**ГАОУ ВО «Дагестанский государственный университет народного хозяйства»**

Факультет информационных технологии и инженерии

Кафедра «Информационные технологии и информационная безопасность»

Очная форма обучения

Направление подготовки 10.03.01 Информационная безопасность

Профиль подготовки «Безопасность автоматизированных систем»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**Анхаева Абдуллы Шамилевича**

**Тема «Разработка веб-приложения для безопасного хранения паролей»**

|  |  |
| --- | --- |
| ВКР допущена к защите  И.о. зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Гасанова Зарема Ахмедовна, к.п.н.,  «\_\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. | Руководитель ВКР  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Гасанова Зарема Ахмедовна, к.пед.н., и.о. заведующего кафедрой «Информационные технологии и информационная безопасность» |

Результаты проверки на объем заимствования - 90 % оригинального текста

ВКР представлена на выпускающую кафедру «08» июня 2024 г.

ВКР представлена в ГЭК «24» июня 2024 г.

Дата защиты ВКР «29» июня 2024 г.

Подпись студента\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Махачкала – 2024 г.

**ГАОУ ВО «Дагестанский государственный университет народного хозяйства»**

Факультет информационных технологии и инженерии

Кафедра «Информационные технологии и информационная безопасность»

Очная форма обучения

Направление подготовки 10.03.01 Информационная безопасность

Профиль подготовки «Безопасность автоматизированных систем»

|  |
| --- |
| «Утверждаю»  И.о. зав. кафедрой\_\_\_\_\_\_\_\_  Гасанова З.А., к.п.н., доцент  «23» сентября 2023 г. |

**ЗАДАНИЕ**

**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

Студенту(ке) Анхаева Абдуллы Шамилевича

Тема ВКР «Разработка веб-приложения для безопасного хранения паролей»

Исходные данные к ВКР:

* План ВКР
* Список литературы и электронных ресурсов

Перечень подлежащих разработке вопросов:

1. Понятие и виды менеджеров паролей
2. Обзор существующих решений
3. Требования к программным средствам хранения паролей
4. Проектирование и реализация веб-приложения для безопасного хранения паролей

**КАЛЕНДАРНЫЙ ГРАФИК ВЫПОЛНЕНИЯ ВКР БАКАЛАВРА**

Срок сдачи 1-й главы – 31 декабря 2023 г.

Срок сдачи 2-й главы – 30 марта 2024 г.

Срок сдачи 3-й главы – 8 июня 2024 г.

Дата выдачи задания – 24 сентября 2024 г.

Срок представления ВКР руководителю – 7 июня 2024 г.

Проверка текста ВКР на процент заимствования – 7 июня 2024 г.

Срок представления ВКР на выпускающую кафедру 8 июня 2024 г.

Представление ВКР на предварительную защиту – 10 июня 2024 г.

Получение отзыва руководителя – 22 июня 2024 г.

Срок представления ВКР в ГЭК – 24 июня 2024 г.

|  |  |
| --- | --- |
| Руководитель ВКР  Гасанова Зарема Ахмедовна, к.пед.н., и.о. заведующего кафедрой «Информационные технологии и информационная безопасность»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись  «23» сентября 2023 г. | Задание принял к исполнению  Студент 4 курса 1 группы  очной формы обучения  Анхаев Абдулла Шамилевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись  «23» сентября 2023 г. |

**Оглавление**

[Введение 5](#_Toc170290129)

[Глава 1. Анализ предметной области 7](#_Toc170290130)

[1.1. Понятие и виды менеджеров паролей 7](#_Toc170290131)

[1.2. Требования к программным средствам хранения паролей 10](#_Toc170290132)

[1.3. Обзор существующих решений 13](#_Toc170290133)

[Глава 2. Методы обеспечения безопасного хранения паролей 23](#_Toc170290134)

[2.1. Шифрование данных 23](#_Toc170290135)

[2.2. Аутентификация и управление доступом 32](#_Toc170290136)

[Глава 3. Проектирование и реализация веб-приложения для безопасного хранения паролей 40](#_Toc170290137)

[3.1. Формирование требований к разрабатываемому приложению 40](#_Toc170290138)

[3.2. Выбор реализуемых технологий и средств разработки 43](#_Toc170290139)

[3.3 Реализация приложения 48](#_Toc170290140)

[3.4 Демонстрация приложения 59](#_Toc170290141)

[Заключение 66](#_Toc170290142)

[Список литературы и источников 68](#_Toc170290143)

# Введение

В современном мире, где количество онлайн-сервисов постоянно растет, пользователи сталкиваются с проблемой безопасного хранения большого количества паролей. Каждый день мы используем различные платформы, от социальных сетей до банковских систем, и для доступа к каждому из этих сервисов требуется уникальный пароль. Часто пользователи сталкиваются с ситуацией, когда необходимо запомнить десятки, а то и сотни различных паролей. Хранение этих паролей на бумажных носителях или в текстовых файлах на компьютере не только неудобно, но и небезопасно.

**Актуальность** выбранной темы обусловлена стремительным ростом числа онлайн-сервисов и повышенными требованиями к безопасности пользовательских данных. В условиях, когда каждое устройство и каждый сервис требуют уникальной аутентификации, необходимость в надежном и удобном способе хранения паролей становится все более очевидной. Традиционные методы хранения паролей, такие как записные книжки или текстовые файлы, не отвечают современным требованиям безопасности и удобства.

**Объектом** данной работы является система безопасного хранения паролей.

**Предметом** же выступают методы, технологии безопасного хранения паролей.

**Целью** данной выпускной квалификационной работы является разработка веб-приложения для безопасного хранения паролей. Это приложение позволит пользователям хранить все свои пароли от различных сервисов в одном месте и получать к ним доступ из любой точки мира с любого устройства, будь то телефон, планшет, ноутбук или компьютер. Такое решение значительно упростит процесс управления паролями и повысит безопасность личных данных.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Изучить предметную область и существующие решения в области управления паролями.
2. Спроектировать веб-приложение, отвечающее современным требованиям безопасности и удобства.
3. Реализовать веб-приложение, обеспечивающее безопасное хранение паролей и доступ к ним из любой точки мира.

В рамках **исследования** будет проведен анализ существующих решений, применяемых для безопасного хранения паролей. **Практическая значимость** данной работы заключается в создании удобного и безопасного инструмента для управления паролями, который позволит пользователям избежать проблем, связанных с утечкой данных и неудобством хранения большого количества паролей.

**Научная новизна** работы состоит в том, что разработанное приложение будет иметь открытый исходный код, что позволит любому желающему развернуть его на собственном сервере, использовать и при необходимости модифицировать в соответствии с индивидуальными потребностями. Это делает решение не только удобным, но и гибким, что особенно важно в условиях быстро меняющихся требований к безопасности и функциональности программного обеспечения.

Работа состоит из введения, трех глав, заключения и списка литературы.

# Глава 1. Анализ предметной области

## 1.1. Понятие и виды менеджеров паролей

**Менеджер паролей** — это программное обеспечение, которое помогает пользователям создавать, хранить и управлять паролями для различных учетных записей и веб-сайтов. Основная цель таких приложений — повысить безопасность и удобство при использовании множества различных учетных записей.

Менеджеры паролей обеспечивают пользователям следующие возможности:

* **Хранение паролей.** Менеджеры паролей хранят все пароли пользователя в зашифрованном виде, что исключает возможность их компрометации в случае несанкционированного доступа к устройству.
* **Автоматическая генерация паролей.** Менеджеры паролей могут автоматически генерировать сложные и уникальные пароли для каждой учетной записи, что значительно повышает уровень безопасности.
* **Автозаполнение форм.** Многие менеджеры паролей поддерживают функцию автозаполнения форм на веб-сайтах, что делает процесс авторизации быстрым и удобным.
* **Синхронизация между устройствами.** Современные менеджеры паролей предоставляют возможность синхронизации данных между различными устройствами, что позволяет пользователю получить доступ к своим паролям из любой точки мира.

Менеджеры паролей можно разделить на несколько основных категорий:

**1.** **Локальные менеджеры паролей.** Эти менеджеры хранят пароли на устройстве пользователя. Примером локального менеджера паролей является KeePass — один из первых бесплатных менеджеров с открытым исходным кодом, который совместим практически со всеми операционными системами и устройствами. KeePass обладает всеми необходимыми функциями для персонального использования и обеспечивает высокий уровень безопасности, так как данные хранятся локально.

**Преимуществом** этой категории можно выделить высокий уровень безопасности, так как данные хранятся только на устройстве пользователя.

Сущими **недостатками** же являются отсутствие синхронизации между устройствами и сложность резервного копирования.

**2.** **Облачные менеджеры паролей.** Облачные менеджеры паролей хранят пароли в облаке и синхронизируют их между устройствами. Примеры облачных менеджеров включают LastPass и BitWarden. LastPass позволяет пользователям вести цифровой архив, формировать надежные пароли и совместно использовать пароли. BitWarden, бесплатный менеджер с открытым исходным кодом, также предлагает возможность использования собственного сервера для хранения данных.

**Преимущества** этих менеджеров перекрывают все недостатки локальных менеджеров, это - удобство использования, синхронизация между устройствами и возможность доступа из любого места.

А **недостатками** являются следующее - потенциальные риски безопасности при взломе облачных серверов, зависимость от интернета, а также стоимость.

**3. Встроенные менеджеры паролей.** Эти менеджеры интегрированы в операционные системы или браузеры. Примером встроенного менеджера паролей является Google Password Manager, который встроен в браузер Chrome и учетные записи Google. Другим примером является iCloud Keychain, встроенный в устройства Apple и учетные записи iCloud.

Здесь **преимуществом** над остальными является - интеграция с операционной системой или браузером, что в последствии даёт удобство использования и автоматическую синхронизацию.

**Недостатками** же выделяют ограниченные функции по сравнению с специализированными менеджерами паролей и самое главное - зависимость от конкретной экосистемы.

## 1.2. Требования к программным средствам хранения паролей

Современные программные средства для хранения паролей должны удовлетворять ряду критически важных требований, обеспечивающих их надежность, безопасность и удобство использования. Рассмотрим основные требования, которым должно соответствовать подобное программное обеспечение.

**1. Безопасность хранения данных:**

**Шифрование данных.** Пароли и другие конфиденциальные данные должны храниться в зашифрованном виде с использованием современных криптографических алгоритмов, таких как AES-256. Это предотвращает несанкционированный доступ к данным.

**Хэширование.** Для защиты паролей используется хэширование, при котором пароль преобразуется в хэш-функцию. При вводе пароля пользователем, он также хэшируется и сравнивается с хранимым значением. Это позволяет избежать хранения паролей в открытом виде.

**Главный пароль.** Доступ к хранилищу паролей защищен главным паролем, который должен быть сложным и уникальным. Это минимизирует риск его взлома.

**Двухфакторная аутентификация (2FA).** Дополнительный уровень защиты при входе в систему, который требует подтверждения личности через второй фактор (например, SMS-код, приложение-аутентификатор или одноразовый код на E-mail).

**Регулярные обновления безопасности.** Программное обеспечение должно регулярно обновляться для защиты от новых уязвимостей и атак.

**2. Удобство использования:**

**Интуитивный интерфейс.** Пользовательский интерфейс должен быть простым и интуитивно понятным, что позволяет пользователям легко управлять своими паролями без необходимости в технических знаниях.

**Автозаполнение форм.** Возможность автоматического заполнения полей для ввода логина и пароля на веб-сайтах существенно упрощает процесс авторизации для пользователей. Для реализации такого функционала необходима интеграция менеджера паролей в браузер посредством расширения для них, либо же установки десктопного приложения, который будет работать с любыми формами любых браузеров.

**Генерация паролей.** Встроенный генератор паролей, который позволяет создавать сложные и уникальные пароли для каждой учетной записи, повышая общую безопасность.

**Кроссплатформенность.** Приложение должно быть доступно на различных операционных системах и устройствах (Windows, macOS, iOS, Android), что обеспечивает доступ к паролям в независимости платформы.

**3. Масштабируемость и производительность:**

**Высокая производительность.** Приложение должно быстро обрабатывать запросы пользователя и обеспечивать мгновенный доступ к паролям.

**Масштабируемость.** Программное обеспечение должно быть способно справляться с увеличением объема данных без потери производительности.

**4. Синхронизация и резервное копирование:**

**Синхронизация данных между устройствами.** Менеджеры паролей должны поддерживать синхронизацию данных между устройствами пользователя, чтобы обеспечить доступ к актуальным паролям в любом месте и в любое время.

**Резервное копирование.** Возможность создания резервных копий данных позволяет восстанавливать информацию в случае утраты устройства или повреждения данных.

**5. Управление доступом:**

**Журналирование действий.** Ведение журналов действий пользователей позволяет отслеживать, кто и когда получил доступ к данным, что повышает уровень контроля и безопасности.

**Разделение прав доступа.** Для корпоративных пользователей важно иметь возможность настройки прав доступа, чтобы сотрудники могли получать доступ только к тем паролям, которые необходимы для выполнения их служебных обязанностей.

**6. Поддержка и обслуживание:**

**Регулярные обновления безопасности.** Периодические обновления программного обеспечения с устранением выявленных уязвимостей и добавлением новых функций.

**Техническая поддержка.** Пользователям должна быть доступна квалифицированная техническая поддержка, способная быстро и эффективно решать возникающие проблемы.

**Документация.** Подробная документация и инструкции по использованию, позволяющие пользователям быстро освоить все возможности приложения.

Таким образом, высококачественный менеджер паролей должен обеспечивать высокий уровень безопасности данных, удобство использования, масштабируемость, кроссплатформенность и надежную поддержку. Эти требования помогают создать надежное и удобное решение для хранения и управления паролями в современном цифровом мире.

## 1.3. Обзор существующих решений

На данный момент существует не так уж много менеджеров паролей для бизнеса, которые обеспечивают надежность и безопасность, а также имеют большой функционал. Можно выделить следующие аналоги:

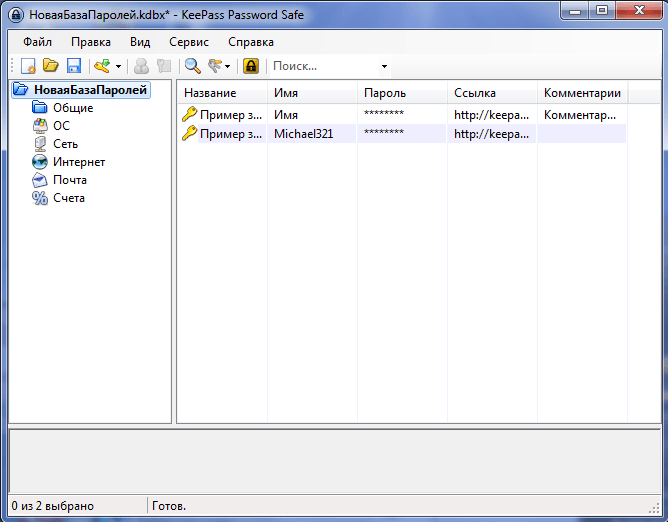
**1) KeePass** — это один из самых известных и широко используемых менеджеров паролей с открытым исходным кодом. KeePass хранит все пароли в зашифрованной базе данных, защищенной паролем или ключевым файлом. Приложение поддерживает множество алгоритмов шифрования, включая AES и Twofish. KeePass отличается высокой степенью кастомизации и расширяемости за счет множества плагинов. Поскольку база данных хранится локально, KeePass предоставляет высокий уровень безопасности, что делает его идеальным выбором для пользователей, ценящих конфиденциальность своих данных.

Рис. 1.1 - Интерфейс KeePass

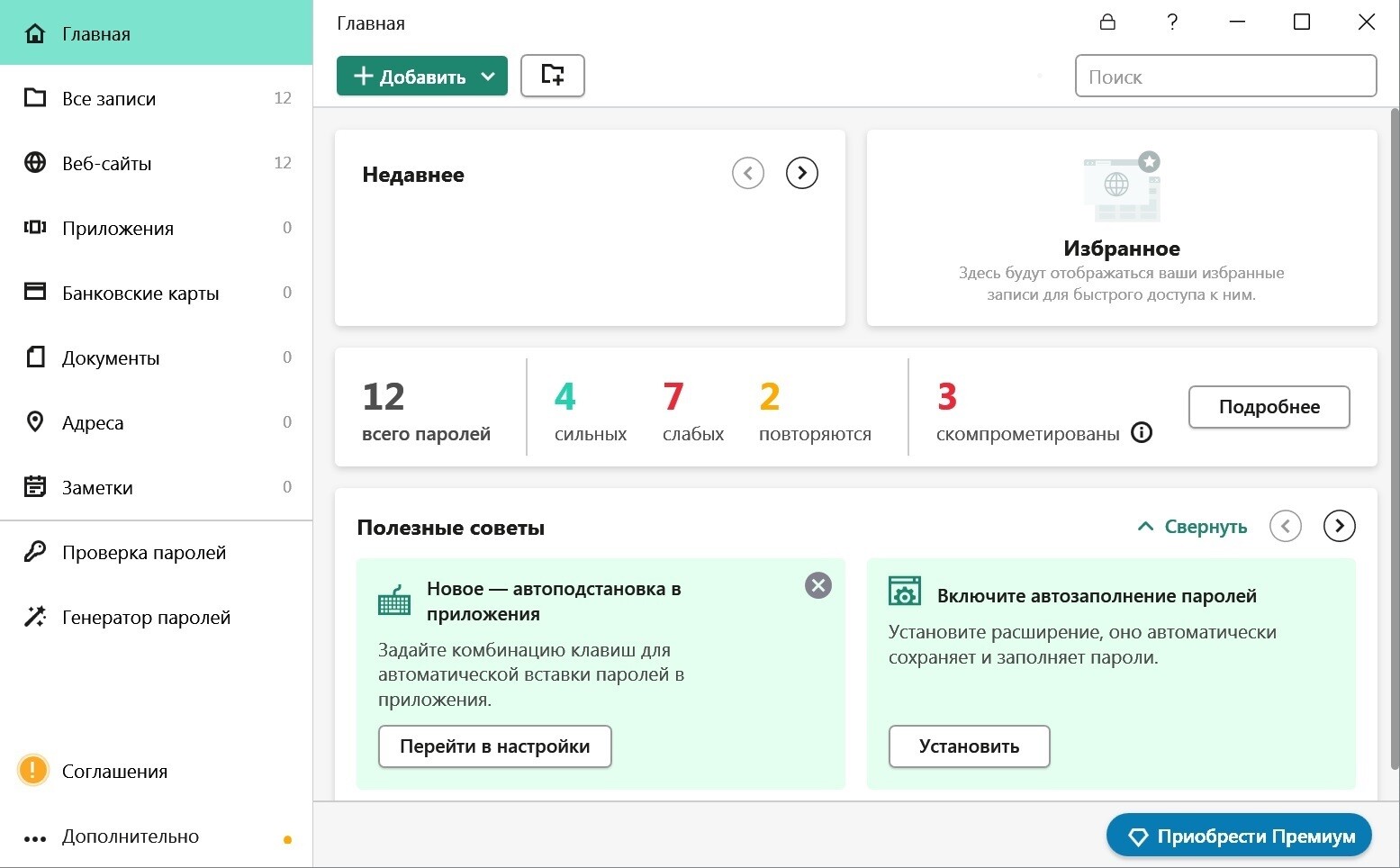
**2) Kaspersky Password Manager** — это продукт, разработанный одной из ведущих компаний в области кибербезопасности. Этот менеджер паролей предлагает шифрование на уровне AES-256 и обеспечивает безопасное хранение паролей, кредитных карт, изображений и сканов документов. Kaspersky Password Manager синхронизирует данные между устройствами через облако, что позволяет пользователю получать доступ к паролям из любого места. Продукт также включает функцию автозаполнения и генерации сложных паролей, что повышает удобство использования. Отличительной особенностью этого менеджера является интеграция с другими продуктами Kaspersky, что обеспечивает комплексный подход к защите данных.

