

АЛГОРИТМЫ РЕАЛИЗАЦИИ АУТЕНТИФИКАЦИИ ПОЛЬЗОВАТЕЛЕЙ ДЛЯ ВЕБ-ПРИЛОЖЕНИЙ НА ОСНОВЕ БИБЛИОТЕКИ STREAMLIT

© 2025 Пономарёв Д.С.

Streamlit является сравнительно новой, быстроразвивающейся библиотекой, которая ориентирована на разработку и развертывание веб-приложений для научных исследований, машинного обучения и статистического анализа данных. Однако, на сегодняшний день для данной библиотеки отсутствуют готовые решения по аутентификации пользователей, что может явиться серьезным вопросом при разработке программных продуктов, поддерживающих приватность пользователей или ограниченность доступа пользователей. Поэтому, поставленной целью исследования явилась разработка алгоритмов и подходов для создания системы аутентификации пользователей с возможностью администрирования (с учетом возможностей интеграции в веб-приложения созданных на основе Streamlit). Основное внимание в работе было уделено обеспечению безопасности через хеширование паролей с использованием библиотеки Wscrypt и управлению пользовательскими данными через файловое хранилище в формате JSON. Рассмотрены варианты администрирования разработанной системы с возможностью обеспечения ограничений доступа для определенных пользователей. Представлены ключевые аспекты архитектуры системы, включая регистрацию пользователей, аутентификацию, управление данными, а также запись в файл JSON. В представленных примерах был использован функциональный подход: рассмотрена разработка как функций, так и отдельных составляющих разрабатываемой системы, их взаимодействие между собой, а также их алгоритмическая реализация. Рассмотрено применение общеизвестных паттернов проектирования и системного дизайна, обработка ошибок. Приведены примеры записи зашифрованных паролей в хранилище. Результаты представленной работы включают разработку архитектуры системы аутентификации, алгоритмов шифрования паролей, методов управления учетными записями и их интеграцию в веб-приложения. Практическая полезность представленной разработки заключается в возможности интеграции (без каких-либо значимых изменений в исходном коде) системы аутентификации пользователей в приложение или информационную систему, которые основаны на библиотеке Streamlit (в частности, это могут быть системы для проведения научных исследований, для работы с статистическими данными или создания и использования моделей машинного обучения).

Ключевые слова: системы контроля управления доступом, паттерны проектирования, алгоритмы шифрования, Python, Wscrypt, Streamlit.

Введение. На сегодняшний день использование в научных исследованиях таких библиотек Python как NumPy, Pandas, SciKit-Learn, SciPy, TensorFlow, Keras, PyTorch и др. становится все более и более популярным решением [1]. Одной из проблем в их применении является дальнейшее представление научных результатов для ученых-коллег и общественности, которые не знакомы с языками программирования (здесь стоит отметить, что практика развертывания научных проектов при помощи контейнеров Docker частично может решить данную проблему, однако не закрывает данный вопрос полностью [2]). Поэтому, в сфере разработки программного обеспечения в последние годы стали появляться библиотеки Python, которые позволяют создавать веб-приложения с ориентиром на интеграцию с научными исследованиями и библиотеками для статистической обработки данных и машинного обучения. Среди таких библиотек можно выделить Streamlit, Solara, Gradio. Актуальность применения Streamlit в научных исследованиях были рассмотрены в опубликованных ранее работах [3, 4]. Так как данная библиотека является сравнительно новой, то по многим вопросам ее применения отсутствуют готовые общепринятые шаблонные решения.

В частности, на сегодняшний день отсутствуют готовые решения по аутентификации пользователей. Поэтому, в данной работе представлено возможное решение по программной реализации системы аутентификации пользователей для веб-приложений на основе библиотеки Streamlit с использованием методов шифрования паролей.

Предложенная система включает такие функциональные элементы как загрузку данных пользователей, аутентификацию, регистрацию. Хранение паролей было реализовано при помощи безопасного хранения информации в формате JSON. (использование данного формата в подобных задачах является уже хорошо зарекомендовавшей себя практикой [5]). Шифрование (хеширование) паролей было рассмотрено на примере bcrypt, актуальность использования данной библиотеки было обосновано в ранее опубликованной работе [6].

В качестве примеров реализации в коде был использован Python. Рассмотрена реализация систем аутентификации для веб-приложений, которые разработаны на основе библиотеки Streamlit. Представленные методы, алгоритмы и подходы прежде всего ориентированы на веб-приложения и информационные системы (далее – ИС), для которых рассматривается доступ ограниченного количества пользователей (который в свою очередь согласован с руководством научного подразделения, предприятия, организации или компании, для которых они разработаны).

Разработка подходов для реализации. Система аутентификации должна быть загружена для пользователя в первую очередь перед непосредственно запуском сервисов ИС. Рассмотрим разработку на примере систем аутентификации «Логин-Пароль». Для начала представим простой пример, который, однако, не следует применять на практике:

```
def authenticate(username, password):
    users = {
        "user_1": "password_1",
        "user_1": "password_1",
        "admin": "password_3"
    }
    if username in users and users[username] == password:
        return True
    return False
```

– в данном примере логины и пароли хранятся непосредственно в самом коде ИС и на сегодняшний день это является плохим решением с позиций информационной безопасности. Для безопасной аутентификации логины и пароли не должны храниться в коде приложения. Вместо этого они могут быть сохранены во внешней базе данных или файле с применением хеширования паролей [7].

Рассмотрим, как можно реализовать аутентификацию с использованием файлового хранилища для логинов и паролей, где пароли будут храниться в зашифрованном виде.

