**Практическое занятие 9: Изменение элементов интерфейса**

**Цели:**

1. Научиться изменять и настраивать различные элементы интерфейса в приложении Flutter.
2. Ознакомиться с методами стилизации и настройки элементов пользовательского интерфейса.

**Оборудование:**

1. Компьютер с установленным Visual Studio Code.
2. Установленный Flutter SDK.
3. Установленный Dart SDK.

**Контрольные вопросы:**

1. Что такое виджет в Flutter? Приведите примеры базовых виджетов.
2. Какое различие между Stateless и Stateful виджетами? Приведите примеры.
3. Как изменить текст и его стиль с помощью виджета Text? Какие параметры доступны для изменения стиля?
4. Какие типы кнопок существуют в Flutter? Чем они отличаются друг от друга?
5. Как используется метод setState() для изменения состояния в Stateful виджете? Почему он важен для изменения интерфейса?
6. Что такое контейнер (Container) и какие его основные параметры могут быть изменены?
7. Как можно стилизовать кнопку в Flutter? Приведите пример использования ButtonStyle.
8. Как в Flutter работает система компоновки элементов? В чем разница между виджетами Row и Column?
9. Что такое GestureDetector и для чего он используется? Как обработать нажатие на текст с его помощью?
10. Как можно глобально задать тему для приложения в Flutter? Какие параметры можно настроить через ThemeData?
11. Что такое адаптивный интерфейс? Какие инструменты предоставляет Flutter для адаптации приложения под разные размеры экранов?
12. Для чего используется LayoutBuilder в Flutter? Приведите пример его применения.
13. Как изменить цвет и размер шрифта текста внутри контейнера в Flutter?
14. Как в Flutter задать отступы (padding) и поля (margin) для элементов интерфейса?
15. Что такое ScaffoldMessenger и как его использовать для отображения всплывающих сообщений (SnackBar)?

**Дополнительные задания:**

**Задание 1**: Добавьте визуальные индикаторы прогресса для отображения скорости загрузки и отдачи во время теста скорости.

**Задание 2**: Улучшите внешний вид кнопок "Начать тест скорости" и "Скачать результаты в PDF" с помощью добавления иконок и кастомных стилей.

**Задание 3**: Добавьте возможность переключения между светлой и темной темой приложения через Switch или иконку в AppBar.

**Теоретическая часть**

Каждый элемент пользовательского интерфейса в Flutter представляет собой виджет. Виджеты могут быть простыми (например, текст или кнопка) или сложными, состоящими из комбинации других виджетов (например, целые экраны). Важной особенностью Flutter является то, что виджеты отвечают за внешний вид и поведение приложения, а также могут быть легко изменены и настроены.

**Элементы интерфейса и виджеты в Flutter**

1. **Текстовые элементы (Text)**: Текст — один из самых базовых виджетов в Flutter. Виджет Text позволяет отображать строки текста с возможностью изменения стиля, размера, цвета и других параметров.

Пример:

Text(

'Пример текста',

style: TextStyle(fontSize: 20, color: Colors.blue),

)

1. **Кнопки (Button)**: Flutter предоставляет несколько типов кнопок, таких как ElevatedButton, TextButton, IconButton и другие. Кнопки позволяют взаимодействовать с пользователем через нажатие, а также могут быть стилизованы.

Пример:

ElevatedButton(

onPressed: () {},

child: Text('Нажми меня'),

)

Виджет кнопки может изменяться по стилю, цвету, размерам и анимациям.

1. **Контейнеры (Container)**: Контейнер — это универсальный виджет, который позволяет управлять размером, отступами, расположением и фоном содержимого. Он часто используется для группировки и размещения других виджетов.

Пример:

Container(

padding: EdgeInsets.all(16.0),

color: Colors.green,

child: Text('Текст внутри контейнера'),

)

**Принципы изменения элементов интерфейса в Flutter**

1. **Использование состояния (State)**: Чтобы изменять элементы интерфейса на основе взаимодействия с пользователем, Flutter использует Stateful виджеты. Виджет с состоянием может динамически изменяться, реагируя на действия пользователя (например, нажатие на кнопку).

Пример использования состояния:

class MyStatefulWidget extends StatefulWidget {

@override

\_MyStatefulWidgetState createState() => \_MyStatefulWidgetState();

}

class \_MyStatefulWidgetState extends State<MyStatefulWidget> {

String buttonText = "Нажми меня";

void changeText() {

setState(() {

buttonText = "Текст изменен!";

});

}

@override

Widget build(BuildContext context) {

return ElevatedButton(

onPressed: changeText,

child: Text(buttonText),

);

}

}

Здесь функция setState() используется для того, чтобы сигнализировать системе Flutter об изменении состояния, что приводит к перерисовке пользовательского интерфейса.

1. **Стилизация и тема приложения**: Flutter позволяет глобально задавать тему приложения с помощью виджета ThemeData. Это позволяет единообразно изменять внешний вид всего приложения — от цветов до шрифтов.

