УДК 004.423

АУТЕНТИФИКАЦИЯ И АВТОРИЗАЦИЯ В ВЕБ-ПРИЛОЖЕНИЯХ НА FLASK В PYTHON

Халевин Т.А.

студент направления подготовки информатики и вычислительной техники,

Хакасский государственный университет имени Н.Ф. Катанова,

г. Абакан, Россия [1]

Аннотация: В настоящем документе рассматривается процесс реализации систем аутентификации и авторизации в веб-приложениях на основе фреймворка Flask. Описаны основные концепции и принципы работы с расширениями Flask-Login и Flask-Security, которые обеспечивают удобные механизмы для управления пользователями и сессиями. Представлены примеры кода с подробными комментариями, позволяющие разработчикам глубже понять механизмы работы с аутентификацией и авторизацией в своих приложениях.

Ключевые слова: Python, Flask, аутентификация, авторизация, Flask-Login, Flask-Security, веб-приложение, управление пользователями, сессии.

AUTHENTICATION AND AUTHORISATION IN FLASK WEB APPLICATIONS IN PYTHON

Khalevin T.A.

student of computer science and computer engineering department,

N.F. Katanov Khakass State University,

Abakan, Russia

[1] Научный руководитель: Голубничий А.А. старший преподаватель кафедры ПОВТиАС, Хакасский государственный университет имени Н.Ф. Катанова, г. Абакан, Россия

Abstract: This document discusses the process of implementing authentication and authorisation systems in web applications based on the Flask framework. It describes the basic concepts and principles of Flask-Login and Flask-Security extensions, which provide convenient mechanisms for user and session management. Code samples with detailed comments are presented, allowing developers to gain a deeper understanding of authentication and authorisation mechanisms in their applications.
Keywords: Python, Flask, authentication, authorisation, Flask-Login, Flask-Security, web application, user management, sessions.

В данной статье мы рассмотрим фреймворк Flask и его функциональность. Flask является одним из самых популярных фреймворков для создания веб-приложений на языке Python. Он обладает множеством инструментов для создания высококачественных веб-приложений и может быть использован как для небольших проектов, так и для крупномасштабных систем [1].

Flask – это легковесный фреймворк для разработки веб-приложений на языке программирования Python. Он обеспечивает минимальный набор инструментов и библиотек, необходимых для создания веб-приложений, но при этом обладает достаточной гибкостью для расширения и адаптации под конкретные потребности разработчика [2].

Flask является одним из самых популярных фреймворков для разработки веб-приложений на Python, благодаря своей гибкости, простоте и легковесности [3].

Разработка безопасных веб-приложений требует реализации надежных систем аутентификации и авторизации. Flask, будучи одним из самых популярных фреймворков для создания веб-приложений на Python, предлагает разработчикам гибкие инструменты для этих целей. В частности, расширения Flask-Login и Flask-Security позволяют легко и эффективно управлять пользователями и сессиями, обеспечивая безопасность приложений.

В данной статье не будет указано, как установить Flask, подразумевается, что он установлен на вашем компьютере. Также вам необходимо установить модуль Flask_Login, если он не установлен. Для этого введите команду pip install flask-login.

Аутентификация – это процесс верификации идентификатора пользователя, обычно осуществляемый путем проверки имени пользователя и пароля. В контексте веб-приложений на Flask, это обычно включает в себя создание формы для входа и обработку отправленных данных для проверки соответствия в базе данных.

Создадим простейший пример аутентификации пользователя. Для этого необходима простая форма входа. В корневой директории создайте папку templates и в ней создайте файл login.html. Пример данного файла представлен на рисунке 1.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
    <form method="post">
        Username: <input type="text" name="username"><br>
        Password: <input type="password" name="password"><br>
        <input type="submit" value="Login">
    </form>
</body>
</html>
```

Рисунок 1 – Пример формы для входа [разработано автором]

Необходимо также написать код для аутентификации пользователя. В корневой директории создайте файл application.py. Код будет разбит на небольшие части. Начнем с импортов необходимых библиотек. Импорты представлены на рисунке 2.

```
from flask import Flask, request, redirect, render_template
from flask_login import LoginManager, UserMixin, login_user, login_required
```

Рисунок 2 – Импорты необходимые для реализации простейшей аутентификации [разработано автором]

Опишем каждый импорт отдельно:

- Flask – основной класс фреймворка Flask, используемый для создания экземпляра веб-приложения.
- request – глобальный объект запроса в Flask, содержащий данные запроса от клиента к серверу.
- redirect – функция для перенаправления пользователя на другой маршрут в приложении.
- render_template – функция для рендеринга HTML-шаблонов на основе Jinja2 с передачей контекста шаблона.
- LoginManager – класс, управляющий процессом аутентификации пользователей в Flask-Login.
- UserMixin – класс-примесь, предоставляющий реализацию базовых методов для класса пользователя.
- login_user – функция, регистрирующая пользователя как вошедшего в систему.
- login_required – декоратор, ограничивающий доступ к определенным маршрутам только для аутентифицированных пользователей.

Перейдем к базовой настройке приложения. Она представлена на рисунке 3.

