ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Омский государственный университет путей сообщения»

(ОмГУПС (ОмИИТ))

Кафедра «Информационная безопасность»

РЕШЕНИЯ ЗАДАЧ КИБЕРБЕЗОПАСНОСТИ С ПРИМЕНЕНИЕМ ЯЗЫКА ППРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ

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

Кафедра «Информационная безопасность»

Задание

на курсовую работу

по дисциплине «Технологии и методы программирования»

Тема курсовой работы

Реализация шифрования и дешифрования текста с использованием шифра Цезаря

Цель работы

*Разработка приложения «Менеджер паролей»*

Срок сдачи студентом законченной курсовой работы: 31 мая 2025 г

Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

– титульный лист;

– задание;

– реферат;

– содержание;

– введение;

– теоретическая часть;

– практическая часть (алгоритм и описание программы, тестирование);

– заключение;

– библиографический список;

– приложения (руководство пользователя и т. п.).

5. Перечень графического материала (в приложении пояснительной записки)

– исходные кода программы;

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

*(подпись)*

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

(дата)

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

(подпись)

Реферат

УДК 004.056.52

Курсовая работа содержит 29 страниц, 3 рисунка, 5 библиографических источника, 1 приложение.

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

Предмет исследования – программная реализация приложения “менеджер паролей”.

Содержание

[Введение 5](#__RefHeading___Toc2893_2984651232)

[1 Обзор литературы 6](#__RefHeading___Toc2895_2984651232)

[2 Теоретическая часть 8](#__RefHeading___Toc2897_2984651232)

[2.1 Анализ функционала приложения «менеджер паролей» 8](#__RefHeading___Toc2899_2984651232)

[3 Практическая часть 9](#__RefHeading___Toc2901_2984651232)

[3.1 Анализ предметной области и требований к программе 9](#__RefHeading___Toc2903_2984651232)

[3.2 Выбор языка программирования и среды разработки 9](#__RefHeading___Toc2905_2984651232)

[3.3 Реализация программных блоков 10](#__RefHeading___Toc3004_1555932910)

[3.4 Тестирование и отладка программы 19](#__RefHeading___Toc2909_2984651232)

[Заключение 21](#__RefHeading___Toc2911_2984651232)

[Библиографический список 22](#__RefHeading___Toc2913_2984651232)

[Приложение А 23](#__RefHeading___Toc3013_794317725)

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.

Ниже приведены несколько популярных примеров приложения «менеджер паролей»

LastPass: Предлагает облачное хранение паролей с доступом с любого устройства, что обеспечивает удобство и мобильность пользователя. Использует современные стандарты шифрования и многофакторную аутентификацию для защиты данных. Встроенный генератор паролей позволяет создавать уникальные и сложные пароли для всех учетных записей. Автоматическое заполнение форм и логинов экономит время и снижает вероятность ошибок при вводе данных. Реализована функция мониторинга утечек данных в даркнете и система мгновенных оповещений о компрометации учетных данных. Предусмотрена возможность безопасного обмена паролями между пользователями и предоставления экстренного доступа доверенным лицам. Для корпоративных клиентов доступны централизованное управление политиками безопасности, интеграция с каталогами пользователей и расширенные административные инструменты.

1Password: Высокий уровень безопасности за счет использования архитектуры "нулевого знания" (Zero-Knowledge), при которой даже разработчики не имеют доступа к вашим данным. Поддержка совместной работы в командах: можно безопасно делиться паролями и другими конфиденциальными данными между сотрудниками. Интеграция с корпоративными инструментами и поддержка единого входа (SSO), что облегчает внедрение в бизнес-процессы. Функция Watchtower отслеживает утечки данных и предлагает рекомендации по усилению безопасности. Кроссплатформенность: поддержка всех основных операционных систем и браузеров.

Dashlane: Интуитивно понятный интерфейс и удобство использования как на компьютерах, так и на мобильных устройствах. Функция "Password History" позволяет восстанавливать ранее использованные пароли, что полезно при случайной потере доступа. Персонализированный анализ безопасности паролей с приоритетом для наиболее важных учетных записей (например, банковских и почтовых). Мониторинг даркнета с уведомлениями о возможных утечках персональных данных. Встроенный VPN для защиты интернет-соединения, особенно при использовании открытых Wi-Fi сетей. Реализована функция безопасного обмена паролями и цифровым кошельком для хранения платежных данных. Поддержка входа без пароля и многофакторной аутентификации.

