Яндекс 360

Пишем на KMP

Но какой ценой?

Денис Александров
Тимлид Яндекс 360

Об меня

15

Годиков программирую

7

Годиков преподаю

7

Годиков тимлидирую

Вечно ищу приключений (и не всегда только себе)

Об доклад

1. Что мы разрабатывали
2. Путь от идеи KMP
   до реализации в цифрах
3. Ретроспектива
   (а стоило ли того)
4. Выводы
5. Неожиданные выводы

Словарь

KMP

Kotlin Multiplatform
Android, IOS, JVM (Backend+ desktop), Web, etc.

CMP

Compose Multiplatform
Android, Desktop,
IOS (Beta), Web (Alpha)

Про проект

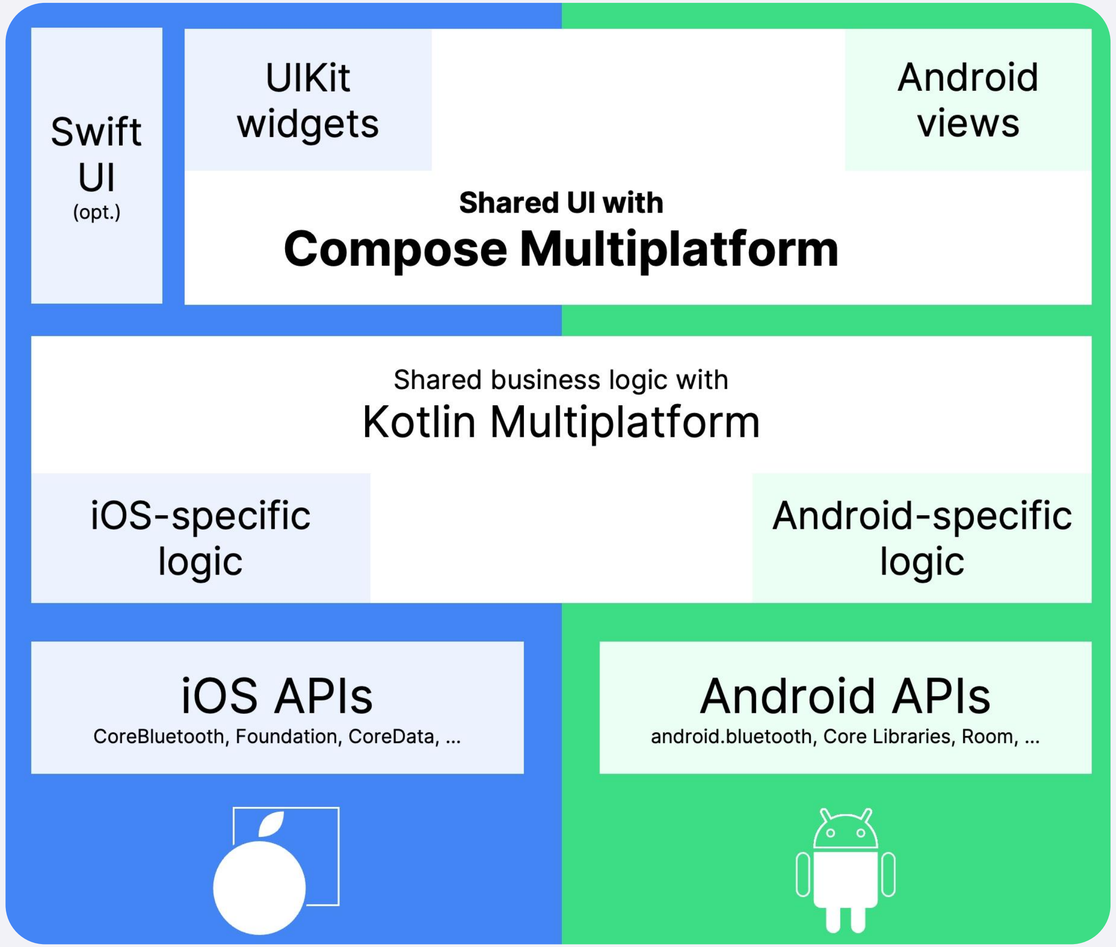
Стартовали разработку 1 апреля 2024

Хотим в прод в октябре – ноябре

Изначальная концепция — KMP + Compose/ SwiftUI

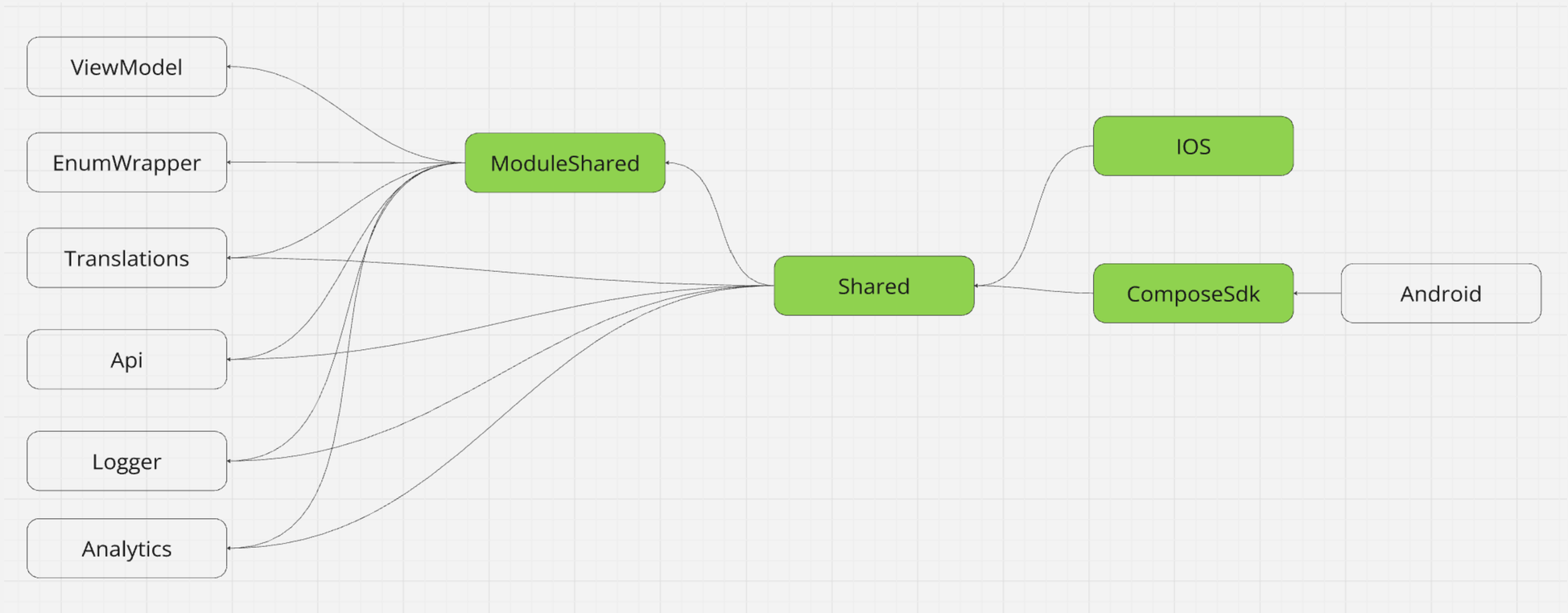
Про KMP концепцию

1. MVVM подход
2. На нативном IOS только верстка
3. DI, логика, переводы — все в KMP
4. Android — на CMP с входом через Activity