Для поставленной задачи следует определить следующие шаги:

- организация хранения хешированных паролей в отдельном файле (например, JSON, также может быть использована и база данных (далее – БД));
- создание аутентификации через сравнение введенного пароля с хешем пароля;
- следует также принять во внимание возможность расширения созданного хранилища паролей.

Ментальная карта (Mind Map) разработки СКУД представлена на рисунке 1 далее.

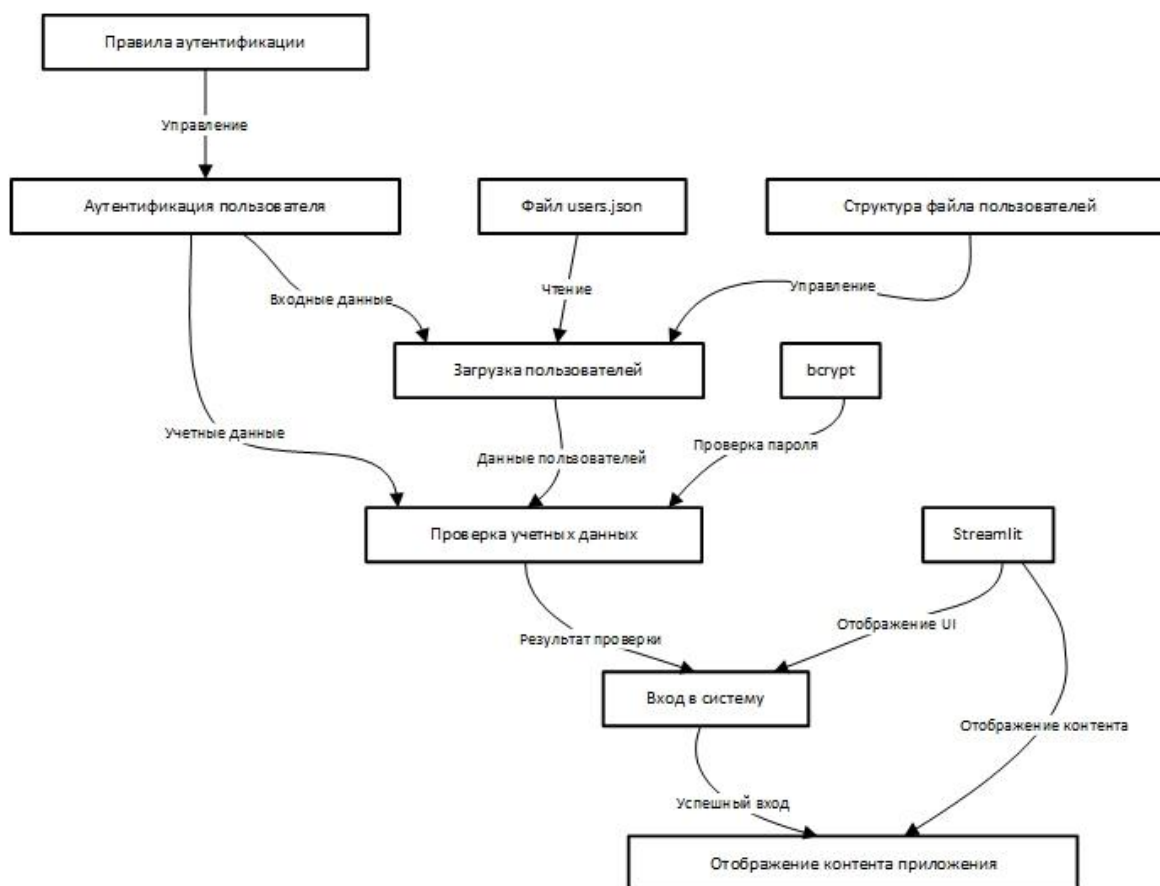


Рис. 1. Ментальная карта (Mind Map) разработки

Рассмотрим разработку системы контроля управления доступом (далее – СКУД) для ИС, которые построены на основе библиотеки Streamlit, для хеширования паролей будет использована библиотека Bcrypt.

В первую очередь, для реализации системы аутентификации потребуется импортировать библиотеки Streamlit, Bcrypt, а также библиотеку для работы с JSON-файлами (применение данных библиотек будет рассмотрено далее более подробно):

```
import streamlit
import bcrypt
import json
```

Рассмотрим шифрование (хеширование) паролей при помощи библиотеки bcrypt. Основное преимущество bcrypt заключается в том, что она использует медленный хеш (время выполнения увеличивается для предотвращения атак методом полного перебора) [8].

Процесс хеширования. Когда пользователь регистрируется, пароль преобразуется в хеш с помощью метода `bcrypt.hashpw()`. Этот метод генерирует случайные данные и объединяет их с паролем для создания стойкого к атакам хеша. Хешированный пароль сохраняется в файл (или базу данных) вместо реального пароля.

Пример хеширования: `hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')`. Здесь: `bcrypt.gensalt()` – генерирует уникальные случайные данные (в некоторой литературе для этого используется термин «Соль»); `Bcrypt.hashpw()` – создает хеш, комбинируя «Соль» и пароль; `Decode('utf-8')` – переводит байтовый объект в строку для удобства хранения (выбран тип Unicode-стандарта utf-8).

Проверка пароля. Когда пользователь вводит пароль для аутентификации, то его пароль преобразуется в байты и сравнивается с хешем, хранящимся в файле: `bcrypt.checkpw(password.encode('utf-8'), stored_password_hash.encode('utf-8'))`. Этот метод берет ввод пользователя, хеширует его и проверяет, совпадает ли результат с ранее сохранённым хешем. Таким образом, система никогда не хранит реальные пароли, а использует только безопасные хеши.

