**Практическое занятие 11: Компоновка элементов интерфейса**

**Цель:**

Научиться создавать и компонировать элементы интерфейса в приложении Flutter. В этом занятии мы будем работать с визуальными элементами, такими как индикаторы загрузки, информационные карточки и кнопки.

**Описание изменений:**

В этом занятии мы будем использовать изменения кода для улучшения интерфейса экрана с информацией о батарее.

**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

**Дополнительные задания:**

1. **Изменение цветовой схемы интерфейса:** Измените цветовую схему приложения. Используйте различные цветовые палитры для каждого состояния батареи: например, зеленый — для заряда более 75%, желтый — для среднего уровня и красный — для низкого уровня. Попробуйте использовать разные оттенки для фона и текста, чтобы улучшить контраст и читаемость.
2. **Добавление индикатора уровня заряда в виде полосы:** Добавьте горизонтальный индикатор уровня заряда в виде полосы (LinearProgressIndicator). Разместите его под текстом с процентом заряда. Сделайте так, чтобы цвет индикатора изменялся в зависимости от уровня заряда батареи, как это сделано с круговым индикатором.
3. **Добавление истории изменений уровня заряда:** Реализуйте функционал для отслеживания истории изменений уровня заряда батареи. Сохраните значения заряда, обновляемые каждые 10 секунд, и отобразите их в виде списка или графика под основной информацией на экране.
4. **Уведомление при низком уровне заряда:** Добавьте функцию, которая будет отправлять уведомление пользователю при снижении уровня заряда до 20%. Используйте локальные уведомления Flutter для этого задания. Убедитесь, что уведомления корректно отображаются при закрытом приложении.
5. **Реализация адаптивного интерфейса:** Адаптируйте интерфейс для работы на устройствах с различными размерами экранов (смартфоны, планшеты). Используйте MediaQuery для определения размера экрана и изменяйте размеры шрифтов, иконок и других элементов интерфейса в зависимости от доступного пространства.

**Контрольные вопросы:**

1. Что представляет собой иерархическая модель интерфейса в Flutter?
2. Какие виджеты отвечают за компоновку элементов интерфейса в виде строк и столбцов?
3. Какую роль играет виджет Container при компоновке элементов?
4. В чем различие между виджетами Row и Column?
5. Как использовать виджет Stack для наложения элементов друг на друга?
6. Какую функцию выполняет виджет Scaffold в структуре приложения?
7. Какие виджеты можно использовать для создания прокручиваемых списков и областей в приложении?
8. Как с помощью виджета Expanded можно адаптировать интерфейс для разных размеров экранов?
9. Какие свойства виджета позволяют задать отступы между элементами интерфейса?
10. Как реализовать адаптивный интерфейс, учитывающий ориентацию экрана?
11. Что такое "отзывчивый дизайн" (Responsive Design) и как его можно реализовать в Flutter?
12. Какую роль играют виджеты SingleChildScrollView и ListView в построении интерфейса?
13. Как с помощью метода MediaQuery можно получить размеры экрана устройства?
14. Как создать и использовать кнопки с иконками и текстом в Flutter?
15. Что нужно сделать, чтобы обновление данных на экране происходило через регулярные промежутки времени?

**Теоретическая часть**

Компоновка элементов интерфейса в мобильных приложениях является ключевой задачей при разработке пользовательского интерфейса (UI). В Flutter эта задача решается с помощью виджетов, которые позволяют размещать, структурировать и визуально оформлять элементы интерфейса на экране. Виджеты могут располагаться внутри других виджетов, образуя дерево интерфейса (widget tree), что позволяет строить сложные структуры UI.

