1. Кроссплатформенная разработка мобильных приложений. Определение, преимущества и недостатки.

Кросплатформенность – пособностьпрограммного обеспеченияработать с двумя и болееаппаратными платформамии (или)операционными системами

Плюсы кроссплатформенности:

•Единая логика приложения –логика приложения будет одинаково работать для всех платформ.

•Разработка кроссплатформенных приложений экономически эффективна

•Простое и быстрое развертывание

•Кроссплатформенные приложения покрывают более широкую аудиторию

•Кроссплатформенные приложения допускают одинаковый интерфейс и UX

•Поддержка и обновление продукта –добавление функционала или исправление ошибок сразу для всех платформ;

Минусы кроссплатформенности:

•Кроссплатформенные приложения не являются такими гибкими, как нативныеприложения

•Кроссплатформенные приложения не работают так же хорошо, как нативныеприложения

•Возможное несоответствие UI в различных платформах

•Отправка кроссплатформенных приложений в соответствующие Магазины приложений может иметь сложности.

1. Характеристика языка программирования Dart. Преимущества и недостатки.

Dart представляет язык программирования общего назначения от компании Google, который предназначен прежде всего для разработки прикладных приложений. Основной сферой применения языка Dart является разработка графических приложений с помощью кроссплатформенного фреймворка Flutter.

-объектно-ориентированный язык программирования общего назначения.

-С-подобный синтаксис

-Dart VM

-JIT и AOT, dart2js

-Hot Reload

Особенности языка:

1. Совмещение JIT и AOT компиляции. Just-in-time компиляция предполагает трансляцию машинного кода “на лету”. Это дает определенные преимущества непосредственно во время разработки - тестирование и отладка происходят быстрее, правки вносятся оперативнее. Правда, при использовании готового приложения на практике оно будет загружаться слишком. Ahead-of-time компилятор сначала обрабатывает код, и только потом начинается само выполнение программы. Это замедляет разработку, но приложение будет более “шустрым”. Dart успешно применяет обе концепции, сочетая их плюсы и предоставляя максимальную производительность.
2. Промежуточная компиляция. Программу на Dart с помощью виртуальной машины без труда можно скомпилировать в JavaScript для выполнения в браузере. Это повышает кроссплатформенность кода
3. Опциональные типы. В Dart можно как строго указывать типы, так и отдавать их определение на усмотрение компилятора.
4. Асинхронность. В Дарт предусмотрено асинхронное выполнение операций. Благодаря фиче Event Loop обработка “тяжелой” части кода временно откладывается, и блокировки потока не происходит.
5. Основные типы данных в языке программирования Dart. Определение и назначение типов данных: Number, String, Boolean, List, Set, Map, Rune and Grapheme Cluster, Symbol.
6. Структура класса в языке программирования Dart. Типы наследования. Определение, назначение и пример использования Dart Interface.

Наследование позволяет передавать одним классам функционал других. В языке Dartнаследование реализуется с помощью ключевого словаextends(как в Java)

class Employee extends Person{ /\*…\*/ }

Поля и методы базового класса наследуются, а конструкторы нет. Если базовый класс явным образом определяет конструктор, то его необходимо вызвать в классе-наследнике при определении конструктора

class Employee extends Person{ Employee(name) : super(name); }

Классы-наследники могут определять свои поля и методы, но также могут переопределять, изменять поведение методов базового класса. Для этого применяется аннотация @override

1. Определение, назначение и пример использования Dart Mixin.
2. Функции в языке программирования Dart. Типы параметров и функций.
3. Определение, назначение и пример использования Dart Future.
4. Определение, назначение и пример использования Dart Stream.
5. Определение Dart Isolate. Алгоритм создания и взаимодействия между несколькими изолятами.
6. Характеристика фреймворка Flutter. Архитектура фреймворка.
7. Описаний организации структуры проекта на Flutter.
8. Понятия, назначение и пример использования Flutter Widget и Flutter State.
9. Понятия StatelessWidget и StatefulWidget в Flutter. Отличия и примеры использования.
10. Понятия Widget Tree, Element Tree и Render Tree в Flutter. Отличия и назначение.
11. Определение и особенности Single-child layout Widgets в Flutter. Виджеты: Align, AspectRatio, Baseline, Center, ConstrainedBox, Container, MediaQuery.
12. Определение и особенности Single-child layout Widgets в Flutter. Виджеты: Expanded, Padding, SizedBox, Transform, OverflowBox, FittedBox.
13. Определение и особенности Multi-child layout Widgets в Flutter. Виджеты: Column, GridView, Stack, IndexedStack.
14. Определение и особенности Multi-child layout Widgets в Flutter. Виджеты: Row, ListView, ListTile, Table, Wrap.
15. Определение и особенности Component Widgets в Flutter. Виджеты: AppBar, RaisedButton, Scaffold, FloatingActionButton.
16. Определение и особенности Component Widgets в Flutter. Виджеты: Drawer, PopupButton, Date&Time PickerDialog.
17. Определение и особенности Component Widgets в Flutter. Виджеты: AlertDialog, BottomSheet, ExpansionPanel, SimpleDialog, SnackBar.
18. Определение и особенности Component Widgets в Flutter. Виджеты: Card, Divider, Stepper, Chip.
19. Взаимодействие виджетов в Flutter с пользователем. Виджеты: Dismissible, DragTarget, Draggable, GestureDetector, LongPressDraggable.
20. Понятие и назначение библиотеки Cupertino в Flutter. Виджеты: CupertinoActionSheet, CupertinoSegmentedControl, CupertinoPicker.
21. Понятие и назначение библиотеки Cupertino в Flutter. Виджеты: CupertinoPageTransition, CupertinoPageScaffold, CupertinoNavigationBar.
22. Понятие и назначение библиотеки Cupertino в Flutter. Виджеты: CupertinoDialog, CupertinoButton, CupertinoContextMenu, CupertinoDatePicker, CupertinoTimerPicker.
23. Понятие и назначение библиотеки Cupertino в Flutter. Виджеты: CupertinoTabScaffold, CupertinoTabView, CupertinoTextField, CupertinoButton.
24. Навигация в приложении Flutter. Определение и назначение Navigator, MaterialPageRoute, CupertinoPageRoute.
25. Понятие Flutter Platform Channel. Процесс создания и способы применения.
26. Алгоритм работы с файловой системой в Flutter.
27. Назначение библиотек Sqflite и SharedPreference в Flutter.
28. Принцип обработки разрешений в приложениях Flutter. Инструменты для управления разрешениями.
29. Механизм отправки и обработки уведомлений в Flutter. Способы и инструменты.
30. Понятие Firebase. Виды и назначение сервисов Firebase, доступные для интеграции с Flutter.
31. Понятия FutureBuilder и StreamBuilder в Flutter. Принципы работы и примеры использования.
32. Структура и назначение архитектурного шаблона Bloc в Flutter.
33. Преимущества использования пакета Provider в Flutter. Применение Provider для управления состоянием.
34. Принцип использования библиотеки Hive для локального хранения данных в приложениях Flutter.
35. Определение и назначение юнит-тестов в Flutter. Роль библиотеки Mockito в тестировании.
36. Тестирование виджетов в Flutter. Методы поиска и проверки конкретных виджетов в приложении.
37. Интеграционное тестирование в приложениях Flutter. Инструменты для проверки взаимодействия компонентов.