**Практическое занятие 6.1: Создание нового проекта мобильного приложения**

**Цель занятия:**

Научиться создавать новый проект мобильного приложения на платформе Flutter, настроить среду разработки и запустить базовое приложение.

**Оборудование и программное обеспечение:**

1. Компьютер с доступом в интернет
2. Установленная операционная система (Windows, macOS или Linux)
3. Установленный Flutter SDK
4. Установленная Visual Studio Code (VS Code) или другая IDE, поддерживающая Flutter
5. Установленный Android Studio

**Подготовка:**

1. Убедитесь, что Flutter SDK и необходимые инструменты (например, Android Studio) установлены и настроены. Проверьте это с помощью команды flutter doctor.
2. Убедитесь, что Visual Studio Code или другая поддерживаемая IDE установлена.

**Контрольные вопросы:**

1. Какую команду нужно использовать для создания нового Flutter проекта?
2. Какие команды помогают в проверке и анализе кода проекта?
3. Как собрать APK файл из Flutter проекта?
4. Для чего используется команда flutter clean?

**Теоретическая часть**

Создание нового проекта мобильного приложения на Flutter включает в себя несколько ключевых этапов: от установки необходимых инструментов и настройки среды разработки до создания проекта и первого запуска приложения. В этой теоретической части мы подробно рассмотрим каждый шаг процесса создания нового проекта на Flutter, а также объясним основные концепции и инструменты, которые будут использоваться.

**1. Flutter: Основные понятия**

**Flutter** — это UI toolkit от Google, предназначенный для создания высококачественных нативных приложений для мобильных, веб- и настольных платформ из единого кода. Он использует язык программирования Dart и предоставляет богатый набор виджетов для создания пользовательского интерфейса.

**Dart** — это объектно-ориентированный язык программирования, на котором основана работа Flutter. Он отличается простым синтаксисом и современными возможностями, что делает его удобным для разработки приложений.

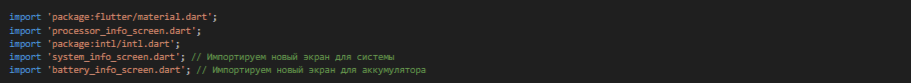
**Практическая часть**

1. **Изменение приложения**

1. **Откройте файл lib/home\_screen.dart**

2. **Внесите изменения в код home\_screen.dart**

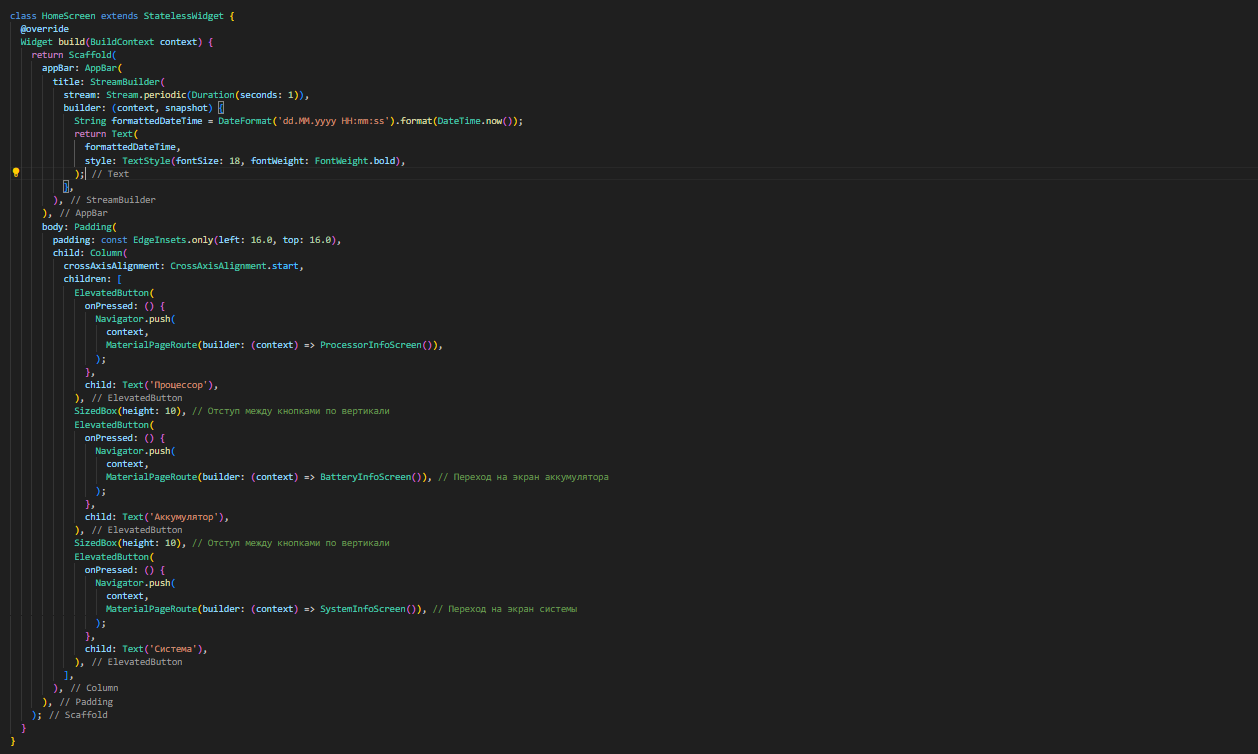
Файл lib/home\_screen.dart представляет собой экран домашней страницы приложения, который предоставляет пользователю несколько кнопок для перехода на различные экраны с информацией о процессоре, аккумуляторе и системе. Давайте подробно разберем содержимое этого файла.



