МОБИЛЬНАЯ РАЗРАБОТКА.

РЫНОК МОБИЛЬНЫХ ОПЕРАЦИОННЫХ СИСТЕМ:

- Android + iOS: 99% рынка
- Android: ~2.5 млн. приложений в Google Play Store
- iOS: ~1.8 млн. приложений в App Store
- Еще варианты: Windows Phone, Аврора
- Мобильные телефоны на основе ОС Linux: PinePhone

ОТДЕЛЬНЫЕ НАТИВНЫЕ ПРИЛОЖЕНИЯ

- Android: Java, Kotlin.
- iOS: Objective-C, Swift.
- Доступ к функциям ОС "из коробки" (календарь, файловая система, контакты и др.).
- Высокая производительность.
- "Родной" интерфейс.

ОТДЕЛЬНЫЕ НАТИВНЫЕ ПРИЛОЖЕНИЯ: НЕДОСТАТКИ.

- Две кодовые базы: поддерживать две команды разработчиков.
- Дорого.
- Необходимо "выравнивать" приложения по функциональности.

ANDROID

- Основные компоненты приложений.
- Многопоточность в приложениях.
- Сеть.
- Adapter Views. Службы (Services).
- Работа с данными и файлами.
- Локализация и работа с ресурсами.

ИНСТРУМЕНТАРИЙ

- JDK
- Android SDK
- Android Studio
- Gradle
- Android смартфон

JAVA

- Текущая версия 24
- Java Runtime Enviroment (JVM + java, выполнение программ)
- Java Development Kit (JRE + средства компиляции и отладки)
- Проект OpenJDK - исходный код JRE/JDK
- Сборки и сертификация.

JAVA

- Один из поставщиков и сборщиков JRE/JDK - Oracle.
- Сборка OpenJDK от Oracle (обновления прекращаются как только выходит новая версия).
- OracleJDK, бесплатно на этапе разработки, платно в рабочей среде.
- Сборка AdoptOpenJDK: длительная поддержка, выбор JVM.

JVM

- Запускать приложения на Java.
- Управлять и оптимизировать память.
- Реализации: HotSpot, OpenJ9.
- Scala, Groovy и Kotlin

ПРОГРЕССИВНОЕ ВЕБ-ПРИЛОЖЕНИЕ.

- Трансформация сайта в веб-приложение.
- Обычно на отдельном домене (определение по User-Agent).
- Не нужно устанавливать отдельное приложение.
- Нет доступа к функциям ОС.
- Нет возможности использовать другие приложения (например, для входа).
- Браузер для ОС достаточно много потребляет энергии.

ОБЩАЯ АРХИТЕКТУРА.

REACT NATIVE.

- Ресурс: https://reactnative.dev/
- Появился в 2015.
- Веб-фрейморк: React (JSX).
- Могут быть отличия ios/android.
- Поддерживается механизм hot-reload.

REACT NATIVE.

- React (также известный как ReactJS) - это библиотека , используемая для создания интерфейса веб-сайта.
- Как React, так и React Native используют смесь JavaScript и специального языка разметки JSX.
- Однако синтаксис, используемый для отображения элементов в компонентах JSX, отличается между React и React Native.
- Кроме того, React использует некоторые HTML и CSS, в то время как React Native позволяет использовать собственные элементы мобильного пользовательского интерфейса.

JSX

- JSX – это синтаксическая обертка над JS.
- Его нет в стандартах ECMAScript, так что инструменты вроде Babel и React занимаются его транспиляцией в обычный JavaScript код.