Рис. 1.2 - Интерфейс Kaspersky Password Manager

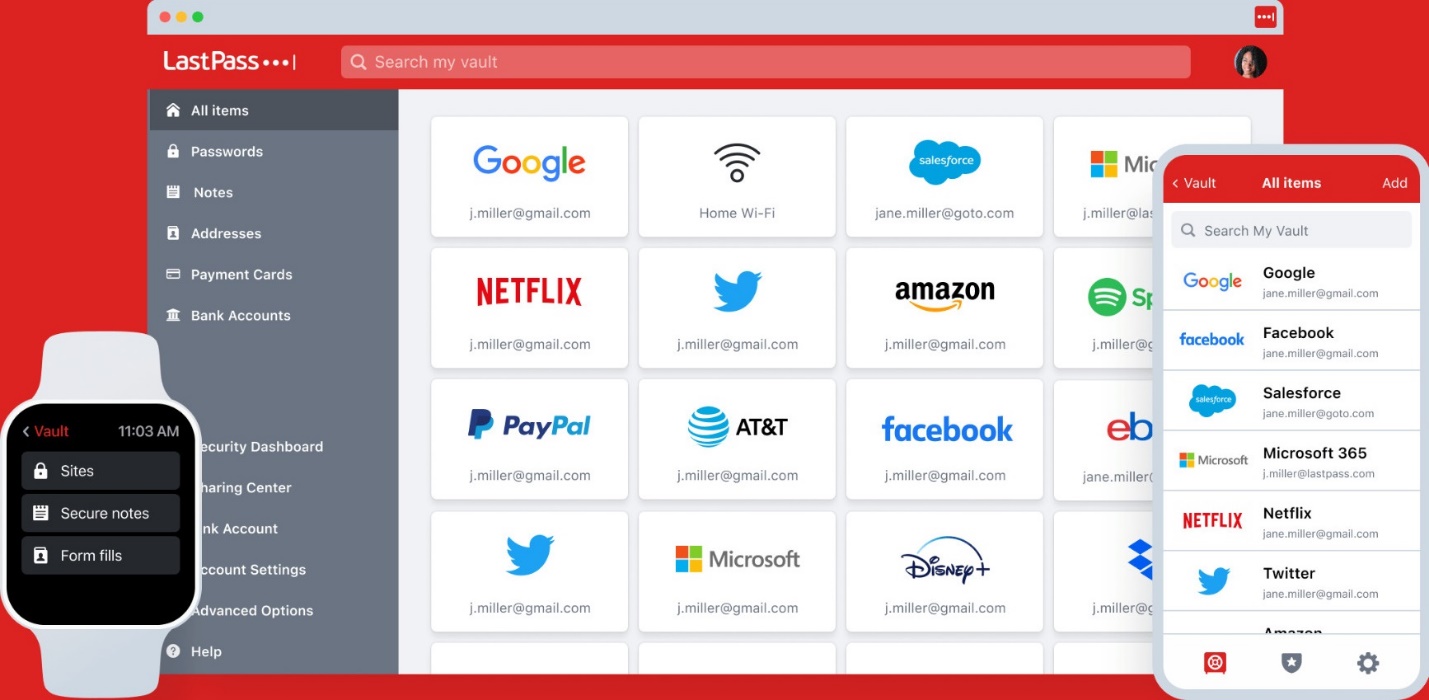
**3) LastPass** является одним из наиболее популярных облачных менеджеров паролей. Он предлагает высокую степень безопасности благодаря шифрованию AES-256 и использованию мастер-пароля. LastPass позволяет хранить пароли, заметки и другую конфиденциальную информацию в зашифрованном виде. Одной из ключевых особенностей LastPass является функция автозаполнения форм и синхронизация данных между устройствами. LastPass также поддерживает двухфакторную аутентификацию и предоставляет пользователям возможность обмена паролями. Приложение доступно на различных платформах, включая Windows, macOS, iOS и Android.

Рис. 1.3 - Интерфейс LastPass

**4) BitWarden** — это менеджер паролей с открытым исходным кодом, который получил признание за свою безопасность и функциональность. Он предлагает шифрование на основе AES-256 и хранение данных как локально, так и в облаке. BitWarden поддерживает автозаполнение форм и генерацию сложных паролей. Приложение также предлагает синхронизацию данных между устройствами и двухфакторную аутентификацию. Благодаря открытости кода, BitWarden привлекает пользователей, которые ценят возможность проверки безопасности и настройки программы под свои нужды.

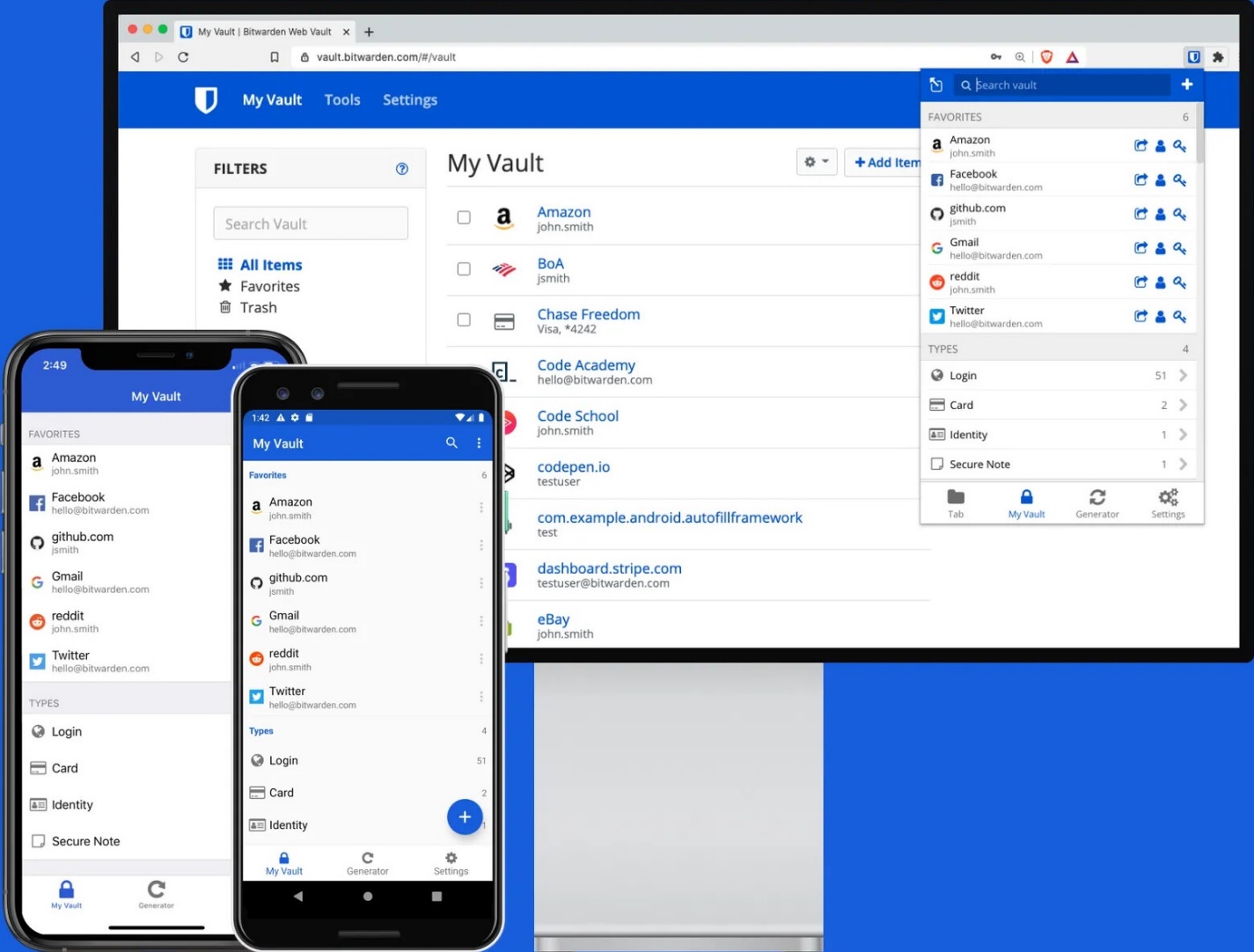
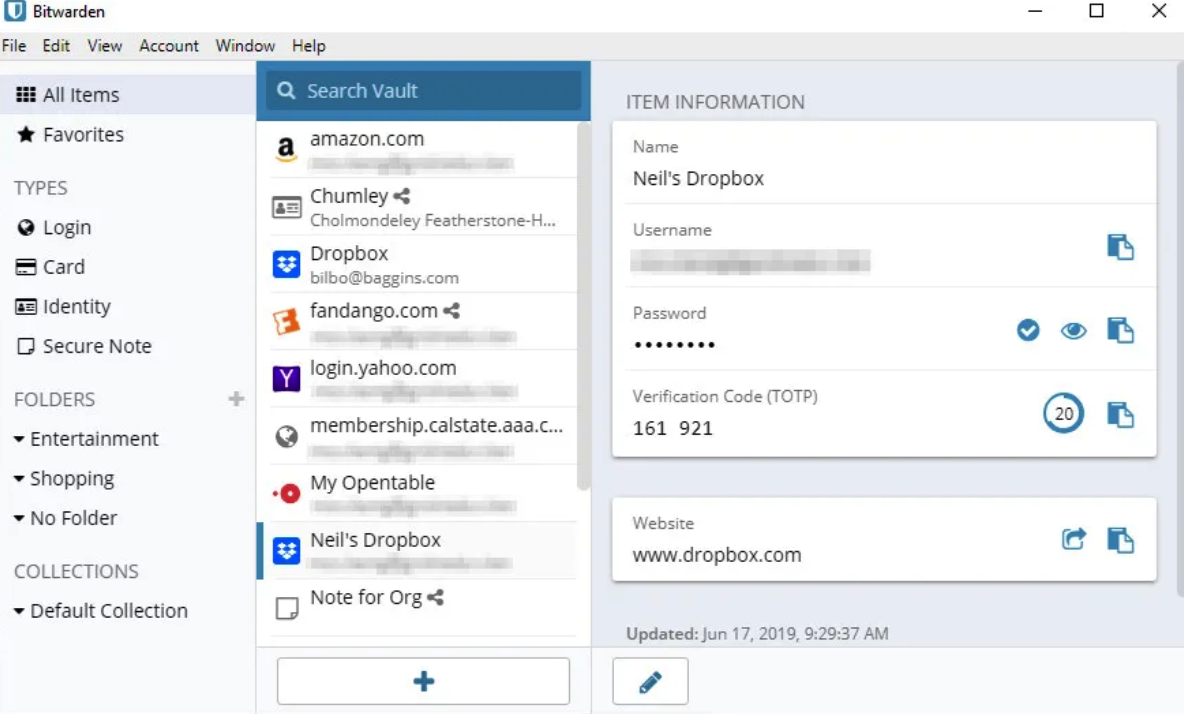


Рис. 1.4 - Интерфейс BitWarden на ПК

Рис. 1.5 - Интерфейс BitWarden

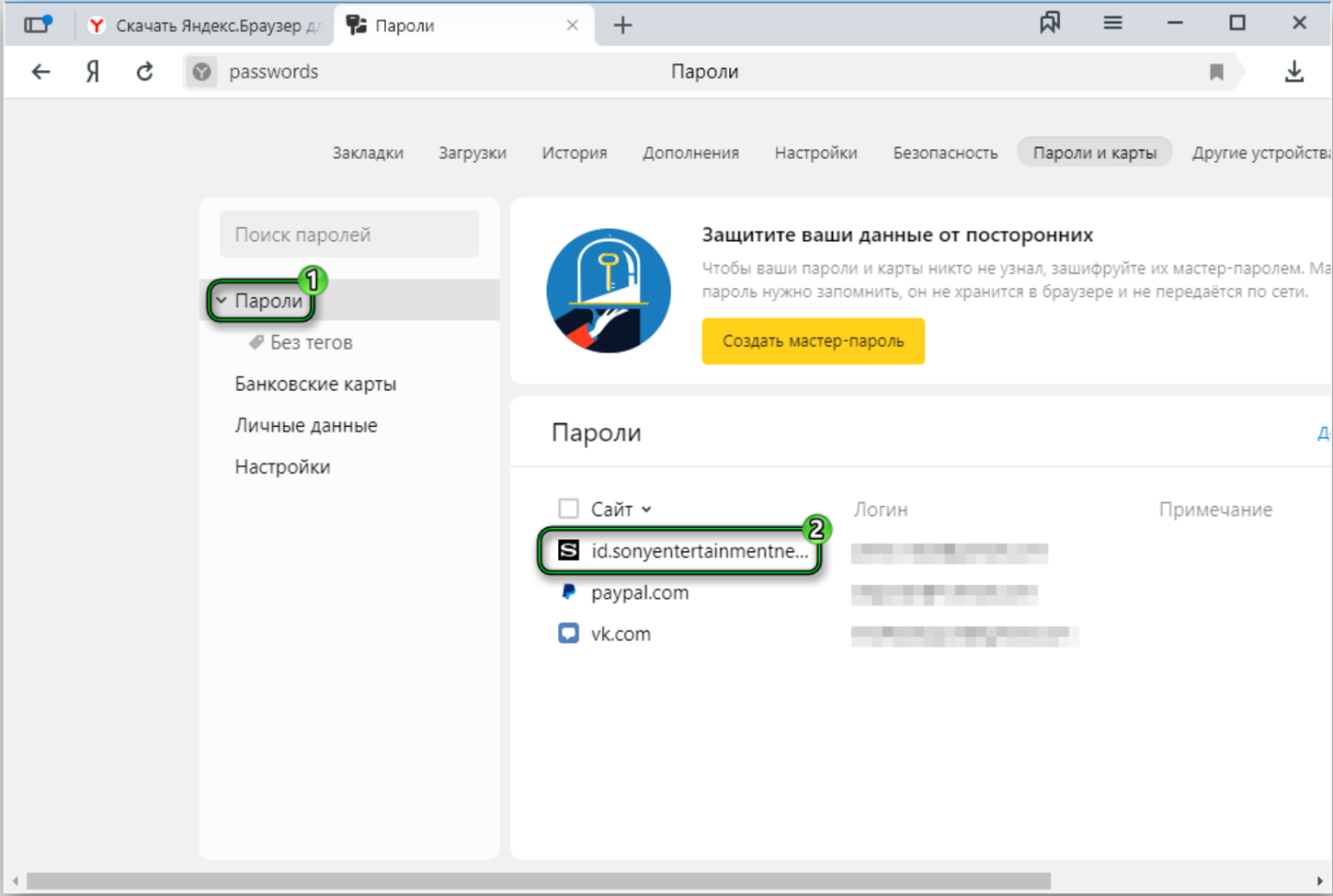
**5) Менеджер паролей Яндекс.Браузера** интегрирован в одноименный браузер и предлагает функции автоматического сохранения и заполнения паролей. Продукт синхронизирует данные между устройствами пользователя через облако и использует шифрование для защиты данных. Яндекс.Браузер обеспечивает удобство использования и безопасность, предлагая пользователям дополнительные функции, такие как проверка надежности паролей и защита от фишинга. Этот менеджер паролей особенно удобен для пользователей, активно использующих другие сервисы Яндекса.