**Импорты**

1. **import 'package:flutter/material.dart';**
   1. Импортирует основной пакет Flutter для работы с виджетами и материал-дизайном.
2. **import 'processor\_info\_screen.dart';**
   1. Импортирует экран для отображения информации о процессоре.
3. **import 'package:intl/intl.dart';**
   1. Импортирует пакет для интернационализации и форматирования дат и времени.
4. **import 'system\_info\_screen.dart';**
   1. Импортирует экран для отображения информации о системе.
5. **import 'battery\_info\_screen.dart';**
   1. Импортирует экран для отображения информации об аккумуляторе.

**Класс HomeScreen**



**Основные компоненты и их функции**

1. **Класс HomeScreen**:
   1. Наследует StatelessWidget, что означает, что этот виджет не имеет внутреннего состояния и не меняется после создания.
2. **Метод build**:
   1. Создает пользовательский интерфейс для экрана домашней страницы.
3. **Scaffold**:
   1. Обеспечивает структуру для визуального оформления экрана, включая AppBar, body, и другие элементы интерфейса.
4. **AppBar**:
   1. Верхняя панель приложения, содержащая текущую дату и время.
5. **StreamBuilder**:
   1. Виджет, который строит пользовательский интерфейс на основе данных из Stream. В данном случае используется для обновления времени на экране каждую секунду.
   2. Stream.periodic(Duration(seconds: 1)) создает поток, который эмитирует событие каждую секунду.
   3. builder метод форматирует текущее время в строку формата dd.MM.yyyy HH:mm:ss и отображает ее в Text виджете.
6. **Padding**:
   1. Оборачивает основной контент экрана с отступами от краев.
7. **Column**:
   1. Располагает дочерние виджеты по вертикали.
8. **ElevatedButton**:
   1. Кнопка с поднятым эффектом. При нажатии выполняет действие перехода на другой экран.
   2. **Кнопка "Процессор"**:
      1. При нажатии вызывает Navigator.push для перехода на экран ProcessorInfoScreen.
   3. **Кнопка "Аккумулятор"**:
      1. При нажатии вызывает Navigator.push для перехода на экран BatteryInfoScreen.
   4. **Кнопка "Система"**:
      1. При нажатии вызывает Navigator.push для перехода на экран SystemInfoScreen.
9. **SizedBox**:
   1. Виджет для создания отступов между элементами.

Файл home\_screen.dart определяет главный экран приложения, который отображает текущее время и предоставляет кнопки для перехода к экранам с информацией о процессоре, аккумуляторе и системе. Этот экран использует StreamBuilder для обновления текущего времени каждую секунду и Navigator для навигации между экранами.

Далее давай создадим файл processor\_info\_screen.dart реализует экран, который отображает информацию о процессоре устройства Android.



**Импорты**

1. **import 'dart:io';**
   1. Импортирует библиотеку Dart для работы с файловой системой и выполнения команд процесса.
2. **import 'package:flutter/material.dart';**
   1. Импортирует пакет Flutter для создания UI-компонентов.
3. **import 'package:device\_info\_plus/device\_info\_plus.dart';**
   1. Импортирует пакет для получения информации об устройстве.