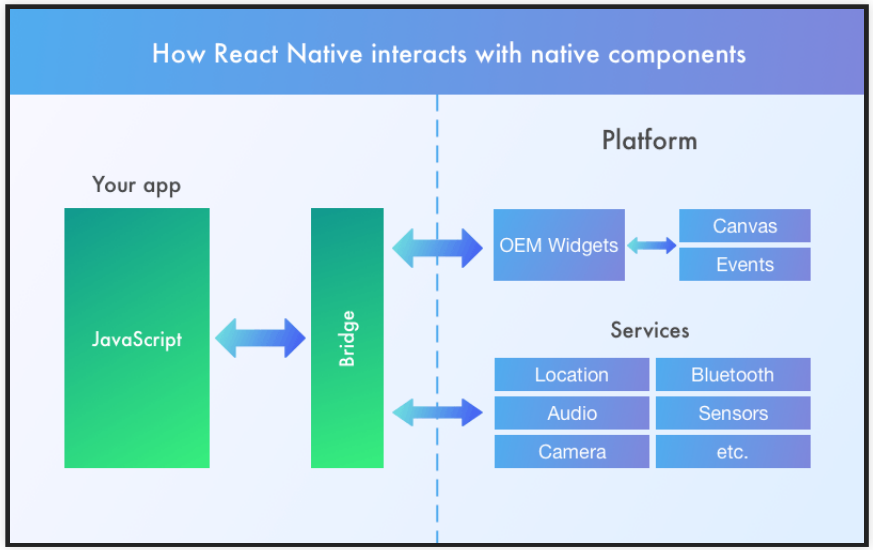
```
const profile = (
  <div>
    <img src="avatar.png" className="profile" />
    <h3>{[user.firstName, user.lastName].join(" ")}</h3>
  </div>
);
```

```
const profile = React.createElement(
  "div",
  null,
  React.createElement("img", { src: "avatar.png", className: "profile" }),
  React.createElement("h3", null, [user.firstName, user.lastName].join(" "))
);
```

TYPESCRIPT

- Является обратно совместимым с JavaScript и компилируется в последний.
- Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js.
- TypeScript отличается от JavaScript возможностью явного статического назначения типов, поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках).
- TypeScript является строго типизированным языком, и каждая переменная и константа в нем имеет определенный тип.
- При этом в отличие от javascript мы не можем динамически изменить ранее указанный тип переменной.

REACT NATIVE. АРХИТЕКТУРА.



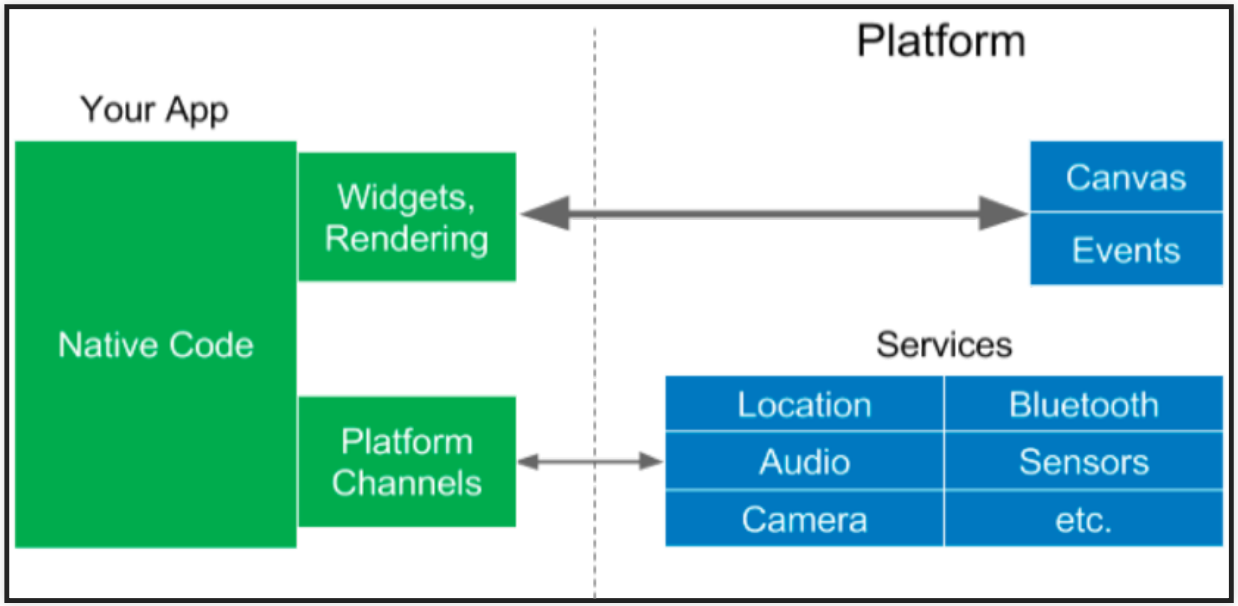
REACT NATIVE. ПРИМЕР.

```
import React from 'react';
import { Text, View } from 'react-native';

const YourApp = () => {
  return (
    <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
      <Text>
        Some text
      </Text>
    </View>
  );
}

export default YourApp;
```

FLUTTER.

