ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Омский государственный университет путей сообщения»

(ОмГУПС (ОмИИТ))

Кафедра «Информационная безопасность»

РАЗРАБОТКА ПРИЛОЖЕНИЯ «МЕНЕДЖЕР ПАРОЛЕЙ»

Пояснительная записка к курсовой работе

По дисциплине «технология и методы программирования»

Студент гр. 23-п \_\_\_\_\_\_\_\_\_\_\_ Е.Г. Лукьяненко

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2025г.

Проверил к.т.н., зав. кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Елизаров

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2025г.

Омск 2025

Федеральное агентство железнодорожного транспорта

Федеральное государственное бюджетное образовательное учреждение

высшего образования  
«Омский государственный университет путей сообщения (ОмГУПС (ОмИИТ))»

Кафедра «Информационная безопасность»

Задание

на курсовую работу

по дисциплине «Технология и методы программирования»

Студент группы 23-п Лукьяненко Егор Григорьевич

1. Тема курсовой работы

*Разработка приложения «Менеджер паролей»*

1. Срок сдачи студентом законченной курсовой работы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Исходные данные к курсовой работе

1. Содержание пояснительной записки (перечень подлежащих разработке вопросов):

­– генерация надежных паролей;

– сохранение паролей в памяти устройства;

– защита паролей от всевозможных атак;

– ограничение доступа к работе с приложением.

5. Перечень графического материала (в приложении пояснительной записки):

– диаграммы, схемы программы;

– подробная блок-схема ключевого фрагмента программы.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Реферат

УДК 004.056.52

Курсовая работа содержит 19 страниц, 3 рисунков, 4 библиографических источника.

Объект исследования – приложение “менеджер паролей”, хранение паролей, защита паролей.

Предмет исследования – программная реализация приложения “менеджер паролей”.

Содержание

[Введение 5](#_Toc199427607)

[1 Обзор литературы 6](#_Toc199427608)

[2 Теоретическая часть 6](#_Toc199427609)

[3 Практическая часть 8](#_Toc199427611)

[3.1 Анализ предметной области и требований к программе 8](#_Toc199427612)

[3.2 Выбор языка программирования и среды разработки 8](#_Toc199427613)

[3.3 Реализация программных блоков 10](#_Toc199427614)

[3.4 Тестирование и отладка программы 11](#_Toc199427615)

[Заключение 13](#_Toc199427616)

[Библиографический список 14](#_Toc199427617)

Введение

Менеджер паролей — это специализированное программное обеспечение, предназначенное для безопасного хранения и управления паролями, а также другими конфиденциальными данными, такими как номера кредитных карт, PIN-коды и заметки. Он помогает пользователям создавать сложные пароли, запоминать их и автоматически заполнять формы входа на сайтах и приложениях.

1. Обзор литературы

Менеджеры паролей — это инструменты, предназначенные для безопасного хранения и управления паролями пользователей. Их основная цель — упростить процесс создания, хранения и использования сложных паролей, что помогает защитить личные данные и аккаунты от несанкционированного доступа.

Хорошим примером такого приложения может послужить Proton pass. Proton Pass — это менеджер паролей, разработанный компанией Proton, известной своими акцентами на конфиденциальности и безопасности данных. Вот несколько ключевых аспектов Proton Pass:

1. Конфиденциальность: Proton Pass использует сквозное шифрование, что означает, что только пользователь может получить доступ к своим паролям. Даже разработчики Proton не могут видеть ваши данные;
2. Интеграция с другими продуктами: Proton Pass интегрируется с другими сервисами Proton, такими как Proton Mail и Proton Drive, обеспечивая комплексное решение для управления личной информацией;
3. Удобство использования: Интерфейс Proton Pass интуитивно понятен, что делает его доступным для пользователей с разным уровнем технической подготовки;
4. Безопасность: Proton Pass предлагает дополнительные функции безопасности, такие как возможность восстановления паролей через безопасные каналы.
5. Теоретическая часть

2.1 Анализ функционала приложения «менеджер паролей»

Основные функции менеджера паролей:

**Хранение паролей**:

1. Менеджер паролей хранит все ваши пароли в зашифрованном виде, защищая их от несанкционированного доступа;
2. Данные обычно защищены мастер-паролем, который известен только вам.