**Класс ProcessorInfoScreen**

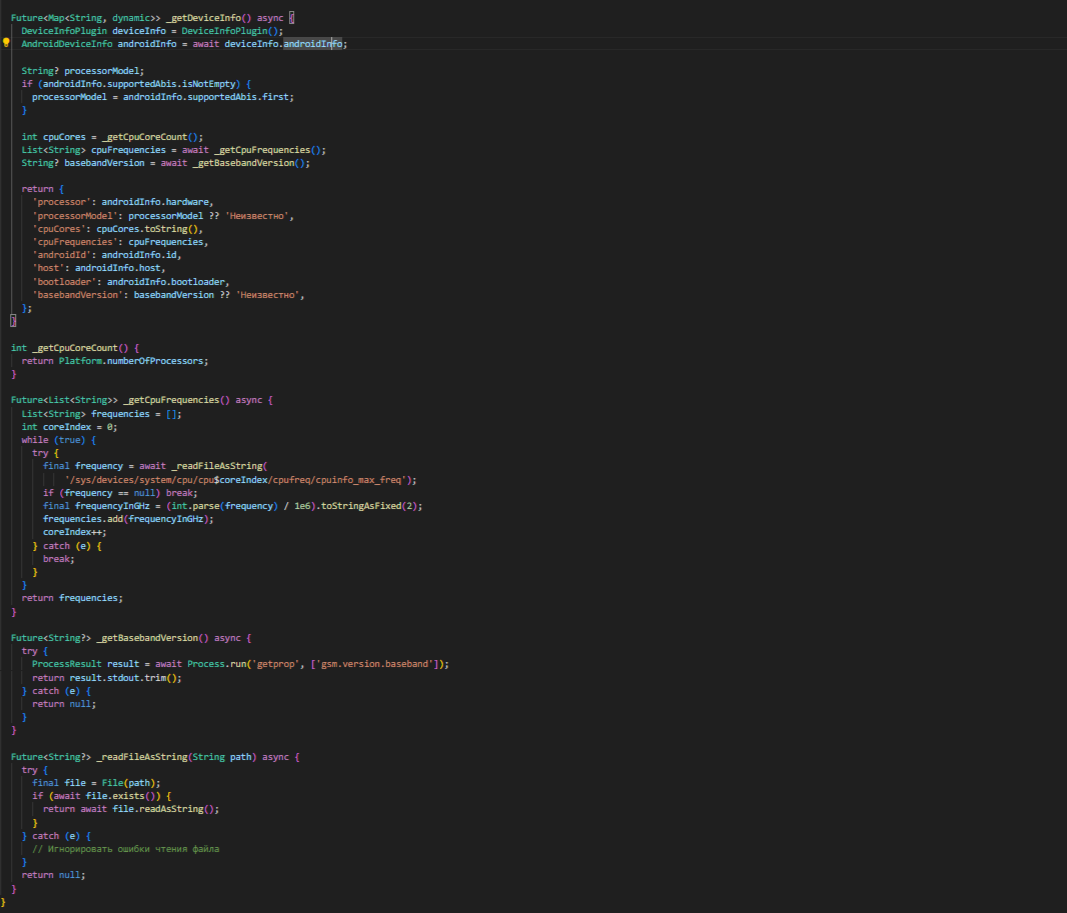


**ProcessorInfoScreen**:

1. Виджет экрана, использующий состояние (StatefulWidget), поскольку информация может изменяться.







**Обзор основных компонентов**

1. **\_ProcessorInfoScreenState**:
   1. Содержит реализацию состояния для виджета ProcessorInfoScreen.
2. **Метод build**:
   1. Строит пользовательский интерфейс для экрана.
   2. **Scaffold**: Создает структуру экрана с AppBar и основным содержимым.
   3. **AppBar**: Панель с заголовком и кнопкой "Назад".
   4. **FutureBuilder**: Используется для асинхронного получения данных устройства и их отображения:
      1. **future**: Указывает метод \_getDeviceInfo(), который возвращает Future с информацией об устройстве.
      2. **builder**: Отображает данные в зависимости от состояния snapshot:
         1. **ConnectionState.waiting**: Пока данные загружаются, показывается индикатор загрузки.
         2. **hasError**: При ошибке отображается сообщение об ошибке.
         3. **hasData**: Отображает данные о процессоре, модели, количестве ядер и частотах.
3. **Методы**:
   1. **\_getDeviceInfo()**:
      1. Использует DeviceInfoPlugin для получения информации о устройстве.
      2. Получает информацию о процессоре, модели, количестве ядер, частотах, идентификаторе устройства и других параметрах.
   2. **\_getCpuCoreCount()**:
      1. Возвращает количество процессорных ядер, используя Platform.numberOfProcessors.
   3. **\_getCpuFrequencies()**:
      1. Асинхронно получает частоты процессоров из файловой системы.
      2. Считывает частоту каждого ядра из соответствующего файла.
   4. **\_getBasebandVersion()**:
      1. Получает версию базовой полосы через команду getprop.
   5. **\_readFileAsString()**:
      1. Читает содержимое файла по указанному пути и возвращает его в виде строки.