Рис. 1.6 - Интерфейс Менеджер паролей Яндекс.Браузера

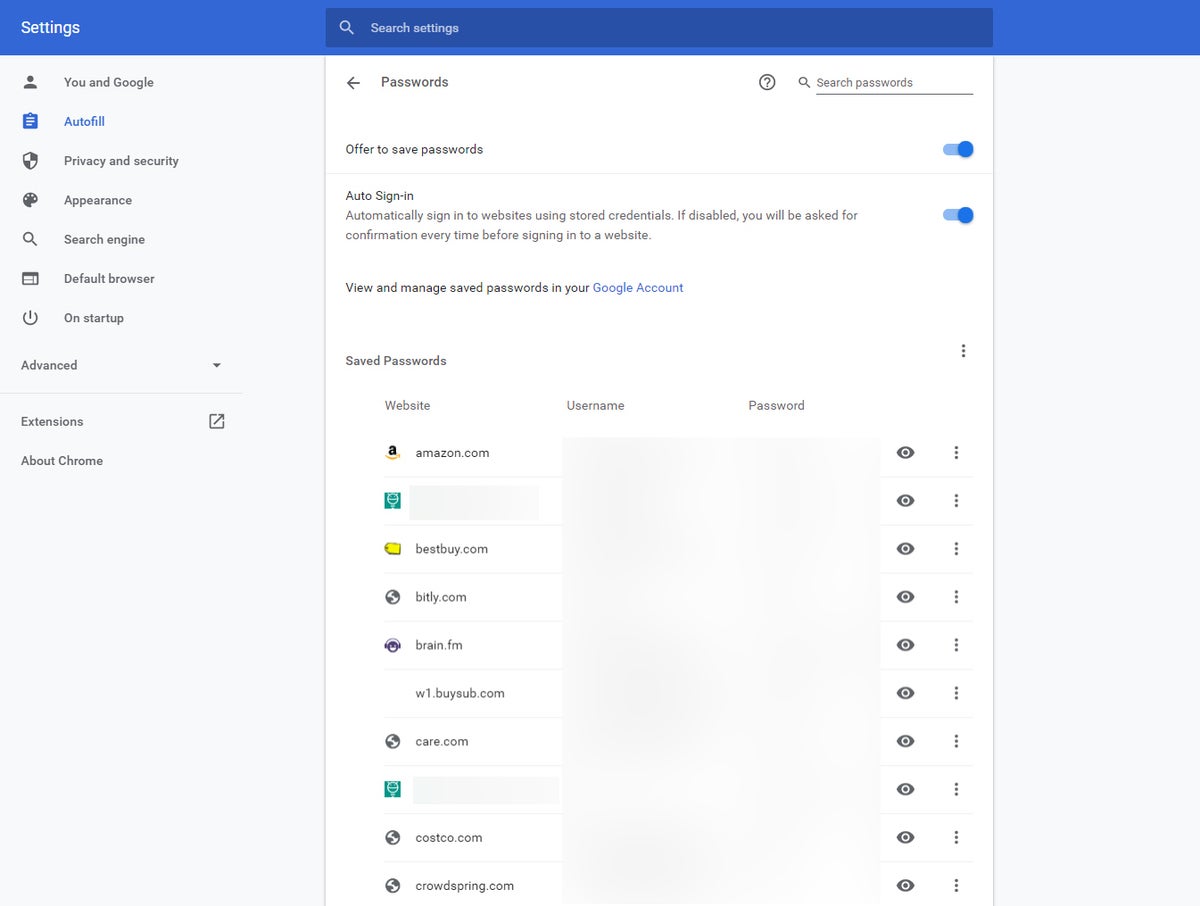
**6) Google Password Manager** встроен в экосистему Google и предлагает пользователям удобный способ управления паролями. Он автоматически сохраняет пароли, вводимые на различных сайтах, и синхронизирует их между всеми устройствами пользователя, связанными с аккаунтом Google. Google Password Manager использует шифрование на основе AES-256 и предлагает двухфакторную аутентификацию. Продукт интегрирован с браузером Google Chrome и другими сервисами Google, что обеспечивает высокую степень удобства, особенно для пользователей, активно использующих экосистему Google.

Рис. 1.7 - Интерфейс Google Password Manager

**7) iCloud Keychain** — это менеджер паролей, разработанный Apple и интегрированный в экосистему iOS и macOS. Он автоматически сохраняет и заполняет пароли, а также данные кредитных карт и Wi-Fi сетей. Все данные синхронизируются между устройствами пользователя через облако iCloud и защищены шифрованием на уровне AES-256. iCloud Keychain предлагает высокий уровень безопасности и удобства, особенно для пользователей, полностью интегрированных в экосистему Apple. Продукт также включает генератор сложных паролей и двухфакторную аутентификацию.

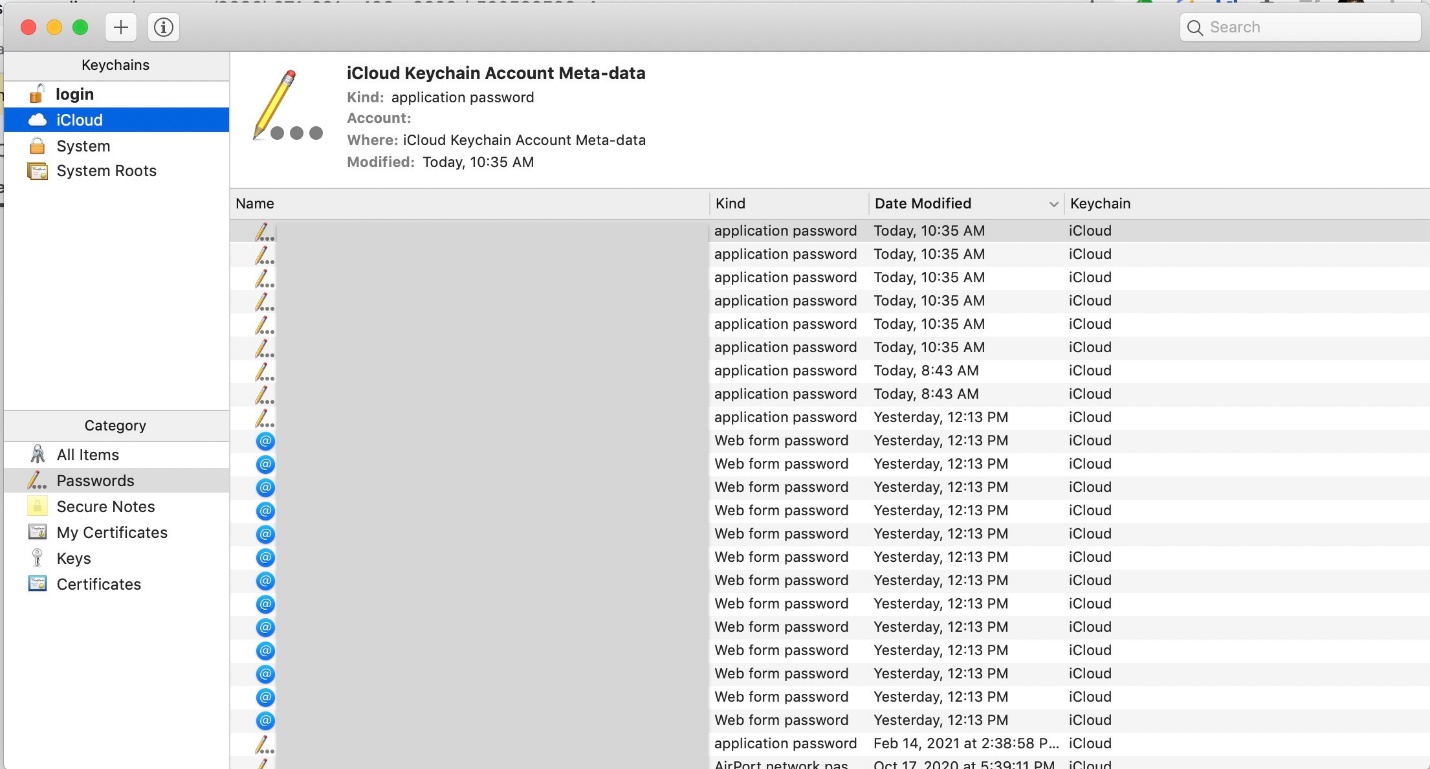
**8) PassWork** — это российский менеджер паролей, специально разработанный для корпоративного использования. Продукт занесен в **реестр отечественного ПО**, что делает его рекомендованным для **государственных** и коммерческих организаций. PassWork устанавливается на сервер компании, обеспечивая полный контроль над данными и исключая зависимость от сторонних облачных сервисов. Это особенно важно в условиях необходимости соблюдения российских законов о защите персональных данных.

Рис. 1.8 - Интерфейс iCloud Keychain

PassWork предлагает шифрование на уровне **AES-256** и **ГОСТ** на выбор, и предоставляет функции управления доступом, что позволяет администратору разграничивать права пользователей и обеспечивать безопасное хранение конфиденциальной информации. Продукт поддерживает генерацию сложных паролей, автозаполнение форм и синхронизацию данных между устройствами. Важной особенностью PassWork является гибкость настройки и интеграция с другими корпоративными системами, что делает его идеальным выбором для организаций, требующих высокого уровня безопасности и функциональности.

Благодаря своему функционалу и **соответствию российским стандартам безопасности**, PassWork является надежным выбором для государственных учреждений и крупных корпораций, требующих максимальной защиты данных.

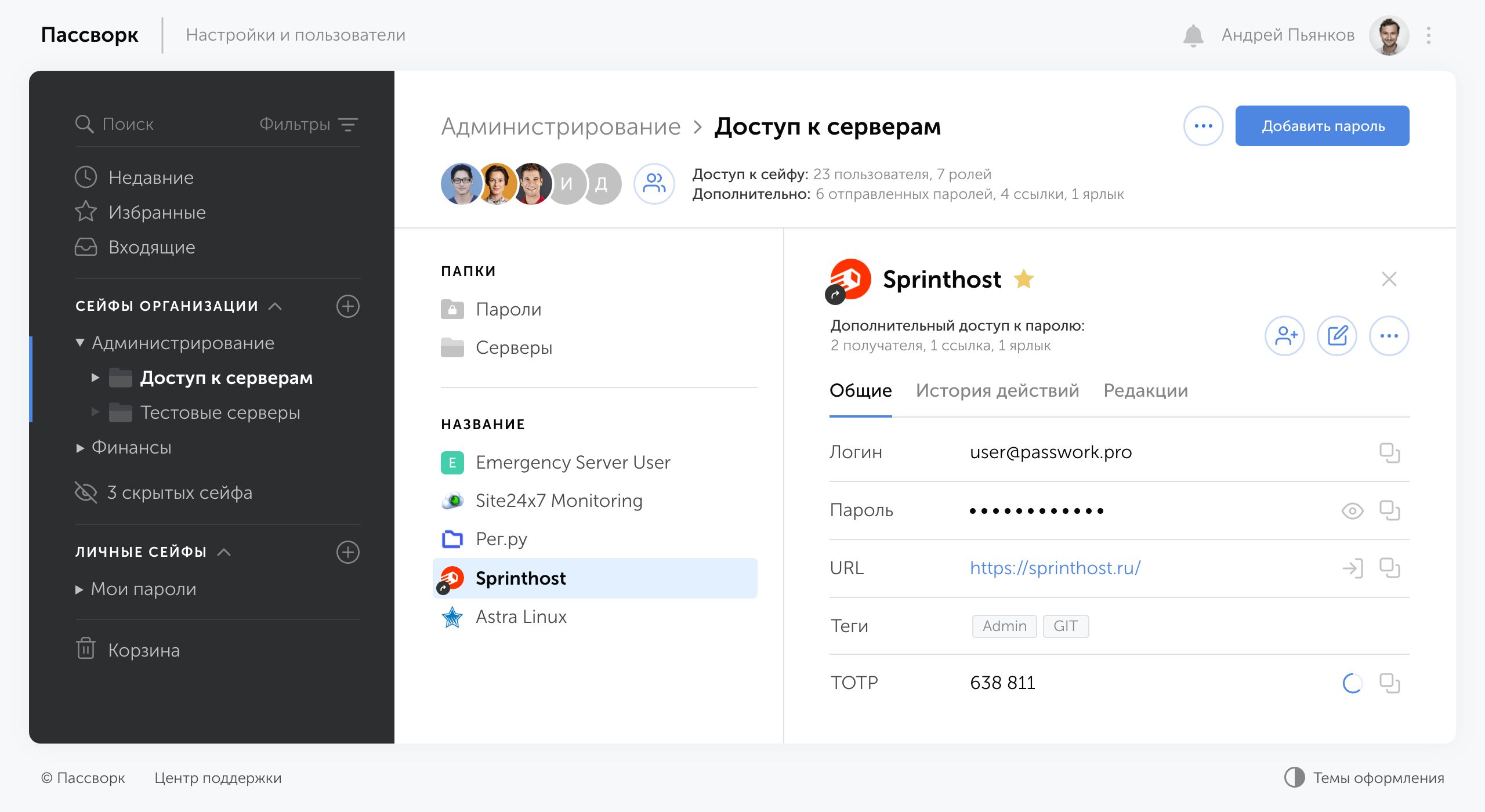
PassWork так же является сертифицированным партнером «Astra Linux», «РЕД Софт» и ОС «Атлант», и участвует в закупках по 44 и 223 ФЗ.

Рис. 1.9 - Интерфейс PassWork

В табл. 1.1. представлены основные критерии приложений и их сравнительная характеристика.

**Таблица 1.1 - Сравнение аналогов.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **KeePass** | **Kaspersky Password Manager** | **LastPass** | **BitWarden** |
| **Категория** | Локальный | Облачный | Облачный | Облачный, Локальный |
| **Тип приложения** | Десктопный | Десктопный, мобильный | Десктопный, мобильный | Все |
| **Шифрование** | AES-256 | AES-256 | AES-256 | AES-256 |
| **ОС** | Windows, macOS, Linux | Windows, macOS, iOS, Android | Windows, macOS, iOS, Android | Windows, macOS, Linux, iOS, Android |
| **Расширения для браузеров** | Присутствует | Присутствует | Присутствует | Присутствует |
| **Страна производителя** | Германия | Россия | США | США |
| **Панель администратора** | Отсутствует | Присутствует | Присутствует | Присутствует |
| **Бесплатная версия** | Присутствует | Пробная версия | Присутствует | Присутствует |
| **Доступность функционала в бесплатной версии** | Полный функционал | Ограниченный функционал | Ограниченный функционал | Ограниченный функционал |
| **Техническая поддержка** | Форум | Электронная почта, Телефон | Чат, Электронная почта | Чат, Электронная почта |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Менеджер паролей Яндекс.Браузера** | **Google Password Manager** | **iCloud Keychain** | **PassWork** |
| **Категория** | Встроенный | Встроенный | Встроенный | Облачный, локальный |
| **Тип приложения** | Веб, Мобильный | Веб, Мобильный | Веб, Мобильный | Все |
| **Шифрование** | AES-256 | AES-256 | AES-256 | AES-256, ГОСТ |
| **ОС** | Windows, macOS, Linux, iOS, Android | Windows, macOS, Linux, iOS, Android | macOS, iOS | Windows, macOS, Linux, iOS, Android |
| **Расширения для браузеров** | - | - | Отсутствует | Присутствует |
| **Страна производителя** | Россия | США | США | Россия |
| **Панель администратора** | Отсутствует | Отсутствует | Отсутствует | Присутствует |
| **Бесплатная версия** | - | - | - | Пробная версия |
| **Доступность функционала в бесплатной версии** | Полный функционал | Полный функционал | Полный функционал | Ограниченный функционал |
| **Техническая поддержка** | Электронная почта, Форум | Чат, Форум | Электронная почта | Электронная почта, Телефон, Форум |

Эти менеджеры паролей предлагают различные уровни безопасности и функциональности, удовлетворяя потребности как индивидуальных пользователей, так и корпоративных клиентов. Особое внимание стоит уделить PassWork, который благодаря своим возможностям по локальному развертыванию и гибкой системе управления доступом, а также наличия в едином реестре Российского ПО идеально подходит для использования в государственных компаниях и других организациях, где требуется высокий уровень контроля и безопасности данных.

Одна особенность, которая объединяет все эти инструменты управления паролями, - их стоимость. Бесплатные или пробные версии менеджеров часто имеют ограниченный функционал. Например, Kaspersky Password Manager позволяет пользователям хранить только пять записей, а LastPass не предлагает синхронизацию между устройствами.

Кроме того, встроенные решения для управления паролями часто требуют от пользователей использования конкретной экосистемы, с которой они интегрированы. Это привело к росту интереса к теме независимых менеджеров паролей.

# Глава 2. Методы обеспечения безопасного хранения паролей

## 2.1. Шифрование данных

**Криптография** — это наука о методах защиты информации путем ее преобразования таким образом, чтобы она стала недоступной для несанкционированного доступа. Основные цели криптографии включают конфиденциальность, целостность данных, аутентификацию и непризнание. В современной криптографии используются различные алгоритмы и протоколы для достижения этих целей.

История криптографии уходит корнями в древние времена, когда шифрование использовалось для защиты военных и государственных секретов. Одним из самых известных примеров ранней криптографии является шифр Цезаря, использовавшийся римским императором Юлием Цезарем. Современная криптография значительно более сложна и основывается на математических принципах и вычислительных технологиях.

Основным методом обеспечения безопасности хранения паролей является шифрование данных. Этот процесс включает в себя шифрование паролей, баз данных и другой конфиденциальной информации.

**Шифрование** — это процесс преобразования обычного текста в шифрованный с помощью алгоритма и секретного ключа. Этот процесс также может быть осуществлен без использования ключа, с помощью **хэширования**, что делает дешифрование невозможным. В процессе **дешифрования** происходит обратное преобразование, и шифротекст снова превращается в открытый текст, либо с использованием оригинального ключа, либо другого ключа, в зависимости от конкретного метода шифрования.

Рассмотрим наиболее распространенные алгоритмы. К ним относятся хэш-функция SHA-256, алгоритм симметричного шифрования AES-256 и асимметричный алгоритм RSA.

