**Практическое занятие 6: 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. **Создание нового проекта Flutter**

1. **Откройте терминал (или командную строку)**

2. **Перейдите в директорию, где вы хотите создать проект**

3. **Введите команду для создания нового проекта**

flutter create my\_new\_app

Замените my\_new\_app на имя вашего проекта. Название проекта должно быть без пробелов и с использованием нижнего регистра.

4. **Перейдите в созданную папку проекта**

cd my\_new\_app

1. **Открытие проекта в Visual Studio Code**

1. **Запустите Visual Studio Code**

2. **Откройте проект**

* 1. Выберите меню File -> Open Folder... и выберите папку my\_new\_app.

3. **Установите расширения для Flutter и Dart**

* 1. Перейдите на вкладку Extensions (Расширения) в VS Code (иконка квадратика в боковой панели).
  2. Найдите и установите расширения Flutter и Dart от Dart Code.
  3. Перезапустите VS Code после установки расширений.

1. **Запуск приложения на эмуляторе или устройстве**

1. **Запустите эмулятор Android или подключите физическое устройство**

2. **В VS Code откройте терминал (меню Terminal -> New Terminal)**

3. **Запустите приложение с помощью команды**

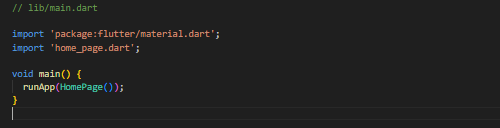
flutter run

Это скомпилирует и запустит приложение на подключенном устройстве или эмуляторе. Вы увидите приветственное сообщение "You have pushed the button this many times:" на экране приложения.

1. **Изменение приложения**

1. **Откройте файл lib/main.dart**

2. **Внесите изменения в код**



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

1. Этот импорт подключает пакет Flutter Material Design, который содержит виджеты, стили и темы для создания интерфейсов по Material Design. Этот пакет предоставляет такие элементы, как кнопки, текстовые поля и многие другие компоненты, используемые для построения пользовательских интерфейсов в приложениях Flutter.

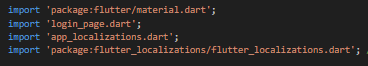
2. import 'home\_page.dart';:

1. Этот импорт подключает файл home\_page.dart, который, вероятно, содержит определение виджета HomePage. Этот файл должен находиться в той же директории, что и main.dart, или путь должен быть корректно указан, если структура проекта отличается. В этом файле описан основной экран или страница вашего приложения.
   1. void main():
      1. Это основная функция, с которой начинается выполнение любого Dart-приложения. В контексте Flutter это точка входа, где начинается выполнение вашего приложения.
   2. runApp(HomePage()):
      1. runApp — это функция Flutter, которая запускает приложение. Она принимает виджет в качестве аргумента и отображает его в корневом элементе приложения.
      2. В данном случае, HomePage() — это виджет, который, как предполагается, находится в файле home\_page.dart. Этот виджет становится корневым виджетом приложения. Если HomePage — это StatefulWidget или StatelessWidget, он будет определять основной экран или страницу приложения.

**Что происходит при выполнении этого кода:**

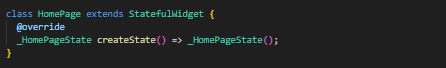
1. Когда вы запускаете приложение Flutter, функция main вызывается автоматически.
2. Внутри main вызывается функция runApp, которая принимает виджет HomePage.
3. Виджет HomePage становится корневым виджетом вашего приложения и отображается на экране. Этот виджет определяет, какой интерфейс будет показан пользователю, и, скорее всего, представляет собой основной экран вашего приложения.

Создадим файл home\_page.dart это будет страница при запуске нашего приложения.

