Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Менеджер паролей

Студент ИУ8-101 (группа) _____ (подпись, дата) М. Е. Шаповалов (И.О. Фамилия)

Руководитель курсовой работы _____ (подпись, дата) А.В. Клюкин (И.О. Фамилия)

Москва, 2025

// ЛИСТ ИЗ

// ЛИСТ КП

Аннотация

В курсовой работе выполняется разработка программного обеспечения «Менеджер паролей» с графическим веб-интерфейсом для хранения паролей в зашифрованном виде, с поддержкой генерации, проверки стойкости и компрометации паролей.

Разработка ведется на платформе .NET на языке C# с расширением в виде библиотек ASP.NET и Entity Framework Core.

Содержание

ВВЕДЕНИЕ

В современном цифровом мире пользователи сталкиваются с необходимостью запоминать множество паролей для различных сервисов и приложений. Это создаёт риски забывания паролей или использования слабых и легко угадываемых комбинаций, что может привести к компрометации личных данных. Менеджер паролей — это специализированное программное обеспечение, которое помогает пользователям безопасно хранить и управлять своими паролями.

Курсовая работа посвящена разработке менеджера паролей, который будет обеспечивать шифрование хранимых данных, будет предоставлять функционал проверки паролей на стойкость и компрометацию, а также функционал генератора паролей. Также для разрабатываемого менеджера паролей будет реализован веб-интерфейс.

ОСНОВНАЯ ЧАСТЬ

В данном описан ход непосредственно разработки ПО «Менеджер паролей».

В первом разделе идет разработка архитектуры разрабатываемой программы: ее функциональные блоки, что в этих блоках должно быть расположено и какой функционал должен быть реализован. В заключение раздела приведена схема архитектуры.

Во втором разделе определяется вспомогательная логика программы – логика, которая не определяет, что должна делать программа, а скорее – как. Определяются методы проверки паролей на стойкость и скомпрометированность, а также методы шифрования данных.

В третьем разделе идет разработка непосредственно программы по разработанной ранее архитектуры. Для разработки выбран язык программирования C#, и разработка ведется на платформе .NET [Ошибка: источник перекрёстной ссылки не найден] с расширением в виде ASP.NET Core [2], Entity Framework Core [3].

1 Разработка архитектуры

Начнем разработку менеджера паролей с разработки архитектуры. Определим 4 основных функциональных блока:

- Блок интерфейса (БИ) – является входной точкой для пользователя, его задача принимать данные от пользователя и адаптировать их в вид, удобный для приложения;
- Блока данных (БД) – блок, отвечающий за хранение зашифрованных данных аккаунтов, а также информации о ключе шифрования;
- Блок вспомогательной логики (БВ) – этот блок обеспечивает БЛ дополнительной логикой: алгоритмы шифрования, алгоритмы проверки стойкости паролей, алгоритмы проверки компрометации паролей; эта логика вынесена в отдельный блок, т. к. ее реализация может меняться независимо от логики БЛ.

- Блок логики (БЛ) – блок программы, содержащий основную бизнес-логику по шифрованию и расшифрованию данных аккаунтов;

Теперь начнем по порядку прорабатывать каждый из блоков.

Будем разрабатывать приложение с веб-интерфейсом, а значит БИ должен состоять из двух основных частей: веб-сайта и веб-сервера. Со стороны веб-сайта должны быть реализована простая логика обработки пользовательского ввода и формирования на основе этого ввода веб-запросов. Со стороны веб-сервера должна быть реализована простая логика обработки запросов и перевода их в объекты, которые могут обрабатывать остальные блоки программы. В качестве основы возьмем паттерн Модель-Представление-Контроллер (MVC), где за модель будет отвечать БЛ, за представление – веб-сайт, а за контроллер – веб-сервер.