**SHA-256 (Secure Hash Algorithm 256-bit)** — это криптографическая хэш-функция, которая производит хэш длиной 256 бит (32 байта) из произвольного входного сообщения. Этот алгоритм был разработан Агентством национальной безопасности США (NSA) и опубликован Национальным институтом стандартов и технологий (NIST) как часть семейства SHA-2. SHA-256 используется в различных приложениях и протоколах, включая SSL/TLS для обеспечения безопасности интернет-соединений и криптовалюты, такие как Bitcoin, для защиты транзакций.

**Основные характеристики SHA-256:**

* **Фиксированный размер выхода.** Независимо от длины входного сообщения, SHA-256 всегда производит 256-битный (32-байтовый) хэш. Это позволяет легко сравнивать и хранить хэши.
* **Односторонняя функция.** Хэш-функция SHA-256 необратима. Это означает, что невозможно восстановить оригинальные данные из хэша, что делает его идеальным для хранения паролей и других конфиденциальных данных.
* **Высокая скорость выполнения.** SHA-256 оптимизирован для быстрой обработки больших объемов данных, что делает его пригодным для использования в реальном времени.
* **Устойчивость к коллизиям.** Алгоритм SHA-256 разработан так, чтобы минимизировать вероятность коллизий, когда два различных входных сообщения производят одинаковый хэш. Это критически важно для обеспечения уникальности и целостности данных.

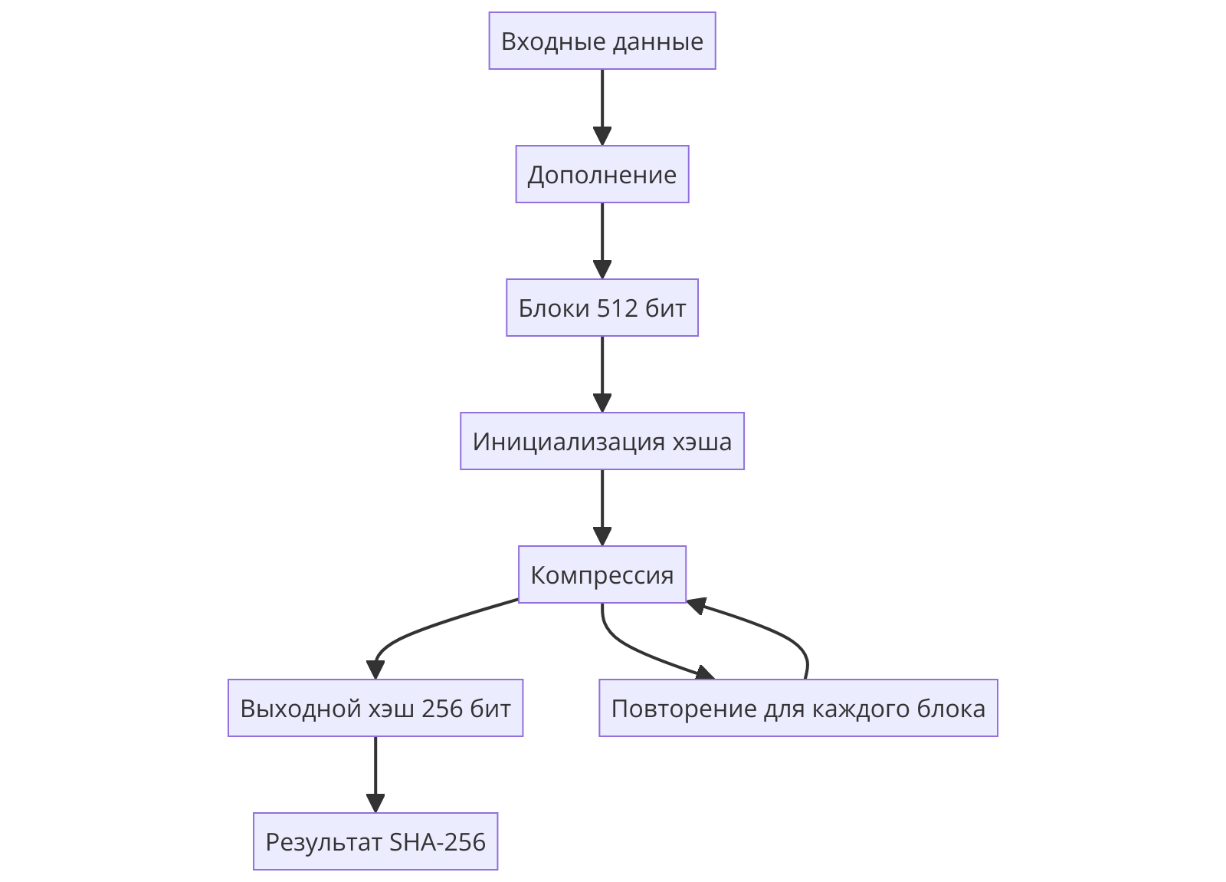
Процесс хэширования с использованием SHA-256 включает несколько этапов:

Рисунок 2.1 - Процесс хэширования SHA-256

1. **Входные данные:** исходные данные, которые нужно захэшировать, подаются на вход алгоритма.
2. **Дополнение:** входные данные дополняются таким образом, чтобы их длина стала кратной 512 битам. Дополнение включает добавление одного бита "1" и необходимых нулей, а также длины исходных данных в битах.
3. **Блоки 512 бит:** дополненные данные разбиваются на блоки размером 512 бит каждый.
4. **Инициализация хэша:** устанавливаются начальные значения переменных хэша. В алгоритме SHA-256 используются восемь начальных значений, которые фиксированы и определены стандартом.
5. **Компрессия:** основная часть алгоритма, где происходит обработка каждого блока. Каждый блок обрабатывается с использованием сложной последовательности битовых операций и математических функций. Промежуточные значения хэша обновляются после обработки каждого блока.
6. **Повторение для каждого блока:** процесс компрессии повторяется для каждого блока данных. Каждый новый блок использует результаты компрессии предыдущего блока.
7. **Выходной хэш 256 бит:** после обработки всех блоков формируется итоговое значение хэша.
8. **Результат SHA-256:** полученное 256-битное значение является результатом хэширования исходных данных.

SHA-256 широко используется для хэширования паролей в веб-приложениях. При регистрации пользователя его пароль хэшируется с использованием SHA-256 и сохраняется в базе данных. При последующих попытках входа введённый пользователем пароль хэшируется и сравнивается с хэшем, хранящимся в базе данных. Если хэши совпадают, пользователь аутентифицируется.

Для повышения безопасности часто используется техника, известная как "соль". Соль — это случайное значение, которое добавляется к паролю перед хэшированием. Это предотвращает использование предвычисленных таблиц (таблицы радуги) для нахождения оригинального пароля по хэшу. В итоге хэш пароля и соль хранятся вместе, и при аутентификации соль извлекается и используется повторно для хэширования введённого пароля.

Одним из самых надежных и широко используемых методов шифрования является алгоритм **AES-256 (Advanced Encryption Standard)** с ключом длиной 256 бит. Этот алгоритм был принят в качестве стандарта шифрования правительством США и используется для защиты как конфиденциальной, так и не конфиденциальной информации.

Алгоритм AES был разработан бельгийскими криптографами Винсентом Рейменом и Джоаном Дайменом. В 1998 году они представили алгоритм Rijndael, который впоследствии был выбран Национальным институтом стандартов и технологий США (NIST) в качестве стандарта шифрования. В 2001 году алгоритм был официально утвержден как стандарт AES (FIPS 197). AES-256 является одним из вариантов этого алгоритма, где используется ключ длиной 256 бит для шифрования и дешифрования данных.

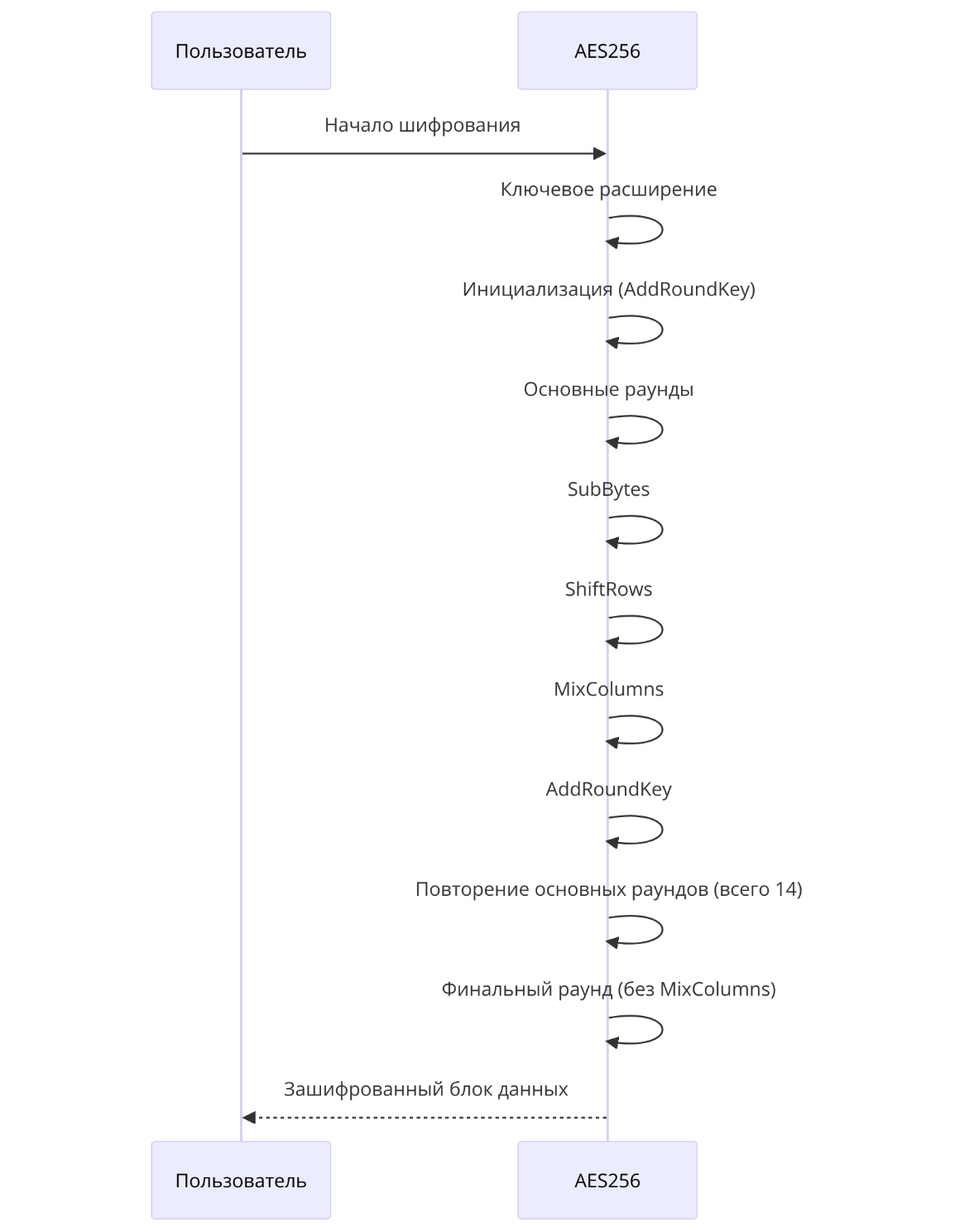
Алгоритм AES-256 представляет собой блочный шифр, который работает с фиксированными блоками данных размером 128 бит. Процесс шифрования включает в себя несколько этапов, каждый из которых добавляет уровень сложности и безопасности к зашифрованным данным.

Рисунок 2.2 - Этапы работы AES-256

Основные этапы работы AES-256 включают:

1. **Ключевое расширение (Key Expansion):** исходный ключ длиной 256 бит расширяется до набора субключей, которые используются на каждом раунде шифрования. Этот процесс включает в себя использование S-боксов (подстановочных блоков) и циклических сдвигов для генерации уникальных субключей.
2. **Инициализация (Initial Round):** в начальном раунде исходный блок данных XOR-ится с начальным субключом. Это этап называется "AddRoundKey".
3. **Основные раунды (Main Rounds):** AES-256 использует 14 основных раундов для шифрования данных. Каждый раунд включает четыре основных операции:
   * **SubBytes:** каждый байт блока данных заменяется на соответствующий байт из S-бокса.
   * **ShiftRows:** строки блока данных циклически сдвигаются на определенное количество байтов.
   * **MixColumns:** столбцы блока данных перемешиваются с использованием линейных преобразований.
   * **AddRoundKey:** текущий блок данных XOR-ится с соответствующим субключом.
4. **Финальный раунд (Final Round):** последний раунд шифрования аналогичен основным раундам, за исключением того, что операция MixColumns пропускается. После выполнения всех раундов получаем зашифрованный блок данных.

AES-256 является одним из самых безопасных алгоритмов шифрования благодаря длине ключа в 256 бит, обеспечивающей высокий уровень защиты от атак методом перебора, стойкости к криптоанализу из-за сложной структуры и множества раундов шифрования, а также широкому применению в государственных и коммерческих системах, банковских и финансовых приложениях, личных устройствах и облачных сервисах.

В менеджерах паролей алгоритм AES-256 используется для шифрования хранимых данных, таких как пароли, ключи доступа и конфиденциальная информация пользователей, обеспечивая их защиту при хранении в базе данных, передаче данных между клиентом и сервером, а также для создания безопасных токенов аутентификации.

Алгоритм **RSA (Rivest-Shamir-Adleman)** является одним из наиболее широко используемых методов асимметричного шифрования и аутентификации. Этот алгоритм был разработан в 1977 году и назван в честь его создателей Рональда Ривеста, Ади Шамира и Леонарда Адлемана. Основная идея RSA заключается в использовании двух различных ключей для шифрования и дешифрования информации: публичного и приватного ключей.

В отличие от симметричных алгоритмов шифрования, которые используют один и тот же ключ для шифрования и расшифровки данных, RSA использует два различных ключа: открытый и закрытый. Эти ключи связаны математически, но зная один ключ, невозможно вычислить другой за разумное время. Это свойство делает RSA чрезвычайно полезным для обеспечения конфиденциальности и аутентичности в цифровых коммуникациях.

**Открытый ключ** используется для шифрования данных и может свободно распространяться. **Закрытый ключ**, с другой стороны, используется для расшифровки данных и должен храниться в тайне.

Процесс шифрования и расшифровки:

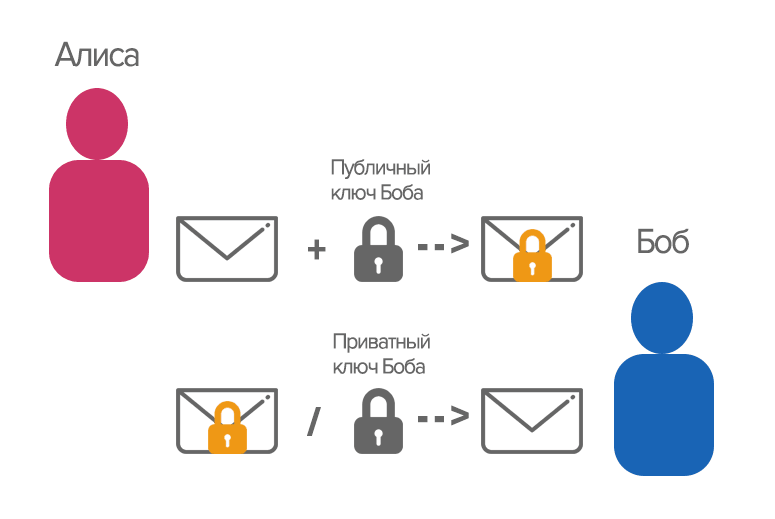
1. **Генерация ключей:**
   * Создание пары ключей начинается с генерации двух больших случайных простых чисел. Эти числа затем используются для создания открытого и закрытого ключей.
   * Открытый ключ состоит из модуля (произведения этих двух чисел) и экспоненты, которая обычно является небольшим фиксированным числом, например, 65537.
   * Закрытый ключ генерируется на основе тех же простых чисел и открытого ключа, но процесс его создания гораздо более сложен и требует значительных вычислительных ресурсов.
2. **Шифрования:**
   * Отправитель использует открытый ключ получателя для шифрования сообщения. Алгоритм RSA преобразует исходное сообщение в зашифрованный текст с помощью открытого ключа.
   * Процесс шифрования заключается в возведении исходного сообщения в степень, равную экспоненте открытого ключа, и взятии остатка от деления на модуль.
3. **Расшифровка:**
   * Получатель использует свой закрытый ключ для расшифровки сообщения. Закрытый ключ позволяет вычислить исходное сообщение из зашифрованного текста.
   * Процесс расшифровки заключается в возведении зашифрованного текста в степень, равную закрытому ключу, и взятии остатка от деления на модуль.

Рисунок 2.3 - Процесс шифрования и расшифровки RSA

RSA используется в различных приложениях, включая цифровую подпись (для подтверждения подлинности и целостности сообщений с помощью закрытых и открытых ключей), шифрование данных (для защиты конфиденциальной информации путем обмена симметричными ключами) и аутентификацию (для проверки пользователей и устройств, например, в протоколах SSL / TLS для установления безопасных соединений между веб-браузерами и серверами).

Менеджеры паролей используют RSA для шифрования паролей перед передачей. В частности, открытый ключ сервера применяется для шифрования паролей, сгенерированных на стороне клиента, перед их передачей на сервер для безопасного хранения. Кроме того, мастер-ключи, используемые для шифрования паролей пользователей, защищены с помощью RSA и хранятся на сервере. Доступ к таким ключам возможен только через соответствующий закрытый ключ, что обеспечивает защиту данных даже в случае взлома сервера.

## 2.2. Аутентификация и управление доступом

Безопасное хранение паролей невозможно без надежных методов аутентификации и управления доступом. Эти методы обеспечивают защиту от несанкционированного доступа и минимизируют риск компрометации данных.