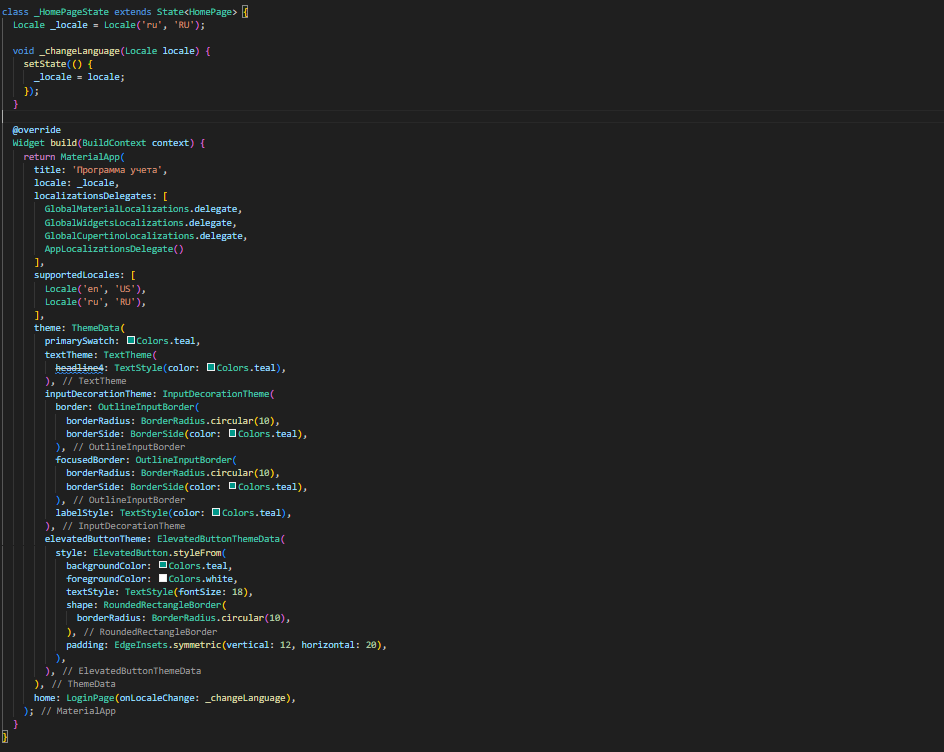


**1. Импорты**

1. import 'package:flutter/material.dart';
   1. Подключает пакет Flutter Material Design, который содержит все необходимые виджеты и инструменты для создания пользовательских интерфейсов по стандартам Material Design.
2. import 'login\_page.dart';
   1. Подключает файл, содержащий виджет LoginPage. Этот виджет будет отображен в качестве основного экрана (home) приложения.
3. import 'app\_localizations.dart';
   1. Подключает файл, в котором, вероятно, определены локализации и делегаты для поддержки нескольких языков.
4. import 'package:flutter\_localizations/flutter\_localizations.dart';
   1. Подключает пакет Flutter Localizations, который содержит делегаты для поддержки различных локализаций, таких как GlobalMaterialLocalizations, GlobalWidgetsLocalizations, и GlobalCupertinoLocalizations.



1. HomePage — это StatefulWidget, что означает, что этот виджет имеет состояние, которое может изменяться. В данном случае, он управляет локалью приложения (языком).
2. createState создает состояние виджета HomePage, которое управляется \_HomePageState.



**Поля и методы класса \_HomePageState**

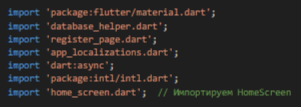
1. **Поле \_locale**
   1. Locale \_locale = Locale('ru', 'RU');
      1. Хранит текущую локаль (язык и регион) приложения. Изначально установлена на русский (ru-RU).
2. **Метод \_changeLanguage**
   1. void \_changeLanguage(Locale locale) { setState(() { \_locale = locale; }); }
      1. Изменяет локаль приложения и обновляет состояние виджета, что приводит к перерисовке экрана с новой локалью.
3. **Метод build**
   1. @override Widget build(BuildContext context) { ... }
      1. Строит интерфейс пользователя для HomePage. В данном случае, он возвращает виджет MaterialApp.

**Виджет MaterialApp**

1. **title: 'Программа учета'**
   1. Заголовок приложения, который отображается в некоторых системных интерфейсах.
2. **locale: \_locale**
   1. Устанавливает текущую локаль для приложения. Это позволяет отображать текст и элементы интерфейса на выбранном языке.
3. **localizationsDelegates: [...]**
   1. Список делегатов локализации, которые предоставляют переводы и поддержку для различных языков.
   2. GlobalMaterialLocalizations.delegate, GlobalWidgetsLocalizations.delegate, и GlobalCupertinoLocalizations.delegate обеспечивают базовую поддержку локализации для виджетов Material, Widgets и Cupertino соответственно.
   3. AppLocalizationsDelegate() — пользовательский делегат, который должен содержать логику для загрузки локализованных строк из ваших файлов локализации.
4. **supportedLocales: [...]**
   1. Указывает список поддерживаемых локалей. В данном случае, поддерживаются английский (США) и русский (Россия).
5. **theme: ThemeData(...)**
   1. Определяет тему приложения, включая цветовую палитру, стиль текста, стили для полей ввода и кнопок.
   2. primarySwatch: Colors.teal — основная цветовая схема приложения.
   3. textTheme — задает стили для текстов, в частности, для заголовков.
   4. inputDecorationTheme — задает стиль для полей ввода, включая оформление границ и цвет.
   5. elevatedButtonTheme — задает стиль для кнопок, включая цвет фона, цвет текста, размер шрифта, форму и отступы.
6. **home: LoginPage(onLocaleChange: \_changeLanguage)**
   1. Устанавливает начальный экран приложения. В данном случае, это виджет LoginPage.
   2. Передает функцию \_changeLanguage как параметр onLocaleChange, чтобы LoginPage мог вызывать эту функцию для изменения локали.