Пример:

MaterialApp(

theme: ThemeData(

primaryColor: Colors.blue,

textTheme: TextTheme(

bodyText1: TextStyle(fontSize: 18.0, color: Colors.black),

),

),

home: MyHomePage(),

)

Также можно использовать кастомные стили для каждого конкретного элемента интерфейса. Для этого применяются такие параметры, как TextStyle, BoxDecoration, ButtonStyle и другие.

1. **Системы компоновки (Layouts)**: Flutter предлагает несколько гибких систем компоновки элементов интерфейса:
   1. Row и Column: позволяют располагать элементы горизонтально или вертикально.
   2. Stack: позволяет накладывать элементы друг на друга.
   3. Expanded и Flexible: управляют гибкостью и адаптивностью размеров элементов в зависимости от пространства экрана.

Пример:

Column(

children: <Widget>[

Text('Первый элемент'),

Text('Второй элемент'),

],

)

**Взаимодействие с пользователем**

Элементы интерфейса должны быть интерактивными. Flutter предоставляет несколько способов добавления взаимодействия:

1. **Нажатия на кнопки** (onPressed).
2. **Жесты** с использованием GestureDetector (например, обработка нажатий, свайпов и других жестов).

Пример использования GestureDetector:

GestureDetector(

onTap: () {

print('Элемент был нажат');

},

child: Container(

color: Colors.blue,

padding: EdgeInsets.all(16.0),

child: Text('Нажми меня'),

),

)

**Адаптивность интерфейса**

В современных приложениях важно учитывать разные размеры экранов. Flutter предоставляет инструменты для адаптивного дизайна:

1. **MediaQuery**: позволяет получать информацию о размерах экрана и изменять интерфейс в зависимости от них.
2. **LayoutBuilder**: помогает адаптировать компоненты в зависимости от доступного пространства.

Пример:

Widget build(BuildContext context) {

return LayoutBuilder(

builder: (context, constraints) {

if (constraints.maxWidth > 600) {

return Text('Большой экран');

} else {

return Text('Маленький экран');

}

},

);

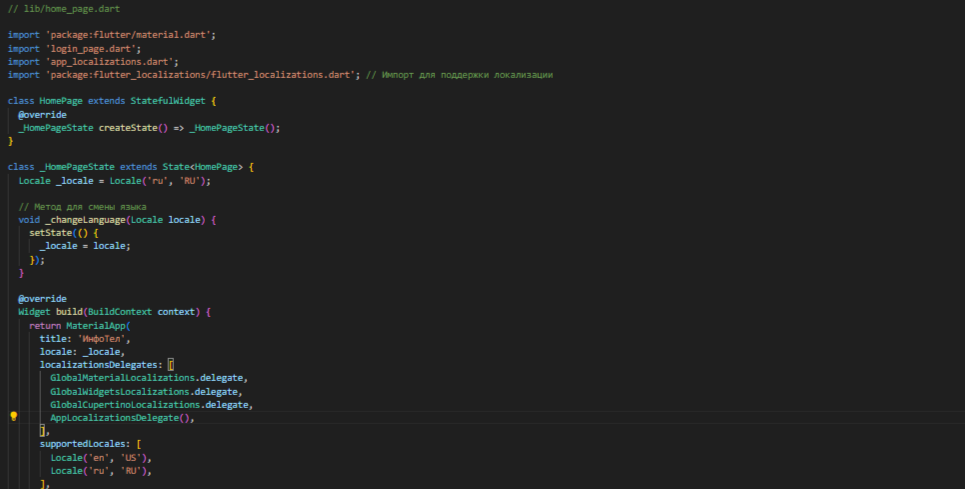
}

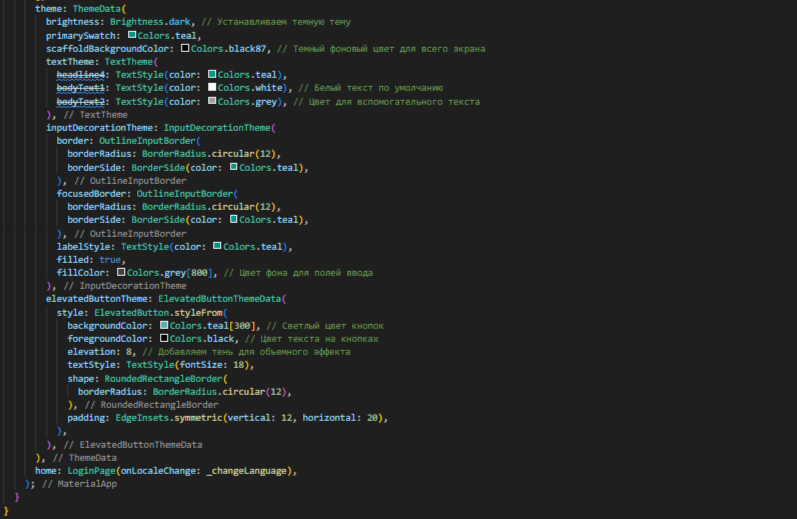
**Заключение**

Изменение и настройка элементов интерфейса в Flutter — это мощный инструмент, который позволяет создавать как простые, так и сложные, адаптивные интерфейсы для мобильных и веб-приложений. Благодаря обширной библиотеке виджетов, гибким средствам компоновки и возможности работы с состояниями, Flutter предоставляет разработчикам полный контроль над визуальной частью приложения.