Одним из ключевых методов, обеспечивающих высокий уровень защиты, является **двухфакторная аутентификация (2FA).** Этот метод требует от пользователя подтверждения личности с использованием двух различных факторов: что-то, что он знает (например, пароль), и что-то, что он имеет (например, мобильный телефон для получения SMS-кода, приложения-аутентификатор или электронный адрес). 2FA значительно повышает уровень безопасности, так как даже при компрометации пароля злоумышленнику потребуется второй фактор для доступа.

Основные этапы работы 2FA включают:

1. **Ввод пароля:** пользователь вводит свой стандартный пароль для входа в систему. Это первый фактор аутентификации, который подтверждает, что пользователь знает секретный код.
2. **Проверка второго фактора:** после ввода пароля система запрашивает второй фактор аутентификации. Это может быть одноразовый код, отправленный на мобильное устройство пользователя, или биометрический идентификатор, такой как отпечаток пальца или скан лица.
3. **Подтверждение личности:** пользователь вводит одноразовый код или предоставляет биометрические данные. Система проверяет этот второй фактор и, если он подтвержден, предоставляет доступ к учетной записи.

Виды двухфакторной аутентификации:

1. **SMS-коды.** Наиболее распространенным методом двухфакторной аутентификации (2FA) является отправка одноразового пароля на зарегистрированный номер мобильного телефона пользователя по SMS, затем пользователь вводит код для аутентификации. Однако у этого метода есть недостаток - SMS-сообщения могут быть перехвачены злоумышленниками.
2. **Приложения-аутентификаторы.** Такие приложения, как Google Authenticator и Authy, генерируют одноразовые пароли (Otps), которые обновляются каждые 30/60 секунд. Эти пароли синхронизируются с сервером и обеспечивают более надежный метод аутентификации, чем SMS.
3. **Аппаратные токены.** Физические устройства, такие как YubiKeys, которые генерируют одноразовые коды или обеспечивают аутентификацию по USB не подвержены фишинговым атакам или перехвату, что делает их надежным вариантом.
4. **Биометрические методы.** Использование биометрических методов аутентификации, таких как сканирование отпечатков пальцев, распознавание лица или сканирование радужной оболочки глаза, становится все более распространенным. Эти методы обеспечивают высокий уровень безопасности и удобства, поскольку каждый биометрический признак уникален для человека и его трудно воспроизвести.
5. **Push-уведомления.** Этот метод отправляет уведомление на мобильное устройство пользователя с просьбой подтвердить вход в систему. Пользователь может одобрить или отклонить запрос, просто нажав на уведомление.
6. **Одноразовые коды на E-mail.** Тоже самое что и SMS-коды, только уже на зарегистрированный E-mail пользователя.

Двухфакторная аутентификация является важным компонентом современных систем безопасности, включая менеджеры паролей. Она повышает безопасность, обеспечивая защиту от попыток фишинга и других угроз. Это добавляет дополнительный уровень контроля и обеспечивает удобство пользователей, а также отвечает требованиям нормативно-правового соответствия. Внедрение двухфакторной аутентификации в системы управления паролями позволяет создать надежную и безопасную платформу для работы с конфиденциальными данными.

**Управление правами доступа (Access Control)** играет не последнюю роль в обеспечении безопасности данных, так как позволяет регулировать и ограничивать доступ пользователей к конфиденциальной информации на основе установленных правил и политик. В менеджерах паролей, особенно в корпоративных системах, эффективное управление правами доступа минимизирует риск утечек и обеспечивает контроль над доступом к информации.

Основные концепции управления правами доступа:

1. **Мандатное управление доступом (Mandatory Access Control, MAC):** в системе с мандатным управлением доступом права доступа определяются центральным администратором на основе предопределенных правил безопасности. Пользователи не могут изменять свои права доступа. Этот подход используется в высокозащищенных системах, где требуется строгий контроль за доступом к информации.
2. **Дискреционное управление доступом (Discretionary Access Control, DAC):** в системе с дискреционным управлением доступом владелец информации (например, пользователь, создавший запись) может самостоятельно определять, кто будет иметь доступ к его данным. Это более гибкий подход, позволяющий пользователям управлять доступом к своей информации.
3. **Ролевое управление доступом (Role-Based Access Control, RBAC):** ролевое управление доступом основано на назначении прав доступа пользователям на основе их ролей в организации. Роли определяют, какие действия могут выполнять пользователи и к какой информации они могут получить доступ. Например, в корпоративном менеджере паролей может быть создана роль "Администратор" с полным доступом ко всем паролям и роль "Сотрудник" с доступом только к личным паролям. RBAC обеспечивает централизованное управление доступом и упрощает администрирование.
4. **Атрибутивное управление доступом (Attribute-Based Access Control, ABAC):** в атрибутивном управлении доступом решения о предоставлении доступа принимаются на основе множества атрибутов, таких как роль пользователя, время доступа, местоположение и другие. Этот подход позволяет создавать более гибкие и детализированные правила доступа.

Реализация управления правами доступа в корпоративных менеджерах паролей:

1. **Определение ролей и прав:** на этапе проектирования системы администраторы определяют роли и права доступа, соответствующие организационной структуре и требованиям безопасности. Это может включать создание ролей для администраторов, менеджеров и сотрудников с различными уровнями доступа.
2. **Назначение прав доступа:** пользователям назначаются роли на основе их должностных обязанностей и уровня доверия. Например, администраторы могут назначать пользователям роли "Сотрудник" с доступом к личным паролям и "Менеджер" с доступом к паролям своей команды.
3. **Мониторинг и управление:** администраторы регулярно мониторят журналы действий и анализируют отчеты об активности для выявления и предотвращения возможных угроз. При необходимости права доступа могут быть изменены или отозваны.
4. **Обучение и информирование пользователей:** важным аспектом управления доступом является обучение пользователей и информирование их о политике безопасности. Пользователи должны быть осведомлены о важности соблюдения правил доступа и последствиях их нарушения.

Эффективное управление правами доступа в менеджерах паролей обеспечивает надежную защиту данных, предотвращает несанкционированный доступ и минимизирует риск компрометации. Это важный элемент комплексной стратегии безопасности, который помогает организациям защищать свои конфиденциальные данные и поддерживать высокий уровень безопасности. Менеджеры паролей, такие как PassWork, предоставляют администратору инструменты для аудита и мониторинга, что способствует поддержанию высокого уровня безопасности.

**Политика сложности паролей** – важный компонент системы безопасной аутентификации, поскольку она также помогает обеспечить надежность и целостность учетных записей пользователей. Она призвана предотвратить использование слабых паролей, которые могут быть легко скомпрометированы злоумышленниками. Средства управления паролями часто включают в себя функции, которые обеспечивают соблюдение этих политик, гарантируя, что пользователи придерживаются лучших практик в области безопасности паролей.

Основные аспекты политики паролей:

* **Минимальная длина пароля.** Одним из основных требований комплексной политики безопасности является установление минимальной длины пароля. Как правило, до недавних пор, рекомендовалось использовать пароли длиной не менее 8 символов, а на сегодняшний день это значение изменилось на 12 символов, поскольку более короткие пароли более восприимчивы к атакам методом перебора (brute-force). Для обеспечения базового уровня защиты система может обязать использовать пароли длиной не менее 12 символов.
* **Использование символов разных типов.** Для повышения надежности паролей рекомендуется использовать различные типы символов, такие как заглавные и строчные буквы, цифры и специальные символы (например, !, @, #, $). Такая практика делает пароли более сложными для угадывания и менее восприимчивыми к различным формам атак. Например, надежный пароль должен включать как минимум одну заглавную букву, одну строчную букву, одну цифру и один специальный символ.
* **Запрет на использование общих и часто используемых паролей.** Многие пользователи предпочитают выбирать простые и легко запоминающиеся пароли, такие как «password», «123456» или «qwerty». Политика сложности паролей должна предотвращать использование таких паролей, а также общих фраз и последовательностей. Система может, например, сверяться со списками слабых паролей и блокировать их использование.
* **Регулярная смена паролей.** Важным аспектом политики безопасности является требование регулярной смены паролей. Пользователи должны менять свои пароли через определенные промежутки времени (например, каждые 90 дней), чтобы минимизировать риск компрометации. Например, система может напоминать пользователям о необходимости обновлять пароль каждые 90 дней и ограничивать доступ, если они этого не делают.
* **История паролей.** Для предотвращения повторного использования старых паролей рекомендуется вести историю паролей. Пользователи не должны иметь возможность использовать свои старые пароли. Пример: система может хранить последние 5 паролей пользователя и запрещать их повторное использование.
* **Обязательное использование уникальных паролей.** Пользователи должны использовать уникальные пароли для каждой учетной записи. Это предотвращает сценарии, когда компрометация одного пароля ведет к компрометации других учетных записей. К примеру, система может предупреждать пользователя, если новый пароль совпадает с паролем, использованным для другой учетной записи.
* **Политика использования паролей в корпоративной среде.** В корпоративных менеджерах паролей политика сложности паролей может включать дополнительные требования, такие как использование корпоративных стандартов и согласование с внутренними политиками безопасности. Корпоративная политика может требовать пароли длиной не менее 15 символов и включать дополнительные меры, такие как обязательное использование двухфакторной аутентификации.

**Удобство использования:** строгие требования к сложности паролей могут создавать для пользователей проблемы с запоминанием паролей, что может привести к хранению паролей в незащищенных местах. **Потребности в обучении:** организациям следует проводить обучение пользователей, чтобы они лучше понимали важность надежных паролей и соблюдения установленной политики паролей.

Примеры реализации политики паролей в менеджерах паролей:

1. **BitWarden.** Позволяет администраторам настраивать политику сложности паролей, включая минимальную длину, обязательное использование символов разных типов и запрет на использование слабых паролей.
2. **LastPass.** Включает встроенный генератор паролей, который создает сложные пароли, соответствующие установленным требованиям, и напоминает пользователям о необходимости смены паролей.

Политика сложности паролей является основополагающим элементом стратегии безопасности менеджеров паролей, обеспечивая надежную защиту учетных записей и конфиденциальной информации пользователей.

# Глава 3. Проектирование и реализация веб-приложения для безопасного хранения паролей

## 3.1. Формирование требований к разрабатываемому приложению

Основные требования можно разделить на **функциональные** и **нефункциональные**. Функциональные и нефункциональные требования к приложению включают регистрацию и аутентификацию пользователей, хранение и управление паролями, сброс и изменение паролей, двухфакторную аутентификацию, генератор паролей, безопасность, производительность и масштабируемость, кроссплатформенность, синхронизацию и резервное копирование, удобство использования, документацию и поддержку.

Пользователь должен иметь возможность зарегистрировать аккаунт, указав уникальные логин и электронную почту, пароль для авторизации и мастер-пароль. Аутентификация должна предоставлять возможность авторизации пользователей с использованием логина и пароля для авторизации, проверяя введенные данные с хэшами, хранящимися в базе данных. Пароли для авторизации и мастер-пароль должны храниться в базе данных в виде хэшей. Приложение должно предоставлять возможность пользователям добавлять, просматривать, редактировать и удалять данные своих аккаунтов различных сервисов (логины и пароли). Пароли от аккаунтов сервисов должны храниться в базе данных в зашифрованном виде с использованием алгоритма AES-256, а ключ шифрования должен быть производным от мастер-пароля пользователя.

Пользователи должны иметь возможность сброса пароля для авторизации посредством запроса письма на свою почту и ввода нового пароля через сгенерированную ссылку. В личном кабинете пользователи могут изменить свой пароль для авторизации, введя старый и новый пароли. Для повышения безопасности пользователи должны подтвердить свою личность, введя мастер-пароль при каждой новой сессии, чтобы получить доступ к сохраненным паролям от аккаунтов. Веб-приложение должно включать страницу с генератором паролей, где можно выбирать длину пароля, символы (заглавные, прописные, пунктуация) и возможность вводить свои символы.

Все пароли и мастер-пароли должны быть защищены с использованием алгоритма PBKDF2 с хэш-функцией SHA-256, а данные аккаунтов различных сервисов должны шифроваться алгоритмом AES-256. Все данные должны передаваться по защищенному протоколу HTTPS. Приложение должно соответствовать современным стандартам безопасности для защиты данных пользователей, включая регулярные обновления безопасности и устранение уязвимостей. Приложение должно обеспечивать быстрое время отклика для операций регистрации, авторизации и управления данными аккаунтов, при этом время шифрования и дешифрования данных должно быть минимальным, чтобы не замедлять работу пользователя. Система должна быть масштабируемой, чтобы справляться с увеличением объема данных без потери производительности.

Приложение должно быть адаптировано под различные устройства, включая персональные компьютеры, ноутбуки, планшеты и смартфоны. Система должна поддерживать синхронизацию данных между устройствами пользователя и возможность создания резервных копий данных для восстановления информации. Интерфейс приложения должен быть интуитивно понятным и удобным для пользователя, а процессы регистрации, авторизации и управления аккаунтами должны быть максимально упрощены для удобства пользователей.

Вся функциональность приложения должна быть подробно задокументирована, включая инструкции по установке и использованию приложения. Пользователям должна быть доступна квалифицированная техническая поддержка для решения возникающих проблем. Приложение должно обеспечивать высокую степень надежности и доступности, минимизируя время простоя, а все критические операции, такие как регистрация и авторизация, должны быть защищены от возможных сбоев.

**Таблица 3.1.**

**Основные требования к приложению.**

|  |  |
| --- | --- |
| Функциональные требования | Нефункциональные требования |
| Регистрация пользователей | Безопасность |
| Аутентификация пользователей | Производительность |
| Управление аккаунтами сервисов | Масштабируемость |
| Безопасное хранение данных | Кроссплатформенность |
| Восстановление доступа | Синхронизация и резервное копирование |
| Управление сессиями | Удобство использования |
| Сброс и изменение паролей | Надежность |
| Двухфакторная аутентификация | Документация |
| Генератор паролей | Техническая поддержка |

## 3.2. Выбор реализуемых технологий и средств разработки

Для выбора технологий и средств разработки веб-приложения для безопасного хранения паролей необходимо учитывать множество факторов, таких как безопасность, производительность, удобство использования и совместимость с различными платформами. Современные технологии предлагают широкий спектр возможностей для создания надежных и функциональных веб-приложений.

Первоначально следует рассмотреть язык программирования для серверной части приложения. Одним из наиболее популярных и надежных решений является использование языка ***Python*** в сочетании с фреймворком ***Django***. ***Python*** известен своей простотой и читаемостью, что ускоряет процесс разработки и упрощает сопровождение кода. ***Django***, в свою очередь, предоставляет широкий набор инструментов для построения безопасных и масштабируемых веб-приложений. Этот фреймворк поддерживает архитектуру ***MVT (Model-View-Template)***, что способствует упорядочению кода и упрощает его поддержку. Важным аспектом является встроенная система управления пользователями и аутентификации, которая предоставляет готовые решения для хэширования паролей. Для хранения паролей пользователей и мастер-паролей будет использоваться стандартный алгоритм и хэш-функция ***Django***, что обеспечивает высокий уровень безопасности. ***Django***, как один из ведущих веб-фреймворков для ***Python***, зарекомендовал себя благодаря высокому уровню встроенных механизмов защиты. Ниже мы рассмотрим основные аспекты безопасности ***Django***, обеспечивающие защиту приложений от различных угроз:

1. **Защита от XSS (Межсайтовый скриптинг)**

Django эффективно предотвращает XSS-атаки за счёт автоматического экранирования опасных символов в шаблонах. Однако, при использовании тегов и методов, таких как «safe», «mark\_safe» и «is\_safe», необходимо проявлять осторожность, так как они могут снизить уровень защиты. Особого внимания требует хранение HTML-кода в базе данных, что может представлять дополнительные риски.

1. **Защита от CSRF (Межсайтовая подделка запросов)**Django включает встроенную защиту от CSRF-атак, проверяя наличие секретного токена в каждом POST-запросе. Отключение данной защиты допустимо лишь в исключительных случаях. Дополнительно, использование HTTPS рекомендуется для повышения уровня безопасности данных.
2. **Защита от SQL-инъекций**

Django ORM автоматически параметризует запросы, что существенно снижает риск SQL-инъекций. Однако, при использовании сырых SQL-запросов через методы «raw», «extra», «RawSQL», необходимо вручную экранировать параметры для предотвращения уязвимостей.

1. **Защита от Clickjacking**

Для предотвращения Clickjacking-атак Django предоставляет Middleware, использующий заголовок X-Frame-Options. Это позволяет избежать отображения вашего сайта в рамках других сайтов, что может использоваться злоумышленниками.

1. **SSL/HTTPS**Использование HTTPS является критически важным для защиты данных при их передаче. Django поддерживает настройки для перенаправления HTTP-запросов на HTTPS, использования защищённых cookies и заголовка HSTS (HTTP Strict Transport Security), который гарантирует, что все будущие подключения будут осуществляться только по HTTPS.
2. **Проверка заголовка Host**

Django проверяет заголовок Host против списка допустимых значений, заданного в настройке ALLOWED\_HOSTS. Это предотвращает атаки типа Cross-Site Request Forgery и другие возможные угрозы.

1. **Политика Referrer**

Настройка политики Referrer помогает защитить конфиденциальность пользователей, ограничивая условия, при которых заголовок Referer передаётся браузером.

1. **Политика Cross-Origin Opener**

Заголовок COOP (Cross-Origin Opener Policy) позволяет изолировать контекст верхнего уровня от других документов, обеспечивая защиту от межсайтовых атак.

1. **Безопасность сессий**

Django предлагает встроенные механизмы защиты сессий, однако важно учитывать ограничения и следовать рекомендациям по защите от атак на сессии.