Bitwarden: Открытый исходный код, что позволяет проводить независимый аудит безопасности. Возможность самостоятельного размещения сервера (self-hosting), что особенно важно для организаций с повышенными требованиями к контролю над данными. Бесплатные и платные версии для частных лиц и организаций, гибкая система лицензирования. Поддержка всех популярных платформ и браузеров. Функции генерации паролей, автоматического заполнения и безопасного обмена данными.

KeePass: Бесплатное программное обеспечение с открытым исходным кодом, не требующее облачного хранения — все данные хранятся локально на устройстве пользователя. Расширенные возможности настройки безопасности, включая поддержку различных алгоритмов шифрования. Большое количество плагинов и расширений для интеграции с другими сервисами и повышения функциональности. Не требует подключения к интернету для работы, что минимизирует риск удаленного взлома. Поддержка портативного режима, позволяющего запускать программу с внешних носителей.

1. 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.

Подключим необходимые библиотеки

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

Листинг 3.1 — подключение библиотек

Библиотека «dart:convert» необходима для конвертирования полученных от пользователя данный в кодировку UTF-8. Библиотека «dart:io» необходима для получения данных из текстовых полей. Библиотека «package:flutter/material.dart» необходима для корректной работы приложения, в нее входят весь функционал для отрисовки графического интерфйса. Библиотека «package:path\_provider/path\_provider.dart» необходима для получения пути к корневой папке приложения и создания в ней файлов. Библиотека «package:crypto/crypto.dart» необходима для обеспечения шифрования паролей и хеширования мастер-пароля. Библиотека «package:flutter/services.dart» необходима для копирования паролей в буфер обмена.

Вызвем функцию запуска приложения.

void main() {

runApp(MyApp());

}

Листинг 3.2 — Вызов функции запуска приложения, лист 1

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(),

);

}

}

Листинг 3.2, лист 2

Данная функция вызывает класс LoginScreen как домашний экран приложения а также задает заголовок приложения и базовые настройки цветовой темы приложения. Рассмотрим класс LoginScreen.

//класс экрана входа

class LoginScreen extends StatefulWidget {

const LoginScreen({super.key});

@override

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

}

Листинг 3.3 — класс LoginScreen, лист 1

Класс LoginScreen расширяет класс StatefulWidget и имеет единственные параметр «состояние»

class \_LoginScreenState extends State<LoginScreen> {

final TextEditingController \_passwordController = TextEditingController();

String? masterPasswordHash;

Листинг 3.3, лист 2

Класс \_LoginScreenState расширяет «состояние» класса LoginScreen, именно класс \_LoginScreenState содержит исполняемый код виджета и функции, необходимые для экрана входа.

//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'];

}

}

Листинг 3.3, лист 3

Функция \_loadMasterPassword выполняет загрузку мастер-пароля из файла в память приложения.

//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'];

}

Листинг 3.3, лист 4

Функция \_saveMasterPassword выполняет хеширование и запись мастер пароля в файл master\_password.json.

//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')));

}

}

}

Листинг 3.3, лист 5

Функция \_Login получает из текстового поля мастер-пароль, хеширует его и сравнивает с хешом записаного в файле мастер-пароля; если хеш совпадает то производит переход на страницу менеджер паролей, в противном случае выводит уведомление «Неверный мастер-пароль»; в случае если файл с хешем мастер-пароля пуст или не существует вызывает функцию \_ShowSetMasterPasswordDialog.

//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'),

),

],

);

},

);

}

Листинг 3.3, лист 6

Функция \_ShowSetMasterPasswordDialog создает диалоговое окно для задания мастер-пароля, после ввода мастер-пароля и нажатия на кнопку «сохранить» диалоговое окно закроется и выведется уведомление «Мастер-пароль установлен».

@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'),

),

],

),

),

),

);

}

}

Листинг 3.3, лист 7

Функция Build имеет тип Widget и содержит в себе инструкции для построение интерфейса экрана входа приложения.

//функия для хэширования паролей

String \_hashPassword(String password) {

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

}

Листинг 3.4 — функция для хеширования паролей

функция \_hashPassword выполняет конвертацию паролей в кодировку UTF-8 и хеширование с помощью алгоритма SHA-256.

//класс экрана менеджера паролей

class PasswordManagerScreen extends StatefulWidget {

const PasswordManagerScreen({super.key});

@override

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

}

Листинг 3.5 — класс \_PasswordManagerScreen, лист 1

Класс PasswordManagerScreen расширяет класс StatefulWidget и имеет единственные параметр «состояние»

class \_PasswordManagerScreenState extends State<PasswordManagerScreen> {

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

@override

void initState() {

super.initState();

\_loadPasswords();

}

Листинг 3.5, лист 2

Класс \_PasswordManagerScreenState расширяет «состояние» класса LoginScreen, именно класс PasswordManagerScreen содержит исполняемый код виджета и функции, необходимые для экрана входа.

