(instance, result);
  /* Null out the reference to the provider. We
   * can make it eligible for GC. */
  provider = null;
}
```

Всегда ли нужен double check?

Сравнение


Dagger Lazy
10 844 ns
19 299 ns
Kotlin Synchronized
10 493 ns
17 189 ns
Kotlin Publication
4 702 ns
8 027 ns

Custom интерфейс?

Custom интерфейс?

- Можно оптимизировать

Custom интерфейс?

- Можно оптимизировать
- Можно изменить логику

Custom интерфейс?

Можно оптимизировать

Можно изменить логику

Можно проконтролировать целостность

Custom интерфейс?

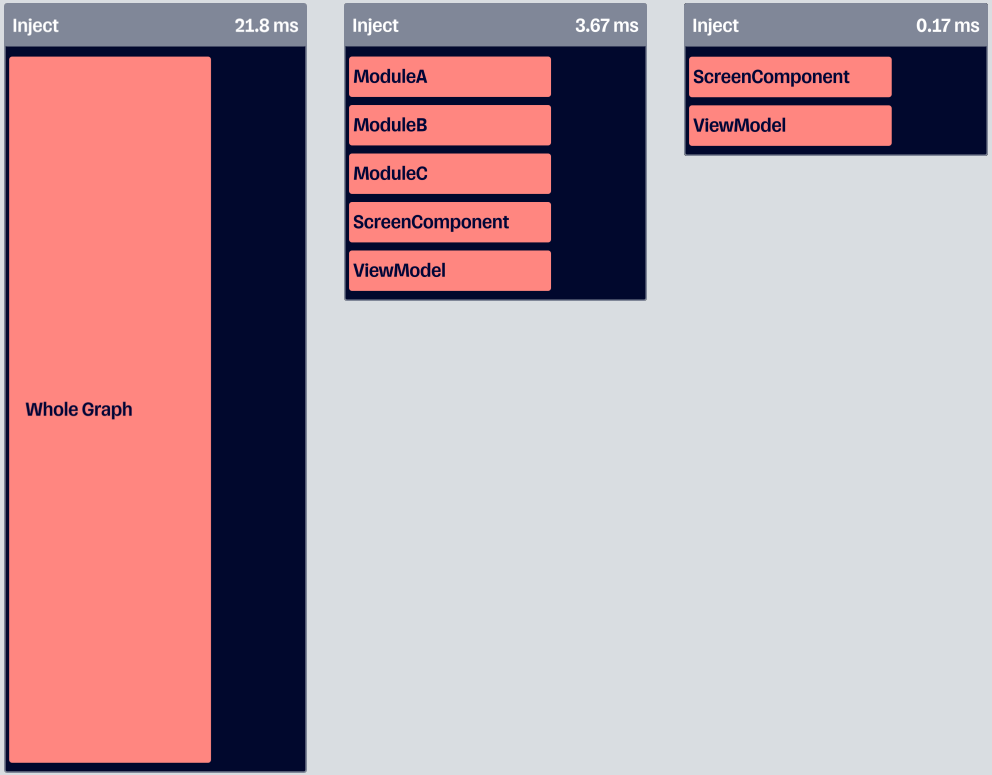
- Можно оптимизировать
- Можно изменить логику
- Можно проконтролировать целостность
- Не работает с Dagger

Выводы

Для чего нужна ЛИ?