**Основные принципы компоновки**

1. **Иерархическая модель интерфейса** Все элементы интерфейса в Flutter представляют собой виджеты. Виджеты могут быть как "листовыми" (отображающие конкретный элемент интерфейса, например, кнопку или текст), так и контейнерными (включающие в себя другие виджеты). Такая иерархическая модель позволяет гибко организовывать и модифицировать структуру интерфейса приложения.
2. **Виджеты контейнеры** Контейнеры отвечают за размещение и компоновку других виджетов. Например:
   1. Column — размещает виджеты вертикально.
   2. Row — располагает виджеты горизонтально.
   3. Stack — накладывает виджеты друг на друга.
   4. Container — универсальный виджет, позволяющий задавать отступы, размеры и фон для дочернего виджета.
3. **Принципы компоновки элементов** Компоновка интерфейса зависит от нескольких аспектов:
   1. **Размеры**: Указание размеров виджетов может быть как фиксированным (например, 100x100), так и гибким (в процентах от экрана или на основе содержимого).
   2. **Отступы и поля (padding и margin)**: Отступы между элементами интерфейса помогают сделать UI более упорядоченным и удобным для восприятия.
   3. **Выравнивание**: Выравнивание элементов позволяет располагать их в нужном порядке относительно родительского контейнера (по центру, по краям, с равными промежутками и т.д.).
4. **Композиционные виджеты** Компоновка элементов интерфейса может осуществляться с помощью композиционных виджетов, которые организуют дочерние виджеты по определенным правилам:
   1. Scaffold — основной виджет структуры страницы, который поддерживает стандартные компоненты интерфейса, такие как AppBar, BottomNavigationBar и FloatingActionButton.
   2. SingleChildScrollView — виджет, позволяющий прокручивать контент, если он не помещается на экране.
   3. Card — виджет для создания карточек с тенью и закругленными углами, часто используемый для отображения информации.
   4. ListView — виджет для создания прокручиваемых списков.

**Пример основных виджетов для компоновки**

1. **Container** Container — это универсальный виджет для управления размерами, отступами и фоном дочерних элементов. Его часто используют для оформления других виджетов, добавляя фон, закругленные углы и тени.

Пример использования:

Container(

padding: EdgeInsets.all(16),

decoration: BoxDecoration(

color: Colors.blue[100],

borderRadius: BorderRadius.circular(10),

),

child: Text('Пример контейнера'),

)

1. **Row и Column** Эти виджеты управляют расположением дочерних виджетов по горизонтали и вертикали соответственно. Внутри Row элементы располагаются в строку, а в Column — в колонку.

Пример использования Column:

Column(

children: [

Text('Элемент 1'),

SizedBox(height: 10), // Отступ между элементами

Text('Элемент 2'),

],

)

1. **Stack** Виджет Stack накладывает виджеты друг на друга. Это полезно для создания сложных композиций, таких как круговые индикаторы, кнопки поверх изображения и т.д.

Пример использования Stack:

Stack(

alignment: Alignment.center,

children: [

CircularProgressIndicator(),

Text('50%'),

],

)

1. **Scaffold** Scaffold — это базовый виджет для страницы приложения. Он обеспечивает базовую структуру и компоненты интерфейса, такие как панель приложения (AppBar), основную область контента (body), боковое меню (Drawer) и плавающую кнопку (FloatingActionButton).

Пример использования Scaffold:

Scaffold(

appBar: AppBar(

title: Text('Пример Scaffold'),

),

body: Center(

child: Text('Основное содержимое страницы'),

),

floatingActionButton: FloatingActionButton(

onPressed: () {},

child: Icon(Icons.add),

),

)

**Основные аспекты адаптивной компоновки**

1. **Отзывчивость (Responsive Design)** Интерфейс должен быть адаптирован под разные размеры экранов. В Flutter отзывчивость достигается за счет гибкого размещения виджетов и использования виджетов, поддерживающих прокрутку, таких как SingleChildScrollView, ListView, GridView и других.
2. **Адаптивные размеры** Для адаптации под различные размеры экрана используют процентные или гибкие размеры (например, MediaQuery или Expanded). Это позволяет автоматически подстраивать элементы интерфейса в зависимости от устройства.
3. **Ориентация экрана** Важно учитывать возможные изменения ориентации экрана (портретная или альбомная). Виджеты в Flutter могут адаптироваться к этим изменениям автоматически или с помощью управления состоянием приложения.