Файл home\_page.dart определяет основной экран приложения и управляет многими аспектами пользовательского интерфейса и локализации. Он предоставляет структуру приложения через MaterialApp, задает тему, поддерживает несколько языков и настраивает начальный экран с передачей функции для изменения языка. Этот файл является ключевой частью вашего приложения, обеспечивая как внешний вид, так и функциональность.

Теперь давайте создадим файл login\_page.dart он описывает экран входа в приложение, который позволяет пользователям вводить свои учетные данные для входа и переключаться между страницами регистрации и домашней страницы.



**1. Импорты**

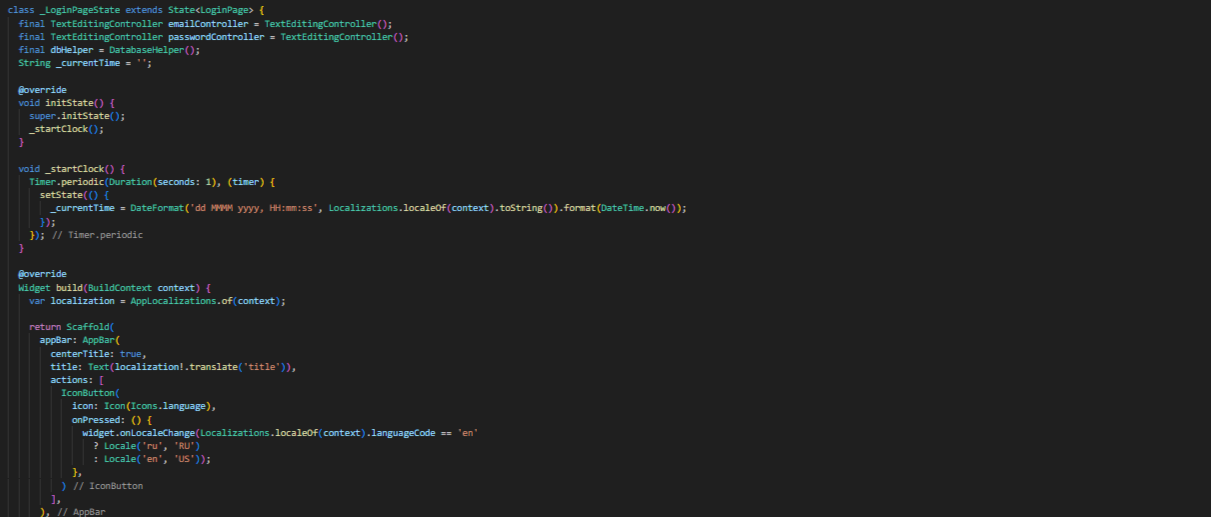
1. import 'package:flutter/material.dart';
   1. Подключает пакет Flutter Material Design для использования виджетов и стилей.
2. import 'database\_helper.dart';
   1. Импортирует DatabaseHelper, который вероятно содержит методы для работы с базой данных, такие как аутентификация пользователей.
3. import 'register\_page.dart';
   1. Импортирует RegisterPage, страницу регистрации пользователей.
4. import 'app\_localizations.dart';
   1. Импортирует файл с локализациями для поддержки многократных языков.
5. import 'dart:async';
   1. Импортирует библиотеку Dart для работы с асинхронными операциями и таймерами.
6. import 'package:intl/intl.dart';
   1. Импортирует библиотеку для форматирования даты и времени.
7. import 'home\_screen.dart';
   1. Импортирует HomeScreen, который представляет собой экран, на который пользователи перенаправляются после успешного входа.