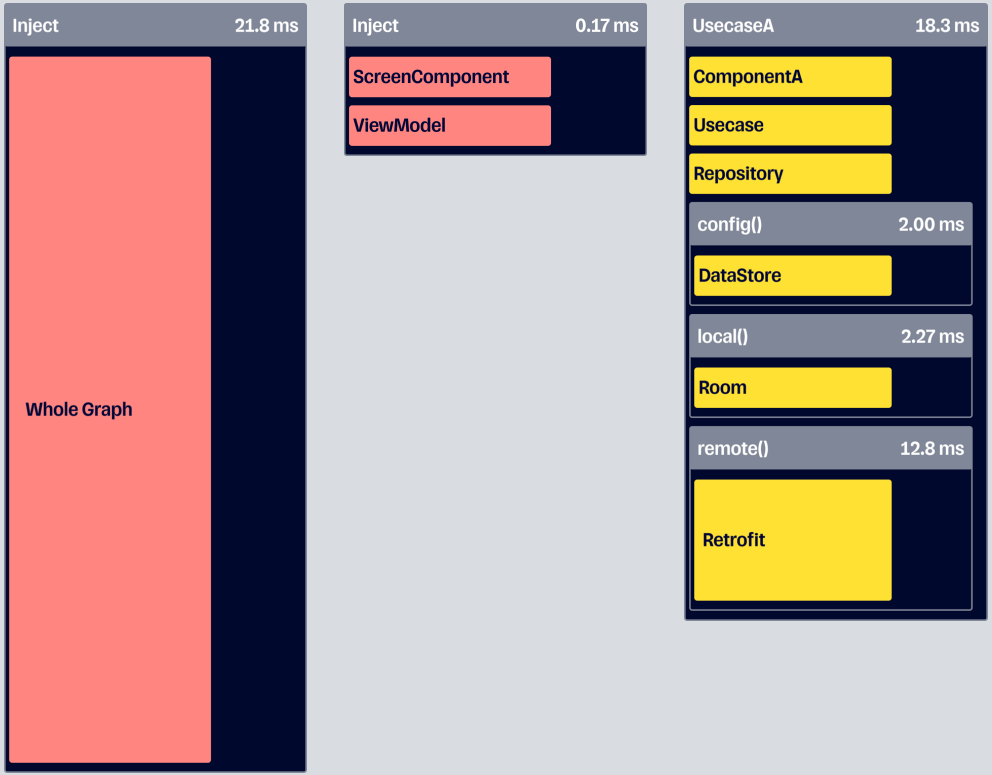
Для чего нужна ЛИ?

- Ускорить старт приложения



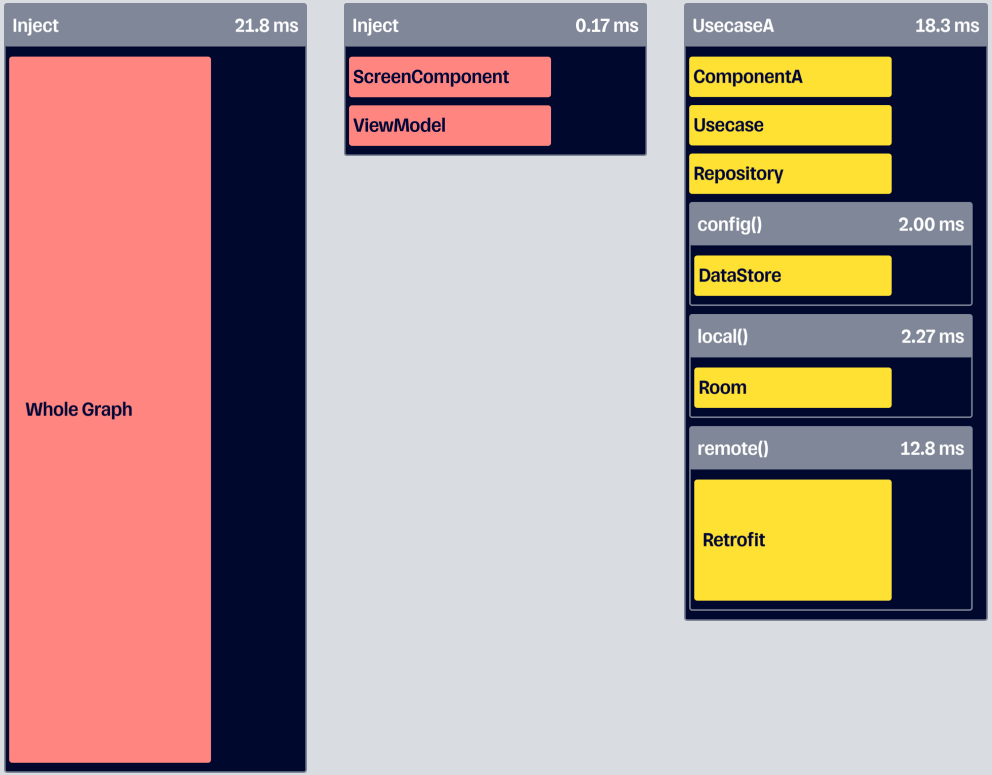
Для чего нужна ЛИ?