//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));

});

}

}

Листинг 3.5, лист 3

Функция \_loadPasswords загружает пароли и логины из файла passwords.json в память приложения.

//функция для 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)));

}

Листинг 3.5, лист 4

Функция \_savePasswords выполняет запись логинов и паролей в файл passwords.json

//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'),

),

],

);

},

);

}

Листинг 3.5, лист 5

функция \_addPassword создает диалоговое окно с двумя текстовыми полями, верхнее записывает полученные данные как логин, нижнее записывает полученные данные как пароль к логину; при нажатии на кнопку «Сохранить» диалоговое окно закрывается, пароль шифруется и записывается в файл passwords.json с логином как элемент списка.

//функция для удаления пароля из памяти

void \_deletePassword(int index) {

setState(() {

\_passwords.removeAt(index);

\_savePasswords();

});

}

Листинг 3.5, лист 6

Функция \_deletePassword удаляет данные о логине и пароле из файла passwors.json.

//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')));

});

}

Листинг 3.5, лист 7

Функция \_copyPassword копирует пароль в буфер обмена устройства.

@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),

),

],

),

);

},

),

);

}

}

Листинг 3.5, лист 8

Функция Build имеет тип Widget и содержит в себе инструкции для построение интерфейса экрана менеджера паролей.

# Тестирование и отладка программы

На рисунке 1 предоставлен снимок программы с видом экрана входа.