Использованные паттерны проектирования. Паттерн «Стратегия» позволит отделить конкретную логику (аутентификацию) от основной логики приложения [9]. Данный паттерн может быть реализован в упрощённом виде. Логика аутентификации может быть вынесена в отдельную функцию (например, назовем ее `authenticate()`) и рассмотрим процесс ее создание далее более подробно). Выделение функциональности от основной бизнес-логики позволяет легко изменять или расширять процесс аутентификации без изменения основной логики приложения.

Если потребуется заменить способ аутентификации, например, на «*Oauth*» или «*JWT*», это можно сделать, не трогая основной код приложения.

Разработанный алгоритм для функции `authenticate()` представлен далее на рисунке 2.

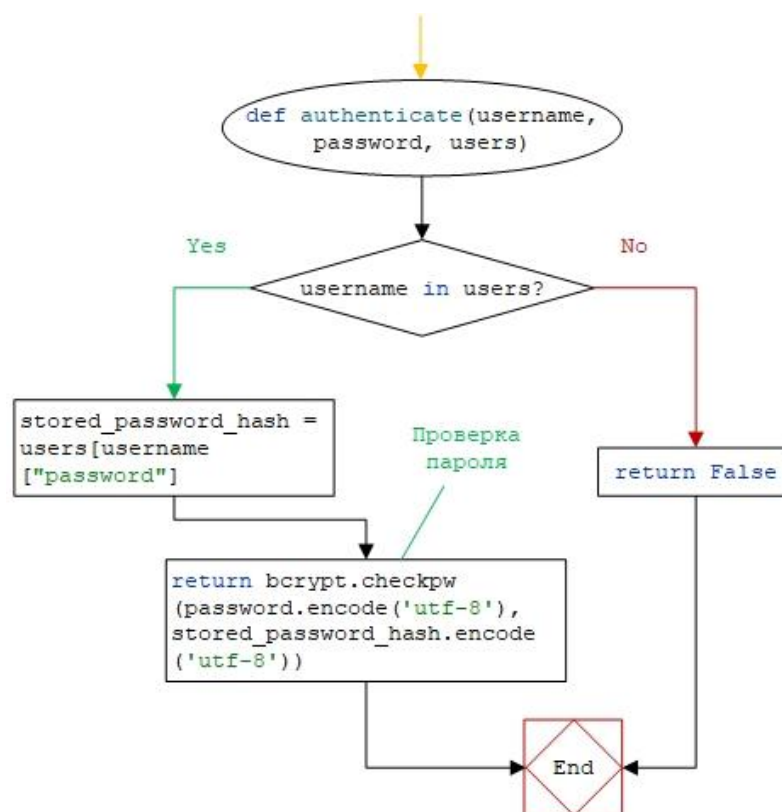


Рис. 2. Реализация функции для аутентификации

Преимуществом такого подхода будет то, что функция «*authenticate*» может легко заменить свою логику на другие методы (например, базу данных), сохранив интерфейс аутентификации.

Применение принципа «Разделение ответственности». Практическое применение принципа «Разделение ответственности» было использовано при разделении функциональности системы на независимые компоненты [10, 11]. Логика приложения может быть разделена на отдельные функции.

«*main()*» – отвечает за отображение интерфейса и взаимодействие с пользователем. Алгоритм для данной функции был реализован следующим образом (рисунок).

«*authenticate()*» – уже была рассмотрена ранее, отвечает за проверку учетных данных.

«*register_user()*» – обрабатывает регистрацию новых пользователей.

«*load_users()*» – отвечает за загрузку пользователей из файла.

Благодаря данному подходу будет упрощена поддержка кода, он станет более систематизированным и разделенным на отдельные составляющие. Например, если нужно будет изменить способ хранения пользователей (например, вместо JSON использовать БД), это можно будет сделать, не изменяя другие частей разработанной СКУД.

Далее рассмотрим более подробно работу функций с файловой системой.

Алгоритм работы с файловой системой – загрузка и сохранение данных в файл. При разработке представленного алгоритма были рассмотрены результаты исследования, представленные в работе [8], а именно: результаты анализа по использованию алгоритма *bcrypt* при хешировании, добавление «соли» при шифровании и подходы для хранения хешированных паролей. Для данного алгоритма была разработана функция *load_users()*. Загрузка данных пользователей из JSON-файла в функции *load_users()* может быть реализована при помощи ключевого слова «*with*» и реализована как:

```
with open (file_path, "r") as file:  
    users = json.load(file)
```

в представленном фрагменте используется *json.load()* для чтения содержимого файла и преобразования его в словарь. Так же была разработана функция *register_user()*, которая позволяет Администратору добавлять новых пользователей и сохранять их в файл. Также данная функция проводит проверку записей в файле JSON для исключения повторяющихся записей. Чтение файла JSON реализовано с помощью стандартных средств Python:

```
with open(file_path, "w") as file:  
    json.dump(users, file)
```

использование *json.dump()* записывает словарь пользователей обратно в JSON-файл.

После загрузки списка пользователей функция проверяет, есть ли уже пользователь с введенным логином в загруженных данных. Если пользователь с таким логином уже существует, программа выдает предупреждение с помощью *st.warning()*, и регистрация прекращается.

Это реализовано при помощи проверки на наличие ключа в словаре. Словарь *users* содержит логины пользователей как ключи, а пароли – как значения. Проверка осуществляется следующим образом:

```
if username in users:  
    st.warning(«Имя пользователя уже существует!»)
```

Данный подход помогает предотвратить создание дубликатов пользователей с одинаковыми именами. Полная версия алгоритма функции *register_user()* представлена далее на рисунке 3. В данной функции применен упомянутый ранее подход для хеширования паролей, а также возможность добавления нового пользователя в словарь. Преимуществами данного алгоритма будут обеспечение простоты хранения данных и независимость от сложных систем управления базами данных.