1. **Загружаемый пользователями контент**

При работе с загружаемыми файлами следует ограничивать их размер и тип, чтобы предотвратить атаки. Рекомендуется использовать отдельный домен для обслуживания загружаемого контента.

Для хранения данных на этапе разработки будет использоваться база данных ***SQLite***.***SQLite*** является легковесной и встроенной базой данных, которая не требует сложной настройки и отлично подходит для разработки и тестирования приложений. При деплое приложения на сервер будет использоваться ***PostgreSQL***, которая является мощной и масштабируемой реляционной базой данных, обеспечивающей высокую производительность и надежность. ***PostgreSQL*** поддерживает сложные запросы, транзакции и репликацию, что делает её идеальным выбором для работы с большими объемами данных.

Фронтенд приложения будет разрабатываться с использованием ***Tailwind CSS*** версии 3.4.3. ***Tailwind CSS*** — это утилитарный CSS-фреймворк, который позволяет создавать адаптивные и настраиваемые интерфейсы без необходимости писать много кастомных стилей. Использование ***Tailwind CSS*** обеспечивает быструю разработку и легкую поддержку стилей, что особенно важно для создания современного и отзывчивого пользовательского интерфейса.

Для шифрования паролей от аккаунтов различных сервисов пользователей будет использоваться алгоритм ***AES-256***, который реализован через библиотеку ***PyCryptodome***. ***AES-256*** признан одним из самых надежных алгоритмов симметричного шифрования, что обеспечивает высокий уровень безопасности данных. Ключ для шифрования паролей будет производной от мастер-пароля пользователя, что добавляет дополнительный уровень защиты, минимизируя риск несанкционированного доступа к зашифрованным данным. Вовсе необязательно использовать алгоритм шифрования AES-256; можно заменить его любым другим криптографическим процессором без каких-либо изменений в функционировании приложения.

Важным аспектом безопасности является двухфакторная аутентификация. В данном приложении двухфакторная аутентификация будет реализована с использованием мастер-пароля пользователя. Пользователь сможет авторизоваться без ввода мастер-пароля, однако для получения доступа к данным своих аккаунтов он должен будет подтвердить свою личность, введя мастер-пароль. Это обеспечит дополнительный уровень защиты конфиденциальной информации.

Для управления версиями кода и организации процесса разработки будет использоваться система контроля версий ***Git*** и платформа для хостинга репозиториев ***GitHub***. ***Git*** является стандартом де-факто для контроля версий, обеспечивая возможность отслеживания изменений в коде, отката к предыдущим версиям и организации командной работы, даже если разработка ведётся в одиночку. ***GitHub*** предоставляет удобный интерфейс для управления репозиториями, а также инструменты для совместной работы и автоматизации процессов развертывания и тестирования.

Таким образом, выбор технологий и средств разработки для веб-приложения по безопасному хранению паролей был сделан с учетом необходимости обеспечения безопасности, производительности и удобства использования. Использование ***Python*** и ***Django*** для серверной части, ***SQLite*** и ***PostgreSQL*** для хранения данных, ***Tailwind*** ***CSS*** для создания адаптивного интерфейса, а также ***AES-256*** для шифрования данных и ***Git*** с ***GitHub*** для управления версиями кода обеспечит создание надежного и функционального веб-приложения.

## 3.3 Реализация приложения

Реализация веб-приложения для безопасного хранения паролей представляет собой сложный и многогранный процесс, включающий в себя разработку серверной части, клиентской части, а также обеспечение безопасности данных на всех этапах их обработки. В данном разделе рассматриваются основные этапы реализации, выбранные технологии и инструменты, а также ключевые аспекты безопасности, которые были учтены при разработке приложения.

**Этап №1: Инициализация проекта**

**Создание виртуального окружения.**

Прежде чем начать разработку, необходимо создать виртуальное окружение для изоляции зависимостей проекта. Для этого используем команду в терминале:

> python -m venv djvenv

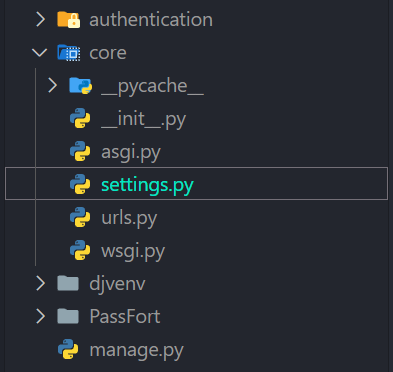
**Установка необходимых зависимостей.**

Активируем виртуальное окружение и устанавливаем Django, PyCryptodome и другие необходимые библиотеки в терминале:

> djvenv\Scripts\activate

> pip install django django-extensions pycryptodome pillow python-slugify

**Создание Django проекта и приложения.**

Создаем новый Djangoпроект под названием ***«core»*** и 2 приложения с названиями ***«authentication»*** и «***PassFort»*** в терминале:

> django-admin startproject core .

> python manage.py startapp authentication

> python manage.py startapp PassFort

У нас получится вот такой каталог (Рис. 3.1) с папками и файлами. Первое приложение для реализации авторизации с регистрацией, а второе – основное приложение с паролями. Для завершения инициализации проекта, необходимо внести правки в некоторые файлы.

Рисунок 3.1. Каталог.

**Внесение правок в файл *settings.py* в папке *core*.**

Включим допустимые адреса для приложения, укажем в списке установленных приложений ***«authentication»*** и ***«PassFort»***, а также дополнения ***«django\_extensions»***. Кроме того, добавим пути для шаблонов HTML, а также для статических и медиа файлов:

ALLOWED\_HOSTS = ["127.0.0.1", 'localhost']

INSTALLED\_APPS = [

……

"django\_extensions",

"authentication.apps.AuthenticationConfig",

"PassFort.apps.PassfortConfig",

]

TEMPLATES = [

{

……

"DIRS": [BASE\_DIR / "templates"],

……

},

]

LANGUAGE\_CODE = "ru-RU"

TIME\_ZONE = "Europe/Moscow"

STATIC\_URL = "static/"

STATICFILES\_DIRS = [BASE\_DIR / "static"]

MEDIA\_URL = "media/"

MEDIA\_ROOT = BASE\_DIR / "media"

Также следует добавить поля для ввода данных электронной почты, чтобы обеспечить возможность отправки писем по email с использованием протокола SMTP:

EMAIL\_BACKEND = "django.core.mail.backends.smtp.EmailBackend"

EMAIL\_HOST = "smtp.mail.ru"

EMAIL\_PORT = 465

EMAIL\_HOST\_USER = "passfort@internet.ru"

EMAIL\_HOST\_PASSWORD = "fwRrmwgvwkuM0ahjyxER"

EMAIL\_USE\_TLS = False

EMAIL\_USE\_SSL = True

EMAIL\_ADMIN = EMAIL\_HOST\_USER

DEFAULT\_FROM\_EMAIL = EMAIL\_HOST\_USER

SERVER\_EMAIL = EMAIL\_HOST\_USER

Ещё необходимо добавить поля, которые указывают пользовательскую модель для аутентификации, а также URL-адреса для перенаправления после входа, выхода и при необходимости авторизации:

AUTH\_USER\_MODEL = "authentication.User"

LOGIN\_REDIRECT\_URL = "passfort:home"

LOGOUT\_REDIRECT\_URL = "authentication:login"

LOGIN\_URL = "authentication:login"

Эти поля не выполняют никаких операций; они являются лишь константами, которые Django использует в процессе инициализации веб-приложения. Последний шаг на этом этапе заключается в добавлении путей в файле ***urls.py***, который находится в той же папке ***core***:

urlpatterns = [

……

path("", include("PassFort.urls", namespace="passfort")),

path("user/", include("authentication.urls", namespace="authentication")),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Этот код конфигурирует URL маршрутизацию для Django приложения, связывая основные пути с другими URL-конфигурациями из приложений ***«PassFort»*** и ***«authentication»***, а также добавляет поддержку для статических медиа файлов.

**Этап №2: Модели данных (Model)**

В Django используется концепция «Модель данных», которая определяет структуру таблицы, включая её столбцы и формат данных в них. Эти модели описываются в файле ***models.py*** в директориях соответствующих приложений. В рамках проекта будут созданы две основные таблицы: ***«User»*** и ***«Account»***. Первая предназначена для таблицы, хранящая данные пользователей, а вторая для хранения аккаунтов, добавляемых пользователями. Модель ***«User»*** будет размещена в папке ***authentication***, а модель ***«Account»*** – в папке ***PassFort***.

Начнем с модели ***«User»***. В Django по умолчанию предоставляется эта модель, которая включает поля: id, username, password, first\_name, last\_name, email и другие. Однако, для наших нужд я планирую установить уникальность для поля email, а также добавить к ней еще два поля. Это поле master\_password для хранения мастер-пароля пользователей и поле img для фотографий пользователей:

Рисунок 3.2. Модель "User"

Этот код определяет модель пользователя в Django, расширяя ***«AbstractUser»***, с уникальным email, мастер-паролем, аватаром и дополнительными методами для хэширования мастер-пароля при создании пользователя и получения мастер-пароля из сессии запроса.

Модель ***«Account»***:

Рисунок 3.3. Модель "Account".

Этот код определяет модель, представляющую учетную запись с полями для URL, имени пользователя, пароля и других данных. Импортируются необходимые библиотеки и модули, включая ***«urlparse»*** для обработки URL-адресов, модули Django для работы с моделями и пользователями, а также ***«AutoSlugField»*** и ***«slugify»*** для создания уникальных строковых идентификаторов (слагов).

Свойство ***username\_hostname*** используется для создания слага, объединяя имя пользователя и hostname URL-адреса, используя функцию ***slugify***.

Метод ***«save»*** переопределяется, чтобы обработать URL-адрес и создать слаг перед сохранением объекта. URL преобразуется для получения только hostname без префикса "www". Если слаг отсутствует, он создается на основе имени пользователя и hostname URL-адреса.

Теперь необходимо создать и применить миграции. В Django **миграции** представляют собой инструмент для управления изменениями в структуре базы данных. Этот механизм позволяет разработчикам без труда создавать, изменять и удалять таблицы и поля в базе данных, поддерживая их синхронизацию с моделями, описанными в коде.

**Создание и применение миграций.**

Миграции создаются и применяются двумя командами:

> python manage.py makemigrations

> python manage.py migrate

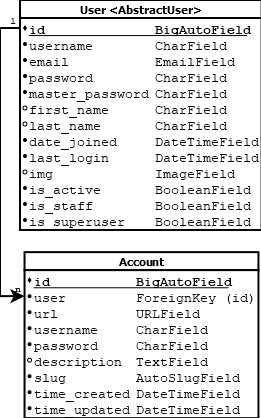
У нас создастся база данных примерно с такой структурой:

Рисунок 3.4. Диаграмма БД.

**Этап №3: Представления (View)**

После выполнения вышеупомянутых этапов следует приступить к разработке и реализации основной логики приложения, а именно его серверной части. За это отвечают контроллеры, которые в паттерне ***MVT (Model-View-Template)*** соответствуют представлениям ***(views)***. В файлах ***views.py*** каждого модуля веб-приложения определяются функции или классы представлений, которые инкапсулируют необходимую логику и активируются при обращении к определённым URL-адресам. Для примера полноценно разберём класс представления для авторизации пользователей:

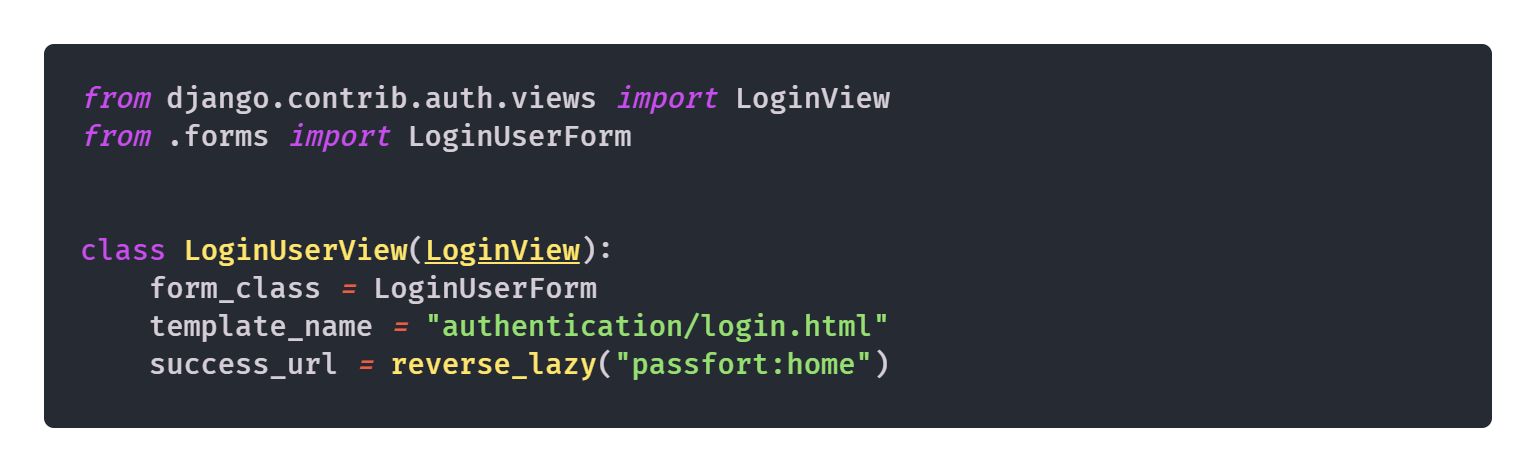
В данном фрагменте кода представлен класс ***LoginUserView***, который наследует стандартный класс представления аутентификации пользователей ***LoginView*** из библиотеки ***django.contrib.auth.views***. Этот код используется для реализации функционала входа пользователей на веб-сайт с использованием пользовательской формы и шаблона. Также импортируется ***LoginUserForm***, который представляет собой пользовательскую форму для входа, определённую в файле ***forms.py*** в текущем приложении. Основные атрибуты этого класса:

Рисунок 3.5. Класс представления для авторизации.

* ***form\_class:***Указывает, что для входа пользователя будет использоваться форма ***LoginUserForm***.
* ***template\_name:*** Определяет путь к HTML-шаблону, который будет использоваться для отображения страницы входа. Это позволяет разделить логику аутентификации и представление.
* ***success\_url:*** Задает URL, на который будет перенаправлен пользователь после успешного входа. Функция reverse\_lazy используется для ленивого разрешения URL-адресов, что полезно для предотвращения проблем с импортами в Django.

Остальные этапы обработки выполняются непосредственно фреймворком Django. Это преимущество Django, поскольку он предоставляет шаблонные и типовые функции, которые можно использовать после минимальных корректировок.

На этом же этапе рассмотрим реализацию пользовательских форм на примере вышеупомянутого ***LoginUserForm***.

Рисунок 3.6. Класс пользовательской формы.

В начале импортируются необходимое для создание пользовательской формы авторизации. Здесь можно выделить импорт ***AuthenticationForm***, который позволяет использовать базовую форму аутентификации, предоставляемую Django. Это встроенный класс, который включает в себя все необходимые поля и логику для проверки учетных данных пользователя.

После этого задаются поля формы для ввода данных, а именно ***username*** и ***password***, которые будут автоматически заполняться на основе ранее введенных значений (autocomplete). Для повышения удобства пользователей, внутри этих полей до ввода данных отображаются тексты-заполнители (placeholder), которые служат указаниями по правильному заполнению полей.

Метакласс ***«Meta»*** служит для определения дополнительных параметров формы. В данном контексте мы указываем, что моделью для формы является пользовательская модель, получаемая с помощью функции ***«get\_user\_model»***. Эта функция предоставляет возможность работы с пользовательскими моделями, если они заданы в проекте. Также в метаклассе определяются поля формы, такие как ***username*** и ***password***.

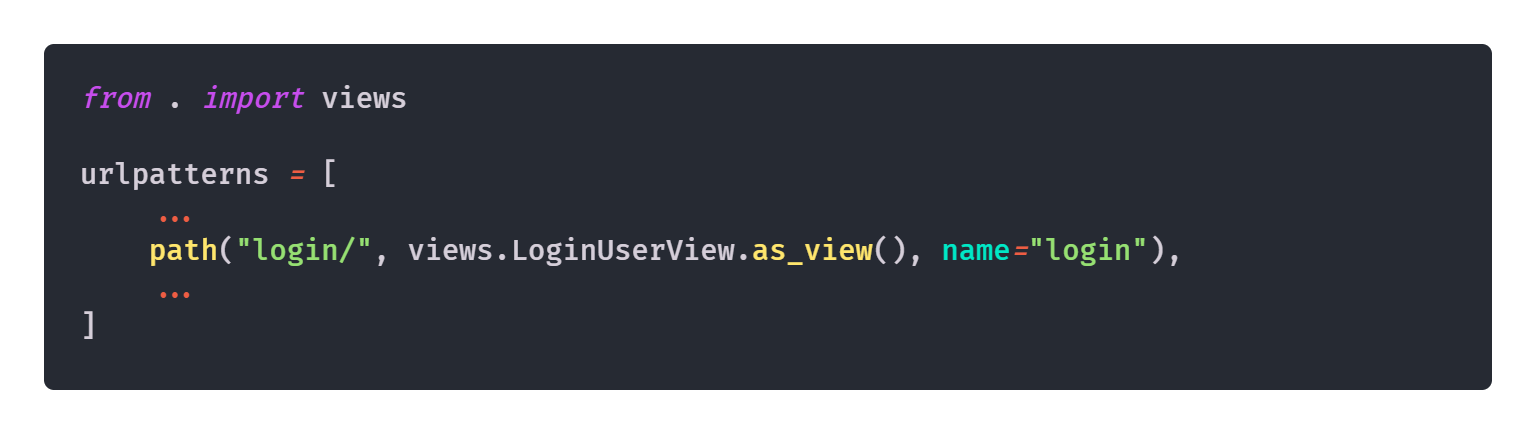
Осталось определить маршрут и метод, который будет вызван при обращении к этому маршруту. Для этого в файле ***urls.py*** соответствующего приложения, в нашем случае расположенного в директории ***«authentication»***, необходимо указать необходимые пути:

Рисунок 3.7. Маршрут для авторизации.