БИ должен содержать страницы:

- Страница первичного входа, на которой пользователь должен ввести мастер-пароль для инициализации менеджера паролей;
- Страница входа, на которой пользователь должен ввести мастер-пароль для входа в менеджер паролей;
- Страница выхода, на которой пользователь сможет завершить сессию;
- Страница со списком аккаунтов, которые хранятся в менеджере паролей;
- Страница аккаунта, на которой пользователь сможет посмотреть данные аккаунта и изменить их;
- Страница, предоставляющая функционал генератора паролей;
- Страница настроек, на которой пользователь сможет поменять мастер-пароль и время до автоматического завершения сессии.

БИ должен содержать контроллеры:

- Контроллер аутентификации для входа и выхода пользователя из сессии;
- Контроллер аккаунтов – для добавления, изменения, удаления, запроса аккаунтов;

- Контроллер паролей – для генерации и проверки аккаунтов;
- Контроллер настроек – для запроса и изменения настроек.

Везде, где предполагается ввод неизвестного пароля, должна быть возможность проверки этого пароля на стойкость.

Веб-сервер должен генерировать авторизационную Cookie для пользователя, после успешного входа, а также инвалидировать ее, когда сессия пользователя завершится.

Веб-сервер должен поддерживать работу по протоколу HTTPS, а также обеспечивать защиту от CSRF атак.

Для БД, так как предполагается, что программа предназначена для личного пользования, будем использовать локальное хранилище. Для удобства будем использовать СУБД SQLite [4]. Помимо самой СУБД данный блок предоставляет репозитории, которые переводят модели базы данных в модели, удобные остальной программе:

- Репозиторий аккаунтов – он хранит аккаунты в зашифрованном виде;
- Репозиторий ключа – он хранит информацию о ключе, чтобы после его верифицировать.

Информацией о ключе может быть, например, сам ключ, зашифрованный самим собой; таким образом, если мы смогли расшифровать информацию о ключе и в результате получили сам ключ – то ключ, переданный для проверки – валиден.

В БВ содержатся различные дополнительные методы; в их числе:

- Методы для определения стойкости паролей;
- Методы определения компрометации паролей;
- Методы генерации паролей;
- Методы шифрования данных.

Наконец БЛ – он предоставляет основные сервисы:

- Хранилище ключа – данный сервис должен хранить ключ в течение заданного периода времени и предоставлять его по запросу;

- Сервис аккаунтов – данный сервис должен уметь добавлять аккаунты БД, изменять и удалять их, а также запрашивать; по необходимости он должен шифровать и расшифровывать данные;
- Сервис ключа – данный сервис должен уметь проверять валидность ключа на основе информации о ключе, которая хранится в БД (и подсчитывать число неудачных попыток, чтобы в случае блокировать последующие попытки), либо, если этой информации нету, добавлять ее в БД; он должен уметь менять эту информацию и, как следствие, перешифровывать аккаунты с учетом новой информации; также он должен записывать ключ в хранилище ключа;
- Сервис паролей – данный сервис должен генерировать пароли и проверять их на стойкость;

В итоге получаем примерный вид архитектуры, представленный на рисунке 1.