- Ресурс: https://flutter.dev
- Первый релиз 2018 год.
- Язык программирования Dart (компилируется в машинный код).
- Набор виджетов: Material для Android и Cupertino для iOS.
- Взаимодействие с платформой через Platform Channels.
- Свой CI/CD: https://codemagic.io/
- Считается на данный момент самым перспективным.

FLUTTER. АРХИТЕКТУРА.



FLUTTER. DART.

- Ресурс: https://dart.dev
- AOT-компиляция. В результате имеем ARM или x86 код с высоким быстродействием на пользовательском устройстве.
- JIT-компиляция. Дает возможность запускать проект в виртуальной машине на эмуляторе и реализовать hot reload длительностью менее одной секунды с сохранением состояния приложения.
- Строгая система типов. Позволяет забыть о выборе между TS и Flow.
- Tree shaking compiler. При сборке приложения удаляет весь неиспользуемый код.
- Generational garbage collection. Быстрое выделение и очистка памяти.

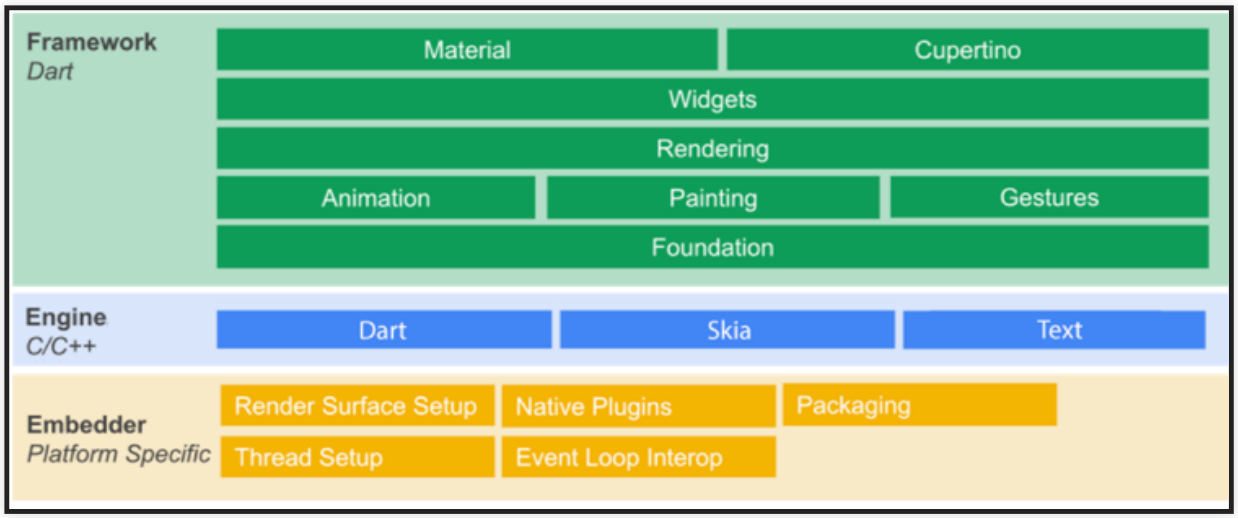
AOT И JIT КОМПИЛЯЦИИ

- Компилятор JIT запускается во время выполнения программы, компилируя ее на лету.
- Первоначальные компиляторы, которые выполняются во время создания программы (до выполнения), называются опережающими компиляторами (AOT).
- Только статические языки поддаются компиляции AOT в собственный машинный код, потому что машинным языкам обычно необходимо знать тип данных
- Следовательно, динамические языки обычно интерпретируются или компилируются JIT.