**Практическая часть**

Давайте обновим интерфейс нашего приложения. Вот как можно обновить код для lib/home\_page.dart, чтобы он стал более читабельным и поддерживал наш текущий функционал.



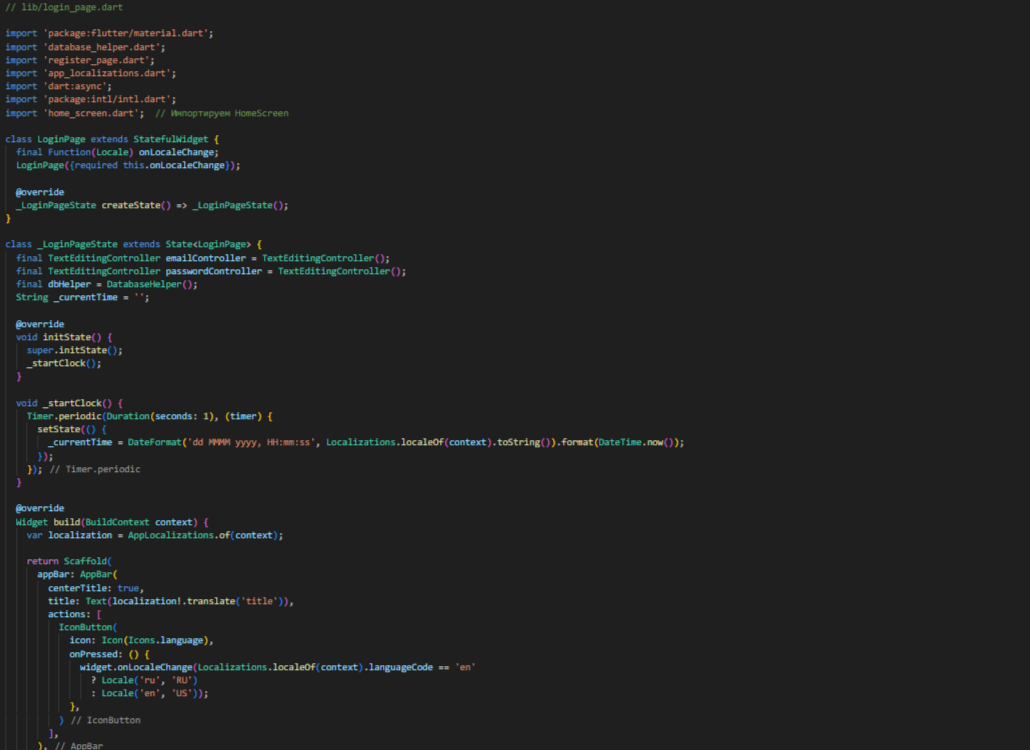


**Основные изменения:**

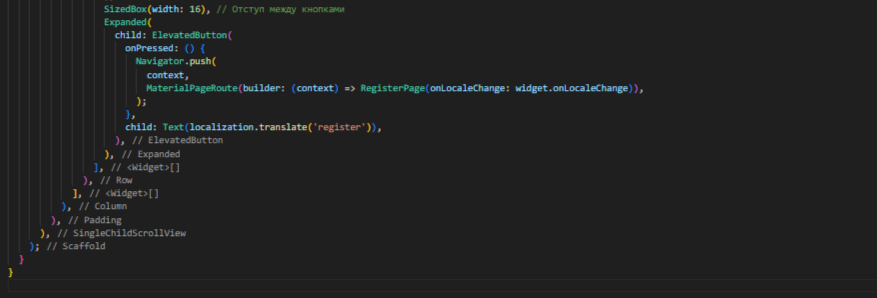
1. **Тема и фон**:
   1. brightness: Brightness.dark: Устанавливает темную тему для всего приложения.
   2. scaffoldBackgroundColor: Colors.black87: Устанавливает темный фон для экрана.
2. **Текст**:
   1. В TextTheme добавлены стили для bodyText1 и bodyText2 с белым и серым цветами соответственно.
3. **Поля ввода**:
   1. inputDecorationTheme теперь использует Colors.grey[800] для фона полей ввода и увеличен радиус границ.
4. **Кнопки**:
   1. Цвет кнопок изменен на более светлый оттенок Colors.teal[300].
   2. Добавлена тень с помощью свойства elevation, чтобы кнопки выглядели объемнее.

Эти изменения придадут приложению более современный и привлекательный вид.