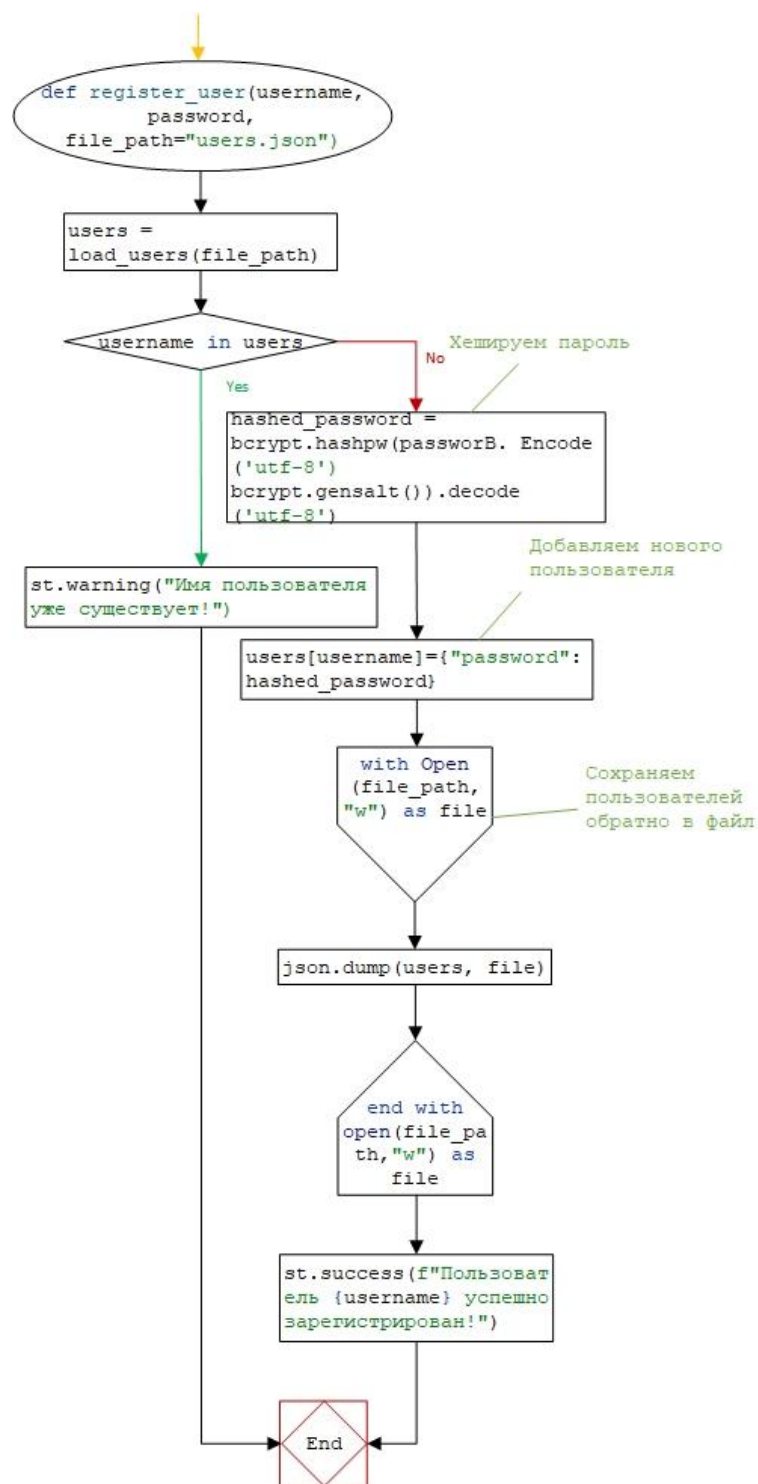


Рис. 3. Алгоритм для реализации функции регистрации новых пользователей

Разработка форм ввода и пользовательского интерфейса. Разработка модуля авторизации пользователей на основе личных данных в веб приложении на Streamlit, была упомянута в работе [12], однако, его создание в данном исследовании не было раскрыто. В представленном разделе будет рассмотрено создание визуализации пользовательского интерфейса для ввода логина и пароля, а также их привязка к внутренней архитектуре рассмотренной разработки и алгоритмам шифрования.

Интерфейс ввода логина и пароля представлен далее (для удобства он был размещен в боковой панели («*sidebar*»)):

```
username = st.sidebar.text_input(«Имя пользователя»)
password = st.sidebar.text_input(“Пароль”, type='password')
```

– здесь используются текстовые поля и функция *Streamlit* для создания безопасного поля ввода пароля с *type='password'*.

Кнопка «Войти» проверяет введенные данные и вызывает функцию аутентификации:

```
if st.sidebar.button(“Войти”):
    if authenticate(username, password, users):
        st.sidebar.success(f”Добро пожаловать, {username}!”)
```

Streamlit API также упрощает организацию интерфейса через использование блоков с основным содержимым, разделяя интерфейс на отдельные логические компоненты (панель авторизации и основная часть приложения) (рисунок 4).