**Генерация сложных паролей**:

Программа может создавать уникальные и сложные пароли, которые трудно взломать.

**Анализ безопасности**:

Некоторые менеджеры паролей предоставляют отчеты о надежности ваших паролей и предупреждают о повторяющихся или устаревших паролях.

Преимущества использования менеджера паролей:

**Улучшение безопасности**: Использование уникальных и сложных паролей для каждого аккаунта снижает риск взлома.

**Удобство**: Вам не нужно запоминать десятки паролей — достаточно помнить один мастер-пароль.

**Экономия времени**: Автозаполнение форм входа ускоряет процесс авторизации.

**Защита от фишинга**: Менеджер паролей заполняет данные только на доверенных сайтах, что снижает риск попадания на фишинговые страницы.

1. Практическая часть

# Анализ предметной области и требований к программе

Анализ предметной области:

Основные сущности:

1. Пароли;
2. Логины;
3. Название сайта и/или приложения.

Отношения между сущностями:

1. К каждому паролю прикрепляется логин и название сайта и/или приложения в котором он зарегестрирован.

Требования к программе;

1. Хранение паролей во внутренней памяти устройства;
2. Хеширование паролей;
3. Доступ к приложению через мастер-пароль;
4. Простой и понятный интерфейс.

# Выбор языка программирования и среды разработки

Для реализации программы использовался язык программирования Dart. Dart — это современный язык программирования, разработанный компанией Google. Он имеет множество преимуществ и возможностей, которые делают его привлекательным для разработчиков. Вот основные из них:

1. **Кроссплатформенная разработка**

Dart является основным языком для разработки приложений с использованием Flutter, что позволяет создавать кроссплатформенные приложения для iOS, Android, веба и настольных систем с единой кодовой базой.

2. **Простота и удобство**

**Легкий синтаксис**: Dart имеет чистый и понятный синтаксис, что облегчает изучение языка для новичков.

**Современные функции**: Поддержка асинхронного программирования, расширяемых типов, а также встроенные коллекции и функции для работы с ними.

3. **Высокая производительность**

**JIT и AOT компиляция**: Dart поддерживает как Just-In-Time (JIT) компиляцию для быстрого запуска и отладки, так и Ahead-Of-Time (AOT) компиляцию для создания высокопроизводительных приложений.

**Оптимизация производительности**: Dart оптимизирован для работы в браузерах и на мобильных устройствах, что обеспечивает высокую скорость выполнения кода.

4. **Безопасность типов**

**Статическая типизация**: Dart поддерживает статическую типизацию, что помогает выявлять ошибки на этапе компиляции, улучшая надежность кода.

**Поддержка null-safety**: Это позволяет избежать распространенных ошибок, связанных с использованием null-значений, что делает код более безопасным.

5. **Инструменты и экосистема**

**Разнообразие библиотек**: Dart имеет обширную стандартную библиотеку и множество пакетов, доступных через Pub, что облегчает разработку.

**Инструменты разработки**: Поддержка таких инструментов, как Dart DevTools, позволяет разработчикам легко отлаживать и профилировать свои приложения.

6. **Сообщество и поддержка**

**Активное сообщество**: Сообщество Dart активно и постоянно растет, что обеспечивает поддержку и обмен знаниями между разработчиками.

**Документация**: Обширная и хорошо структурированная документация делает изучение языка и его возможностей более доступным.

7. **Поддержка веб-разработки**

**Dart для веба**: Dart можно использовать для создания веб-приложений, что позволяет использовать один язык для фронтенда и бэкенда.

Для реализации программы использовалась визуальная среда Visual Studio Code(далее – VSCode). **VSCode** — это мощный и легкий редактор кода, разработанный Microsoft. Он популярен среди разработчиков благодаря своей гибкости, настраиваемости и множеству встроенных функций. VSCode имеет интеграции с Dart и Flutter.

# Реализация программных блоков

Исходный код программы предоставлен в приложении А.

Класс MyApp:

**Описание**: Этот класс является корневым виджетом приложения. Он отвечает за инициализацию приложения и определение его основной темы и домашнего экрана.

**Метод** build: Создает экземпляр MaterialApp, устанавливая заголовок приложения, тему и начальный экран (экран входа).

Класс LoginScreen:

**Описание**: Этот класс представляет экран входа, где пользователи могут ввести свой мастер-пароль для доступа к менеджеру паролей.