Для того чтобы разместить кнопки "Войти" и "Регистрация" на одной строке, можно использовать виджет Row, который позволит выстроить кнопки горизонтально. Вот обновленный код для lib/login\_page.dart, где кнопки размещены на одной строке:







**Объяснение изменений:**

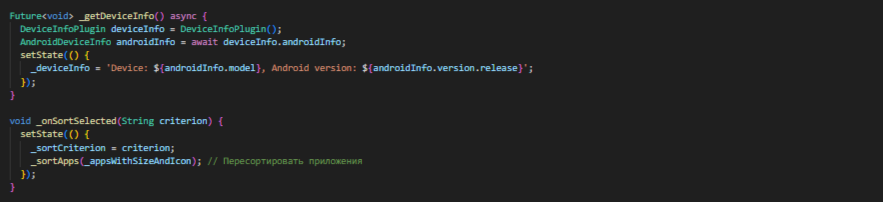
1. **Добавление Row**:
   1. Использован виджет Row для размещения кнопок "Войти" и "Регистрация" на одной строке.
   2. mainAxisAlignment: MainAxisAlignment.spaceBetween распределяет пространство между кнопками равномерно.
2. **Использование Expanded**:
   1. Обернул кнопки в Expanded, чтобы они занимали равное пространство и были выровнены по ширине.
3. **Отступ между кнопками**:
   1. Добавлен SizedBox(width: 16) для создания отступа между кнопками.
4. **Цвет текста даты и времени**: В Text виджете, отображающем дату и время, добавлен параметр color: Colors.white в стиль текста, чтобы установить белый цвет.

Теперь кнопки "Войти" и "Регистрация" отображаются на одной строке и занимать равное пространство.

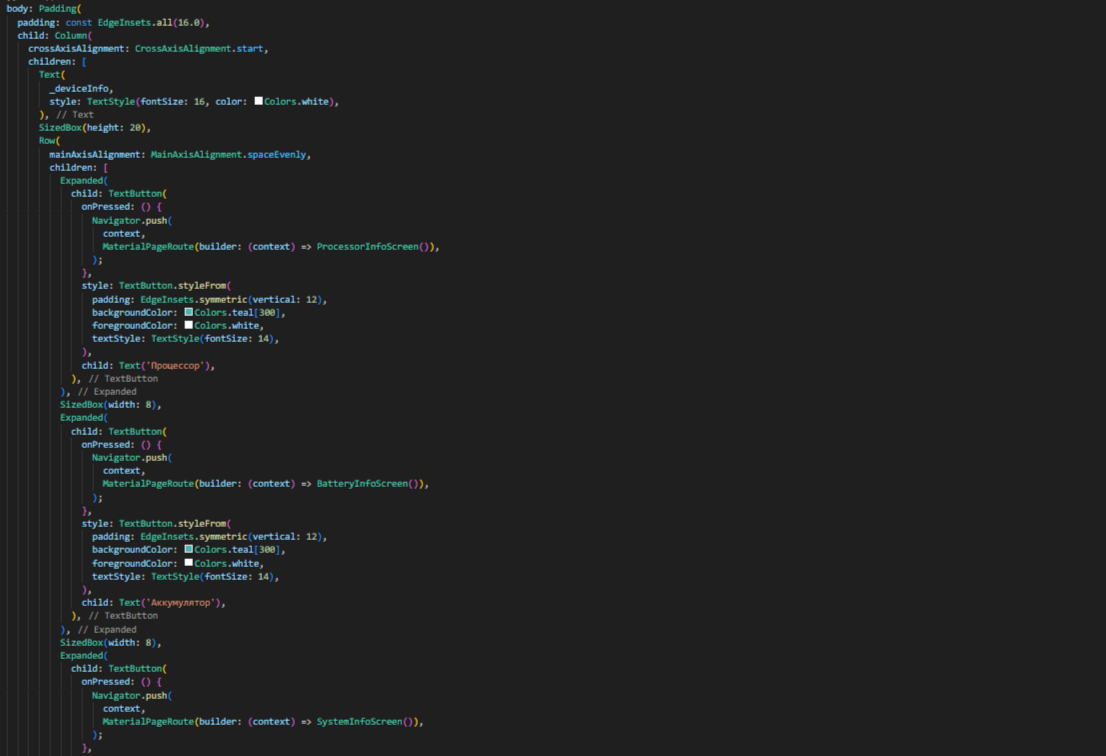
Расположим дату и время по середине AppBar, а также сделаем кнопки "Процессор", "Аккумулятор" и "Система" в одну строку меньшего размера, нужно внести следующие изменения в код:

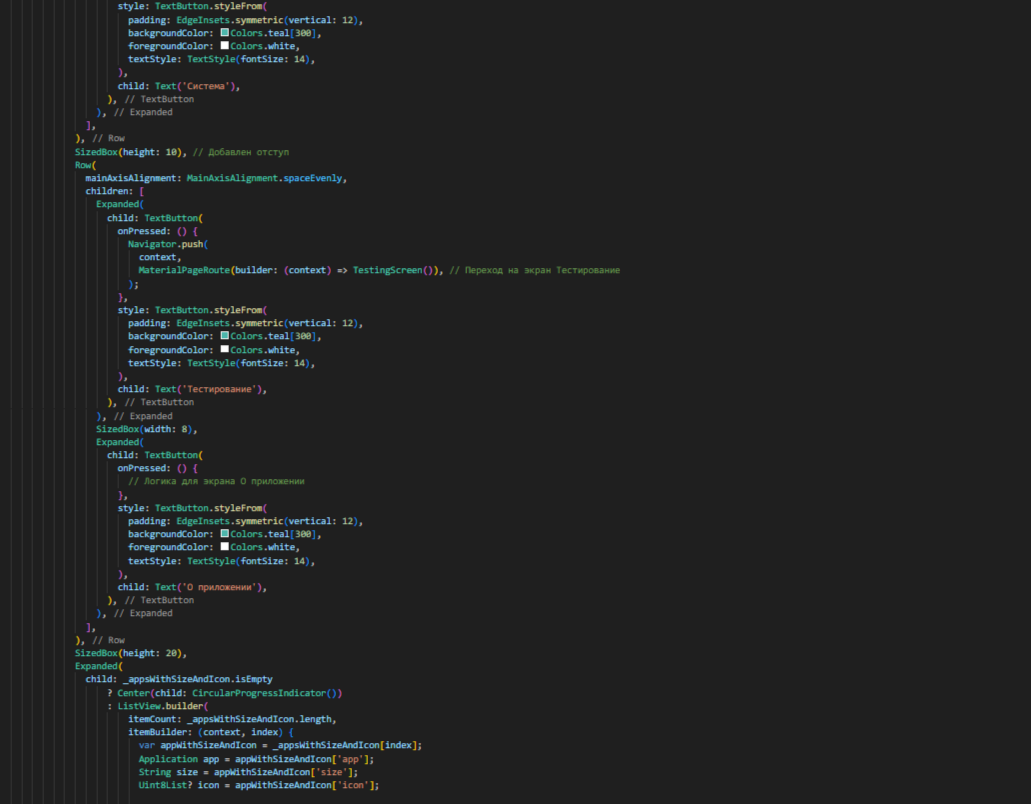
1. **Центрирование даты и времени в AppBar**:
   1. Мы будем использовать Center для размещения текста по середине AppBar.
2. **Размещение кнопок в одну строку**:
   1. Используем виджет Row и изменим размер кнопок с помощью TextButton и параметров стиля.

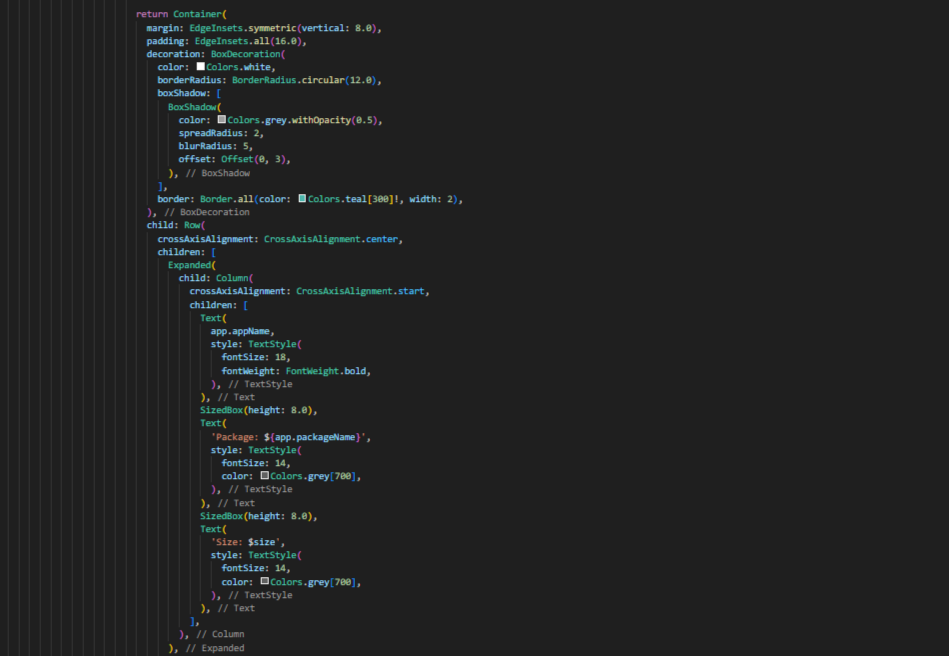














**Объяснение изменений:**

1. **Центрирование даты и времени**:
   1. Обернули StreamBuilder в Center, чтобы текст был выровнен по середине AppBar.
2. **Кнопки в одной строке**:
   1. Обернули кнопки в Row с mainAxisAlignment: MainAxisAlignment.spaceEvenly, чтобы равномерно распределить кнопки.
   2. Заменили ElevatedButton на TextButton с параметром style: TextButton.styleFrom() для настройки стиля кнопок. Установили меньший размер текста и добавили отступы.
3. **Отступы и стили кнопок**:
   1. Использовали SizedBox(width: 8) для отступов между кнопками.
   2. Настроили padding, backgroundColor, и textStyle для кнопок, чтобы они выглядели компактнее и современнее.
4. **Заменили primary на foregroundColor**:
5. В TextButton.styleFrom() использован параметр foregroundColor для установки цвета текста.
6. **Использование backgroundColor для фона кнопок**:
7. backgroundColor используется для установки цвета фона кнопок.

Теперь дата и время будут отображаться по центру AppBar, а кнопки будут расположены в одной строке с меньшим размером текста.