**2. Класс LoginPage**

****

1. LoginPage — это StatefulWidget, что позволяет ему иметь состояние, которое может изменяться.
2. Конструктор принимает функцию onLocaleChange, которая позволяет обновлять локаль (язык) приложения из этого виджета.
3. createState создает состояние для виджета LoginPage, которое управляется \_LoginPageState.

**3. Класс \_LoginPageState**

****

****

**Поля и методы класса \_LoginPageState**

1. **Поле emailController и passwordController**
   1. TextEditingController используется для управления текстовыми полями ввода (email и password).
2. **Поле dbHelper**
   1. DatabaseHelper используется для работы с базой данных, например, для аутентификации пользователей.
3. **Поле \_currentTime**
   1. Хранит текущее время, форматированное как строка.
4. **Метод initState**
   1. initState вызывается при инициализации состояния виджета. В данном случае, он запускает метод \_startClock, который обновляет текущее время каждую секунду.
5. **Метод \_startClock**
   1. Запускает таймер, который каждую секунду обновляет \_currentTime, форматируя текущее время с помощью DateFormat.
6. **Метод build**
   1. build строит виджет Scaffold, который представляет основной экран для входа.
   2. **AppBar**:
      1. Заголовок с текстом, локализованным с помощью AppLocalizations.
      2. Кнопка для переключения языков.
   3. **body**:
      1. SingleChildScrollView обеспечивает возможность прокрутки содержимого, если оно не помещается на экране.
      2. **Padding**:
         1. Оборачивает содержимое с отступами.
      3. **Column**:
         1. Организует дочерние виджеты вертикально.
      4. **FlutterLogo**:
         1. Отображает логотип Flutter.
      5. **Text**:
         1. Показывает текущее время.
      6. **TextField**:
         1. Поля ввода для email и password. Поле пароля скрывает текст.
      7. **ElevatedButton**:
         1. Кнопка для входа в систему. При нажатии проверяет учетные данные и, если они верны, перенаправляет пользователя на HomeScreen. В случае ошибки отображает сообщение об ошибке.
      8. **ElevatedButton**:
         1. Кнопка для перехода на страницу регистрации (RegisterPage).

Файл login\_page.dart управляет интерфейсом экрана входа в приложение. Он включает в себя:

1. Управление состоянием с помощью StatefulWidget.
2. Локализацию текста и изменение языка через AppLocalizations.
3. Отображение текущего времени с использованием таймера.
4. Проверку учетных данных пользователя с использованием DatabaseHelper.
5. Навигацию между экранами входа, домашним экраном и экраном регистрации.

Дальше давай создадим файл register\_page.dart он представляет собой экран регистрации пользователей. Давайте подробно разберем его, чтобы понять, как он работает и как его элементы взаимодействуют.