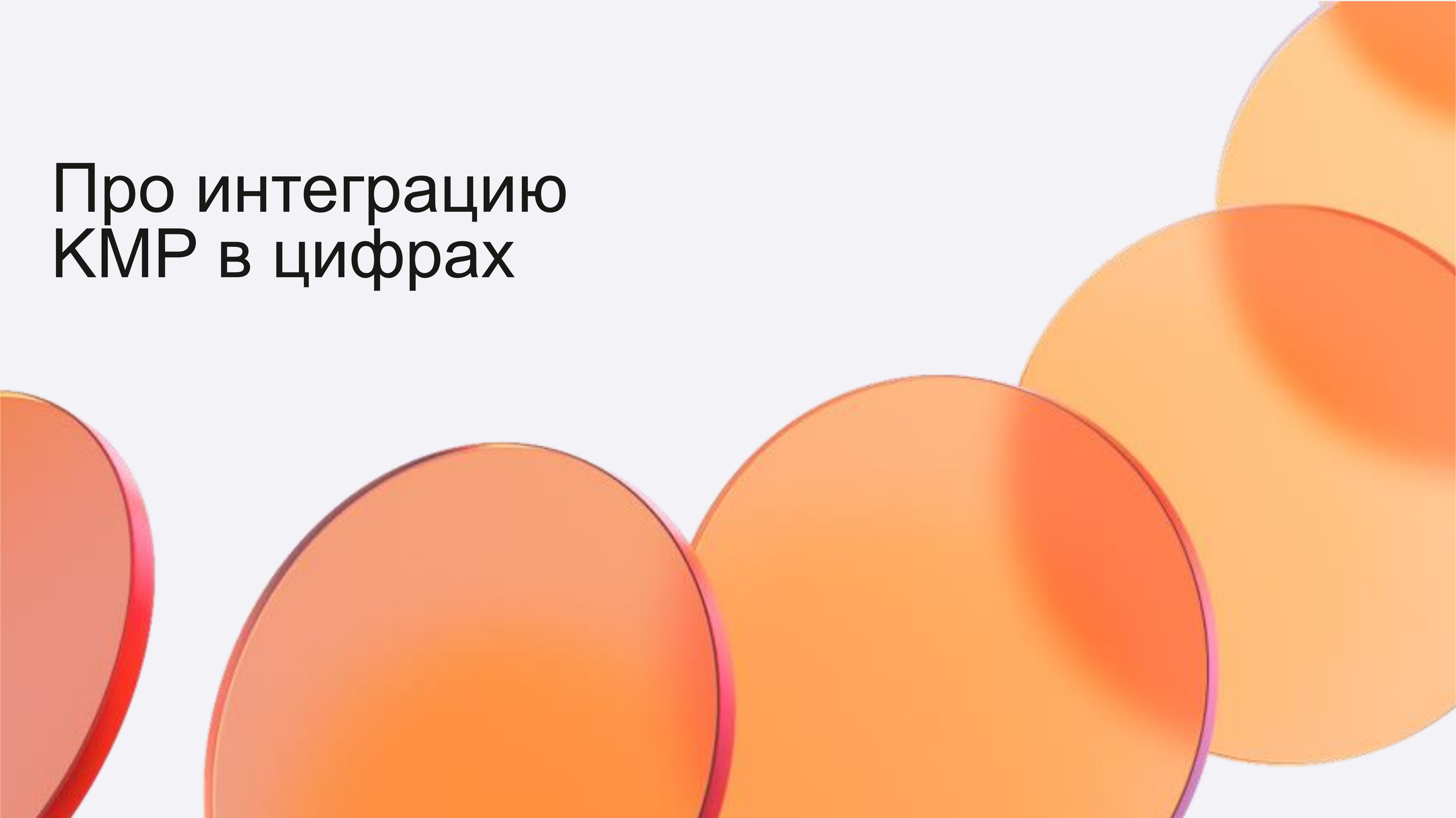


Общая схема

1. Зонтичный модуль shared для интеграции бизнес логики
2. Зонтичный модуль ComposeSdk для написание всего CMP приложения
3. IOS для нативной верстки



Про интеграцию KMP в цифрах



Сколько стоит интеграция в мегабайтах и секундах

Собираем пустые проекты

Поочередно добавляем библиотеки в проект

Замеряем время и размер сборок

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	
Coroutines	7.5	1	67.89	1	3.1	1	

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	
Coroutines	7.5	1	67.89	1	3.1	1	
DateTime	7.5	1	68.44	1	7.6	1	На холодную 20 сек

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	
Coroutines	7.5	1	67.89	1	3.1	1	
DateTime	7.5	1	68.44	1	7.6	1	На холодную 20 сек
ViewModel	7.5	1	68.42	1	7.6	1	

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	
Coroutines	7.5	1	67.89	1	3.1	1	
DateTime	7.5	1	68.44	1	7.6	1	На холодную 20 сек
ViewModel	7.5	1	68.42	1	7.6	1	
Ktor	8.63	2	71.37	4	12.9	1	На 2.X было сильно меньше

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	
Coroutines	7.5	1	67.89	1	3.1	1	
DateTime	7.5	1	68.44	1	7.6	1	На холодную 20 сек
ViewModel	7.5	1	68.42	1	7.6	1	
Ktor	8.63	2	71.37	4	12.9	1	На 2.X было сильно меньше
SqlDelight	8.7	3	85.07	3	12.9	2	Нативные драйвера

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	
Coroutines	7.5	1	67.89	1	3.1	1	
DateTime	7.5	1	68.44	1	7.6	1	На холодную 20 сек
ViewModel	7.5	1	68.42	1	7.6	1	
Ktor	8.63	2	71.37	4	12.9	1	На 2.X было сильно меньше
SqlDelight	8.7	3	85.07	3	12.9	2	Нативные драйвера
Koin	8.8	1	85.47	2	12.9	2	

	Android		Desktop		IOS		
Зависимость	Размер	Время	Размер	Время	Размер	Время	Комментарий
Dry	5.25	1	67.84	1	0.2	1	Compose включен в android и JVM
Kotlin	5.25	1	67.84	1	2.7	1	
Coroutines	7.5	1	67.89	1	3.1	1	
DateTime	7.5	1	68.44	1	7.6	1	На холодную 20 сек
ViewModel	7.5	1	68.42	1	7.6	1	
Ktor	8.63	2	71.37	4	12.9	1	На 2.X было сильно меньше
SqlDelight	8.7	3	85.07	3	12.9	2	Нативные драйвера
Koin	8.8	1	85.47	2	12.9	2	
Compose	8.8	1	85.47	3	45	3	

Выводы

1. Собирается все очень быстро (за исключением десктопа)
2. На KMP можно писать даже библиотеки (на IOS весь около 10мб)
3. Если использовать только стандартный котлин, можно уложиться в пару мегабайт
4. На CMP писать библиотеки сложнее, но тоже можно (около 47мб)
5. Десктоп весит и собирается больше остальных, но в целом числа адекватные