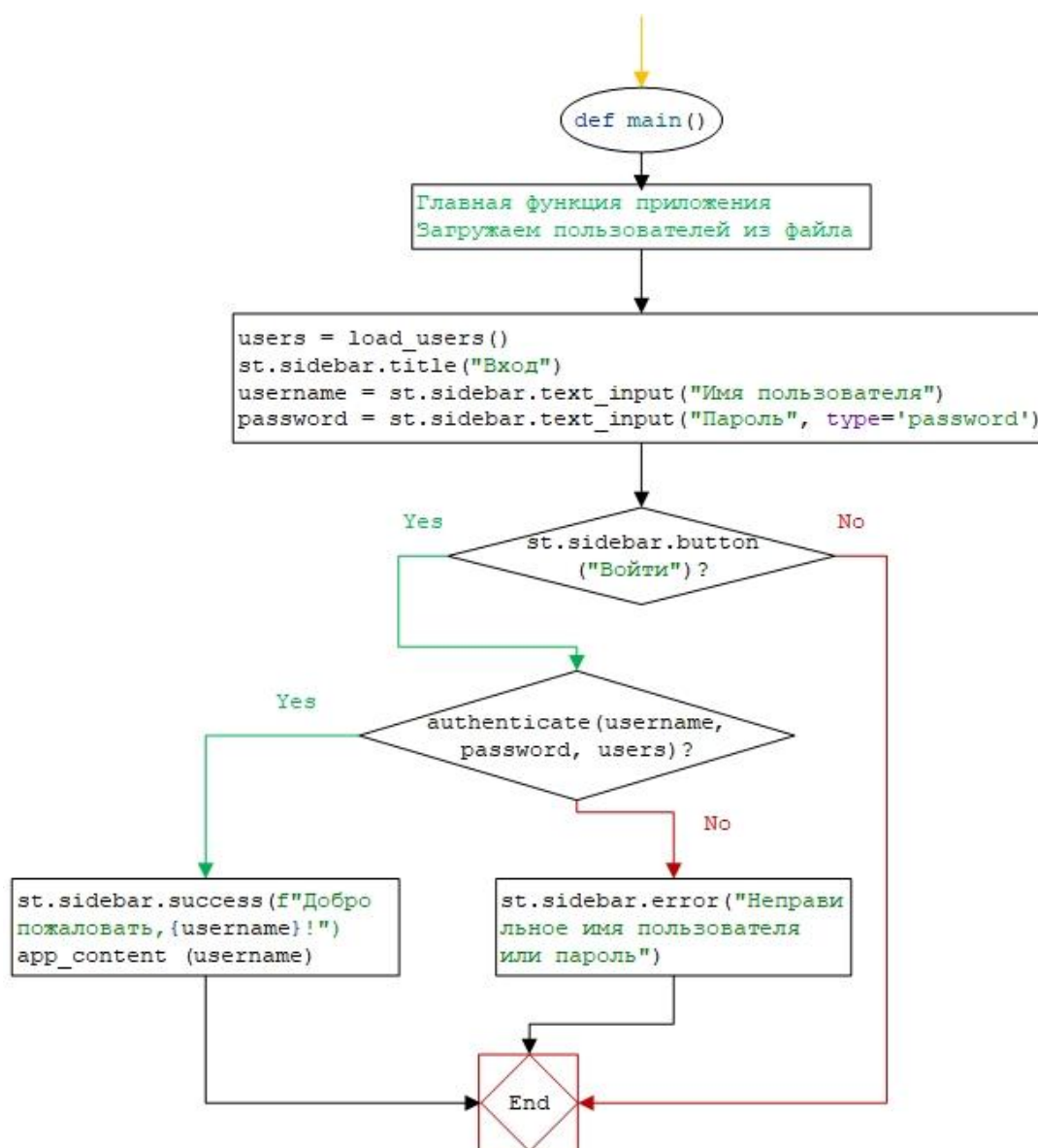


Рис. 4. Реализации функции для создания пользовательского интерфейса

Обработка исключений и ошибок. Для защиты от ошибок, например, если файл с пользователями не найден, используется обработка исключений. Это важная часть защиты программы от непредвиденных ситуаций. В функции *load_users()* это было реализовано при помощи связки из ключевых слов «try-except»:

```
try:
    with open(file_path, "r") as file:
        users = json.load(file)
except FileNotFoundError:
    users = {}
```

В случае отсутствия файла с пользователями – будет создаваться пустой словарь, предотвращая сбой приложения. В итоге алгоритм функции *load_users()* выглядит следующим образом (рисунок 5).

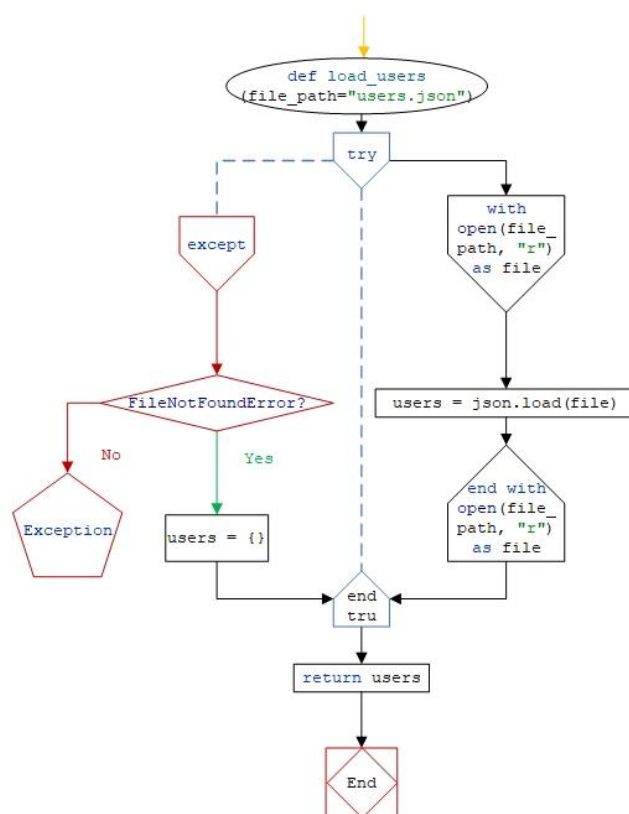


Рис. 5. Алгоритм для функции *load_users()*

Работа кода будет складываться из следующих шагов.