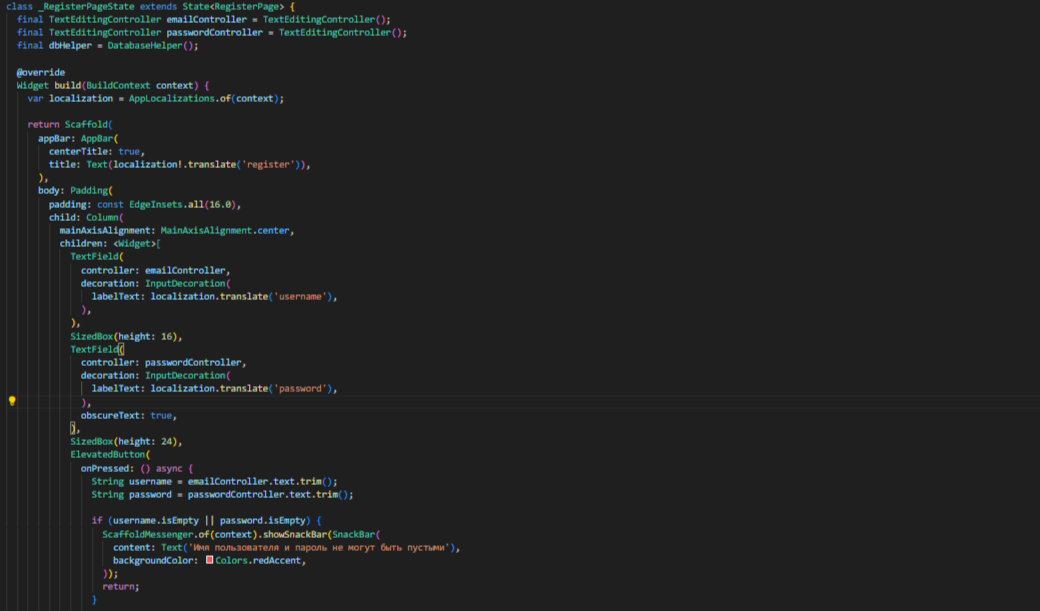
**1. Импорты**

1. import 'package:flutter/material.dart';
   1. Подключает пакет Flutter Material Design для использования виджетов и стилей.
2. import 'database\_helper.dart';
   1. Импортирует DatabaseHelper, который, вероятно, содержит методы для работы с базой данных, включая регистрацию пользователей.
3. import 'app\_localizations.dart';
   1. Импортирует файл с локализациями для поддержки многократных языков.

**2. Класс RegisterPage**

1. RegisterPage — это StatefulWidget, что означает, что он имеет состояние, которое может изменяться.
2. Конструктор принимает функцию onLocaleChange, позволяющую обновлять локаль (язык) приложения из этого виджета.
3. createState создает состояние для RegisterPage, которое управляется \_RegisterPageState.

**3. Класс \_RegisterPageState**

****

****

**Поля и методы класса \_RegisterPageState**

1. **Поле emailController и passwordController**
   1. TextEditingController используется для управления текстовыми полями ввода (email и password).
2. **Поле dbHelper**
   1. DatabaseHelper используется для работы с базой данных, включая регистрацию пользователей.
3. **Метод build**
   1. build создает интерфейс для экрана регистрации, который включает в себя:
      1. **AppBar**:
         1. Заголовок с текстом, локализованным с помощью AppLocalizations.
      2. **body**:
         1. **Padding**:
            1. Оборачивает содержимое с отступами.
         2. **Column**:
            1. Организует дочерние виджеты вертикально.
         3. **TextField**:
            1. Поля ввода для email и password. Поле пароля скрывает текст с помощью obscureText: true.
         4. **ElevatedButton**:
            1. Кнопка для регистрации. При нажатии выполняет следующие действия:

Считывает текст из полей ввода.

Проверяет, что поля не пустые. Если пустые, отображает сообщение об ошибке.

Пытается зарегистрировать пользователя с помощью dbHelper.registerUser(). Если регистрация успешна, отображает сообщение об успехе и возвращает пользователя на предыдущий экран (Navigator.pop(context)).

Если регистрация не удалась (например, пользователь уже существует), отображает сообщение об ошибке.

Файл register\_page.dart управляет интерфейсом экрана регистрации пользователей в вашем приложении. Он включает:

1. Формы для ввода учетных данных пользователя (email и password).
2. Локализацию текста с помощью AppLocalizations.
3. Валидацию ввода (проверка на пустые поля).
4. Регистрацию пользователя через DatabaseHelper.
5. Отображение сообщений об ошибках и успехах с использованием SnackBar.
6. Навигацию обратно на предыдущий экран после успешной регистрации.

Этот файл обеспечивает важную часть процесса регистрации пользователей, позволяя им создать учетную запись и получать соответствующие уведомления.

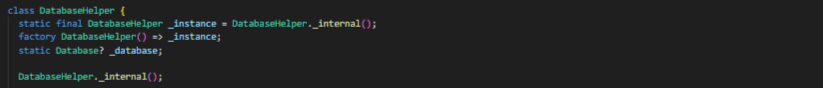
Дальше давайте создадим локальную базу данных , создайте файл database\_helper.dart он содержит реализацию помощника для работы с базой данных SQLite в приложении Flutter. Этот файл обеспечивает взаимодействие с базой данных для регистрации и аутентификации пользователей.



**1. Импорты**

1. import 'package:sqflite/sqflite.dart';
   1. Импортирует пакет sqflite, который предоставляет инструменты для работы с базами данных SQLite в Flutter.
2. import 'package:path/path.dart';
   1. Импортирует пакет path, который используется для работы с путями файлов.

**2. Класс DatabaseHelper**



1. **Паттерн Singleton**
   1. DatabaseHelper реализует паттерн Singleton, что обеспечивает создание только одного экземпляра этого класса. Это полезно для управления подключением к базе данных и предотвращения создания нескольких экземпляров базы данных.
2. **\_instance**:
   1. Статическое поле для хранения единственного экземпляра класса DatabaseHelper.
3. **\_database**:
   1. Статическое поле для хранения экземпляра базы данных SQLite.
4. **DatabaseHelper.\_internal()**:
   1. Конструктор, который инициализирует \_instance.
5. **factory DatabaseHelper() => \_instance;**:
   1. Фабричный конструктор, который возвращает единственный экземпляр класса.