Файл lib/processor\_info\_screen.dart реализует экран, который предоставляет подробную информацию о процессоре устройства Android, используя пакет device\_info\_plus и доступ к файловой системе устройства. Это полезно для приложений, которым нужно предоставлять пользователям информацию о технических характеристиках устройства.

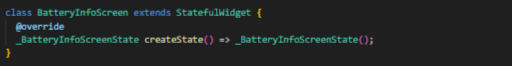
Далее создадим файл battery\_info\_screen.dart реализует экран, отображающий информацию об аккумуляторе устройства Android.



**Импорты**

1. **import 'package:flutter/material.dart';**
   1. Импортирует пакеты Flutter для создания пользовательского интерфейса.
2. **import 'package:battery\_plus/battery\_plus.dart';**
   1. Импортирует пакет для получения информации о заряде аккумулятора и его состоянии.
3. **import 'package:battery\_info/battery\_info\_plugin.dart';**
   1. Импортирует пакет для получения дополнительной информации об аккумуляторе на платформе Android.
4. **import 'package:battery\_info/model/android\_battery\_info.dart';**
   1. Импортирует модели данных, используемые для работы с информацией об аккумуляторе на Android.

**Класс BatteryInfoScreen**



**BatteryInfoScreen**:

1. Виджет экрана, использующий состояние (StatefulWidget), чтобы обновлять информацию об аккумуляторе в реальном времени.



1. **\_BatteryInfoScreenState**:

1. Хранит состояние для BatteryInfoScreen:
   1. **\_battery**: Экземпляр Battery для доступа к информации о заряде и состоянии аккумулятора.
   2. **\_batteryLevel**: Хранит уровень заряда аккумулятора.
   3. **\_batteryState**: Хранит состояние аккумулятора.
   4. **\_batteryTemperature**: Хранит температуру аккумулятора.
   5. **\_batteryVoltage**: Хранит напряжение аккумулятора.

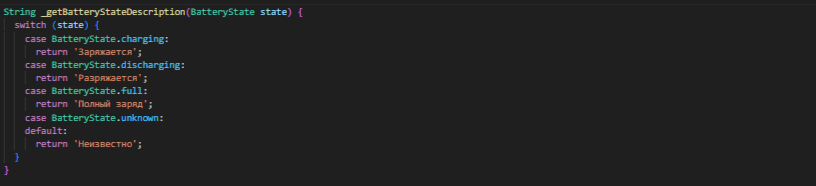
2. **initState()**:

1. Инициализирует состояние виджета:
   1. Вызов \_getBatteryInfo() для получения начальной информации об аккумуляторе.
   2. Подписка на изменения состояния аккумулятора через Battery.onBatteryStateChanged.



**\_getBatteryInfo()**:

1. Асинхронно получает информацию о батарее:
   1. Уровень заряда через Battery.batteryLevel.
   2. Состояние аккумулятора через Battery.batteryState.
   3. Дополнительная информация (температура и напряжение) через BatteryInfoPlugin().androidBatteryInfo.
2. Обновляет состояние виджета с полученной информацией.



**\_getBatteryStateDescription()**:

1. Возвращает строковое описание состояния аккумулятора на основе его состояния (BatteryState).



1. **Метод build**:
   1. Создает пользовательский интерфейс для экрана:
      1. **Scaffold**: Обеспечивает структуру экрана с AppBar и основным содержимым.
      2. **AppBar**: Панель с заголовком.
      3. **Padding**: Добавляет отступы вокруг содержимого.
      4. **Column**: Организует текстовые элементы в вертикальном порядке.
      5. **Text**: Отображает уровень заряда, состояние аккумулятора, температуру и напряжение.
      6. **SizedBox**: Добавляет вертикальные отступы между элементами.

Файл lib/battery\_info\_screen.dart реализует экран для отображения информации о заряде, состоянии, температуре и напряжении аккумулятора устройства. Он использует пакеты battery\_plus и battery\_info\_plugin для получения информации и отображения её в удобном формате. Вы также используете поток onBatteryStateChanged для обновления состояния аккумулятора в реальном времени.

Теперь создадим файл system\_info\_screen.dart он представляет собой экран, на котором отображается информация о системе и устройстве, используя различные пакеты и методы для сбора данных.