Сколько стоит написание модуля в байтах

Пишем модуль логина

Ручки Ktor

3

Проперти
в SharedPreferences

1

MVVM,
Coroutines

Экран
на CMP

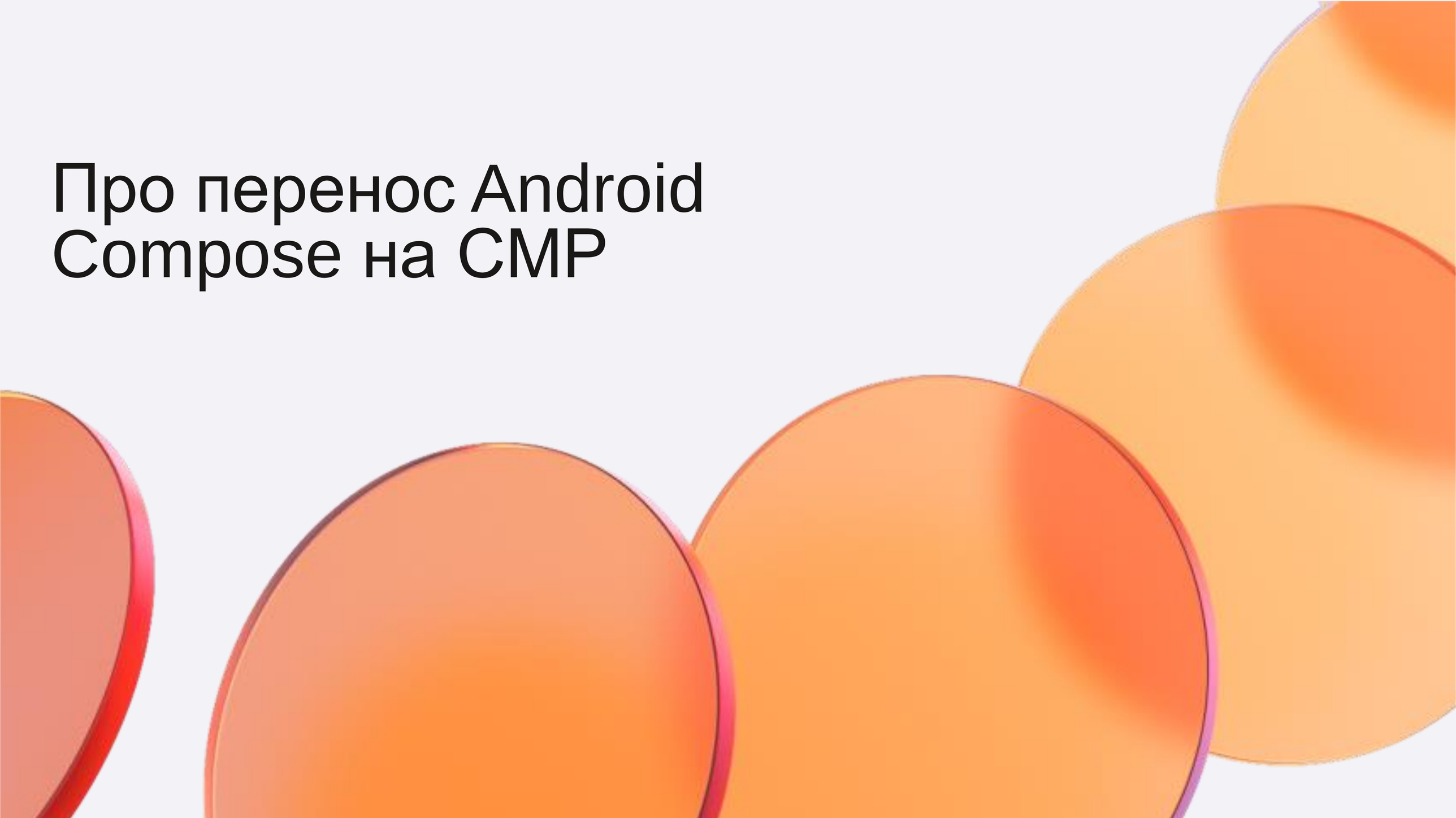
Экран
на SwiftUI

	Android	Desktop	iOS2	iOS
Просто зависимости	12.324.456	93.657.439	51.059.43	50.805.815
+1 модуль	12.407.967	93.815.412	51.230.138	57.882.972
+2 экрана	12.443.616	93.839.303	51.232.140	57.883.020

Выводы

1. Размер увеличивается линейно и полого
2. Все прилаги растут равномерно
   (можно ориентироваться на размер Android)
3. Артефакт со сборкой IOS нужно
   изучить (плавающий инкремент)
4. На время сборки практически не влияет
5. Модульная архитектура не влияет

Про перенос Android Compose на CMP



Сколько стоит перенести дизайн систему Compose на CMP

Имеем

Картинок

400+

Кодген,
размеры и цвета

Нативный
шрифт

Задача

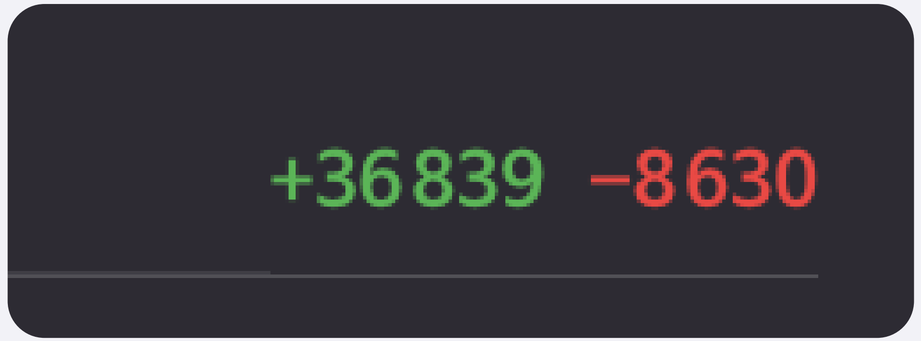
Получить тоже
самое на CMP

Запуск

- 2 задачи
- 3 человеко-дня разработки (без опыта x2)
- 2 ПРа
- 1 хотфикс (шрифты сломались)
- Постоянную (но легкую) боль,
  т.к. превью в 1.6.11 не работало