**3. Инициализация базы данных**



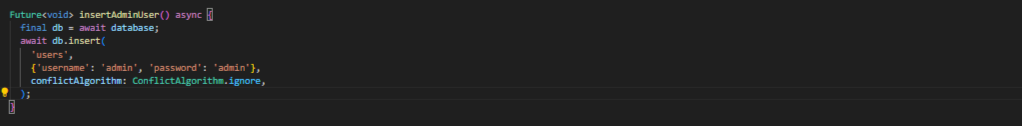
**database**:

1. Геттер для получения экземпляра базы данных. Если база данных уже инициализирована, она возвращается сразу. В противном случае вызывается метод \_initDatabase() для инициализации базы данных.



1. **\_initDatabase**:
   1. Метод для инициализации базы данных.
   2. openDatabase создает и открывает базу данных users.db в указанном пути.
   3. onCreate вызывается при создании новой базы данных и выполняет SQL-запрос для создания таблицы users с полями id, username и password.

**4. Методы для работы с пользователями**



**insertAdminUser**:

1. Метод для вставки учетной записи администратора в базу данных.
2. Использует ConflictAlgorithm.ignore, чтобы игнорировать попытки вставить дублирующую запись.



**authenticateUser**:

1. Метод для проверки учетных данных пользователя.
2. Выполняет запрос к таблице users, чтобы найти пользователя с указанным именем пользователя и паролем.
3. Возвращает true, если пользователь найден, и false в противном случае.



**registerUser**:

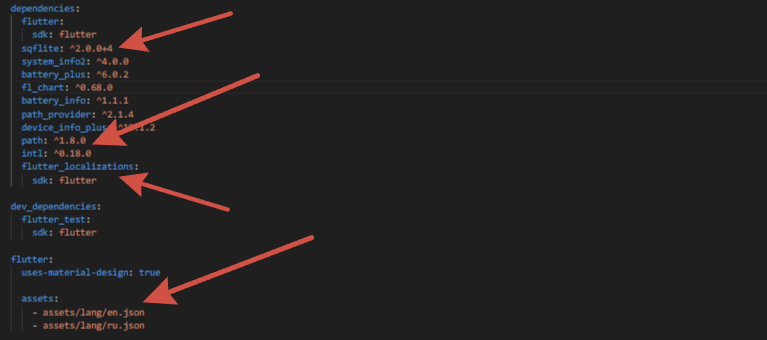
1. Метод для регистрации нового пользователя.
2. Использует conflictAlgorithm: ConflictAlgorithm.fail, чтобы вызывать исключение при попытке вставки пользователя с уже существующим именем пользователя.
3. Если вставка не удалась (например, пользователь уже существует), метод ловит исключение, выводит сообщение об ошибке и генерирует исключение для обработки в вызывающем коде.

Файл database\_helper.dart содержит класс DatabaseHelper, который предоставляет методы для взаимодействия с базой данных SQLite. Этот класс:

1. Обеспечивает реализацию паттерна Singleton для управления экземпляром базы данных.
2. Инициализирует базу данных и создаёт таблицу пользователей при первом запуске.
3. Предоставляет методы для вставки администратора, аутентификации пользователей и регистрации новых пользователей.
4. Обрабатывает возможные ошибки при регистрации и аутентификации.

Этот файл является ключевой частью для работы с данными пользователей в вашем приложении, обеспечивая хранение и управление учетными записями пользователей.

Так же нужно открыть файл pubspec.yaml он является важной частью любого проекта Flutter, поскольку он управляет зависимостями и настройками проекта. В этом файле указываются необходимые пакеты, конфигурации и ресурсы, которые будут использоваться в вашем приложении.



**dependencies**

1. **flutter:**
   1. Указывает на использование SDK Flutter для приложения. Это обязательная зависимость для любого проекта Flutter.
2. **sqflite: ^2.0.0+4**
   1. Пакет для работы с SQLite базами данных в Flutter. Он предоставляет функции для создания, чтения, обновления и удаления данных в SQLite.
3. **path: ^1.8.0**
   1. Пакет для работы с путями файлов. Он помогает создавать и манипулировать путями файлов в Dart.
4. **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.
5. **flutter\_localizations:**
   1. Включает поддержку локализаций в Flutter. Позволяет вашему приложению поддерживать несколько языков и локализаций.