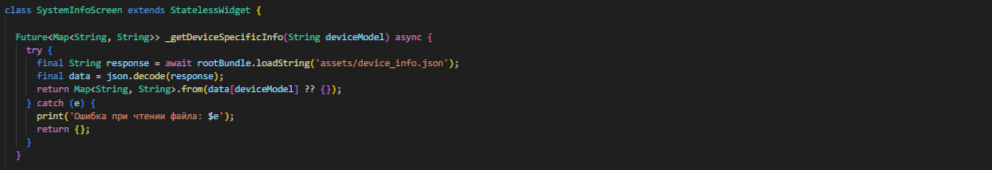
Файл содержит один виджет SystemInfoScreen, который является StatelessWidget. Он извлекает и отображает информацию о системе и устройстве, такую как память, хранилище и спецификации устройства.



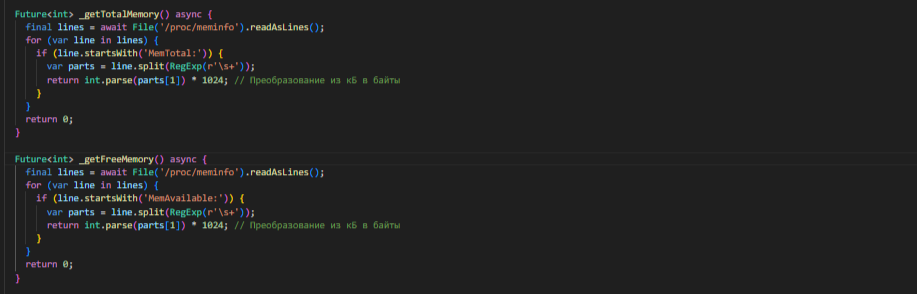
1. **flutter/material.dart**: Пакет Flutter для создания пользовательского интерфейса.
2. **device\_info\_plus/device\_info\_plus.dart**: Пакет для получения информации о устройстве.
3. **dart:io**: Пакет для работы с файловой системой и процессами.
4. **dart:convert**: Пакет для кодирования и декодирования JSON.
5. **flutter/services.dart**: Пакет для доступа к ресурсам пакета и платформенным каналам.

**Методы для получения информации**

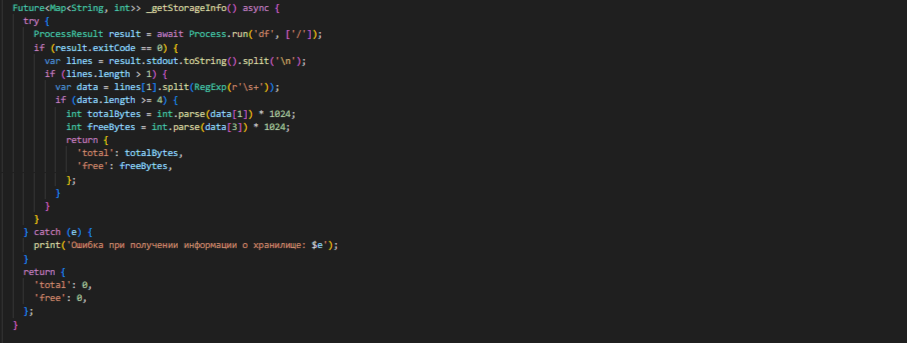
1. **\_getDeviceSpecificInfo(String deviceModel)**: Загружает и декодирует информацию о устройстве из файла device\_info.json, который должен находиться в папке assets.



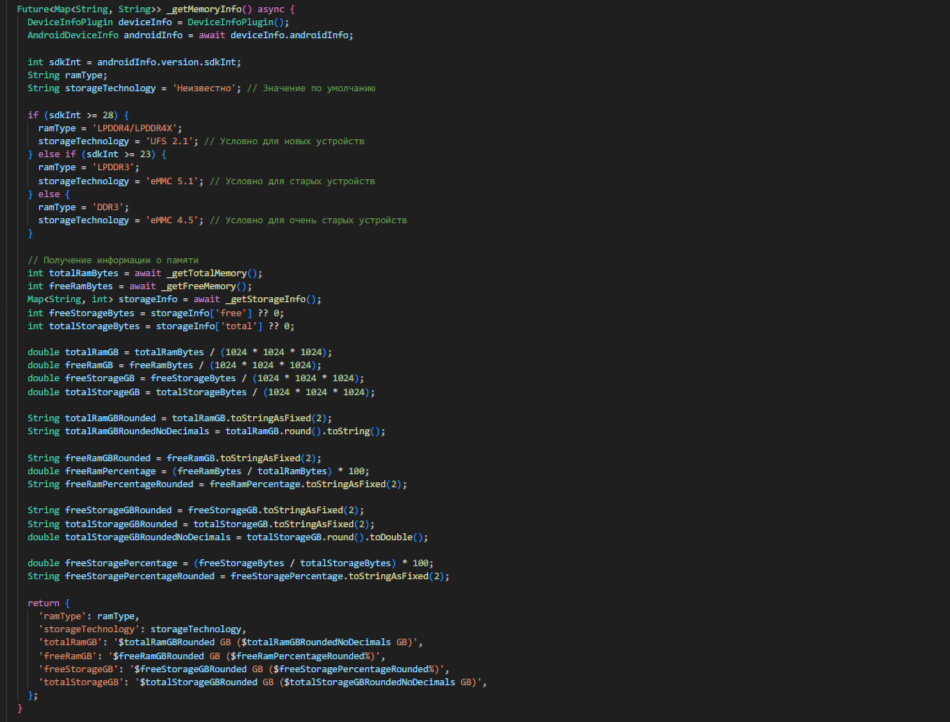
1. **\_getTotalMemory()** и **\_getFreeMemory()**: Читают информацию о памяти из /proc/meminfo. Эти методы доступны только на платформе Android и возвращают объем общей и свободной памяти в байтах.