Нам помогали

- Самопальный генератор верстки из фигмы
- Плагин Svg to Compose
- Бесконечная вера в то,
  что это пригодится



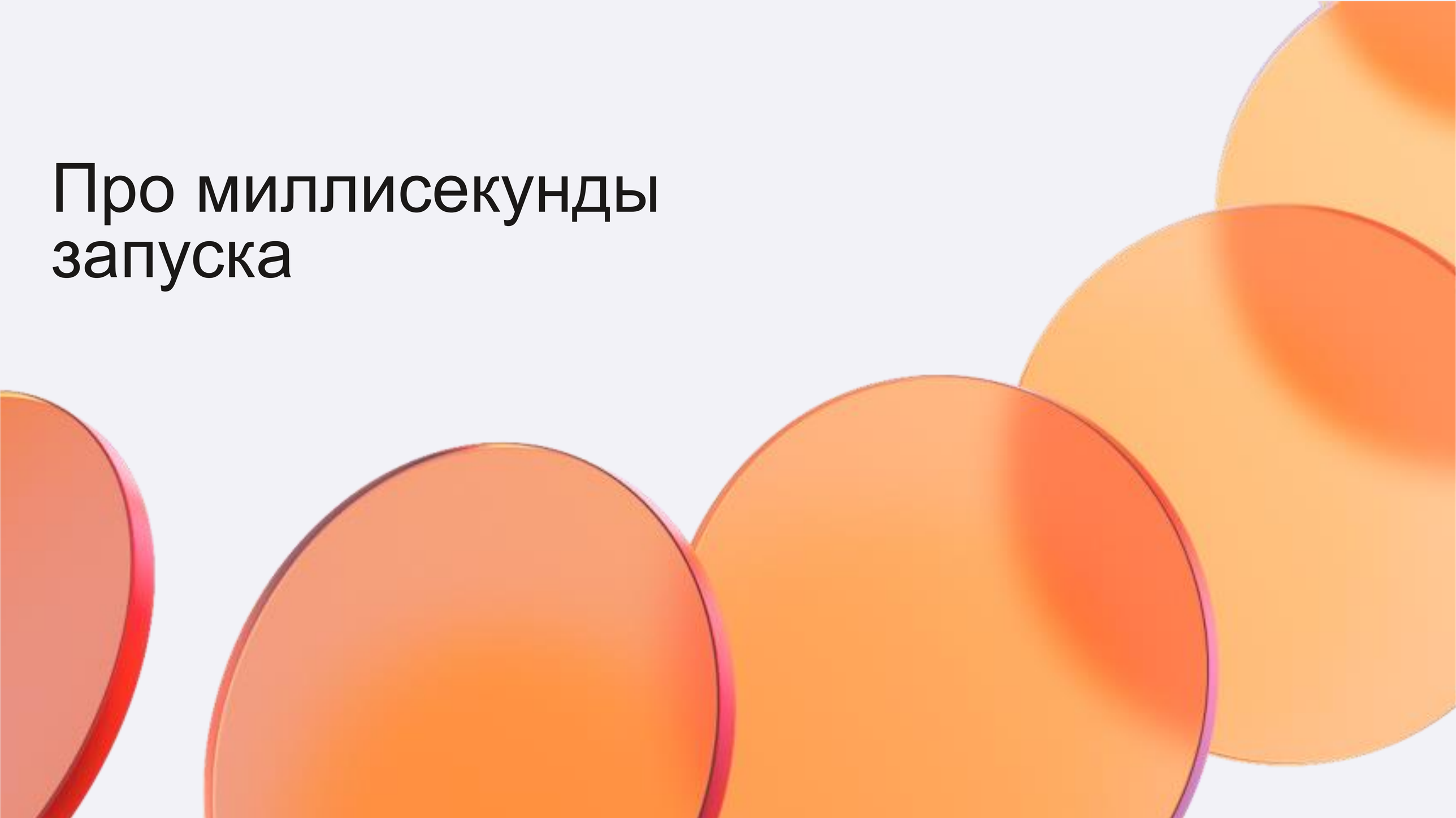
Выводы

1. У нас есть ДС на десктопе (можно запускать без эмулятора, сравнивать с нативкой, растягивать размер приложения и пр.)

2. С точки зрения ресурсов это бесплатно (меньше одного человека дня на запуск)

3. Дизайнеры благодарны

Про миллисекунды запуска



Запуск в попугаях

Меряем запуск пустого приложения

Пустого
с KMP

С Corountines (запустим одну джобу в корутинах, а также синхронно вызовем код)

С CMP (через сколько MC отрисуется первый экран)