**Метод** createState: Создает состояние для LoginScreen, возвращая экземпляр \_LoginScreenState.

Класс \_LoginScreenState:

**Описание**: Состояние для LoginScreen, которое управляет логикой аутентификации и взаимодействием с пользователем.

**Поля**:

\_passwordController: Контроллер для текстового поля ввода пароля.

masterPasswordHash: Хэш мастер-пароля, загружаемый из файла.

**Методы**:

\_loadMasterPassword: Асинхронно загружает хэш мастер-пароля из локального файла. Если файл существует, считывает содержимое и обновляет masterPasswordHash.

\_saveMasterPassword: Асинхронно сохраняет новый мастер-пароль в локальный файл после его хэширования. Обновляет masterPasswordHash после сохранения.

\_login: Проверяет введенный мастер-пароль. Если он совпадает с загруженным хэшем, переходит на экран менеджера паролей. Если нет, отображает сообщение об ошибке.

\_showSetMasterPasswordDialog: Отображает диалоговое окно для установки нового мастер-пароля, где пользователь может ввести и сохранить новый пароль.

**Метод** build: Создает интерфейс экрана входа, включая текстовое поле для ввода пароля и кнопку для входа.

Функция \_hashPassword:

**Описание**: Хэширует переданный пароль с использованием алгоритма SHA-256 и возвращает его в виде строки.

Класс PasswordManagerScreen:

**Описание**: Этот класс представляет экран менеджера паролей, где пользователи могут управлять своими паролями.

**Метод** createState: Создает состояние для PasswordManagerScreen, возвращая экземпляр \_PasswordManagerScreenState.

Класс \_PasswordManagerScreenState:

**Описание**: Состояние для PasswordManagerScreen, которое управляет логикой отображения и управления паролями.

**Поля**:

\_passwords: Список, содержащий пароли, хранящиеся в виде карт (логин и пароль).

**Методы**:

initState: Вызывается при инициализации состояния, загружает пароли из локального файла.

\_loadPasswords: Асинхронно загружает пароли из локального файла и обновляет состояние \_passwords.

\_savePasswords: Асинхронно сохраняет текущий список паролей в локальный файл.

\_addPassword: Отображает диалоговое окно для добавления нового пароля. После ввода логина и пароля добавляет их в список и сохраняет.

\_deletePassword: Удаляет пароль по индексу из списка и обновляет файл с паролями.

\_copyPassword: Копирует указанный пароль в буфер обмена и отображает сообщение о подтверждении.

**Метод** build: Создает интерфейс экрана менеджера паролей, включая список паролей и кнопки для добавления, копирования и удаления паролей.

# Тестирование и отладка программы

На рисунке 1 предоставлен снимок программы с видом экрана входа.