AOT И JIT В DART

- Поддерживает два вида компиляции AOT и JIT
- Компиляция JIT используется во время разработки с использованием особенно быстрого компилятора.
- Затем, когда приложение готово к выпуску, оно компилируется AOT.
- быстрые циклы разработки и быстрое время выполнения и запуска.
- Dart также предоставляет автономную виртуальную машину, которая использует сам язык Dart в качестве своего промежуточного языка (по сути, действуя как интерпретатор).

DART. АРХИТЕКТУРА.



DART. УПРЕЖДАЮЩЕЕ ПЛАНИРОВАНИЕ, РАЗДЕЛЕНИЕ ВРЕМЕНИ И ОБЩИЕ РЕСУРСЫ

- Большинство компьютерных языков, поддерживающих несколько параллельных потоков выполнения (включая Java, Kotlin, Objective-C и Swift), используют вытеснение для переключения между потоками.
- Каждому потоку выделяется "кусочек" времени для выполнения, и если он превышает выделенное время, поток прерывается с помощью переключения контекста.
- Однако если вытеснение происходит при обновлении ресурса, совместно используемого потоками (например, памяти), это вызывает состояние гонки.
- способ исправить состояние гонки — защитить общий ресурс с помощью блокировки

DART. УПРЕЖДАЮЩЕЕ ПЛАНИРОВАНИЕ, РАЗДЕЛЕНИЕ ВРЕМЕНИ И ОБЩИЕ РЕСУРСЫ

- Потоки в Dart, называемые изолятами, не используют общую память, что устраняет необходимость в большинстве блокировок.
- Изоляты общаются, передавая сообщения по каналам, что похоже на акторов в Erlang или web-workers в JavaScript.
- Dart, как и JavaScript, является однопоточным, что означает, что он вообще не допускает вытеснения.
- Вместо этого потоки явно уступают (используя async/await, Futures или Streams).
- Однопоточность помогает разработчику гарантировать, что критически важные функции (включая анимацию и переходы) выполняются полностью, без вытеснения.

DART. СТАТИЧЕСКИЕ АНАЛИЗ КОДА

- Линтер - это инструмент для статического анализа кода, который находит и уведомляет об ошибках, предупреждениях, а также нарушениях стиля кода, чтобы разработчик их исправил.
- В случае использования Flutter, это один из наиболее простых и полезных пунктов, чтобы сохранять код чистым.
- lint: https://pub.dev/packages/lint
- flutter_lints
- very_good_analysis

FLUTTER. ЛОКАЛИЗАЦИЯ

- Встроенная поддержка
- Создаются файлы .arb для каждого языка, который будет в приложении.
- Создаваемые файлы должны быть в папке l10n.
- У файлов должно быть расширение .arb.
- Название ключей должно начинаться с маленьких букв — специальные символы запрещены.
- Не ставьте запятую в конце последней пары «ключ-значение».

DART. ПРИМЕР.

```
import 'package:flutter/material.dart';

void main() async {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        body: MyApp(),
      ),
    ),
  );
}

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}
```

FLUTTER. "ЗА".

- Собственный рендеринг, независимость от OEM-виджетов и богатый инструментарий разработчика существенно ускоряет написание кода и поддержку двух платформ одновременно.
- Web и desktop — в активной разработке.
- За счет pixel perfect rendering, тестирование и устранение платформо-зависимых багов и нюансов верстки будет проходить существенно быстрее.
- Быстродействие.
- Одинаковый UI на всех устройствах. При необходимости написать приложение с уникальным дизайном, который должен быть одинаковым на двух мобильных платформах.

FLUTTER. "ПРОТИВ".

- Молодость платформы.
- Dart добавляет накладные затраты на внедрение Flutter за счет необходимости изучать и вникать в новый язык.
- В случае с React Native, они могут быть подключены в проект достаточно быстро. Во Flutter все новые появляющиеся на стороне платформы элементы UI должны быть заново отрисованы.