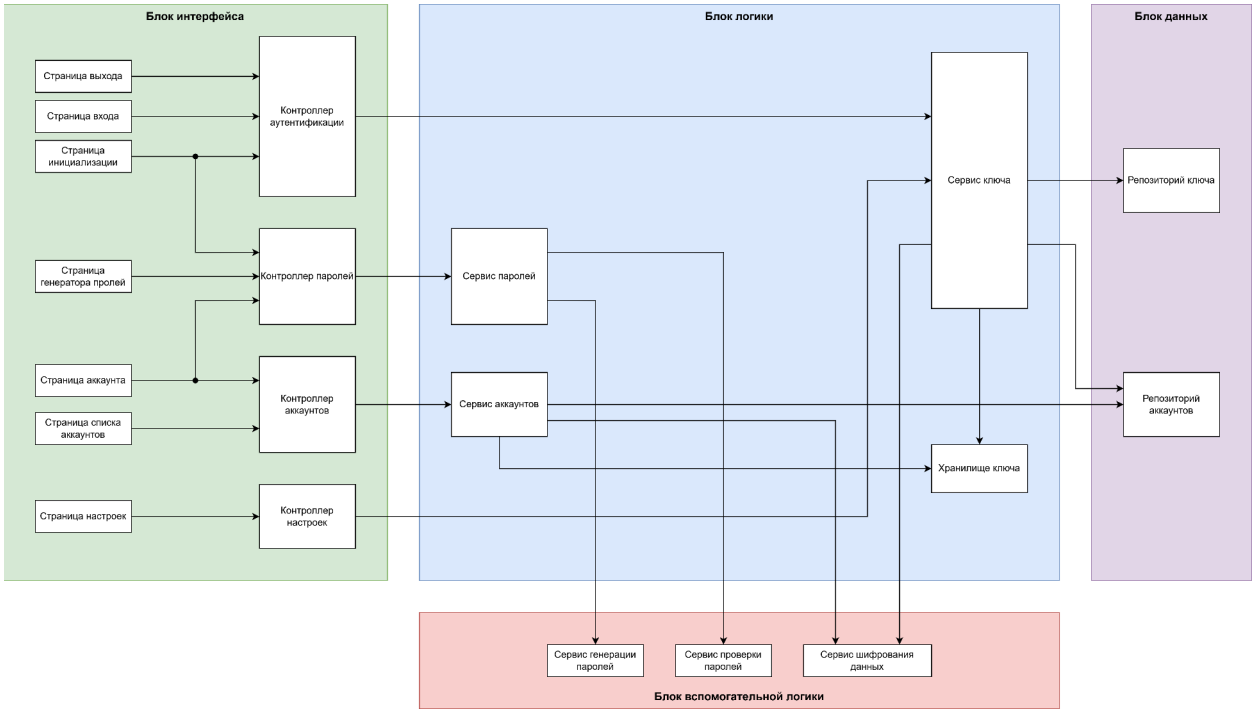

Рисунок 1 – Архитектура программы

2 Определение вспомогательных методов

Как было описано в первом разделе, в БВ расположена вспомогательная логика, которая легко заменяется на любую другую.

Определим заранее, какие именно вспомогательные методы будем использовать при разработке.

Для определения стойкости паролей будем использовать комплексный метод на основе трех подходов.

Первый подход – определение стойкости паролей по энтропии [5]. Данный метод имеет недостаток – он требует для своей работы знание алфавита паролей, что является проблемой: мы не можем знать, какой алфавит использовал пользователь для своих паролей. Ему, конечно, доступны все ASCII символы, но маловероятно, что он их все он принимал во внимание. Тем не менее, этот подход пригодится нам для определения стойкости сгенерированных нами же паролей.

Второй подход – на основе алгоритма проекта SeaMonkey [6]. Этот подход лишен недостатка прошлого и не требует знания алфавита.

Но у этих двух методов есть общий недостаток: они не учитывают словарные пароли, например *qwerty*, *12345*, *password* и т. п. Потому добавим третий подход на основе алгоритма *zxcvbn* [7], который учитывает словарные слова, а также возможные замены символов на похожие, например, замену буквы О на цифру 0.

Таким образом итоговый метод оценки стойкости выглядит так: оценить стойкость тремя подходами и выбрать наименьший из них.

Для оценки компрометации пароля воспользуемся сервисом *';--have i been pwned?* [8], который предоставляет открытое API для проверки скомпрометированных паролей.

Для генерации паролей будем использовать генератор случайных чисел и алфавит, который пользователь сможет настроить в БИ.

Для шифрования данных будем использовать алгоритм AES в режиме CBC с размером ключа 256 бит. Для выработки ключа будем использовать алгоритм PBKDF2 [9] – эта функция хеширование поддерживает «соли», так что у разных пользователей будет свой собственный ключ даже при одинаковом пароле, а также ее рекомендуют использовать NIST [10] и она

входит в состав стандарта PKCS #5 v2.0. Также БВ должен предоставлять сервис для генерации солей для шифрования.