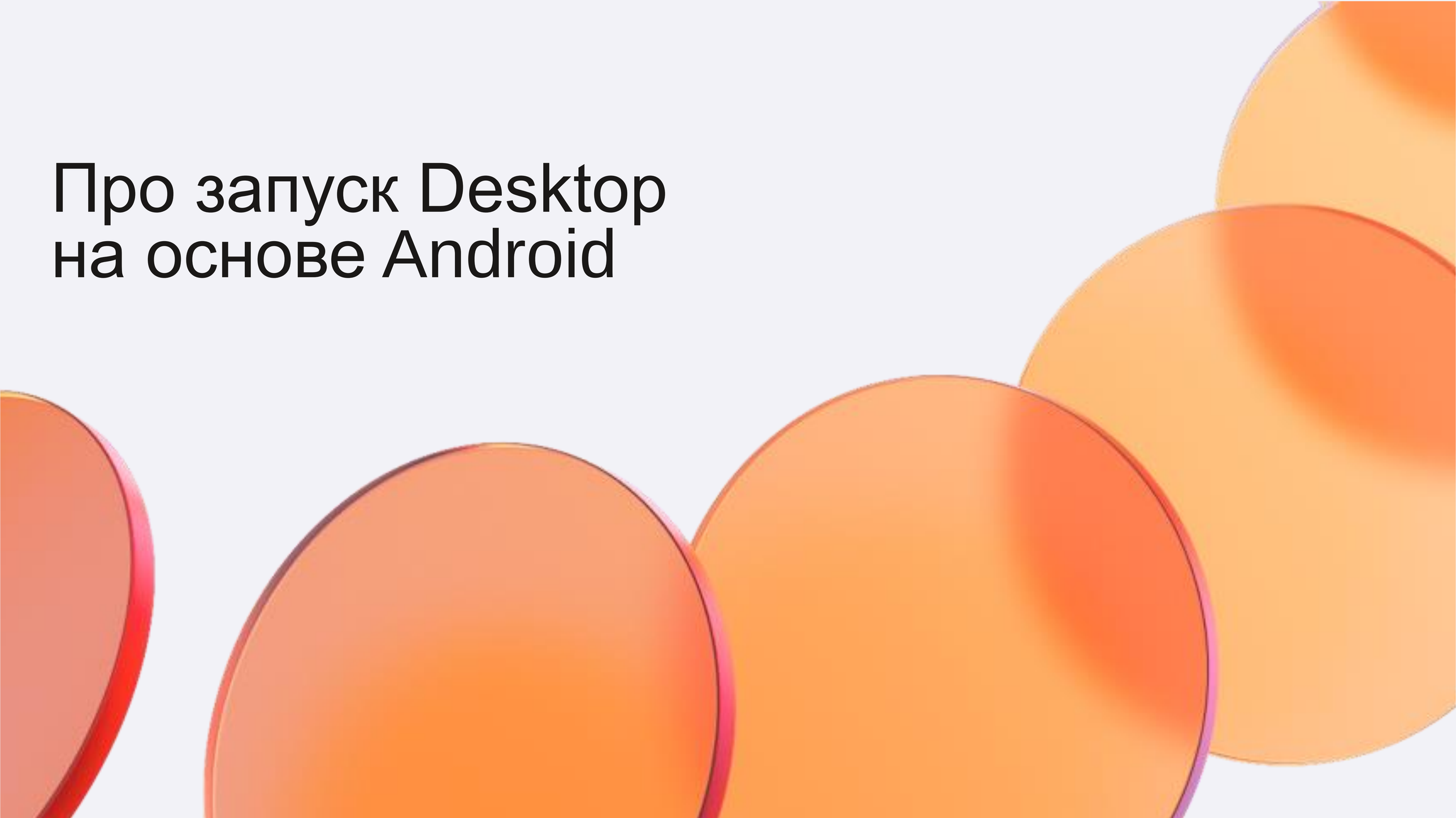
	Время запуска			
	Android	Desktop	IOS	
Пустой проект	165	257	101	Пустой проект
Вызов корутины	144	256	63	Вызов корутины
+1 Compose экран	163	280	73	+1 Compose экран

Выводы

1. Kotlin на IOS работает быстрее андроид (шутка)

2. Compose/Coroutines не влияют на скорость запуска приложения

Про запуск Desktop на основе Android



Поднятие Desktop в живом приложении

Имеем андроид, написанный на технологиях выше

1

Имеем собирающийся десктоп (если чего-то не хватало, просто ставили в actual TODO (надо визуализировать)

2

Надо прикрутить авторизацию (нет библиотеки)

3

Запуск

Задачи

3

Человеко-дня разработки
(на самом деле сделал в отпуск)