Фрагмент кода, представленный выше, является частью конфигурационного файла urls.py, который определяет соответствие между URL-запросами и их обработчиками в виде представлений (views). Рассмотрим данный фрагмент детально.

Первая строка выполняет импорт модуля views, который содержит определение классов и функций, обрабатывающих различные запросы. Это необходимо для того, чтобы маршруты могли использовать эти обработчики.

***urlpatterns*** — это список, в котором каждый элемент представляет собой путь (route), определяющий обработчик для конкретного URL.

Данный маршрут определяет URL для страницы входа пользователя:

* path("login/", ...) указывает, что когда пользователь переходит по адресу *«http://<домен>/login/»,* запрос должен быть обработан указанным представлением, но так, как были прописаны пути в корневом urls.py, то адресом будет *«http://<домен>/user/login/».*
* views.LoginUserView.as\_view() представляет собой обращение к классу LoginUserView, который отвечает за обработку входа пользователя. Метод as\_view() превращает класс представления в функцию, совместимую с маршрутизацией Django.
* name="login" задает имя маршрута. Это имя может быть использовано для ссылки на этот URL в шаблонах и других частях кода приложения. Использование именованных маршрутов повышает читаемость и облегчает поддержку кода.

**Этап №4: Шаблоны (Template)**

Последним и заключающем этапом является разработака шаблонов. Шаблоны являются важным компонентом веб-приложения, обеспечивая отображение данных пользователю в удобной и понятной форме. В Django шаблоны создаются с использованием языка шаблонов Django ***(Django Template Language, DTL)***, который предоставляет мощные инструменты для работы с HTML, CSS и динамическими данными. Создание адаптивного и интуитивно понятного пользовательского интерфейса представляет собой весьма трудоёмкий и сложный процесс. Для достижения удовлетворительного результата требуется проведение множества тестирований и глубокое понимание психологии пользователя. В случае, если итоговый интерфейс не соответствует стандартам удобства и эргономики, его необходимо дорабатывать и совершенствовать до тех пор, пока не будут достигнуты необходимые критерии качества. Мы должны получить файл ***login.html***, расположенный по пути ***«authentication/templates/authentication/login.html»,*** как это указано в соответствующем классе представления. После чего у нас должно получиться вот такая страница авторизации (Рисунок 3.8).

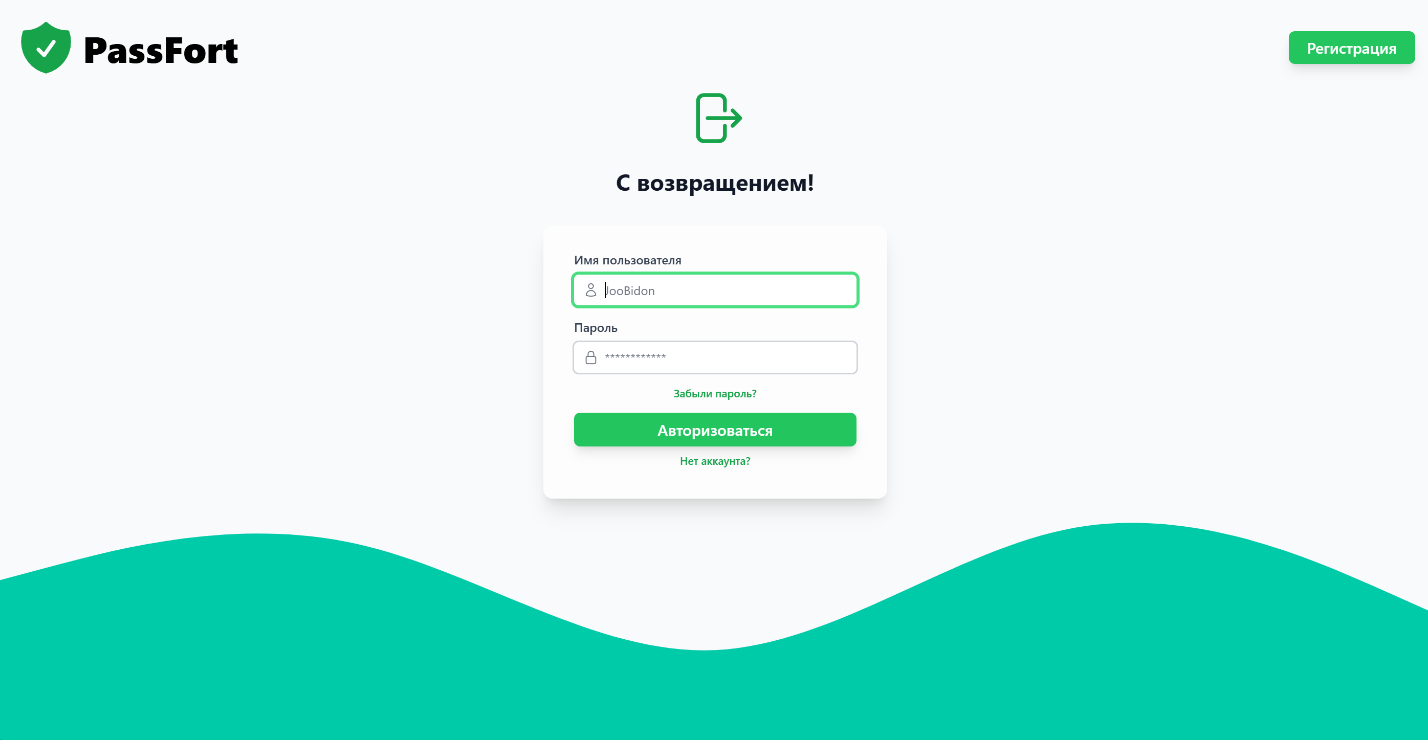
 Следуя данному поэтапному алгоритму разработки, можно создать веб-приложение с нуля до полной реализации. При этом, весь проект будет масштабируемым, что позволит в будущем при необходимости вносить дополнения и изменения.

Рисунок 3.8. Страница авторизации.

## 3.4 Демонстрация приложения

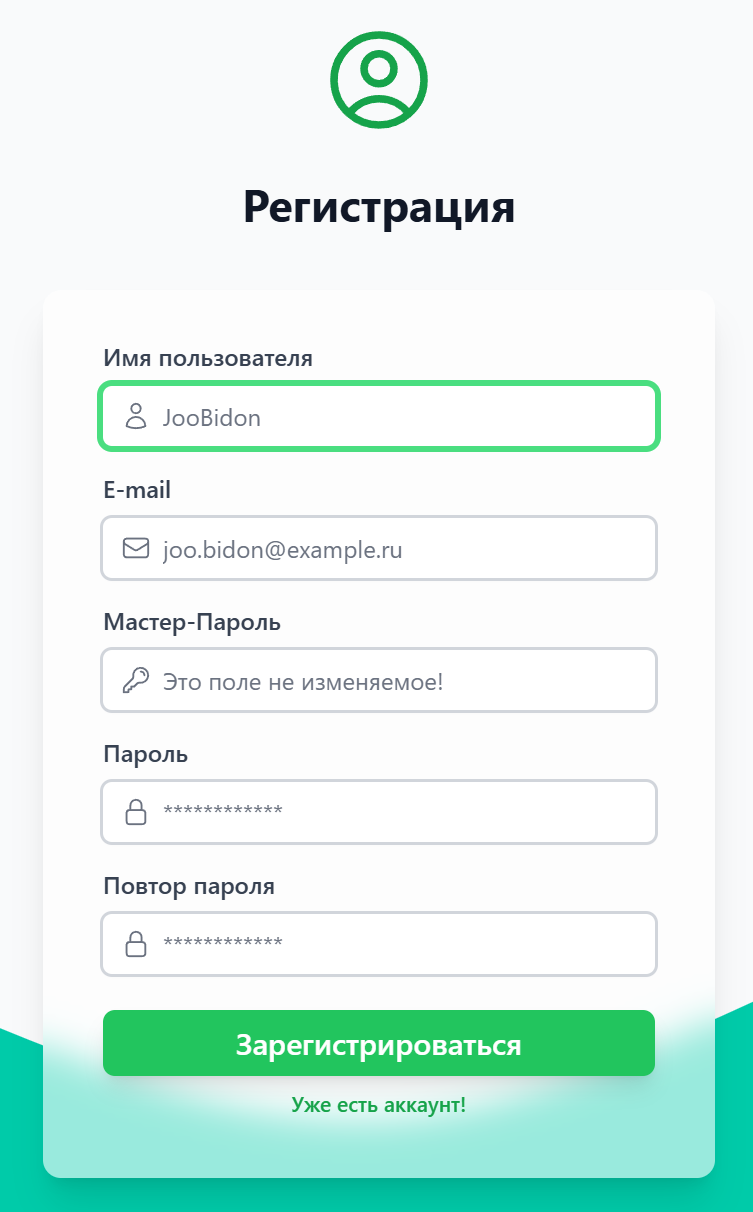
Для использования приложения требуется пройти процедуру авторизации, которая, в свою очередь, требует предварительной регистрации. Процесс регистрации осуществляется на веб-странице по адресу ***«https://<домен>/authentication/registration»:***

Рисунок 3.9. Страница регистрации.

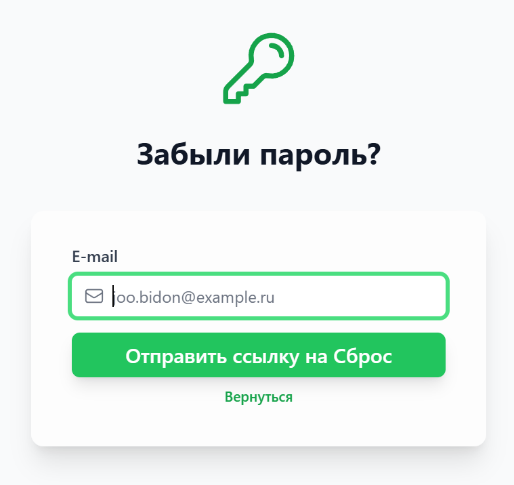
На странице авторизации пользователю предоставляется возможность восстановления пароля. Для этого необходимо подтвердить свою личность, введя адрес электронной почты, на который будет отправлена ссылка для сброса пароля.

Рисунок 3.10. Страница сброса пароля.

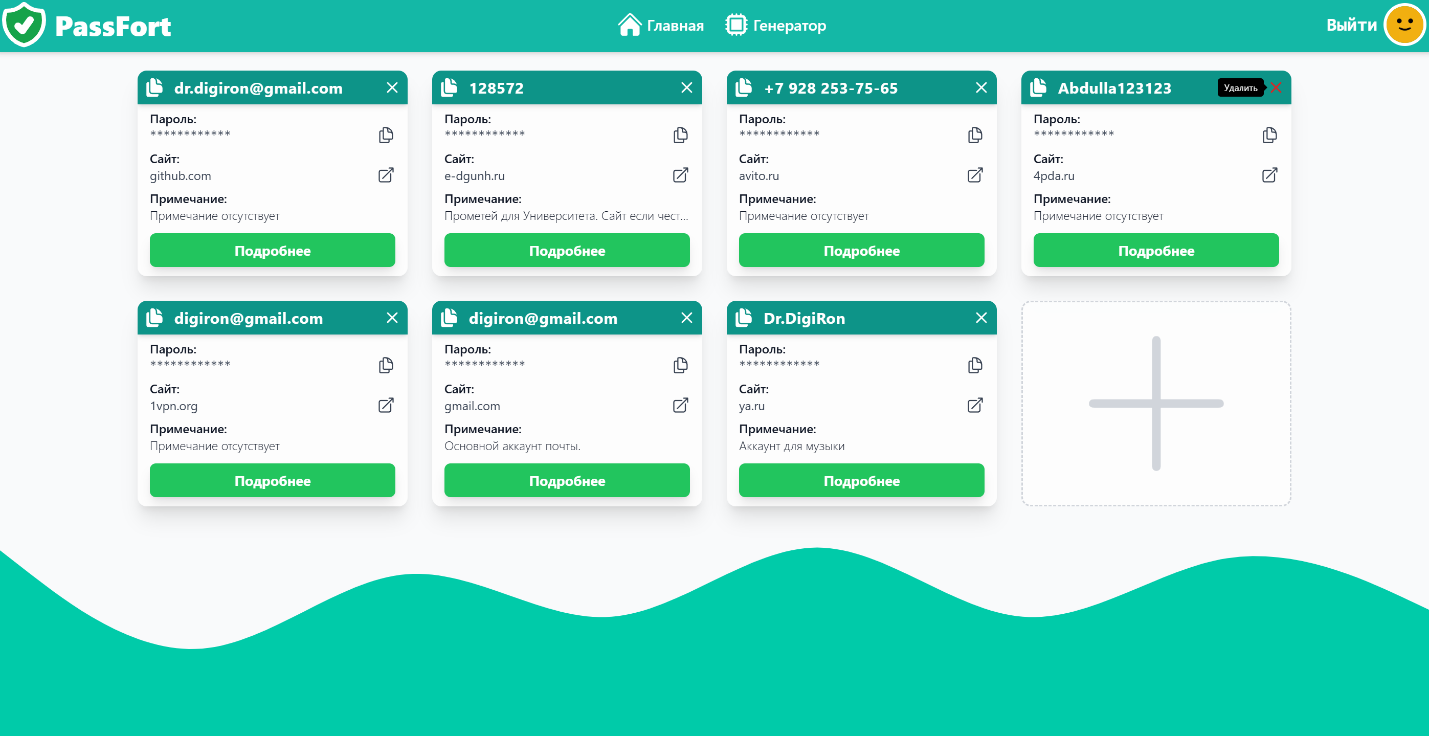
После успешного прохождения процедуры аутентификации пользователь перенаправляется на главную рабочую страницу (Рисунок 3.11). На данной странице предоставляется функциональность для просмотра, добавления, редактирования и удаления данных различных учетных записей. Дополнительно реализована возможность копирования логинов и паролей одним кликом, что существенно упрощает управление учетными данными.

Рисунок 3.11. Главная страница.

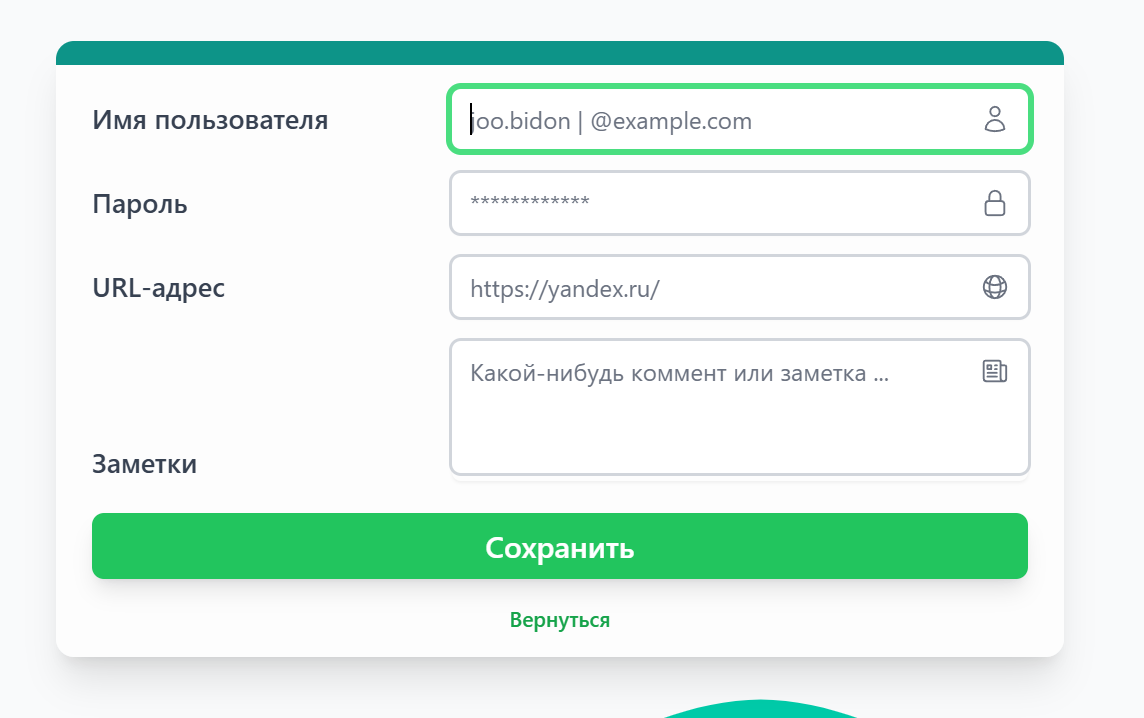
Форма для добавления учётной записи выглядит следующим образом:

Рисунок 3.12. Форма добавления учётной записи.