- Ускорить старт приложения
- Отложить создание зависимостей



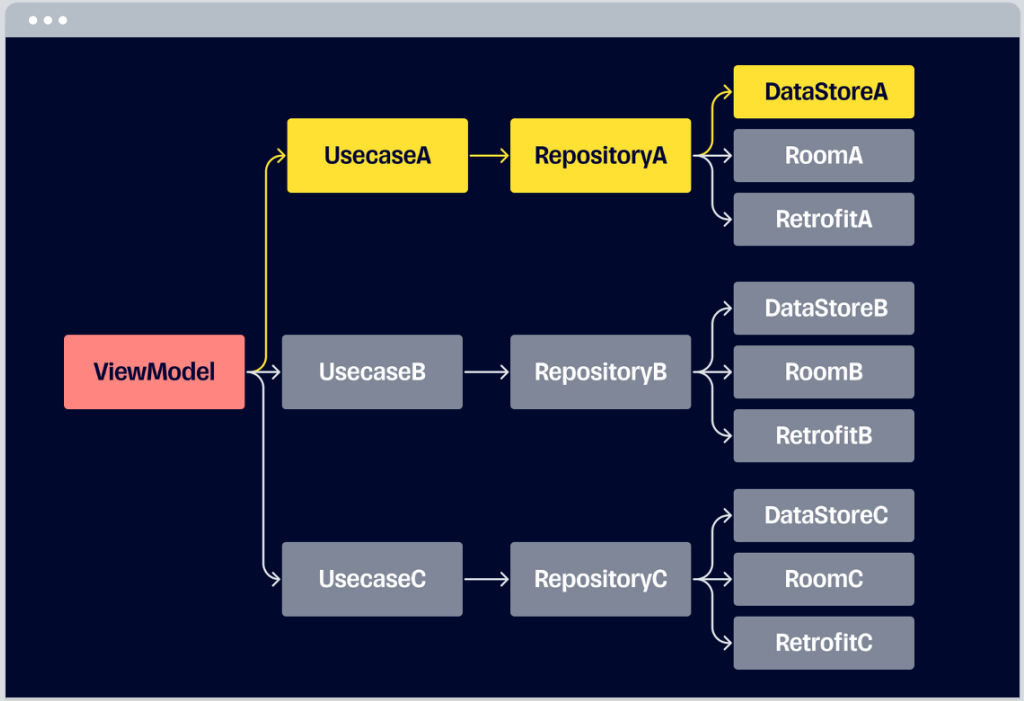
Для чего нужна ЛИ?

- Ускорить старт приложения
- Отложить создание зависимостей
- Вынести создание зависимостей в бекграунд



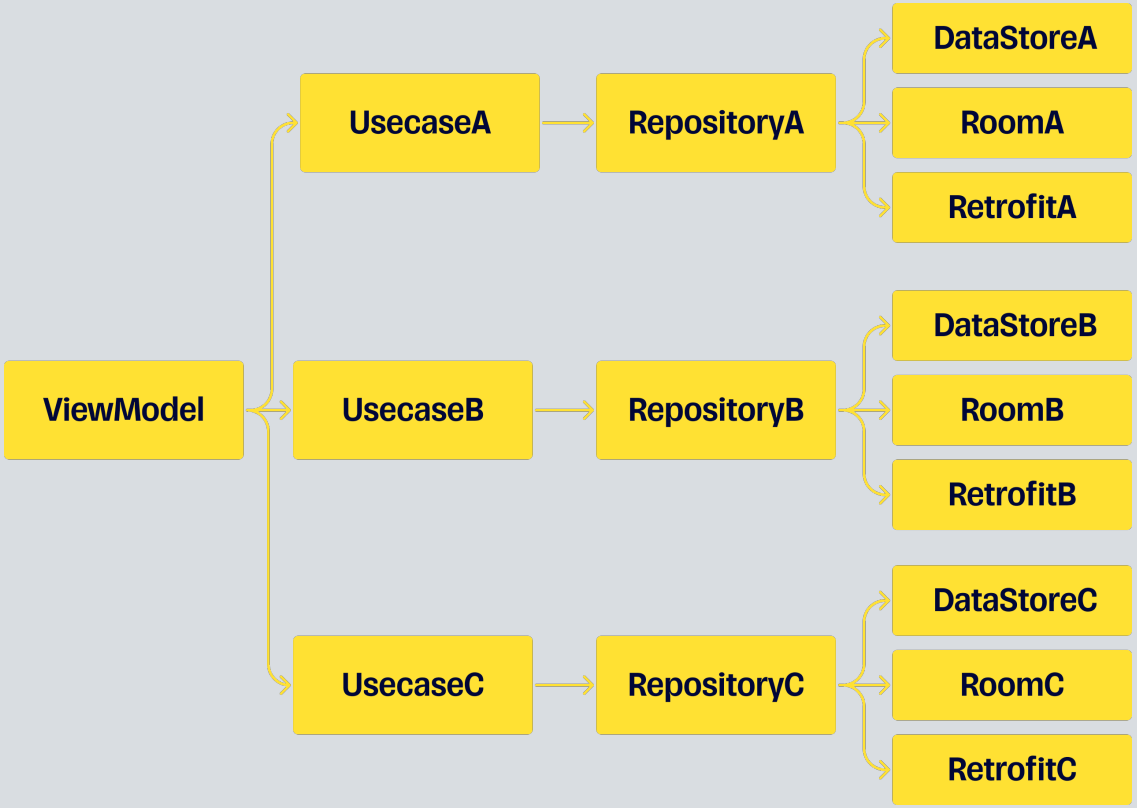
Для чего нужна ЛИ?