4

ПРов

5

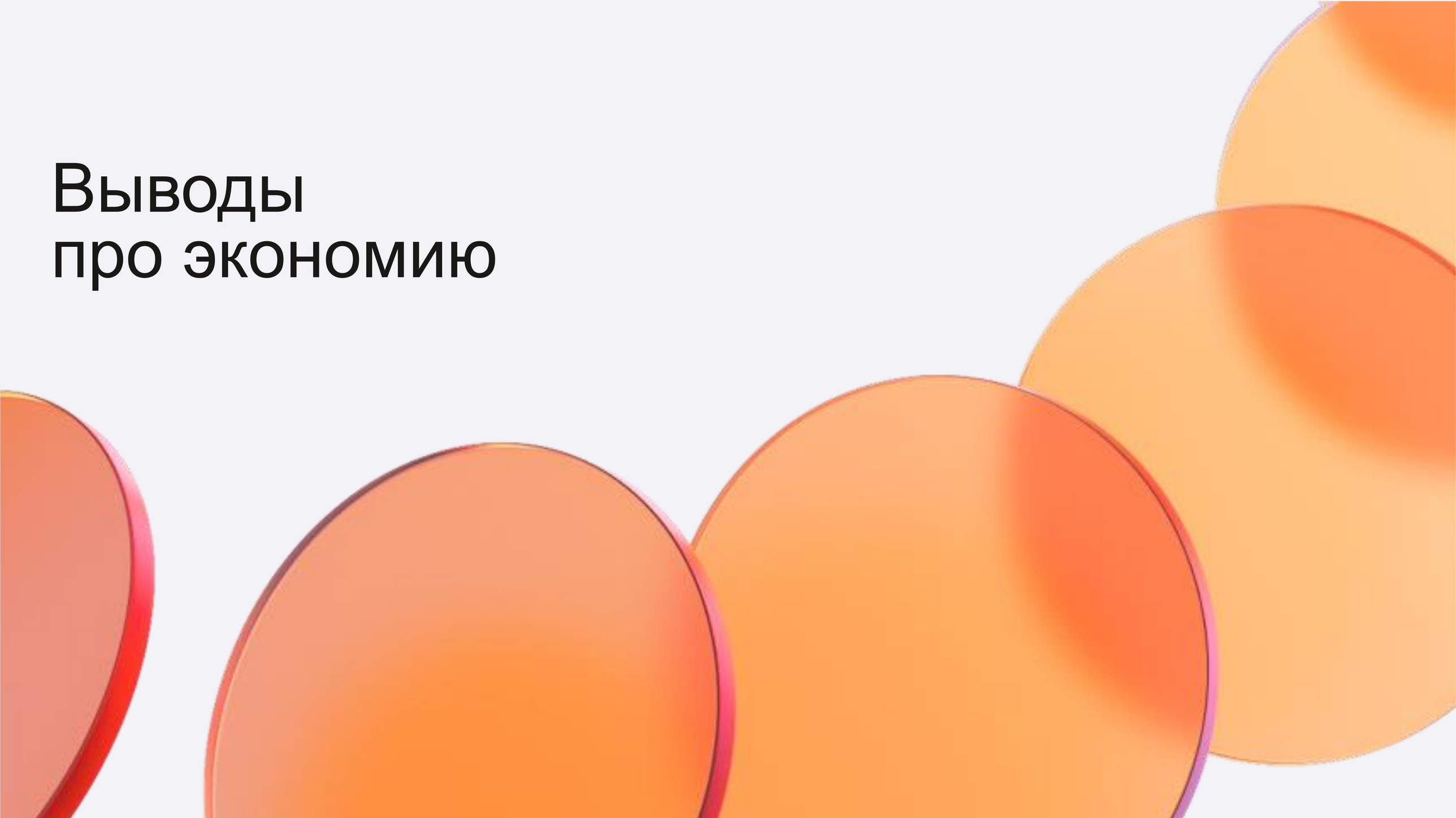
Долго буксовал с подключением
авторизации по апи (кейс редкий)

Дедлайн — вчера

Выводы

1. Многие штуки можно переиспользовать с Android (okHTTP)
2. Нет секьюрных преференсов
3. Нет большого кол-ва библиотек
4. Много UX надо адаптировать (например, работу с файлами)
5. В целом работает так же как на других платформах

Выводы
про экономию



Экономится ли время на разработке

1. Ядро пишется очевидно один раз
2. UI пишется дважды,
   но с уже адаптированным UX
3. По CMP, выгода от не нативного UI пока
   не очевидна (есть ресурс IOS, пока
   не хватает нескольк штук в Compose)
4. В прод пока рано
   (по крайней мере нырять с головой)
5. Реально много времени экономится на багах
   и доработках (80 процентов задач
   дорабатываются сразу на двух платформах)
6. Итого минус 1 платформа в бизнес логике =
   30 процентов времени разработки

API without Android classes in their signatures

Some parts of the API can be available only for the Android target, even if their signatures don't contain `android.*` or `androidx.*` classes, and the API is applicable to other platforms. The reason behind this is usually that the implementation uses many platform specifics and it takes time to write other implementations for other platforms.

Normally, parts of the API like this are ported to Compose Multiplatform after they are introduced in Jetpack Compose for the Android target.

In Compose Multiplatform 1.6.11, the following parts of the API are **not** available in `commonMain`:

- `Modifier.imeNestedScroll()` ↗ function
- `Modifier.systemGestureExclusion()` ↗ function
- `Modifier.magnifier()` ↗ function
- `LocalOverscrollConfiguration` ↗ variable
- `AnimatedImageVector.animatedVectorResource` API ↗
- material3-adaptive ↗ library
- material3-window-size-class ↗ library

Экономится ли время на тестировании

Очевидно дешевле
писать тесты

Пока UI разный,
тестировать надо
и то, и то

И перепроверять
на соседней
платформе – тоже

При норме на тесты 20% времени,
минус одна платформа = 10%
от общего времени разработки

Багов объективно меньше
(они на мультиплатформе,
это 30% на исправление)

Про сборку и пр.