**dev\_dependencies**

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

**flutter**

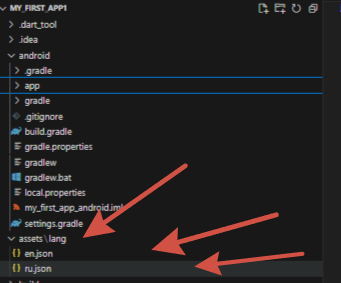
1. **uses-material-design: true**
   1. Указывает, что ваше приложение использует Material Design, что позволяет использовать различные компоненты и стили Material Design в вашем приложении.
2. **assets:**
   1. Определяет список ресурсов (файлов), которые будут включены в приложение и доступны для использования.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

Файл pubspec.yaml управляет зависимостями вашего проекта Flutter и конфигурациями. Он включает:

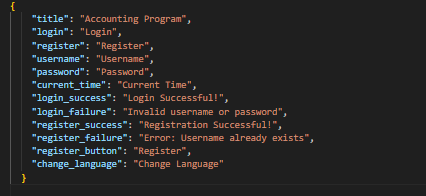
1. **dependencies**: список всех пакетов, которые необходимы для работы вашего приложения.
2. **dev\_dependencies**: пакеты, используемые для разработки и тестирования приложения.
3. **flutter**: настройки, такие как использование Material Design и указание ресурсов (например, локализационных файлов).

Корректное управление зависимостями и настройками в pubspec.yaml обеспечивает правильную работу вашего приложения и его интеграцию с различными библиотеками и инструментами.

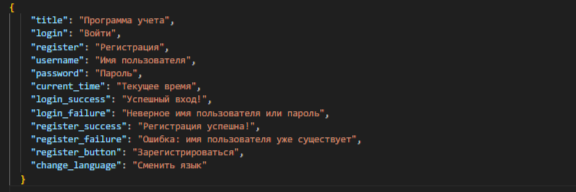
Так же в конфигурации нужно создать папку **assets** для включение в него ресурсов в данный момент нужно создать внутри папку **lang** и в ней два файл **en** и **ru .json**. Эти файлы позволят нам переводить окно входа на два языка.



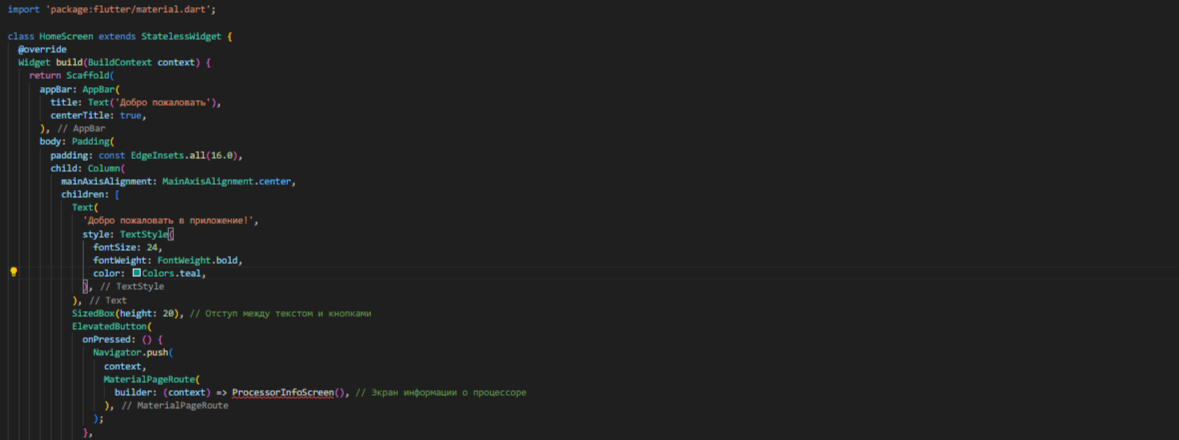
Содержимое файла **en.json** которое нужно заполнить.



Содержимое файла **ru.json** которое нужно заполнить.