```
app = Flask(__name__)
app.config['SECRET_KEY'] = 'секретный ключ'
login_manager = LoginManager(app)
```

Рисунок 3 – Базовая настройка приложения. [разработано автором]

Создается экземпляр Flask-приложения и задается секретный ключ, который используется для подписи сессий и других необходимых элементов

безопасности. Затем инициализируется LoginManager с созданным приложением.

Определение пользователя представлено на рисунке 4.

```
# Создаем "базу данных" пользователя в памяти
users = {'admin': {'password': 'secret'}}

class User(UserMixin):
    def __init__(self, id):
        self.id = id
```

Рисунок 4 – Определение пользователя [разработано автором]

Создается простая "база данных" пользователей в виде словаря для демонстрации. Класс User наследуется от UserMixin, что обеспечивает базовую реализацию методов для работы с пользователем. Конструктор класса принимает id, который является уникальным идентификатором пользователя. Важно отметить, что пользователь создается прямо в коде для большей наглядности. В реальном проекте так делать нельзя, потому как это связано с риском утечки. В реальном проекте используйте базу данных [4].

Рассмотрим метод загрузки пользователя на рисунке 5.

```
@login_manager.user_loader
def load_user(user_id):
    if user_id in users:
        return User(user_id)
    return None
```

Рисунок 5 – Метод загрузки пользователя [разработано автором]

Декоратор @login_manager.user_loader используется для определения функции загрузки пользователя, которая требуется Flask-Login для управления пользовательской сессией. Функция возвращает экземпляр пользователя по его идентификатору.

Для логина пользователя создадим маршрут для входа в систему, представленный на рисунке 6.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users and users[username]['password'] == password:
            user = User(username)
            login_user(user)
            return redirect('/secure-page')
        else:
            return 'Invalid username or password'
    return render_template('login.html')
```

Рисунок 6 – Маршрут для входа в систему [разработано автором]

Маршрут /login обрабатывает страницу входа. Если метод запроса POST (пользователь отправил форму), извлекаются имя пользователя и пароль. Если данные верны, пользователь аутентифицируется, и происходит перенаправление на защищенную страницу. В противном случае выводится сообщение об ошибке. Для GET-запросов отображается форма входа.

Так же для проверки работоспособности кода будет создан защищенный маршрут, к которому могут иметь доступ только пользователи, вошедшие в систему. На рисунке 7 представлен код для создания такого простого защищенного маршрута.

```
@app.route('/secure-page')
@login_required
def secure_page():
    return 'Welcome to the secure page!'
```

Рисунок 7 – Защищенный маршрут [разработано автором]

И последнее что необходимо сделать, это запустить приложение. Код необходимый для запуска представлен на рисунке 8.

```
if __name__ == '__main__':
    app.run(debug=True)
```

Рисунок 8 – Запуск проекта [разработано автором]

Блок кода выше запускает веб-сервер для разработки с включенным режимом отладки, когда скрипт запускается напрямую.

Теперь соберем все эти части воедино. Полный код приложения представлен на рисунке 9.

```
from flask import Flask, request, redirect, render_template
from flask_login import LoginManager, UserMixin, login_user, login_required

app = Flask(__name__)
app.config['SECRET_KEY'] = 'секретный ключ'
login_manager = LoginManager(app)

# Создаем "базу данных" пользователя в памяти
users = {'admin': {'password': 'secret'}}

class User(UserMixin):
    def __init__(self, id):
        self.id = id

@login_manager.user_loader
def load_user(user_id):
    if user_id in users:
        return User(user_id)
    return None

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users and users[username]['password'] == password:
            user = User(username)
            login_user(user)
            return redirect('/secure-page')
        else:
            return 'Invalid username or password'
    return render_template('login.html')

@app.route('/secure-page')
@login_required
def secure_page():
    return 'Welcome to the secure page!'

if __name__ == '__main__':
    app.run(debug=True)
```

Рисунок 9 – Полный код приложения [разработано автором]

Запустим приложение. При попытке перейти по адресу http://127.0.0.1:5000/secure-page без предварительного входа в систему, нам выдаст предупреждение о том, что мы не авторизованы. Но если мы войдем в систему по адресу http://127.0.0.1:5000/login, то мы будем перемещены на защищенную страницу и увидим её содержание.

В заключение, использование Flask-Login для добавления системы аутентификации в веб-приложения на Flask позволяет разработчикам эффективно управлять сессиями пользователей, обеспечивая безопасность и легкость в обслуживании. Это расширение предлагает простой и гибкий интерфейс для реализации входа и выхода из системы, а также для проверки статуса аутентификации пользователя в любой части приложения.

Библиографический список:

1. Доусон М. Программируем на Python [Текст] / Доусон М. – СПб.: Издательский Дом ПИТЕР, 2022. – 416 с.
2. Гэддис Т. Начинаем программировать на Python [Текст] / Т. Гэддис. – СПб.: БХВ-Петербург, 2021. – 768 с.
3. Васильев А.Н. Программирование на Python в примерах и задачах [Текст] / Васильев А.Н. – М.: Бомбора, 2021, 2022.
4. Гуриков, С.Р. Основы алгоритмизации и программирования на Python [Текст] / С.Р. Гуриков. – М.: ИНФРА-М, 2023. – 343 с.

Оригинальность 89%