1. При запуске приложения через Streamlit отображается боковая панель для ввода.
2. Пользователь вводит логин и пароль, и при нажатии кнопки «Войти» вызывается функция *authenticate()*, которая проверяет данные через хеширование с помощью библиотеки *bcrypt*.
3. Если введенные данные верны, пользователь получает доступ к основному содержимому приложения.
4. Вся информация о пользователях хранится в файле *users.json* в зашифрованном виде.
5. Регистрация новых пользователей выполняется через отдельную функцию *register_user()*, которая хеширует пароль и добавляет нового пользователя в файл.

Так как запуск ИС происходит через файл домашней страницы («*Home.py*») то наиболее подходящим решением может стать размещение разработанного скрипта в данном файле.

Асинхронность для представленной разработки в данном случае не рассматривалась, а вставку разработанного скрипта, который реализует систему аутентификации пользователей, оптимальнее всего добавить сразу после импортирования всех нужных библиотек. Таким образом, система аутентификации будет запущена в первую очередь при загрузке домашней страницы.

Система аутентификации реализована таким образом, что при первом входе через домашнюю страницу (*Home.py*), учетная запись будет сохранена на протяжении всего времени работы с ИС (либо до очистки кэша) т.е. при переходе на *Home.py* с других вкладок после прохождения аутентификации повторного запроса логина и пароля не будет.

Администрирование и разработка UML-диаграммы последовательностей СКУД. Отдельным вопросом следует рассмотреть добавление новых пользователей в систему аутентификации. Организация системы аутентификации с позиций пользователя и администратора может быть организована следующим образом: Администратор ИС создает уникальные логин и пароль, которые передает Пользователю и добавляет в скрипт домашней страницы. После чего, происходит хеширование данных и их запись в файл JSON.

Добавленные логин и пароль в код домашней страницы в последующем могут быть удалены: это никак не повлияет на последующий вход в ИС пользователей (предварительно, перед удалением, данные могут быть дополнительно сохранены т.е. может быть сделана резервная копия). Удаление учетной записи может производиться путем нахождения и удаления логина и хешированного пароля непосредственно в самом файле JSON (помимо файла JSON может быть использована и БД). Пример заполнения файла JSON данными логина и хешированного пароля представлено далее на рисунке 6 (на примере логинов «*user1*», «*admin*»).

```
{
  "user1": {
    "password": "$2b$12$eixZ0NUdpXRZ7bXht.5xM.cXZl0c5vQrfr6nLTVcKfhxu8QMeS5C."
  },
  "admin": {
    "password": "$2b$12$6uTvrS9erQ03j.MKNF70EePIay9P6kL4YBe5SzC/.x0qw7x0dKXaK"
  }
}
```

Рис. 6. Пример записи хешированных паролей в файл их хранения JSON

Рассмотрим более подробно непосредственно способы регистрации новых пользователей. Здесь можно выделить два способа.

Способ 1: использование функции «*register_user*» напрямую т.е. можно вручную зарегистрировать пользователя, вызвав функцию «*register_user*» в отдельной части программы или интерактивно в Python-скрипте. Например, если нужно добавить пользователя с логином «*user1*» и паролем «*password1*», можно временно добавить блок кода:

```
if st.sidebar.button(«Зарегистрировать нового пользователя»):
    register_user("user1", "password1")
```

далее запустить домашнюю страницу и нажать кнопку регистрации в боковой панели. После этого пользователь «*user1*» с паролем «*password1*» будет зарегистрирован, и его данные сохранятся в файле *users.json*.

Способ 2: вызов функции вручную. Можно временно вызвать функцию «*register_user*» вне основного кода программы. Для этого, после блока «*if __name__ == '__main__':*» перед «*main()*» следует вызвать данную функцию с нужными логином и паролем, например: «*register_user('user1', 'password1')*». Далее, после записи данных в файл JSON, представленная строка с логином и паролем может быть удалена.

Рассмотрим представление полученных результатов при помощи UML-диаграммы последовательностей. Актуальность и особенности использования UML моделей для исследования и обеспечения информационной безопасности была обоснована в опубликованных ранее работах отечественных и зарубежных авторов [13]. UML-диаграмма последовательностей для проводимого в данной статье исследования представлено на рисунке 7 далее.