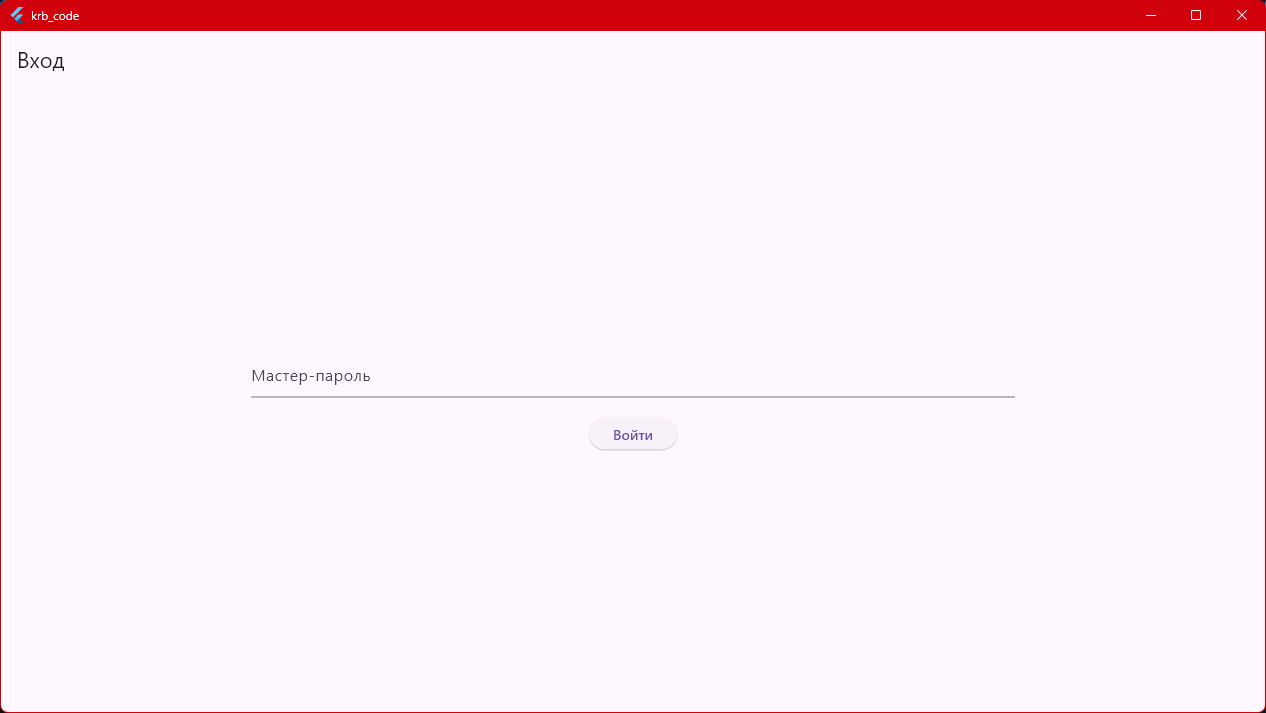


Рисунок 1 – Экран входа в приложение

На рисунке 2 предоставлен снимок программы с видом диалогового окна для задания значения мастер-пароля.