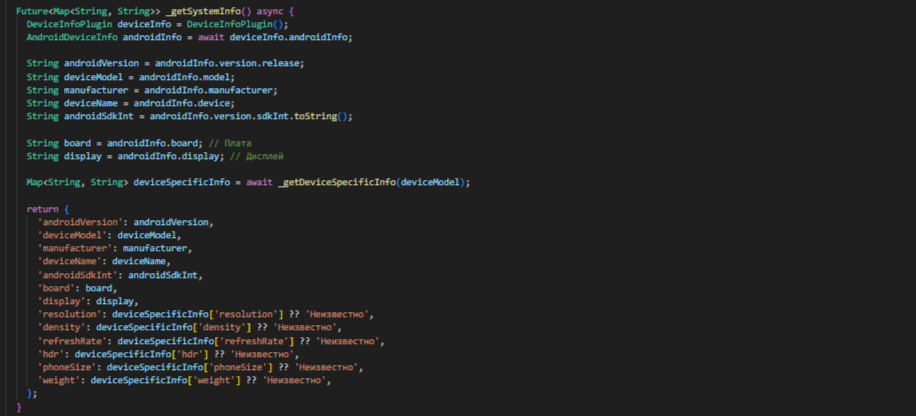
1. **\_getStorageInfo()**: Использует команду df для получения информации о хранилище устройства.



1. **\_getMemoryInfo()**: Собирает информацию о памяти, используя методы \_getTotalMemory(), \_getFreeMemory() и \_getStorageInfo(). Также определяет тип RAM и технологию хранения на основе версии SDK Android.