Добавим кнопку "Выход" на экран, которая перенаправляет пользователя на страницу входа (LoginPage), нужно сделать следующее:

1. **Добавить кнопку "Выход" в интерфейс**.
2. **Обработать нажатие кнопки для навигации к странице входа**.
3. **Добавление кнопки "Выход" в AppBar**:
   1. Использован IconButton с иконкой Icons.exit\_to\_app.
   2. Обработчик onPressed перенаправляет пользователя на страницу входа (LoginPage) при помощи Navigator.pushReplacement. Это обеспечивает замену текущего экрана, чтобы пользователь не мог вернуться к предыдущему экрану (HomeScreen) после выхода.
4. **Импорт LoginPage**:
   1. Импортирован LoginPage для возможности навигации.
5. **Параметры конструктора LoginPage**:
   1. Параметр onLocaleChange передается пустым лямбда-функцией, так как он не используется на странице входа. Это необходимо, чтобы соответствовать конструктору LoginPage.

Теперь у вас будет кнопка "Выход" в AppBar, которая позволяет пользователю вернуться на страницу входа.

Сделаем отображения списка установленных приложений и объема занимаемой ими памяти на странице, вам нужно использовать пакет installed\_apps, который позволяет получать данные об установленных приложениях на устройстве.

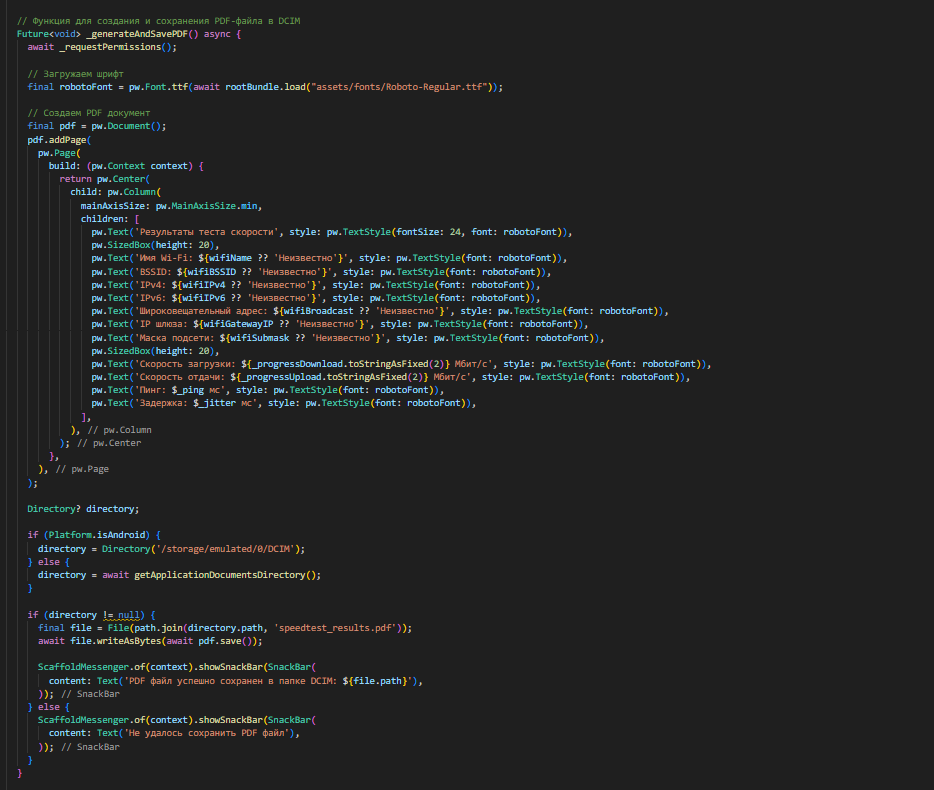
Вот шаги, которые нужно выполнить:

1. **Установка зависимости**:
   1. Добавьте installed\_apps: ^1.5.0 в файл pubspec.yaml и выполните команду flutter pub get, чтобы установить пакет.
2. **Импорт необходимых библиотек**:
   1. Импортируйте пакет installed\_apps в нужный файл.
3. **Получение и отображение списка установленных приложений**:
   1. Используйте installed\_apps для получения списка приложений и их размера, и отобразите их в вашем интерфейсе.

Давайте создадим код для страницы тестирования testing\_screen.dart чтоб выводилась информация о сети wifi, был тест скорости сети и возможно сохранять PDF файл для возможности выгрузки системы.











**Блок 1: Инициализация переменных и объектов**

Этот блок содержит переменные для хранения информации о сети (Wi-Fi), результаты теста скорости (загрузка, отдача, пинг и задержка), флаги для управления состоянием тестирования и таймер для ограничения времени теста.

**Блок 2: initState и инициализация информации о сети**

Здесь происходит начальная настройка экрана. В initState вызывается метод \_initNetworkInfo, который запрашивает разрешения на доступ к информации о местоположении. После получения разрешений приложение собирает информацию о Wi-Fi (например, имя сети, BSSID, IP-адреса) с помощью пакета network\_info\_plus и сохраняет её в соответствующие переменные.