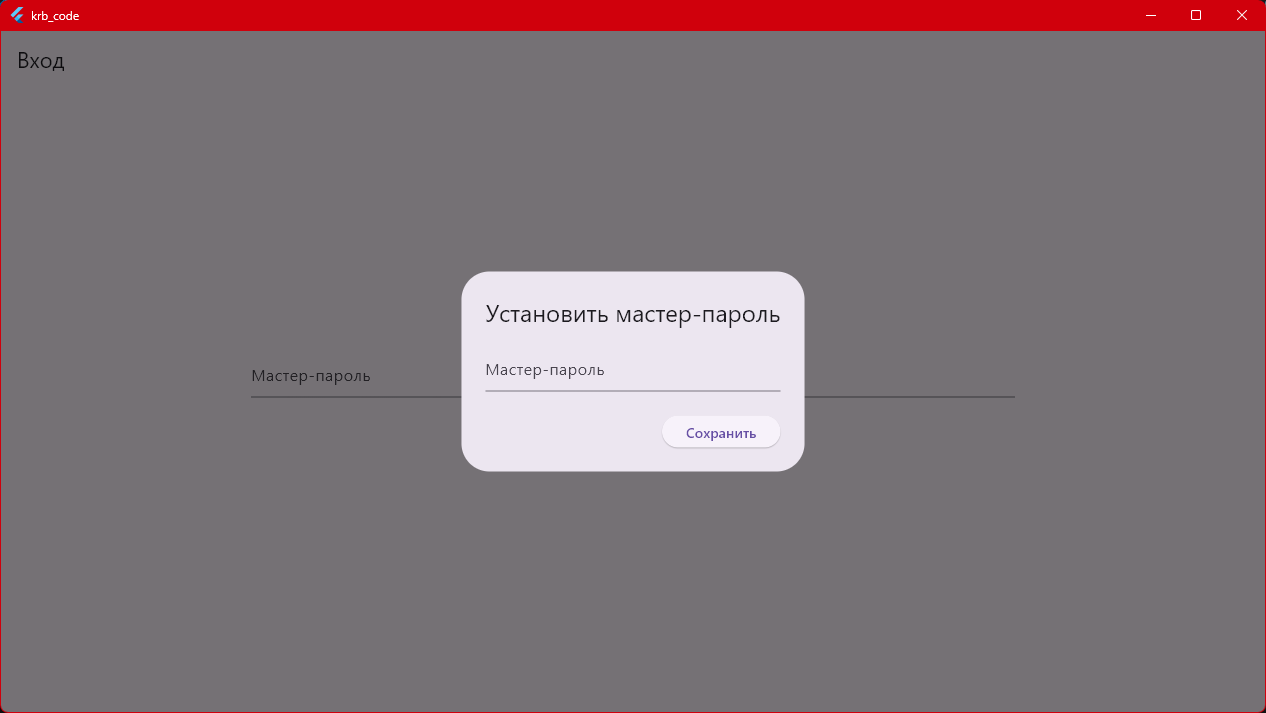


Рисунок 2 – Диалоговое окно для задания мастер-пароля

На рисунке 3 предоставлен снимок экрана основного экрана приложения с двумя экземплярами логинов(ни один логин и пароль не являются действительными и являются исключительно тестовыми).



Рисунок 3 – Основной экран менеджера паролей

Заключение

В ходе разработки приложения «Менеджер паролей» была проведена комплексная работа, охватывающая различные аспекты программирования и проектирования программного обеспечения. В результате была создана функциональная и безопасная программа, которая позволяет пользователям удобно управлять своими паролями. Проект не только углубил знания в области разработки программного обеспечения, но и стал ценным опытом в решении реальных задач, связанных с безопасностью и удобством использования.

Библиографический список

1. «руководство по языку Dart», URL:<https://metanit.com/dart/tutorial>

2. «Dart Documentation», URL: <https://dart.dev/docs>

3. «Руководство по фреймворку Flutter» URL: <https://metanit.com/dart/flutter/>

4. «Flutter Documentation», URL: <https://docs.flutter.dev/>

Приложение А

(обязательное)

Исходный код программы

import 'dart:convert'; //для конвертирования данных в нужную кодировку

import 'dart:io'; //для ввода информации с текстовых полей

import 'package:flutter/material.dart'; //для работы приложения

import 'package:path\_provider/path\_provider.dart'; //для работы с путями

import 'package:crypto/crypto.dart'; //для хеширования паролей

import 'package:flutter/services.dart';//для работы с буфером обмена

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Менеджер паролей',

      theme: ThemeData(

        primarySwatch: Colors.blue,

      ),

      home: LoginScreen(),

    );

  }

}

//класс экрана входа

class LoginScreen extends StatefulWidget {

  const LoginScreen({super.key});

  @override

  \_LoginScreenState createState() => \_LoginScreenState();

}

class \_LoginScreenState extends State<LoginScreen> {

  final TextEditingController \_passwordController = TextEditingController();

  String? masterPasswordHash;

  //функция для чтения мастер-пароля

  Future<void> \_loadMasterPassword() async {

    final directory = await getApplicationDocumentsDirectory();

    final file = File('${directory.path}/master\_password.json');

   if (await file.exists()) {

      String contents = await file.readAsString();

      masterPasswordHash = json.decode(contents)['hash'];

    }

  }

  //функция для записи мастер-пароля(активируется в том случае если он еще не задан)

  void \_saveMasterPassword(String password) async {

    final directory = await getApplicationDocumentsDirectory();

    final file = File('${directory.path}/master\_password.json');

    await file.writeAsString(json.encode({'hash': \_hashPassword(password)}));

    String contents = await file.readAsString();

    masterPasswordHash = json.decode(contents)['hash'];

  }

  //функция для аутентификации

  void \_login() {

    \_loadMasterPassword();

    if (masterPasswordHash == null) {

      \_showSetMasterPasswordDialog();

    }

    else {

      final inputPasswordHash = \_hashPassword(\_passwordController.text);

      if (inputPasswordHash == masterPasswordHash) {

        Navigator.pushReplacement(

          context,

          MaterialPageRoute(builder: (context) => PasswordManagerScreen()),

        );

      }

      else {

        ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Неверный мастер-пароль')));

      }

    }

  }

  //функция для диалогового окна, предназначенного для записи мастер-пароля

  void \_showSetMasterPasswordDialog() {

    showDialog(

      context: context,

      builder: (context) {

        String newPassword = '';

        return AlertDialog(

          title: Text('Установить мастер-пароль'),

          content: TextField(

            onChanged: (value) => newPassword = value,

            decoration: InputDecoration(labelText: 'Мастер-пароль'),

            obscureText: true,

          ),

          actions: [

            ElevatedButton(

              onPressed: () {

                \_saveMasterPassword(newPassword);