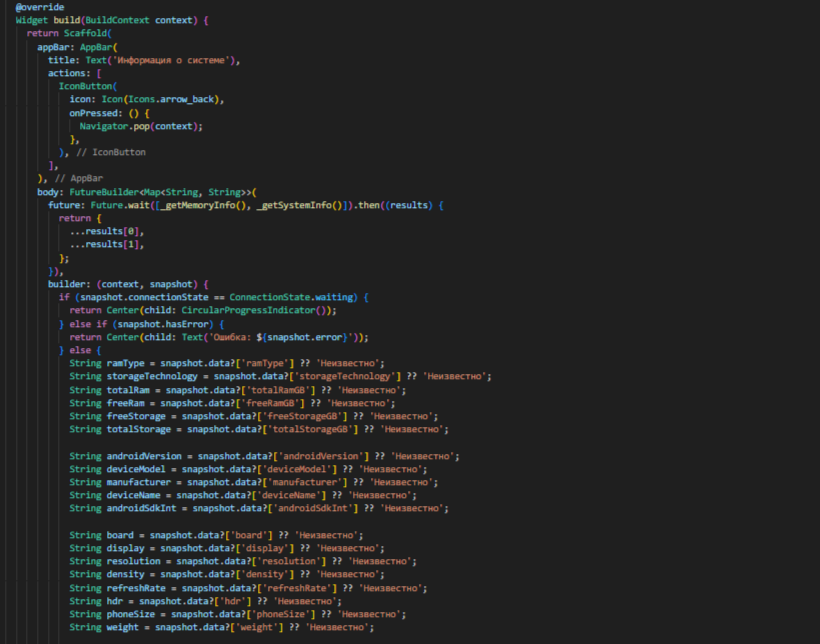


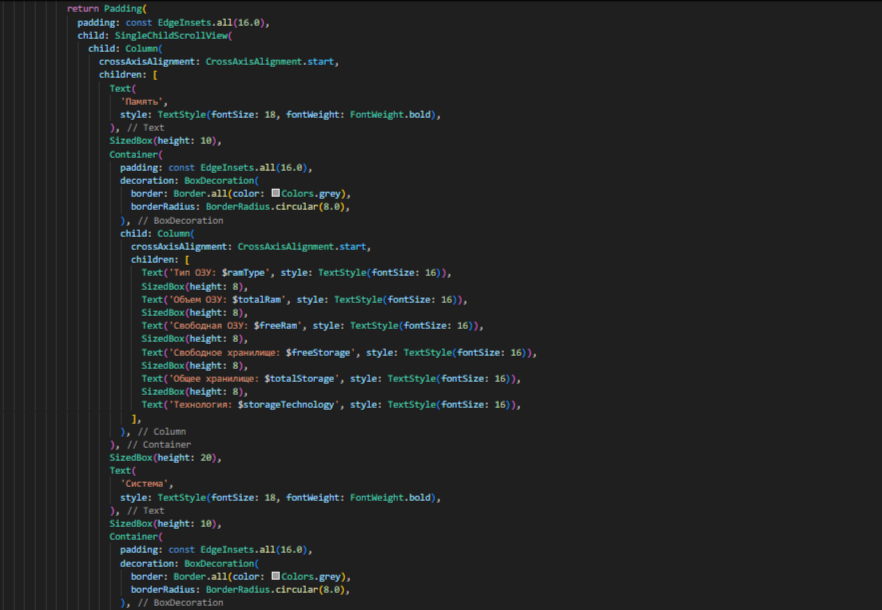
1. **\_getSystemInfo()**: Получает общую информацию о системе и устройстве, используя DeviceInfoPlugin.

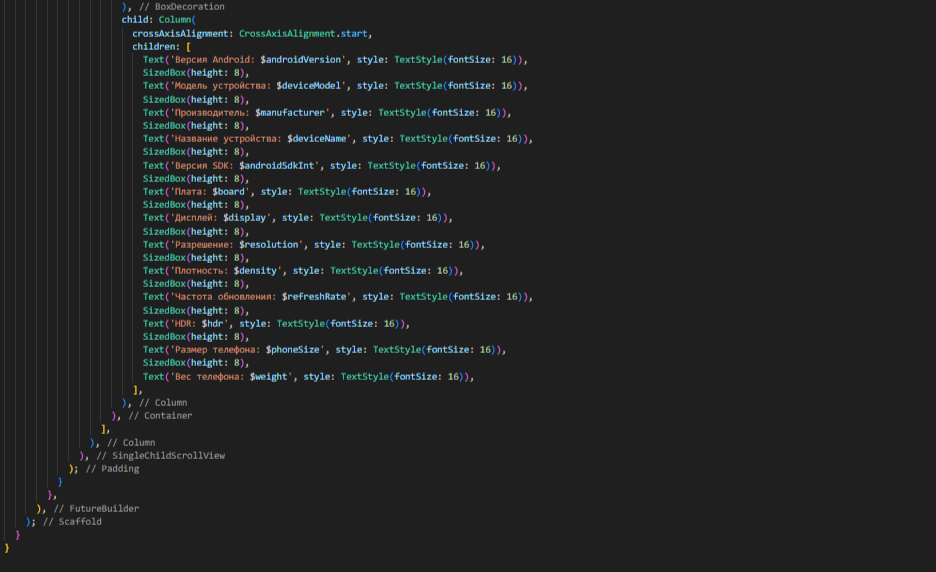


**Метод build**

Метод build создает интерфейс, который отображает собранную информацию о памяти и системе:







1. **Платформенные ограничения**: Некоторые методы (например, доступ к /proc/meminfo и df) работают только на Android-устройствах. Учитывайте это в приложении и, возможно, добавить проверку платформы, если приложение должно работать и на других платформах.

2. **Обработка ошибок**: Правильно обрабатывать возможные ошибки, особенно при работе с файловой системой и внешними командами.

3. **Форматирование данных**: Данные о памяти и хранилище в гигабайтах и округляете их до двух десятичных знаков. Это делает вывод более читабельным.

Так же нам нужно создать файл JSON. В папке assets добавим файл device\_info.json он содержит информацию о различных моделях устройств, которую можно использовать для отображения специфической информации на экране вашего приложения.