- Ускорить старт приложения
- Отложить создание зависимостей
- Вынести создание зависимостей в бекграунд
- Уменьшить количество объектов в памяти

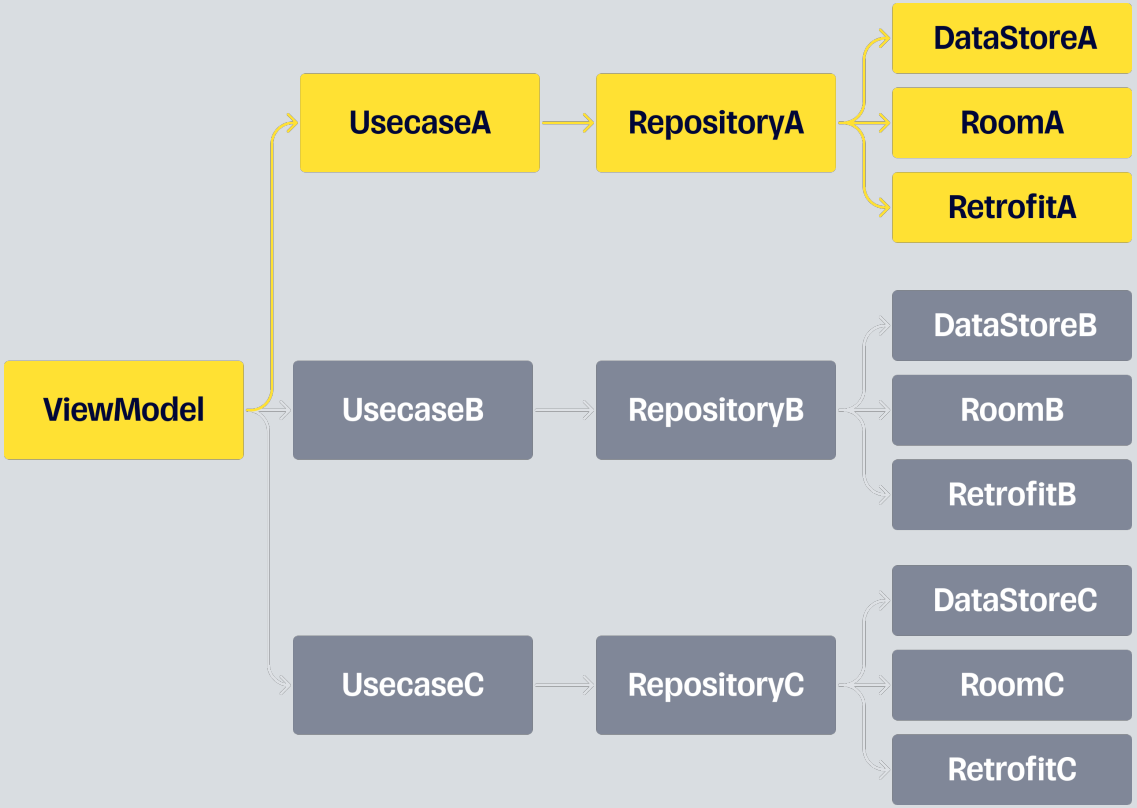


Какую стратегию выбрать?