                Navigator.of(context).pop();

                ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Мастер-пароль установлен')));

              },

              child: Text('Сохранить'),

            ),

          ],

        );

      },

    );

  }

  @override

  Widget build(BuildContext context) {

    \_loadMasterPassword();

    return Scaffold(

      appBar: AppBar(

        title: Text('Вход'),

      ),

      body: Center(

        child: Padding(

          padding: const EdgeInsets.all(250.0),

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              TextField(

                controller: \_passwordController,

                decoration: InputDecoration(labelText: 'Мастер-пароль'),

                obscureText: true,

              ),

              SizedBox(height: 20),

              ElevatedButton(

                onPressed: \_login,

                child: Text('Войти'),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

//функия для хэширования паролей

String \_hashPassword(String password) {

  return sha256.convert(utf8.encode(password)).toString();

}

//класс экрана менеджера паролей

class PasswordManagerScreen extends StatefulWidget {

  const PasswordManagerScreen({super.key});

  @override

  \_PasswordManagerScreenState createState() => \_PasswordManagerScreenState();

}

class \_PasswordManagerScreenState extends State<PasswordManagerScreen> {

  List<Map<String, dynamic>> \_passwords = [];

  @override

  void initState() {

    super.initState();

    \_loadPasswords();

  }

  //функция для чтения всех локальных паролей

  Future<void> \_loadPasswords() async {

    final directory = await getApplicationDocumentsDirectory();

    final file = File('${directory.path}/passwords.json');

    if (await file.exists()) {

      String contents = await file.readAsString();

      setState(() {

        \_passwords = List<Map<String, dynamic>>.from(jsonDecode(contents));

      });

    }

  }

  //функция дл сохранения всех локальных паролей

  Future<void> \_savePasswords() async {

    final directory = await getApplicationDocumentsDirectory();

    final file = File('${directory.path}/passwords.json');

    await file.writeAsString(json.encode((\_passwords)));

  }

  //функция для добавления нового пароля

  void \_addPassword() {

    showDialog(

      context: context,

      builder: (context) {

        String login = '';

        String password = '';

        return AlertDialog(

          title: Text('Добавить новый пароль'),

          content: Column(

            mainAxisSize: MainAxisSize.min,

            children: [

              TextField(

                onChanged: (value) => login = value,

                decoration: InputDecoration(labelText: 'Логин'),

              ),

              TextField(

                onChanged: (value) => password = value,

                decoration: InputDecoration(labelText: 'Пароль'),

                obscureText: true,

              ),

            ],

          ),

          actions: [

            ElevatedButton(

              onPressed: () {

                setState(() {

                  \_passwords.add({'login': login, 'password': password});

                  \_savePasswords();

                });

                Navigator.of(context).pop();

              },

              child: Text('Сохранить'),

            ),

          ],

        );

      },

    );

  }

  //функция для удаления пароля из памяти

  void \_deletePassword(int index) {

    setState(() {

      \_passwords.removeAt(index);

      \_savePasswords();

    });

  }

  //функция для копирования пароля в буфер обмена

  void \_copyPassword(String password) {

    Clipboard.setData(ClipboardData(text: password)).then((\_) {

      ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Пароль скопирован в буфер обмена')));

    });

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text('Менеджер паролей'),

        actions: [

          IconButton( //кпонка для добавления нового пвроля

            onPressed: \_addPassword,

            icon: Icon(Icons.add),

            alignment: Alignment.center,

          )

        ],

      ),

      body: ListView.builder(

        itemCount: \_passwords.length,

        itemBuilder: (context, index) {

          return ListTile(

            title: Text(\_passwords[index]['login']!,selectionColor: Colors.blueAccent,),

            trailing: Row(

              mainAxisSize: MainAxisSize.min,

              children: [

                IconButton( //кнопка для копирования пароля в буфер обмена

                  icon: Icon(Icons.copy, color: Colors.blue),

                  onPressed: () => \_copyPassword(\_passwords[index]['password']!),

                ),

                IconButton( //кнопка для удавления пароля из памяти

                  icon: Icon(Icons.delete, color: Colors.red),

                  onPressed: () => \_deletePassword(index),

                ),

              ],

            ),

          );

        },

      ),

    );

  }

}