**Блок 3: Функция для начала теста скорости**

Метод \_startSpeedTest запускает тест скорости интернета с использованием библиотеки flutter\_speedtest. Он отслеживает прогресс загрузки и отдачи данных через обратные вызовы (callbacks) и обновляет состояние экрана с результатами. Если во время теста возникают ошибки, они логируются с помощью developer.log. Также запускается таймер на 40 секунд, после которого тест завершится.

**Блок 4: Функция для остановки теста скорости**

Метод \_stopSpeedTest останавливает тест, если он был запущен. После этого на экране отображается кнопка для скачивания PDF-файла с результатами. Таймер, запущенный в предыдущем блоке, также останавливается.

**Блок 5: Создание и сохранение PDF-файла**

Метод \_generateAndSavePDF создает PDF-документ с результатами теста скорости, используя библиотеку pdf. Документ включает информацию о сети (имя Wi-Fi, BSSID, IP-адреса и др.), а также результаты теста скорости (скорость загрузки и отдачи, пинг, задержка). После создания PDF сохраняется в папке DCIM на устройстве. Если PDF успешно сохранён, показывается уведомление об успехе через ScaffoldMessenger.

**Блок 6: Запрос разрешений для сохранения файлов**

Метод \_requestPermissions запрашивает у пользователя разрешения на доступ к хранилищу устройства. Это нужно для того, чтобы сохранять PDF-документ в памяти устройства. Если разрешения не были предоставлены, приложение снова запрашивает их.

**Блок 7: Виджеты для интерфейса**

Метод build строит пользовательский интерфейс. Он состоит из:

1. **AppBar**: Заголовок экрана с текстом "Тестирование сети".
2. **Container** с информацией о сети: отображает результаты получения информации о Wi-Fi.
3. **ElevatedButton** для начала теста скорости: активен только если тест не запущен.
4. **Container** с результатами теста скорости: отображает результаты тестирования скорости (загрузка, отдача, пинг, задержка).
5. **ElevatedButton** для скачивания PDF: отображается только после завершения теста.

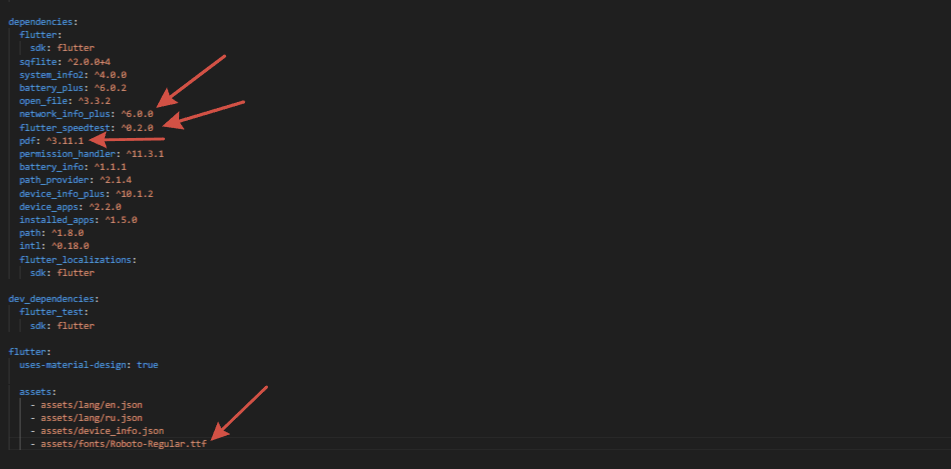
**Блок 8: Действия при закрытии экрана**

Метод dispose используется для очистки ресурсов при закрытии экрана. В данном случае, он останавливает таймер, если тот ещё работает, чтобы избежать утечек памяти.

Таким образом, этот код представляет собой приложение для тестирования скорости интернета и сбора информации о сети Wi-Fi. После завершения теста скорость можно сохранить в PDF-файл.

**Файл pubspec.yaml**

Добавим новые модификации в pubspec.yaml.