**Заключение**

Компоновка элементов интерфейса — это процесс структурирования и оформления пользовательского интерфейса. Правильное использование виджетов компоновки в Flutter позволяет создавать гибкие и отзывчивые интерфейсы, которые адаптируются под различные устройства и экраны. Изучение и понимание работы с виджетами контейнерами и композиционными виджетами является основой успешной разработки приложений на Flutter.

**Практическая часть**

**Описание изменений:**

В этом занятии мы будем использовать изменения кода для улучшения интерфейса экрана с информацией о батарее.

**Шаги:**

1. **Подключение новых библиотек:** В начале кода добавляем пакет dart:async, чтобы использовать таймер для периодического обновления данных о батарее.

import 'dart:async';

1. **Обновление логики и состояния интерфейса:** Для реализации логики обновления данных каждые 10 секунд добавляем флаг для отслеживания загрузки (\_isLoading) и используем таймер (Timer) для периодического обновления состояния.

Добавляем следующие поля в класс состояния:

bool \_isLoading = false; // Флаг для индикации загрузки

Timer? \_timer; // Таймер для обновления информации

Создаем функцию для запуска обновления данных с использованием таймера:

void \_startUpdatingBatteryInfo() {

setState(() {

\_isLoading = true; // Показываем индикатор загрузки

});

\_timer = Timer.periodic(Duration(seconds: 1), (timer) async {

// Обновляем информацию каждую секунду

await \_getBatteryInfo();

if (timer.tick >= 10) {

// После 10 секунд останавливаем таймер

timer.cancel();

setState(() {

\_isLoading = false; // Скрываем индикатор загрузки

});

}

});

}

1. **Индикатор уровня заряда батареи:** Используем CircularProgressIndicator для отображения уровня заряда батареи. В зависимости от уровня заряда меняем цвет индикатора с помощью функции \_getBatteryLevelColor. Например, если заряд больше 75%, индикатор зеленый, если меньше 20% — красный.

Center(

child: Stack(

alignment: Alignment.center,

children: [

SizedBox(

height: 150,

width: 150,

child: CircularProgressIndicator(

value: \_batteryLevel / 100,

strokeWidth: 12,

valueColor: AlwaysStoppedAnimation<Color>(

\_getBatteryLevelColor(\_batteryLevel),

),

backgroundColor: Colors.grey[200],

),

),

Text(

'$\_batteryLevel%',

style: TextStyle(

fontSize: 24,

fontWeight: FontWeight.bold,

color: \_getBatteryLevelColor(\_batteryLevel),

),

),

],

),

),

1. **Создание карточек с информацией:** Для отображения информации о состоянии батареи, температуре и напряжении используем карточки с иконками, описаниями и значениями. Эти элементы созданы через метод \_buildInfoCard, который позволяет легко настраивать контент карточки.

Пример создания карточки:

\_buildInfoCard(

title: 'Состояние',

value: \_getBatteryStateDescription(\_batteryState),

icon: Icons.power,

backgroundColor: Colors.orange[100],

),

Карточка оформлена с помощью контейнера с закругленными углами и тенью для улучшения восприятия.

1. **Кнопка обновления данных:** В центре экрана добавляем кнопку для ручного обновления данных о батарее. При нажатии на кнопку начинается процесс обновления, и отображается индикатор загрузки, если данные обновляются.

Align(

alignment: Alignment.center,

child: ElevatedButton.icon(

onPressed: () {

\_startUpdatingBatteryInfo(); // Запускаем обновление

},

icon: \_isLoading

? CircularProgressIndicator(

color: Colors.white,

strokeWidth: 2.0,

)