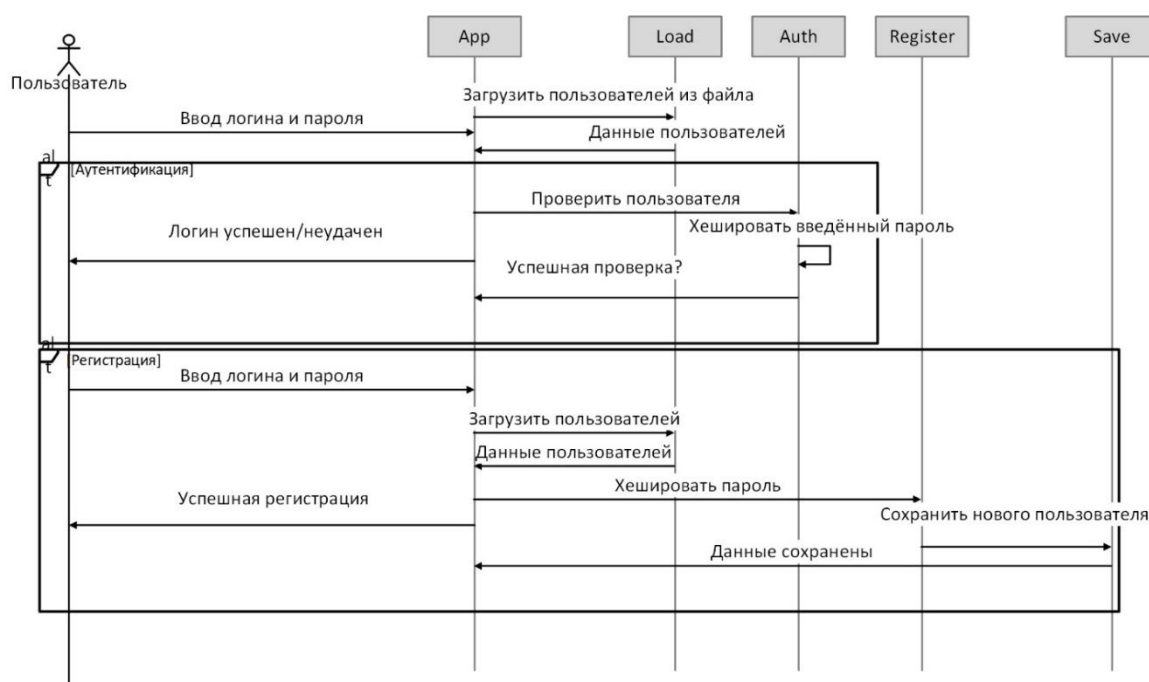


Рис. 7. UML-диаграмма последовательностей для разработанной СКУД

Разберем представленную UML-диаграмму последовательностей более подробно. Актор: Пользователь – взаимодействует с приложением библиотеки *Streamlit*. Далее организован основной поток для аутентификации, который проявляется в следующих шагах: пользователь вводит логин и пароль в приложение; приложение загружает список пользователей из файла через компонент *Load*; компонент аутентификации (*Auth*) проверяет наличие пользователя и хеширует введенный пароль, происходит сверка паролей. И если проверка успешна, пользователь авторизован, иначе – отказ.

Альтернативный поток для регистрации следующий: пользователь вводит новый логин и пароль; далее приложение снова загружает список пользователей через компонент *Load*; новый пароль хешируется через компонент *Register*; после чего новый пользователь сохраняется в файл через компонент *Save*; пользователю сообщается об успешной регистрации. Особенности данной UML-диаграммы следующие. Первое – диаграмма содержит альтернативные блоки (*alt*), которые показывают два различных потока – для аутентификации и регистрации; второе – взаимодействие между компонентами приложений *Streamlit* и вспомогательными функциями, такими как загрузка и сохранение данных.

Таким образом, благодаря разработанной UML-диаграмме последовательностей было рассмотрено графическое проектирование представленных примеров для разработки СКУД.

Заключение. В работе представлен подход к разработке и реализации системы аутентификации пользователей, ориентированной на использование в веб-приложении. Проанализированы ключевые аспекты функциональности, включая регистрацию новых пользователей, аутентификацию существующих, безопасное хеширование паролей, а также управление пользовательскими данными с использованием файлового хранилища в формате JSON.

В работе были рассмотрены: особенности шифрования паролей и их программная реализация; применение функционального подхода и практическая реализация паттернов проектирования «Стратегия», «Разделение ответственности»; примеры работы с файловой системой (для загрузки и сохранения данных в файл); разработка пользовательских интерфейсов и форм ввода; обработка исключений и ошибок; администрирование системы. Полученные результаты были отражены при помощи – UML-диаграммы, ментальной карты (Mind-Map), фрагментов и примеров кода, примера хранения зашифрованных паролей в файле. Представлены примеры использования библиотек `Wcrypt` (для шифрования паролей), `Streamlit` (создание интерфейсов, интеграция в web-приложение), `JSON` (для записей в файл хранения).

Практическая значимость представленной разработки заключается в возможности без каких-либо значимых изменений в исходном коде интегрировать систему аутентификации пользователей с функцией администрирования в веб-приложения и информационные системы, созданные на основе библиотеки `Streamlit` (включая веб-приложения для научных исследований, обработки статистических данных, а также разработки и использования моделей машинного обучения). В качестве продолжения работы могут быть рассмотрены системы многофакторной аутентификации для повышения уровня безопасности.