3 Разработка программы

Наконец приступим к разработке программы. Так как имеется необходимость разработать веб-интерфейс, а также есть необходимость работы с базами данных, то для упрощения будем использовать язык программирования C# и платформу .NET [Ошибка: источник перекрёстной ссылки не найден], к которым есть удобные расширения в виде ASP.NET [2] для разработки веб-сервисов и Entity Framework Core [3] для работы с базами данных.

В рамках данного отчета будут приведены абстрактные и текстовые описания алгоритмов, структур данных и т. д. Исходный код всего менеджера паролей приведен в репозитории на GitHub [11].

Будем продвигаться от бизнес-логики к непосредственно интерфейсу. В первую очередь реализуем интерфейсы и абстрактные вещи, которые будут представлять нашу бизнес логику. Для этого создадим проект PasswordManager.Abstractions.

Данный проект предлагает следующие сущности:

- Account – модель, аккаунта, которая хранится в программе, состоит из идентификатора, названия и данных об аккаунте;
- AccountData – данные об аккаунте: логин и пароль;
- EncryptedItem – зашифрованная информация, которая будет хранится в БД, состоит из идентификатора, названия и непосредственно зашифрованных данных;
- EncryptedData – непосредственно зашифрованные данные, состоит из соли и зашифрованных данных;
- PasswordCheckStatus – статус проверки пароля, его стойкость и компрометация;

Данный проект будет предлагать следующий набор интерфейсов с методами:

- IAlphabet – интерфейс алфавита, который необходим для энтропийного метода стойкости, а также для генерации паролей;
- ICounter – счетчик, нужен для подсчета неверных попыток входа в менеджер паролей;
- IPasswordChecker – интерфейс для проверки паролей;
- IPasswordCheckerFactory – фабрика для создания интерфейса проверки паролей;
- IPasswordGenerator – генератор пароля;
- IPasswordGeneratorFactory – фабрика для генератора паролей;
- ICrypto – интерфейс абстрактного шифровальщика, может шифровать и расшифровывать данные;
- IKeyGenerator – генератор ключа шифрования;
- IKeyGeneratorFactory – фабрика для создания генератора ключей;
- IKeyGeneratorOptions – настройки для генератора ключей;
- IKeyValidator – верификатор ключа на корректность;
- IKeyValidatorFactory – фабрика для верификатора ключа;
- ISaltGenerator – генератор соли для шифрования;
- ISaltValidator – верификатор воли на корректность;
- IKeyDataRepository – репозиторий для информации о ключе;
- IEncryptedItemsRepository – репозиторий для зашифрованных данных;
- IDataContext – контекст работы с данными, нужен для сохранения изменений после работы над какими-либо данными в репозиториях;
- IAccountService – сервис для управления аккаунтами;
- IKeyService – сервис для управления ключом;
- IPasswordService – сервис для паролей, их генерации и проверки;
- IReadOnlyKeyStorage – хранилище ключа только для чтения;
- IKeyStorage – хранилище ключа;

Данный список интерфейсов и методов покрывает всю бизнес-логику нашего приложения. Далее приступим уже к их реализации.

Создадим проект PasswordManager.Aes, который будет предоставлять реализации интерфейсов: ICrypto, IKeyGenerator, IKeyGeneratorFactory, IKeyValidator, IKeyValidatorFactory, ISaltGenerator, ISaltValidator, IKeyGeneratorFactory. Эти реализации заточены, соответственно, под работу с алгоритмом AES. Для IKeyValidator определим две реализации: простую, которая проверят ключ только на общие требования (длина), и расширенную, которая проверят ключ, учитывая информацию о ключе.

Создадим проект PasswordManager.Core, который будет предоставлять реализации интерфейсов: IAplhabet, ICounter, IPasswordGenerator, IPasswordGeneratorFactory, IPasswordChecker, IPasswordCheckerFactory, IKeyService, IKeyStorage, IReadOnlyKeyStorage, IAccountService, IKeyService, IPasswordService. Данный проект будет предоставлять реализации оценки пароля с энтропийным подходом и с алгоритмом SeaMonkey, а в целом данный проект заточен под логику обработки данных, без знания о том, как они хранятся.