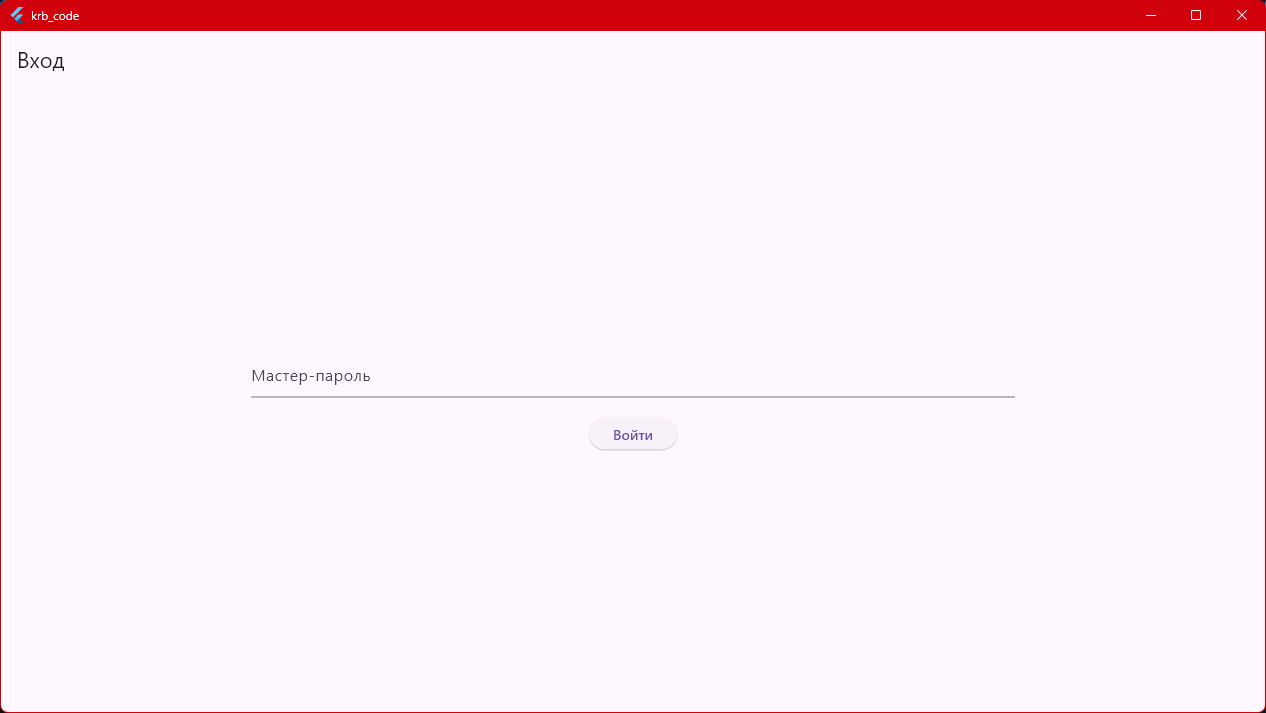


Рисунок 1 – Экран входа в приложение

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

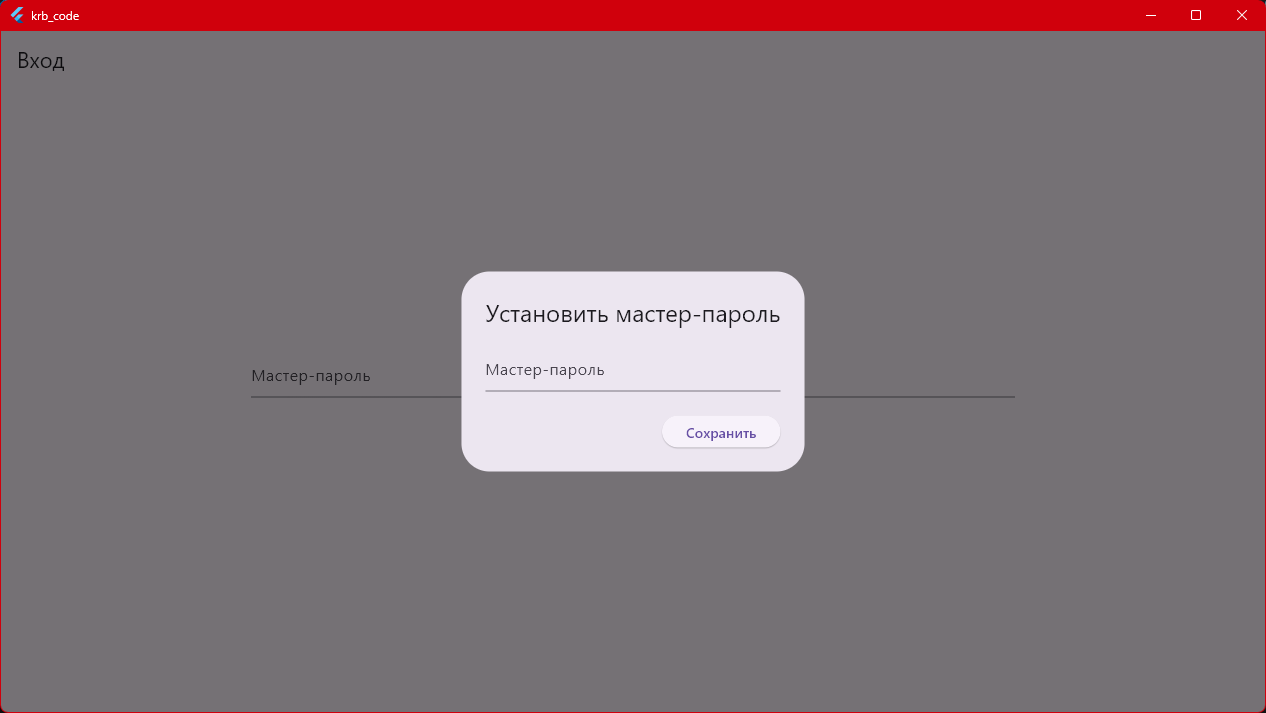


Рисунок 2 – Диалоговое окно для задания мастер-пароля

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», Евгений Попов, metanit.com, 2025, URL: <https://metanit.com/dart/tutorial>

2. «Руководство по фреймворку Flutter», Евгений Попов, metanit.com, 2025, URL: <https://metanit.com/dart/flutter/>

3. Статья «Как защитить данные с помощью менеджера паролей», 2025 АО «Лаборатория Касперского», URL: [https://www.kaspersky.ru/resource-center/preemptive-safety/protecting-your-data-online-password-manager](https://metanit.com/dart/flutter/)

4. Баланов, А. Н. Защита информационных систем. Кибербезопасность: учебное пособие для СПО / А. Н. Баланов. — 2-е изд., стер. — Санкт-Петербург : Лань, 2025. — ISBN 978-5-507-53004-5. URL: [https://e.lanbook.com/book/464183 (дата обращения: 13.06.2025.](https://metanit.com/dart/flutter/)

# 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 {

Листинг А.1 — Исходный код приложения, лист 1

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() {

Листинг А.1, лист 2

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(

Листинг А.1, лист 3

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

Листинг А.1, лист 4

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 для 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,

),

],

Литсинг А.1, лист 5

),

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

Листинг А.1, лист 6

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),

),

],

),

);

},

),

);

}

}

Листинг А.1, лист 7