СПИСОК ИСТОЧНИКОВ

1. Gholizadeh, S. Top Popular Python Libraries in Research / S. Gholizadeh // *Journal of Robotics and Automation Research*. – 2022. – Vol. 3, No. 2. – P. 142-145. – DOI 10.33140/jrar.03.02.02. – EDN GLBISU.
2. Chiang, R. C. Contention-aware container placement strategy for docker swarm with machine learning based clustering algorithms / R. C. Chiang // *Cluster Computing*. – 2023. – Vol. 26, No. 1. – P. 13-23. – DOI 10.1007/s10586-020-03210-2. – EDN SWQZXV.
3. Taufik A. I. Web Application Based on MachineLearning for Diabetes Detection usingMicrostrip Resonator and Streamlit / A. I. Taufik, Y. Rahayu, A. Setiawan // *Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics*. – 2024. – Vol. 6, No. 3. – P. 120-131. – DOI 10.35882/ijeeemi.v6.i3.2. – EDN TPLKYH.
4. Akkem, Ya. Streamlit Application for Advanced Ensemble Learning Methods in Crop Recommendation Systems – A Review and Implementation / Ya. Akkem, B. S. Kumar, A. Varanasi // *Indian Journal of Science and Technology*. – 2023. – Vol. 16, No. 48. – P. 4688-4702. – DOI 10.17485/ijst/v16i48.2850. – EDN QFZSPO.
5. Нурматов, М. Ш. Обеспечение безопасности веб-приложений на основе использования JWT в Lumen с использованием Swagger / М. Ш. Нурматов, А. Н. Скоба // *Перспективные научные исследования: опыт, проблемы и перспективы развития : Сборник научных статей по материалам X Международной научно-практической конференции, Уфа, 04 апреля 2023 года. Том Часть 3. – Уфа: Общество с ограниченной ответственностью "Научно-издательский центр "Вестник науки", 2023. – С. 196-206. – EDN RSALAH.*
6. Свиноухов, Д. Д. Сравнение существующих алгоритмов хеширования / Д. Д. Свиноухов, А. В. Абрамов, Э. Х. Милушев // *Наукосфера*. – 2024. – № 5-1. – С. 71-74. – DOI 10.5281/zenodo.11185942. – EDN BDZLJS.

7. Лейн, Ф. Е. Исследование методов хранения пользовательских паролей и их синхронизации на мобильных устройствах / Ф. Е. Лейн // Вестник науки. – 2024. – Т. 1, № 5(74). – С. 512-522. – EDN ISTUTJ.
8. Batubara, T. P. Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force / T. P. Batubara, S. Efendi, E. B. Nababan // Journal of Physics: Conference Series. – 2021. – Vol. 1811, No. 1. – P. 012129. – DOI 10.1088/1742-6596/1811/1/012129. – EDN TTSZOX.
9. Шмаков, С. Э. Взаимодействие паттернов проектирования для эффективной web-разработки / С. Э. Шмаков, Е. В. Щенникова, А. Е. Определенцева // Ученые записки УлГУ. Серия: Математика и информационные технологии. – 2021. – № 2. – С. 90-96. – EDN YQJJOH.
10. Федорова, О. Single Responsibility Principle / О. Федорова // Системный администратор. – 2024. – № 1-2(254-255). – С. 62-63. – EDN ZHTWLW.
11. Дехтиевский, С. А. Принципы SOLID и их применение в разработке программного обеспечения / С. А. Дехтиевский, Г. С. Бударный // Научный аспект. – 2024. – Т. 43, № 4. – С. 5635-5641. – EDN RBCDIR.
12. Радковский, С. А. Использование фреймворка Streamlit для создания виртуальных лабораторных работ в виде веб-приложений / С. А. Радковский, Г. В. Коваленко // Сборник научных трудов Донецкого института железнодорожного транспорта. – 2024. – № 4(75). – С. 61-72. – EDN TDRGYB.
13. Meziane, N. A Study of Modelling IoT Security Systems with Unified Modelling Language (UML) / N. Meziane, N. Ouerdi // International Journal of Advanced Computer Science and Applications. – 2022. – Vol. 13, No. 11. – DOI 10.14569/ijacsa.2022.0131130. – EDN SMPDJQ.

Поступила в редакцию 19.09.2025 г., рекомендована к печати 08.10.2025 г.

ALGORITHMS FOR IMPLEMENTING USER AUTHENTICATION FOR WEB APPLICATIONS BASED ON THE STREAMLIT LIBRARY

Ponomarev D.S.

Streamlit is a relatively new, rapidly developing library that focuses on developing and deploying web applications for scientific research, machine learning, and statistical data analysis. However, to date, there are no ready-made solutions for user authentication for this library, which may be a serious issue when developing software products that support user privacy or limited user access. Therefore, the goal of the study was to develop algorithms and approaches for creating a user authentication system with the ability to administer (taking into account the possibilities of integration into web applications created on the basis of Streamlit). The main focus of the work was on ensuring security through password hashing using the Bcrypt library and managing user data through file storage in JSON format. Options for administering the developed system with the ability to provide access restrictions for certain users are considered. Key aspects of the system architecture are presented, including user registration, authentication, data management, and writing to a JSON file. The presented examples use a functional approach: the development of both functions and individual components of the developed system, their interaction with each other, as well as their algorithmic implementation are considered. The application of well-known design patterns and system design, error handling are considered. Examples of recording encrypted passwords in a storage are given. The results of the presented work include the development of the authentication system architecture, password encryption algorithms, account management methods and their integration into web applications. The practical usefulness of the presented development lies in the ability to integrate (without any significant changes in the source code) the user authentication system into an application or information system based on the Streamlit library (in particular, these can be systems for conducting scientific research, working with statistical data or creating and using machine learning models).

Keywords: access control systems, design patterns, encryption algorithms, Python, Bcrypt, Streamlit.

Пономарёв Дмитрий Сергеевич

кандидат технических наук, ведущий научный сотрудник филиала (г. Ижевск) ФКУ «Научно-исследовательский институт Федеральной службы исполнения наказаний», доцент кафедры водоснабжения и водоподготовки ФГБОУ ВО «Ижевский государственный технический университет имени М.Т. Калашникова», Российская Федерация, Удмуртская Республика, г. Ижевск.
E-mail: ponomarev.dmitry1990@mail.ru

Ponomarev Dmitrii Sergeevich

Candidate of Technical Sciences, Leading Researcher of the Branch (Izhevsk) of the Federal Penitentiary Service of Russia, Associate Professor of Izhevsk State Technical University, Russian Federation, Udmurt Republic, Izhevsk.