Также в Core реализована модель настроек пользователя, которая реализует интерфейс IKeyGeneratorOptions и дополняет его настройкой времени сессии, по истечению которого ключ будет удален из хранилища ключа.

В дополнение к этому проекту создадим проект PasswordManager.External, который дополнит реализации для интерфейсов IPasswordChecker и IPasswordCheckerFactory для подходов *';--have i been pwned?* и *zxcvbn*.

Для генерации данных воспользуемся встроенным с язык C# генератором случайных чисел [12], который считается криптографически стойким

Создадим проект PasswordManager.Data, который будет предоставлять реализации интерфейсов IKeyDataRepository и IEncryptedItemsRepository. Также в данном проекте определим конфигурацию базы данных. Создадим две таблицы:

- EncryptedItems – в ней будем хранить зашифрованные данные об аккаунтах в форме: идентификатор, название, версия данные, соль, зашифрованные данные;
- KeyData – в ней будет всего одна запись – информация о ключе – в форме: идентификатор, версия, соль, зашифрованные данные.

Для работы с таблицами определим модели:

- EncryptedDataDbModel – абстрактная модель данных в БД, содержащая идентификатор, версию, соль и зашифрованные данные;
- EncryptedItemDbModel – модель для работы с таблицей EncryptedItems, расширяет EncryptedDataDbModel полем с названием;
- KeyDataDbModel – модель для работы с таблицей KeyData, наследует от EncryptedDataDbModel без изменений.

Наконец создадим проект PasswordManager.Web, который будет являться входной точкой для пользователя. Построим данный проект на модели MVC. Определим контроллеры и представления для них:

- AccountController с представлением Account, на котором отображена информация о требуемом аккаунте с возможностью ее изменения; в случае нового аккаунта отображается пустая информация;
- AccountsApiController, который предоставляет функционал сервиса аккаунтов;
- AuthenticationController с представлениями Login, на котором отображена форма ввода мастер-пароля, и Logout, которая при загрузке посылает запрос на завершение сессии;
- AuthenticationApiController, который предоставляет функционал по инициализации хранилища, его очистке, а также создания авторизационной Cookie;
- GeneratorController с представлением Generator, на котором отображены поля ввода параметров для генерации пароля;
- PasswordsApiController, который предоставляет функционал сервиса паролей;

- SettingsController с представлением Settings, на котором отображены текущие настройки с возможностью их изменения;
- SettingsApiController, который предоставляет функционал по запросу текущих настроек, а также их изменению; дополнительно он предоставляет возможность удаления всех данных.
- IndexController с представлением Index, на котором отображены все аккаунты;

Для работы с контроллерами разработаем модели:

- ChangeKeySettingsRequest – модель запроса для изменений настроек ключа: соль, количество итерация, новый мастер-пароль; для подтверждения таких изменений также требует текущий мастер-пароль;
- ChangeSessionTimeoutRequest – модель запроса для изменений времени сессии;
- GeneratePasswordRequest – модель запроса на генерацию пароля, содержит параметры алфавита и запрашиваемую длину;
- LoginRequest – модель запроса на вход в приложений, содержит мастер-пароль;
- UploadAccountRequest – модель запроса на загрузку аккаунта в приложение, содержит название, логин и пароль;
- VerifyPasswordRequest – модель запроса на проверку пароля, содержит пароль;
- AccountHeaderResponse – модель ответа для представления Index, содержит идентификаторы и названия аккаунтов;
- AddAccountResponse – модель ответа на запрос создания аккаунта, содержит идентификатор нового аккаунта;
- GeneratePasswordResponse – модель ответа на запрос генерации пароля, содержит сгенерированный пароль и результат проверки этого пароля на стойкость и компрометацию;

- VerifyPasswordReponse – модель ответа на запрос проверки пароля, содержит уровень стойкости и статус компрометации пароля.

Для обеспечения защиты веб-приложения используем встроенные механизмы ASP.NET. ASP.NET предоставляет защиту от CSRF атак [13] с помощью специального токена, который внедряется в HTML страницу и отправляется в заголовках HTTP запросов для последующей проверки со стороны веб-сервера [14].