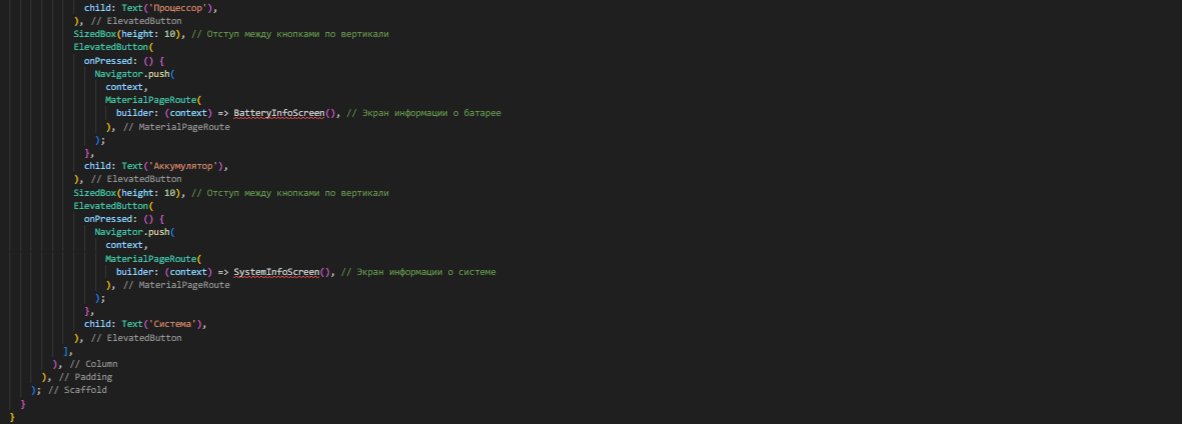


Дальше давайте создадим файл home\_screen.dart он представляет собой виджет HomeScreen, который является экраном приветствия в приложении Flutter. Этот экран предоставляет пользователю три кнопки для перехода на другие экраны, которые отображают информацию о процессоре, батарее и системе.



**Импорт пакета**

1. **import 'package:flutter/material.dart';**
   1. Импортирует пакет material.dart из Flutter, который предоставляет виджеты и темы для использования Material Design в приложении.



**Класс HomeScreen**

1. **class HomeScreen extends StatelessWidget**
   1. Определяет виджет HomeScreen, который наследует от StatelessWidget. Это означает, что виджет не имеет внутреннего состояния и только отображает пользовательский интерфейс.
2. **@override Widget build(BuildContext context)**
   1. Переопределяет метод build, который создает виджеты для отображения в приложении.

**Виджет Scaffold**

1. **Scaffold**
   1. Основной виджет, предоставляющий структуру для экрана, включая AppBar, Drawer, BottomNavigationBar, FloatingActionButton, и основной контент.
2. **appBar: AppBar(...)**
   1. Заголовок экрана с текстом "Добро пожаловать". centerTitle: true центрирует заголовок по горизонтали.

**Основное содержимое экрана**

1. **body: Padding(...)**
   1. Виджет Padding добавляет отступы вокруг дочерних виджетов. В данном случае используется отступ в 16 пикселей со всех сторон.
2. **Column(...)**
   1. Виджет Column размещает дочерние виджеты вертикально.
3. **Text(...)**
   1. Отображает текст "Добро пожаловать в приложение!" с заданным стилем: размер шрифта 24, жирный, и цвет Colors.teal.
4. **SizedBox(height: 20)**
   1. Добавляет отступ между текстом и кнопками.
5. **ElevatedButton(...)**
   1. Кнопки, которые при нажатии выполняют действия:
      1. Первая кнопка для перехода на экран информации о процессоре (ProcessorInfoScreen).
      2. Вторая кнопка для перехода на экран информации о батарее (BatteryInfoScreen).
      3. Третья кнопка для перехода на экран информации о системе (SystemInfoScreen).
   2. **onPressed**:
      1. При нажатии на кнопку используется Navigator.push для перехода к новому экрану. MaterialPageRoute создает новый маршрут для навигации.
   3. **child: Text('Процессор')**:
      1. Указывает текст, отображаемый на кнопке.
6. **SizedBox(height: 10)**
   1. Добавляет отступ между кнопками по вертикали.

**Тестирование и отладка.**

Теперь нужно очистить кеш с помощью команды flutter clean, далее нужно собрать проект с помощью команды flutter pub get, после чего запустите проект с помощью команды flutter run, тут уже вы выбираете либо через эмулятор андроид , но лучше подключить свой телефон (Андроид) и запустить через него, так как это будет быстрее запустить проект , не жили эмулятор.

**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?