Сборка идет параллельно, поэтому всегда упираемся в самый долгий пайплайн (сейчас это IOS)

Стараемся это минимизировать, занимаясь последовательным написанием бизнес логики и UI

На изменение в кор логике тригерится сборка и тест обеих платформ независимо (это разово нагружает инфраструктуру, но в штуках не зависит от KMP)

Есть смысл пускать Android вперед — он собирается быстрее

Больно собирать IOS (как и писать PODы в целом), плюс пересборка долгая

Сборка IOS — 2.5 минуты против 1.15 на Android, плюс пайплайн тоже гоняеся примерно в 2 раза дольше

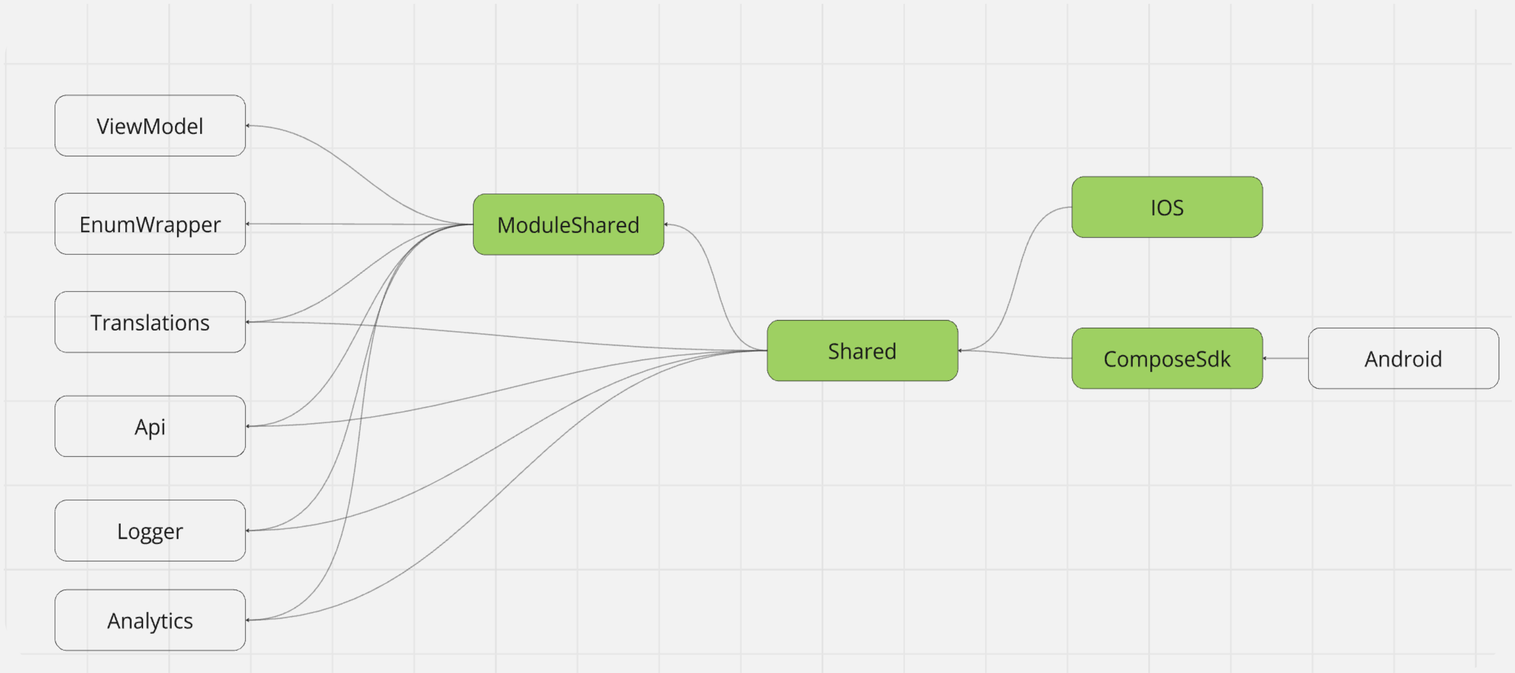
Почему с IOS все не просто

Транзитивные зависимости
не работают

Нужен модуль-зонтик для
подключения и мержа
зависимостей

Отсюда все недостатки
зонтичной системы
наследования

Все не так драматично
(в числах все равно получается
довольно быстро)



Общие выводы

1. Писать на KMP можно,
   это эффективно и не больно

2. Общая бизнес логика экономит около
   30 процентов времени разработки
   и столько же тестирования

3. Писать на Compose надо и обязательно
   с прицелом в CMP (это бесплатно)

4. Писать на CMP пока еще больно,
   но можно начинать экспериментировать
   на небольших фичах

5. Ждем стабильного IOS в CMP

А как во всем этом убедить всех остальных

Оперируйте числами

Вы правда сэкономите на разработке

но не так много, как хотелось бы, но суммарно это все равно хороший результат

KMP + CMP дает наилучшую гибкость в связке с нативом

это все же не фреймворк, можно писать отдельные классы и экраны с интеграцией в натив

...Никак (честно)

Почему убеждение не работает?

1

Предмет этого спора не в плоскости эффективности

2

Мультиплатформа давно имеет подмоченную репутацию

3

Apple будет гнуть свою линию

4

IOS разработчики тоже хотят есть

5

Типичный пример — отношение к китайским брендам

6

Ищите единомышленников, которые любят считать эффективность

7

Мы в 360 — считаем, и иногда пишем на KMP

Яндекс 360

Вопросы?

t.me/guitariz

Денис Александров
Тимлид Яндекс 360