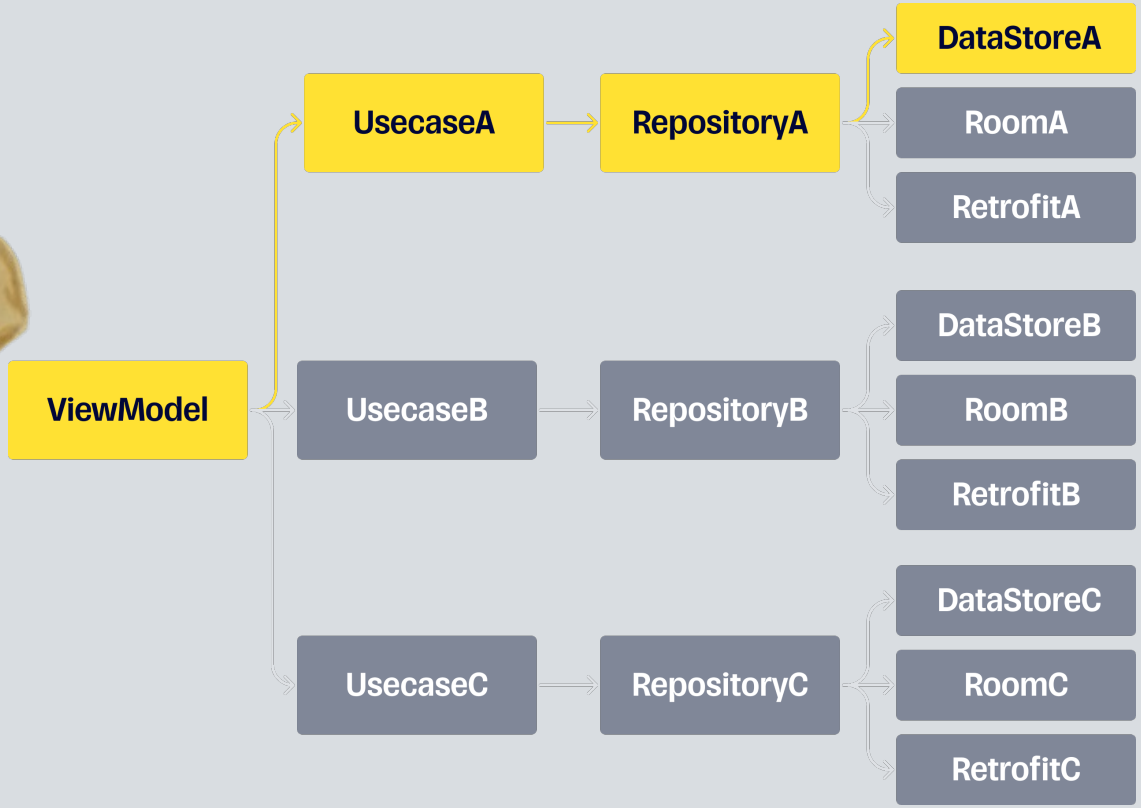
Маленький проект



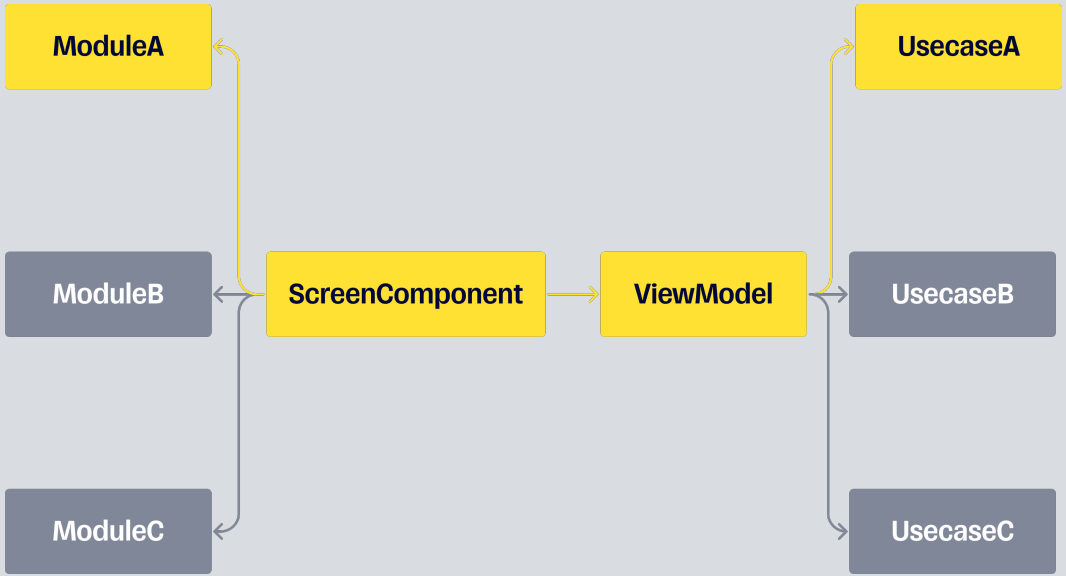
Средний проект



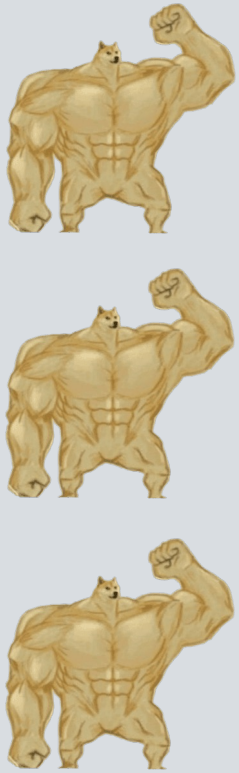
Крупный проект

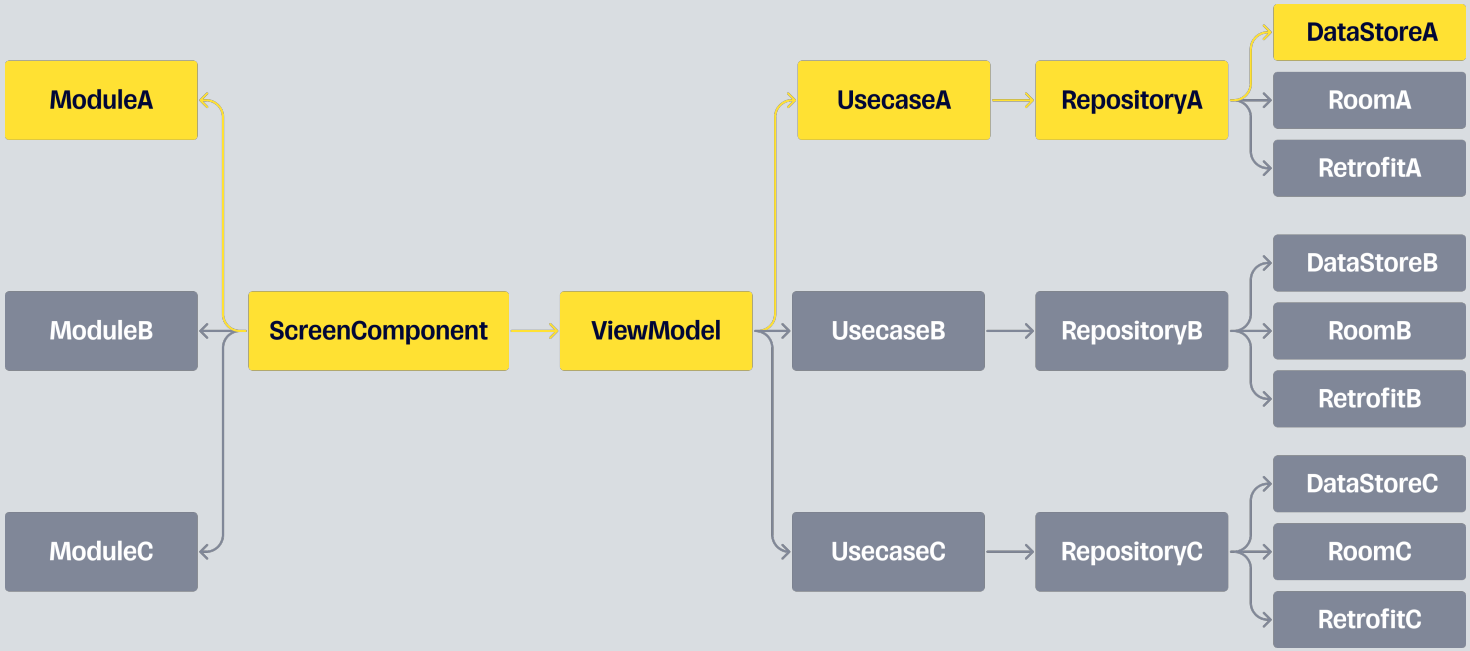


Крупный проект #2



СуперАпп




ModuleA
ModuleB
ModuleC
ScreenComponent
ViewModel
UsecaseA
UsecaseB
UsecaseC
RepositoryA
RepositoryB
RepositoryC
DataStoreA
RoomA
RetrofitA
DataStoreB
RoomB
RetrofitB
DataStoreC
RoomC
RetrofitC

ТИНЬКОФФ

Таганов Александр | разработчик

a.taganov@tinkoff.ru