**Как интегрировать JSON-файл**

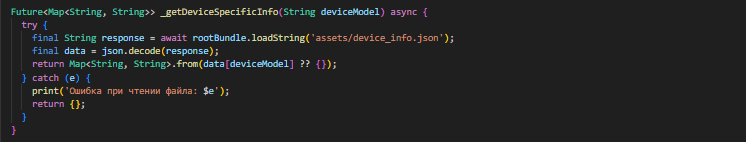
1. **Загрузите JSON-файл в assets**: Убедитесь, что ваш файл device\_info.json добавлен в секцию assets вашего pubspec.yaml:

flutter:

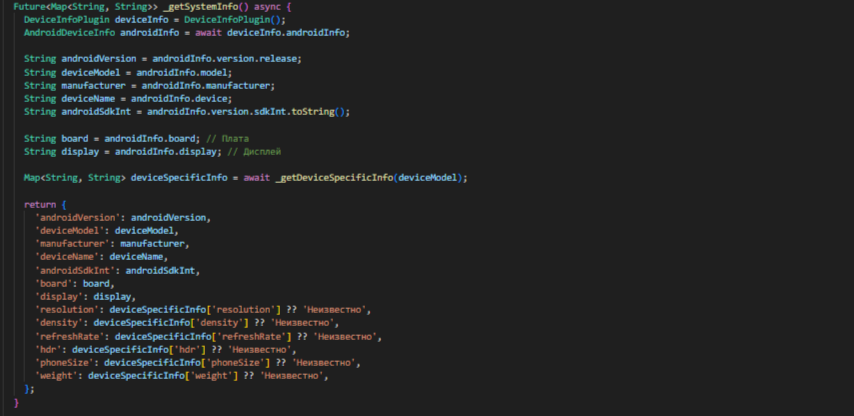
assets:

- assets/device\_info.json

**Чтение JSON-файла**: Метод \_getDeviceSpecificInfo уже правильно читает JSON-файл и ищет информацию по модели устройства. Однако стоит удостовериться, что модель устройства в JSON-файле точно соответствует модели устройства, возвращаемой deviceInfo.androidInfo.model.



**Использование данных из JSON-файла**: В методе \_getSystemInfo() правильно вызывается \_getDeviceSpecificInfo для получения информации о текущем устройстве. Убедитесь, что информация о модели устройства корректно используется для получения данных:



Так же нужно добавить в pubspec.yaml пакеты и json файл  


**dependencies**

1. **flutter:**
   1. Указывает на использование SDK Flutter для приложения. Это обязательная зависимость для любого проекта Flutter.
2. **system\_info2: ^4.0.0**
   1. Пакет для получения информации о системе, такой как операционная система, версия и архитектура.
3. **battery\_plus: ^6.0.2**
   1. Пакет для получения информации о состоянии батареи устройства. Например, уровень заряда батареи.
4. **battery\_info: ^1.1.1**
   1. Пакет для получения подробной информации о батарее устройства. Он может предоставлять такие данные, как состояние батареи и уровень заряда.
5. **path\_provider: ^2.1.4**
   1. Пакет для получения путей к директориям и файлам на устройстве. Это полезно для сохранения данных и доступа к файловой системе.
6. **device\_info\_plus: ^10.1.2**
   1. Пакет для получения информации о устройстве, такой как модель, версия операционной системы и производитель.
7. **path: ^1.8.0**
   1. Пакет для работы с путями файлов. Он помогает создавать и манипулировать путями файлов в Dart.
8. **intl: ^0.18.0**
   1. Пакет для работы с интернационализацией и локализацией в приложениях. Он предоставляет функции для работы с форматированием дат, чисел и текста.

**dev\_dependencies**

1. **flutter\_test:**
   1. Пакет для тестирования приложений Flutter. Он предоставляет инструменты и виджеты для написания и выполнения тестов.

**flutter**

1. **uses-material-design: true**
   1. Указывает, что ваше приложение использует Material Design, что позволяет использовать различные компоненты и стили Material Design в вашем приложении.
2. **assets:**
   1. Определяет список ресурсов (файлов), которые будут включены в приложение и доступны для использования. В данном случае это JSON-файлы для локализаций:
      1. assets/ device\_info.json — файл для моделей телефонов.

**Отладка и тестирование**

Проверьте, что версии ваших зависимостей совместимы друг с другом и с текущей версией Flutter. Если возникают проблемы, попробуйте обновить зависимости до последних стабильных версий, если это возможно.

После выполнения всех шагов, попробуйте запустить ваше приложение на устройстве или эмуляторе и проверьте работоспособность приложения.

**Контрольные вопросы:**

1. Какую команду нужно использовать для создания нового Flutter проекта?
2. Какие команды помогают в проверке и анализе кода проекта?
3. Как собрать APK файл из Flutter проекта?
4. Для чего используется команда flutter clean?