Так же создадим в assets папку font и скачаем в него файл текста .ttf

**Зависимости (dependencies)**

1. **open\_file: ^3.3.2**
   1. **Описание**: Пакет, который позволяет открывать файлы с помощью внешних приложений, доступных на устройстве.
   2. **Использование**: Например, приложение может открывать PDF-файлы или другие типы документов, используя сторонние приложения.
2. **network\_info\_plus: ^6.0.0**
   1. **Описание**: Пакет для получения информации о сети устройства (тип сети, IP-адрес, Wi-Fi).
   2. **Использование**: Может использоваться для анализа сети, к которой подключено устройство, и адаптации функциональности приложения под текущие сетевые условия.
3. **flutter\_speedtest: ^0.2.0**
   1. **Описание**: Пакет для измерения скорости интернет-соединения.
   2. **Использование**: Приложение может проверять скорость интернета и предоставлять пользователю эту информацию для диагностики соединения.
4. **pdf: ^3.11.1**
   1. **Описание**: Пакет для генерации PDF-файлов. Позволяет создавать и экспортировать документы в формате PDF прямо из приложения.
   2. **Использование**: Генерация отчетов, квитанций, документов для отправки или сохранения.
5. **permission\_handler: ^11.3.1**
   1. **Описание**: Пакет для управления разрешениями на устройствах (доступ к камере, микрофону, памяти и т.д.).
   2. **Использование**: Позволяет проверять и запрашивать необходимые разрешения у пользователя для работы с функционалом устройства.
6. **path\_provider: ^2.1.4**
   1. **Описание**: Пакет для определения путей к файлам и директориям на устройстве (например, временные файлы, документы).
   2. **Использование**: Для работы с файлами и сохранения данных в определенные директории на устройстве.
7. **device\_info\_plus: ^10.1.2**
   1. **Описание**: Пакет для получения информации о устройстве (модель, производитель, операционная система и т.д.).
   2. **Использование**: Используется для диагностики устройства, анализа платформы и адаптации приложения под разные устройства.
8. **device\_apps: ^2.2.0**
   1. **Описание**: Пакет для получения информации о приложениях, установленных на устройстве.
   2. **Использование**: Может использоваться для проверки списка установленных приложений или управления ими.
9. **installed\_apps: ^1.5.0**
   1. **Описание**: Пакет, который позволяет получить список установленных приложений на устройстве.
   2. **Использование**: Аналогично device\_apps, используется для получения информации об установленных приложениях.

**assets**

1. **Описание**: Определяет список ресурсов (файлов), которые будут использоваться в приложении. Эти файлы будут упакованы в приложение при сборке.
2. **Файлы**:
   1. assets/fonts/Roboto-Regular.ttf: Шрифт Roboto, который будет использоваться для текста в приложении.

**Контрольные вопросы:**

1. Что такое виджет в Flutter? Приведите примеры базовых виджетов.
2. Какое различие между Stateless и Stateful виджетами? Приведите примеры.
3. Как изменить текст и его стиль с помощью виджета Text? Какие параметры доступны для изменения стиля?
4. Какие типы кнопок существуют в Flutter? Чем они отличаются друг от друга?
5. Как используется метод setState() для изменения состояния в Stateful виджете? Почему он важен для изменения интерфейса?
6. Что такое контейнер (Container) и какие его основные параметры могут быть изменены?
7. Как можно стилизовать кнопку в Flutter? Приведите пример использования ButtonStyle.
8. Как в Flutter работает система компоновки элементов? В чем разница между виджетами Row и Column?
9. Что такое GestureDetector и для чего он используется? Как обработать нажатие на текст с его помощью?
10. Как можно глобально задать тему для приложения в Flutter? Какие параметры можно настроить через ThemeData?
11. Что такое адаптивный интерфейс? Какие инструменты предоставляет Flutter для адаптации приложения под разные размеры экранов?
12. Для чего используется LayoutBuilder в Flutter? Приведите пример его применения.
13. Как изменить цвет и размер шрифта текста внутри контейнера в Flutter?
14. Как в Flutter задать отступы (padding) и поля (margin) для элементов интерфейса?
15. Что такое ScaffoldMessenger и как его использовать для отображения всплывающих сообщений (SnackBar)?

**Дополнительные задания:**

**1. Добавление индикаторов прогресса для теста скорости**

**Задание**: Добавьте визуальные индикаторы прогресса для отображения скорости загрузки и отдачи во время теста скорости.

**Описание**:

1. Используйте виджеты LinearProgressIndicator или CircularProgressIndicator для отображения текущего прогресса загрузки и отдачи.
2. Привяжите индикаторы к переменным \_progressDownload и \_progressUpload, чтобы они динамически изменялись в реальном времени во время теста.

**Подсказка**: Можете использовать значение transferRate / 1000 для отображения прогресса в процентах.

**2. Кастомизация кнопок с использованием иконок**

**Задание**: Улучшите внешний вид кнопок "Начать тест скорости" и "Скачать результаты в PDF" с помощью добавления иконок и кастомных стилей.

**Описание**:

1. Используйте виджет Icon внутри кнопок, чтобы добавить к ним иконки, например, иконку сети для кнопки теста и иконку загрузки для PDF.
2. Измените стиль кнопок: добавьте округленные углы, используйте градиентные фоны для более современного вида и настройте размер и шрифт текста.

**Подсказка**: Можно использовать параметр Icon(Icons.network\_check) для кнопки теста скорости и Icon(Icons.download) для кнопки загрузки PDF.

**3. Добавление темной и светлой темы с переключателем**

**Задание**: Добавьте возможность переключения между светлой и темной темой приложения через Switch или иконку в AppBar.

**Описание**:

1. Реализуйте поддержку двух тем: светлой и темной, используя ThemeData.light() и ThemeData.dark().
2. Добавьте переключатель в верхний AppBar или используйте кнопку с иконкой для переключения темы.
3. Измените цветовую палитру для различных элементов интерфейса в зависимости от выбранной темы (например, цвета фона, текста, кнопок и т.д.).

**Подсказка**: Можно использовать параметр brightness в ThemeData для переключения между светлой и темной темой.