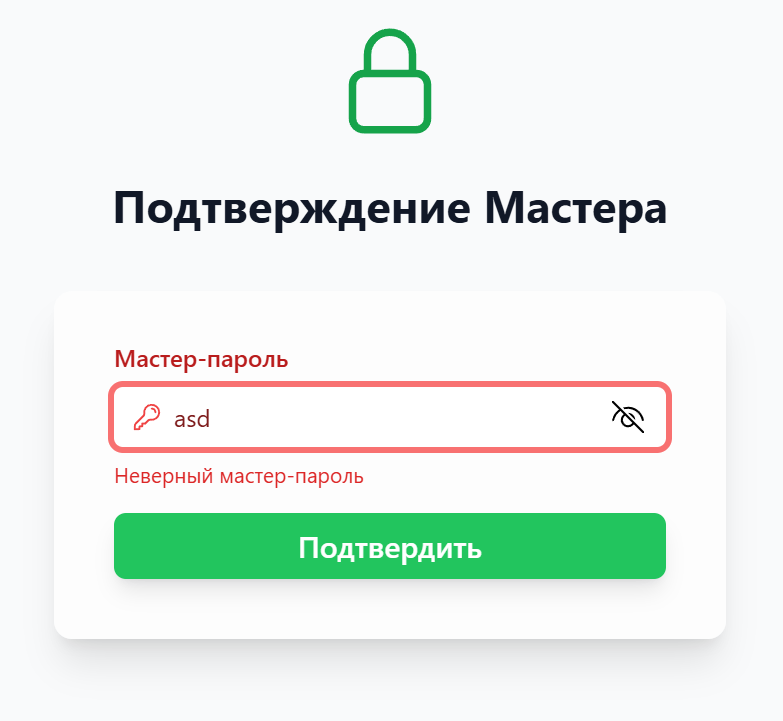
В случае, если в текущей сессии мастер-пароль не был подтверждён, выполнение любых операций с учетными записями становится невозможным. При попытке осуществить какое-либо действие, приложение запросит у пользователя ввод мастер-пароля (Рисунок 3.13).

Рисунок 3.13. Подтверждения мастер-пароля.

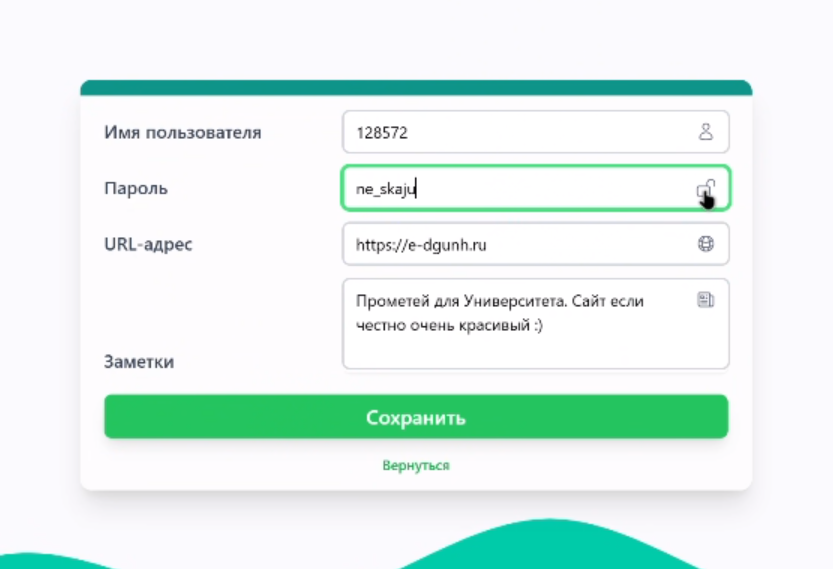
Также можно редактировать уже существующие аккаунты:

Рисунок 3.14. Окно редактирования.

Ещё имеется страница с профилем (Рисунок 3.15), где можно отредактировать данные, добавить/изменить фото и сменить пароль авторизации.

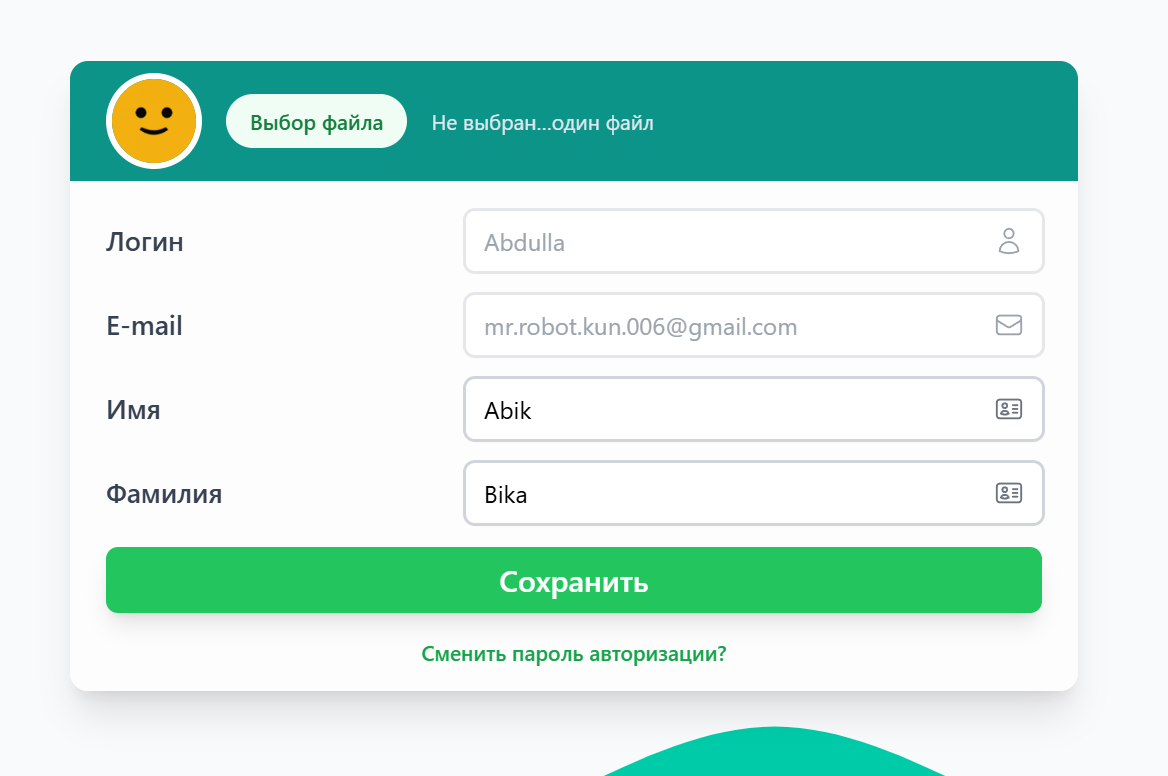
При добавлении или копировании пароля учетной записи, данные подвергаются процессам шифрования и расшифрования, используя мастер-пароль пользователя. Эти функции (Рисунок 3.16) реализованы в модели ***«Account»*** и поддерживаются библиотекой ***«PyCryptodome»***.

Рисунок 3.15. Страница профиля.

Рисунок 3.16. Крипто-функции.

Этот код (Рисунок 3.16) демонстрирует процесс шифрования и расшифрования паролей с использованием алгоритма ***AES-256 (Advanced Encryption Standard)*** в режиме ***GCM (Galois/Counter Mode)***, что обеспечивает как конфиденциальность, так и целостность данных.

Функция ***«generate\_key»*** используется для генерации криптографического ключа на основе мастер-пароля пользователя и соли (случайно сгенерированного значения). Для этого используется алгоритм ***PBKDF2 (Password-Based Key Derivation Function 2)***, который применяет хэш-функцию многократно (в данном случае миллион раз) для усложнения атаки перебором (brute-force attack).

Функция ***«encrypt\_password»*** осуществляет шифрование пароля. Основные шаги:

1. **Генерация соли:** Создается случайное значение соли длиной 16 байт.
2. **Генерация ключа:** Используется функция *«generate\_key»* для получения ключа шифрования на основе мастер-пароля и соли.
3. **Инициализация шифра:** Создается объект шифра AES в режиме GCM.
4. **Шифрование и вычисление тега аутентичности:** С помощью метода *«encrypt\_and\_digest»* шифруется пароль и одновременно вычисляется тег аутентичности.
5. **Формирование итогового зашифрованного сообщения:** Зашифрованные данные (соль, *nonce*, тег и зашифрованный текст) кодируются в формат *«base64»* для удобства хранения и передачи.

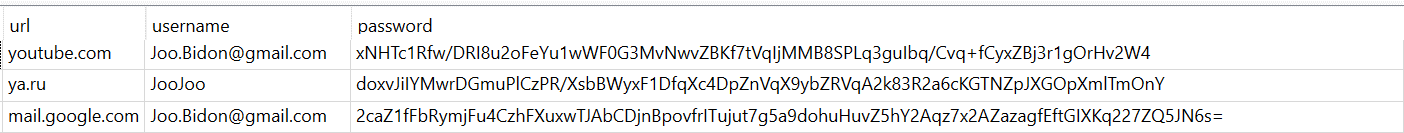
После проведения процедуры шифрования, пароли будут сохранены в следующем виде:

Рисунок 3.17. Зашифрованные пароли.

Функция ***«decrypt\_password»*** занимается расшифрованием зашифрованного пароля. Рассмотрим шаги подробнее:

1. **Декодирование данных:** Входные данные декодируются из формата *«base64».*
2. **Извлечение компонентов:** Из зашифрованных данных извлекаются соль, *nonce*, тег и зашифрованный текст.
3. **Генерация ключа:** Используется функция *«generate\_key»* для восстановления ключа на основе мастер-пароля и соли.
4. **Инициализация шифра:** Создается объект шифра AES в режиме GCM с *nonce*.
5. **Расшифрование и проверка целостности:** С помощью метода *«decrypt\_and\_verify»* происходит расшифрование текста и проверка целостности данных с использованием тега.
6. **Возврат расшифрованного текста:** Расшифрованный текст декодируется и возвращается.

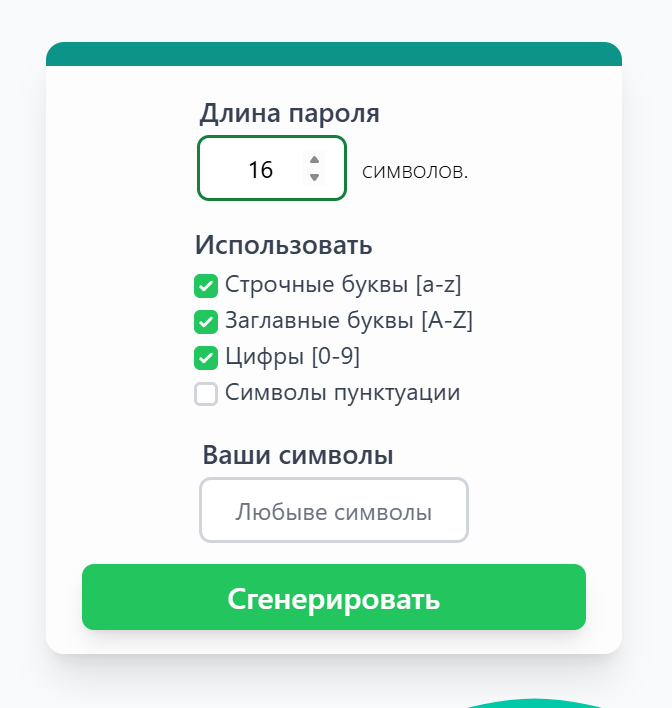
 Для упрощения создания пользователями сложных и надежных паролей, в приложении реализована страница генерации паролей (Рисунок 3.18). На данной странице пользователи могут задать необходимые параметры пароля и сгенерировать криптографически стойкий пароль, соответствующий заданным требованиям.

Рисунок 3.18. Генератор паролей.

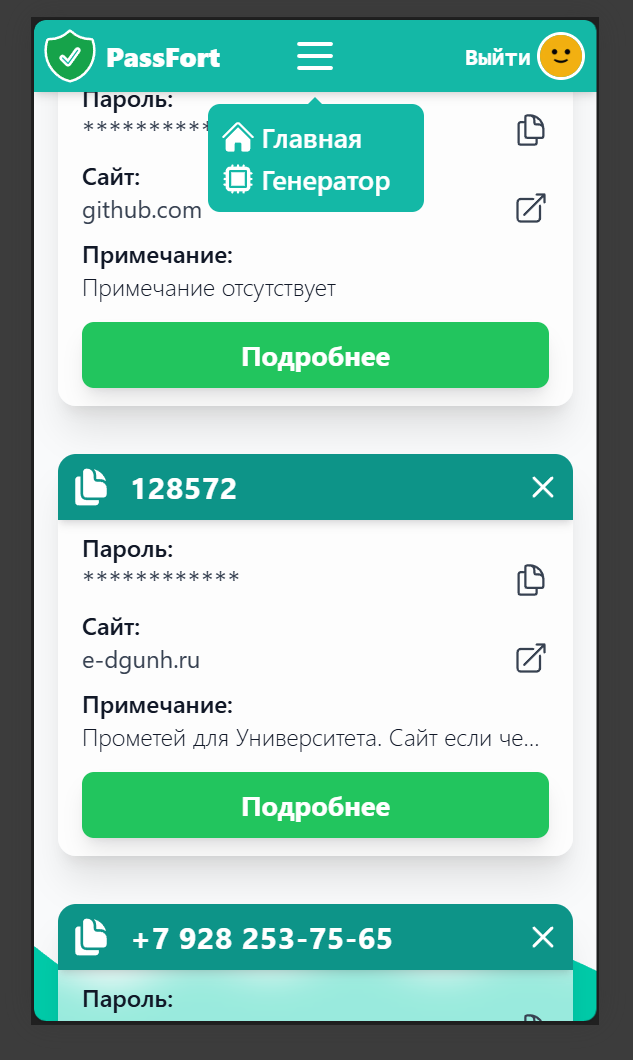
 Пример того, как должно отображаться адаптивное веб-приложение на мобильном устройстве:

Рисунок 3.19. Приложение на смартфоне.

# Заключение

Разработка веб-приложения для безопасного хранения паролей является актуальной и значимой задачей в условиях стремительного роста количества онлайн-сервисов и повышенных требований к безопасности пользовательских данных. В ходе работы были достигнуты следующие результаты:

1. Проведен анализ предметной области, включающий обзор существующих решений для хранения паролей и определение их преимуществ и недостатков. Были рассмотрены как локальные, так и облачные менеджеры паролей, а также встроенные менеджеры паролей, интегрированные в операционные системы или браузеры.
2. Определены основные требования к программным средствам хранения паролей, включающие шифрование данных, двухфакторную аутентификацию, управление правами доступа и другие критически важные аспекты безопасности.
3. Разработано веб-приложение, обеспечивающее безопасное хранение паролей и доступ к ним из любой точки мира. Приложение реализовано с использованием современных технологий и соответствует требованиям безопасности и удобства использования. Для шифрования данных применяется алгоритм AES-256, а для защиты доступа — мастер-пароль и двухфакторная аутентификация.
4. Проведены тестирования и отладка приложения, что позволило устранить выявленные недостатки и улучшить функциональность.

Практическая значимость данной работы заключается в создании удобного и безопасного инструмента для управления паролями, который позволит пользователям избежать проблем, связанных с утечкой данных и неудобством хранения большого количества паролей. Научная новизна работы состоит в том, что разработанное приложение имеет открытый исходный код, что позволяет любому желающему развернуть его на собственном сервере, использовать и при необходимости модифицировать в соответствии с индивидуальными потребностями. Это делает решение не только удобным, но и гибким, что особенно важно в условиях быстро меняющихся требований к безопасности и функциональности программного обеспечения.

Перспективы развития приложения включают:

* Реализацию функций загрузки и выгрузки базы паролей пользователями, что позволит им переносить свои пароли в другие менеджеры паролей.
* Добавление возможности изменения мастер-пароля и email пользователей, что повысит гибкость и удобство использования.
* Внедрение нескольких различных вариантов двухфакторной аутентификации (по SMS, по E-mail, по приложению аутентификации), что обеспечит пользователям дополнительный выбор и повысит уровень безопасности.
* Реализация вариации показа списка учетных записей в виде таблицы, что улучшит визуальное восприятие и удобство работы с приложением.
* Введение тёмной темы с возможностью переключения, что удовлетворит предпочтения пользователей и улучшит визуальный комфорт при использовании приложения в разных условиях освещения.

Потенциал дальнейшего развития приложения практически неограничен.

Таким образом, выполненная работа решает важную проблему обеспечения безопасности пользовательских данных и предлагает эффективное решение для управления паролями в современном цифровом мире, а также имеет значительный потенциал для дальнейших усовершенствований.

# Список литературы и источников

1. Аверьянов, Л. В., Анисимова, Н. А., Бурба, Д. Г. Информационные технологии в образовании: Учебное пособие. – СПб.: Питер, 2010. – 512 с.
2. Головин, С. Ю., Малышев, А. С. Программирование на Python: учебное пособие. – М.: Наука, 2015. – 304 с.
3. Иванов, И. И., Сидоров, П. П. Введение в криптографию: учебное пособие. – М.: Издательство МГУ, 2012. – 256 с.
4. Гейтс, Б. (2022). Пароли: будущее безопасности. Изучено в Интернете. (Дата обращения: 22.04.2024). URL: <https://security.blog/passwords>
5. Рябов, М.С. Программное обеспечение для хранения паролей: Обзор и анализ. Электронный журнал "Информационная безопасность". 2021. (Дата обращения: 02.05.2024). URL: <https://info-sec-journal.ru/articles/2021/06/password-managers-review>
6. Смирнов, И.И. Криптографические методы защиты информации. Онлайн-ресурс "Технологии безопасности". 2020. (Дата обращения: 09.05.2024). URL: <https://security-tech.ru/articles/crypto-methods>
7. Ломов, Е.П. (2021). Облачные технологии в хранении паролей. Онлайн-журнал "Веб-технологии". (Дата обращения: 14.05.2024). URL: <https://web-tech-journal.ru/cloud-password-storage>
8. Сорокин, А.В. Современные методы аутентификации пользователей. Электронный журнал "Технологии безопасности". 2022. (Дата обращения: 29.05.2024). URL: <https://tech-sec.ru/articles/authentication-methods>
9. Python Documentation. URL: <https://docs.python.org/3/index.html>
10. Django Documentation. URL: <https://docs.djangoproject.com/en/5.0/>
11. PyCryptodome Documentation. URL: <https://www.pycryptodome.org/>