Для настройки HTTPS протокола используем настройки приложения в файле appsettings.json: в нем настроим порт, на котором будет открыто HTTPS соединение, а также параметры сертификата [15]. Для удобства используем скрипт, использующий openssl для генерации самоподписанного сертификата [16].

Для защиты авторизационной Cookie будем в нее записывать текущий IP-адрес пользователя, а при каждом HTTP запросе проверять этот IP-адрес. В случае несовпадения IP-адресов Cookie будет инвалидироваться.

Для удобства введем поддержку работы приложения за прокси-сервером и добавим возможность это настроить через appsettings.json. В таком случае вместо IP-адреса пользователя будем использовать IP-адрес, переданный через заголовок X-Forwarded-For, который должен добавлять прокси-сервер [17].

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были разработаны архитектура программного обеспечения «Менеджер паролей», на основе которой была разработана и сама программа.

Также в ходе выполнения работы была определена вспомогательная логика программы: были выбраны методы шифрования данных, методы определения стойкости и компрометации паролей, а также методы генерации паролей.

Результат курсовой работы в виде исходного кода программы приведен в репозитории [11].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 .NET [Электронный ресурс] // Microsoft – URL: https://dotnet.microsoft.com/ (дата обращения: 12.04.2025)

2 ASP.NET Core [Электронный ресурс] // Microsoft – URL: https://dotnet.microsoft.com/en-us/apps/aspnet/ (дата обращения: 12.04.2025)

3 Entity Framework Core [Электронный ресурс] // Microsoft Learn – URL: https://learn.microsoft.com/en-us/ef/core/ (дата обращения: 12.04.2025)

4 SQLite [Электронный ресурс] // Wikipedia – URL: https://ru.wikipedia.org/wiki/SQLite (дата обращения: 12.04.2025)

5 Зачем менять надёжный пароль? Брутфорс и энтропия [Электронный ресурс] // Хабр – URL: https://habr.com/ru/companies/globalsign/articles/697708/ (дата обращения: 12.04.2025)

6 Ещё об оценке стойкости пароля [Электронный ресурс] // Хабр – URL: https://habr.com/ru/sandbox/27520/ (дата обращения: 12.04.2025)

7 [Электронный ресурс] // – URL: https://pangolin.sbertech.ru/center/docs/extensions/zxcvbn/ (дата обращения: 12.04.2025)

8 zxcvbn. Оценщик надежности пароля [Электронный ресурс] // Pangolin DB – URL: https://haveibeenpwned.com/ (дата обращения: 12.04.2025)

9 PBKDF2 [Электронный ресурс] // Wikipedia – URL: https://ru.wikipedia.org/wiki/PBKDF2 (дата обращения: 12.04.2025)

10 NIST Special Publication 800-132 [Электронный ресурс] // NIST – URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf (дата обращения: 12.04.2025)

11 PasswordManager [Электронный ресурс] // GitHub – URL: https://github.com/mrypdm/PasswordManager (дата обращения: 12.04.2025)

12 RandomNumberGenerator [Электронный ресурс] // Microsoft Learn – URL:

https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.randomnumbergenerator (дата обращения: 12.04.2025)

13 Cross Site Request Forgery [Электронный ресурс] // OWASP Foundation – URL: https://owasp.org/www-community/attacks/csrf (дата обращения: 12.04.2025)

14 Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core [Электронный ресурс] // Microsoft Learn – URL: https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery (дата обращения: 12.04.2025)

15 Configure endpoints for the ASP.NET Core Kestrel web server [Электронный ресурс] // Microsoft Learn – URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel/endpoints (дата обращения: 12.04.2025)

16 Generate self-signed certificates with the .NET CLI [Электронный ресурс] // Microsoft Learn – URL: https://learn.microsoft.com/en-us/dotnet/core/additional-tools/self-signed-certificates-guide (дата обращения: 12.04.2025)

17 X-Forwarded-For [Электронный ресурс] // MDN – URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Reference/Headers/X-Forwarded-For (дата обращения: 12.04.2025)