: Icon(Icons.refresh),

label: Text(

\_isLoading ? 'Обновляется...' : 'Обновить информацию',

style: TextStyle(fontSize: 16, color: Colors.white),

),

style: ElevatedButton.styleFrom(

backgroundColor: Colors.teal[300],

padding: EdgeInsets.symmetric(horizontal: 32, vertical: 16),

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(20),

),

),

),

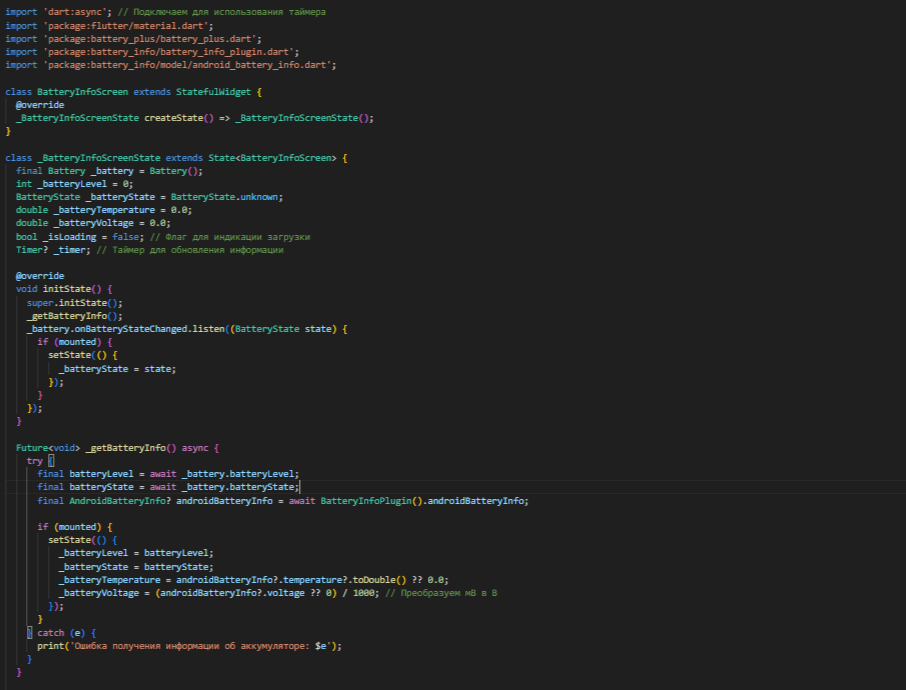
),

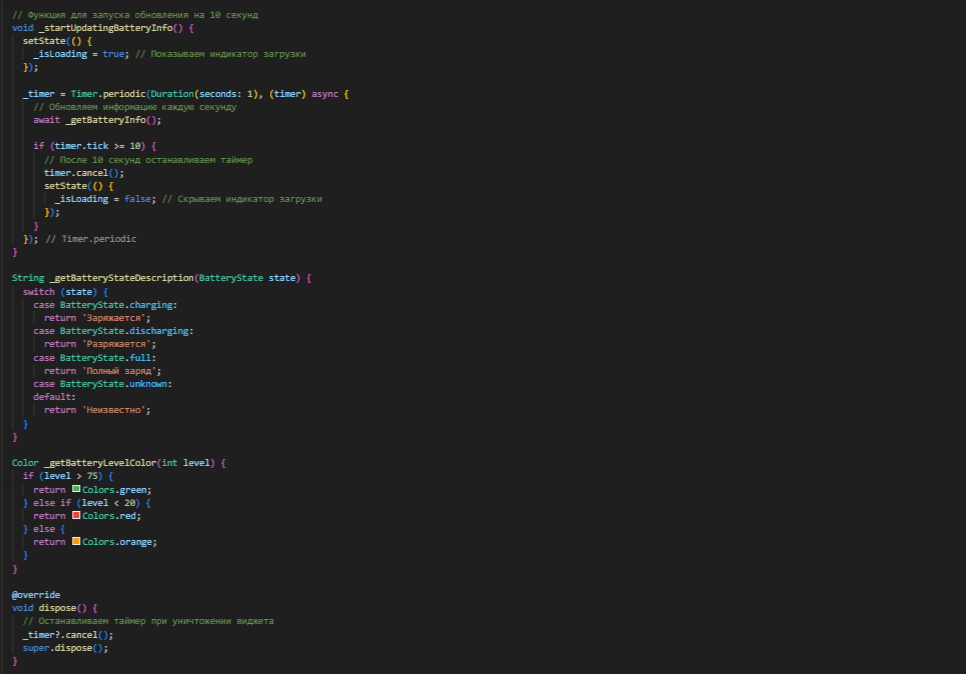
**Результат:**

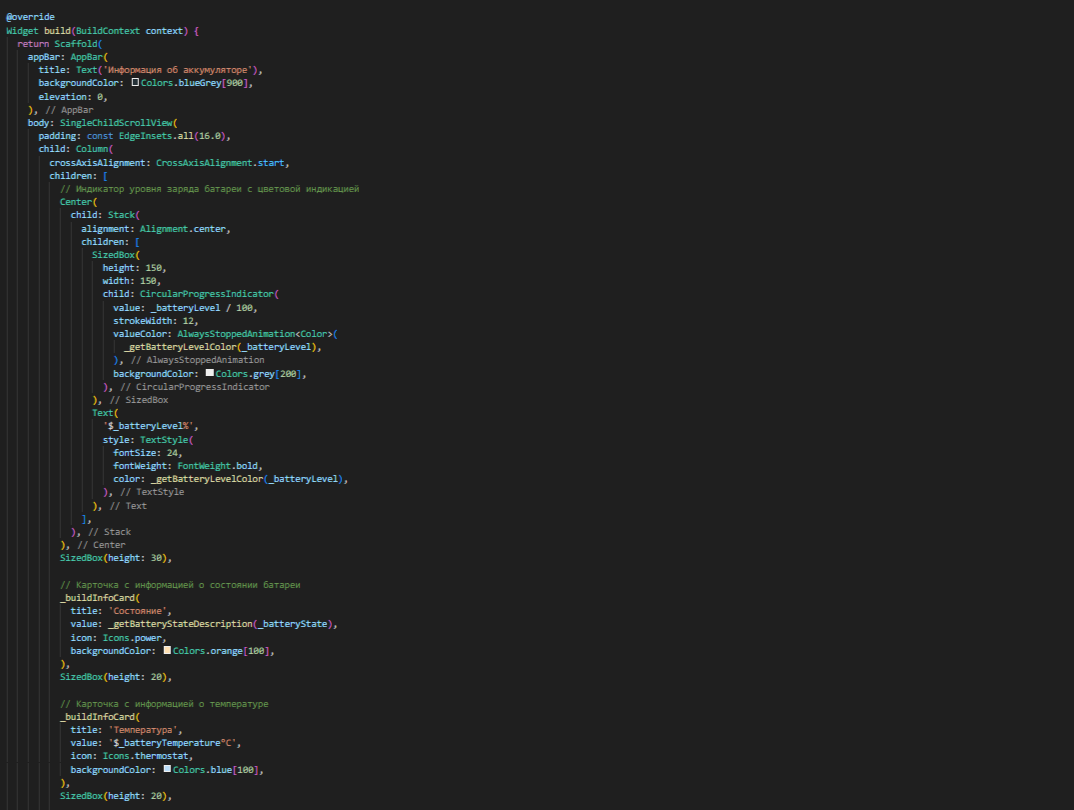
В результате этого практического занятия вы научились:

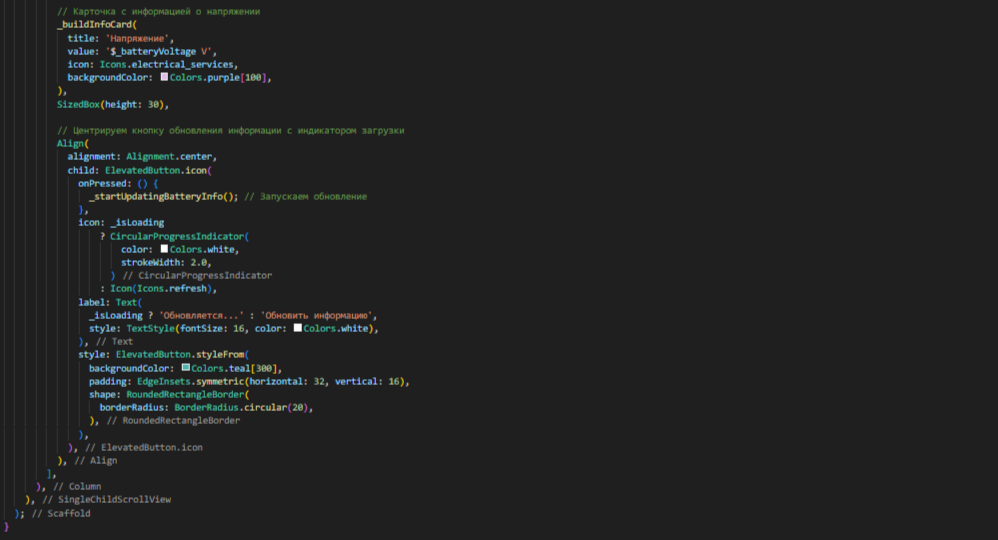
1. Использовать таймер для обновления данных через определенные промежутки времени.
2. Создавать элементы интерфейса, такие как индикаторы и карточки с информацией.
3. Использовать цветовые индикаторы для визуализации состояния.
4. Реализовать кнопку с индикатором загрузки.

**Полный код // lib/battery\_info\_screen.dart**











Этот подход позволяет не только улучшить функциональность приложения, но и сделать его интерфейс более удобным и наглядным для пользователя.

**Дополнительные задания:**

1. **Изменение цветовой схемы интерфейса:** Измените цветовую схему приложения. Используйте различные цветовые палитры для каждого состояния батареи: например, зеленый — для заряда более 75%, желтый — для среднего уровня и красный — для низкого уровня. Попробуйте использовать разные оттенки для фона и текста, чтобы улучшить контраст и читаемость.
2. **Добавление индикатора уровня заряда в виде полосы:** Добавьте горизонтальный индикатор уровня заряда в виде полосы (LinearProgressIndicator). Разместите его под текстом с процентом заряда. Сделайте так, чтобы цвет индикатора изменялся в зависимости от уровня заряда батареи, как это сделано с круговым индикатором.
3. **Добавление истории изменений уровня заряда:** Реализуйте функционал для отслеживания истории изменений уровня заряда батареи. Сохраните значения заряда, обновляемые каждые 10 секунд, и отобразите их в виде списка или графика под основной информацией на экране.
4. **Уведомление при низком уровне заряда:** Добавьте функцию, которая будет отправлять уведомление пользователю при снижении уровня заряда до 20%. Используйте локальные уведомления Flutter для этого задания. Убедитесь, что уведомления корректно отображаются при закрытом приложении.
5. **Реализация адаптивного интерфейса:** Адаптируйте интерфейс для работы на устройствах с различными размерами экранов (смартфоны, планшеты). Используйте MediaQuery для определения размера экрана и изменяйте размеры шрифтов, иконок и других элементов интерфейса в зависимости от доступного пространства.

**Контрольные вопросы:**

1. Что представляет собой иерархическая модель интерфейса в Flutter?
2. Какие виджеты отвечают за компоновку элементов интерфейса в виде строк и столбцов?
3. Какую роль играет виджет Container при компоновке элементов?
4. В чем различие между виджетами Row и Column?
5. Как использовать виджет Stack для наложения элементов друг на друга?
6. Какую функцию выполняет виджет Scaffold в структуре приложения?
7. Какие виджеты можно использовать для создания прокручиваемых списков и областей в приложении?
8. Как с помощью виджета Expanded можно адаптировать интерфейс для разных размеров экранов?
9. Какие свойства виджета позволяют задать отступы между элементами интерфейса?
10. Как реализовать адаптивный интерфейс, учитывающий ориентацию экрана?
11. Что такое "отзывчивый дизайн" (Responsive Design) и как его можно реализовать в Flutter?
12. Какую роль играют виджеты SingleChildScrollView и ListView в построении интерфейса?
13. Как с помощью метода MediaQuery можно получить размеры экрана устройства?
14. Как создать и использовать кнопки с иконками и текстом в Flutter?
15. Что нужно сделать, чтобы обновление данных на экране происходило через регулярные промежутки времени?