

**Анализ и сравнение различных методов аутентификации и авторизации в веб-приложениях на примере OAuth, JWT и session-based authentication**

Содержание

[Обзор проекта 1](#_Toc181057902)

[Термины и определения 1](#_Toc181057903)

[Структура проекта 2](#_Toc181057904)

[Форма Аутентификации 2](#_Toc181057905)

[Форма Регистрации 3](#_Toc181057906)

[Шаблоны страниц 3](#_Toc181057907)

[Структура базы данных 4](#_Toc181057908)

[Анализ методов аутентификации 4](#_Toc181057909)

[Session-based authentication 4](#_Toc181057910)

[JWT (JSON Web Token) 6](#_Toc181057911)

[OAuth 2 10](#_Toc181057912)

[Заключение 14](#_Toc181057913)

[Идеи по модернизации 14](#_Toc181057914)

[Приложение 1. Пример файловой структуры проекта 15](#_Toc181057915)

[Приложение 2. Список необходимых библиотек 16](#_Toc181057916)

[Приложение 3. Ссылки на используемую документацию 17](#_Toc181057917)

# Обзор проекта

Для дипломного проекта была выбрана тема: «Анализ и сравнение различных методов аутентификации и авторизации в веб-приложениях: OAuth, JWT и session-based authentication». Целью дипломного проекта является знакомство с каждым из выбранных методов с последующей их реализацией, а также анализ документации и сбор данных для возможности определения достоинств и недостатков.

Для анализа и сравнения каждого из выбранных методов был создан одностраничный проект на базе Django. На единственной странице которого, отображается форма аутентификации или регистрации (в зависимости от того, что необходимо пользователю). Для создания более приятного интерфейса был использован инструментарий Bootstrap версии 5.1.3, который включает в себя различные CSS и HTML-шаблоны оформления.

# Термины и определения

**Регистрация** — это процесс передачи персональных данных пользователя, на основании которых, пользователь сможет получить доступ к определенным возможностям веб-сайта, в соответствии с его уровнем доступа. При регистрации создаётся личный кабинет (профиль или запись), через который пользователь может, например, смотреть фильмы, заказывать товары и отслеживать их обработку, а также просматривать историю своих запросов и многое другое.

**Аутентификация** — процедура проверки подлинности введённых пользователем данных. В процессе аутентификации пользователь передает серверу данные для входа, чаще используется комбинация логин и пароль. В качестве логина может выступать различная информация, например, адрес электронной почты, а паролем чаще выступает комбинация из цифр, букв (в различном регистре) и специализированных символов.

**Авторизация** — процесс предоставления прав доступа к различным возможностям и областям веб-сайта на основании присвоенных учетной записи прав (или уровня) доступа. Процесс авторизации происходит строго после процедуры аутентификации и может повторяться при каждом обращении пользователя к той или иной области (возможности) веб-сайта. Так же пользователь в процессе авторизации может получить доступ на определенное время. Для этой цели могут быть использованы сессионные ключи, либо токены.

**Токен** — или ключ, это идентификатор, который используется для представления доступа к различным возможностям и областям веб-сайта. Токен может быть представлен в виде последовательности букв и цифр, а также в виде физического носителя (чаще в виде подключаемого usb устройства).

# Структура проекта

Проект включает следующие ключевые компоненты:

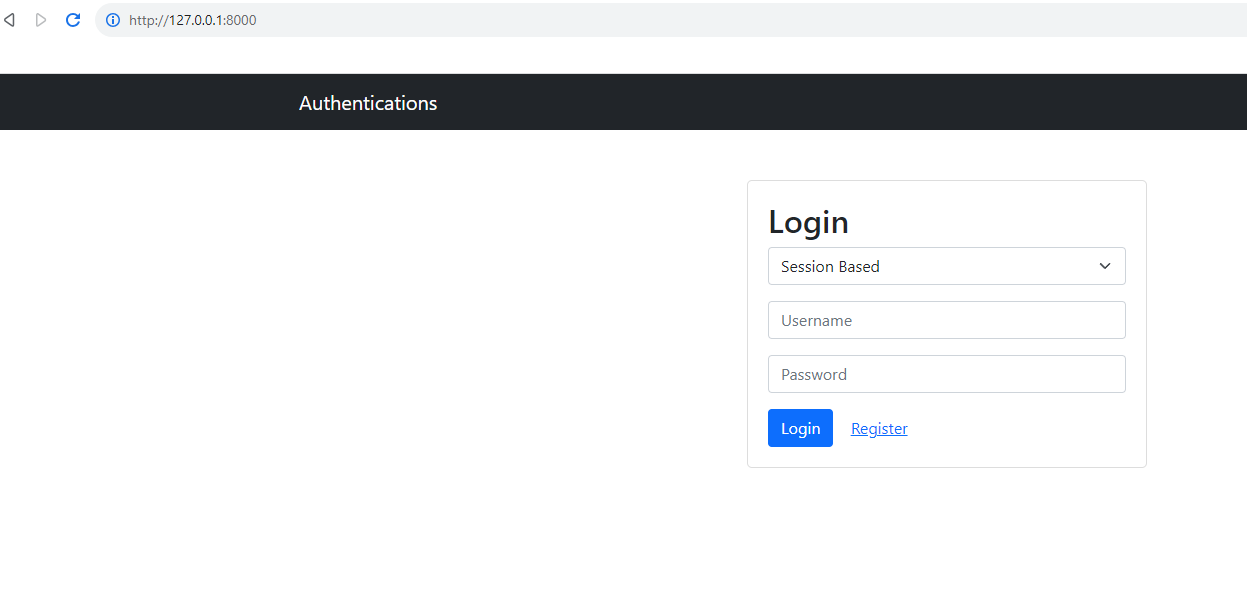
## Форма Аутентификации

Данная страница отображает базовый (base.html) шаблон, который дополняется блоками из шаблона страницы аутентификации (mainpage.html).

На Рисунке 1 показано в каком виде пользователю представляется страница.

На Главной (Домашней) странице отображается название сайта или приложения (Authentications), а также форма аутентификация, в которой можно:

* Выбрать метод (session-based authentication, JWT или OAuth);
* Ввести имя пользователя (username) и пароль (password);
* Кнопка (Login) для передачи на сервер введенной информации с последующей ее проверкой;
* Текстовая ссылка (Register) для перехода к форме регистрации.



*Рисунок 1 Форма аутентификации*

## Форма Регистрации

Данная страница отображает базовый (base.html) шаблон, который дополняется блоками из шаблона страницы аутентификации (mainpage.html).

На Рисунке 2 показано в каком виде пользователю представляется страница.

На Главной (Домашней) странице отображается название сайта или приложения (Authentications), а также форма регистрации, в которой можно:

* Выбрать метод (session-based authentication, JWT или OAuth);
* Ввести данные для регистрации такие как имя пользователя (username), адрес электронной почты (email), пароль (поля password и confirm password);
* Кнопка (Register) для передачи на сервер введенной информации с последующей ее проверкой на наличие использования ранее (в случае если пользователь с такими данными уже зарегистрирован);
* Текстовая ссылка (Back to Login) для возврата к форме аутентификации.



*Рисунок 2 Форма регистрации*

## Шаблоны страниц

base.html – базовый шаблон, в котором подключается инструментарий Bootstrap, а также CSS стиль оформления границ форм. На основании данного шаблона разрабатывается шаблон аутентификации.

mainpage.html – главная (домашняя) страница, с формами аутентификации и регистрации.

## Структура базы данных

После написания кода приложения и выполнения миграции получаем показанную на Рисунке 3 структуру.



*Рисунок 3 Структура базы данных*

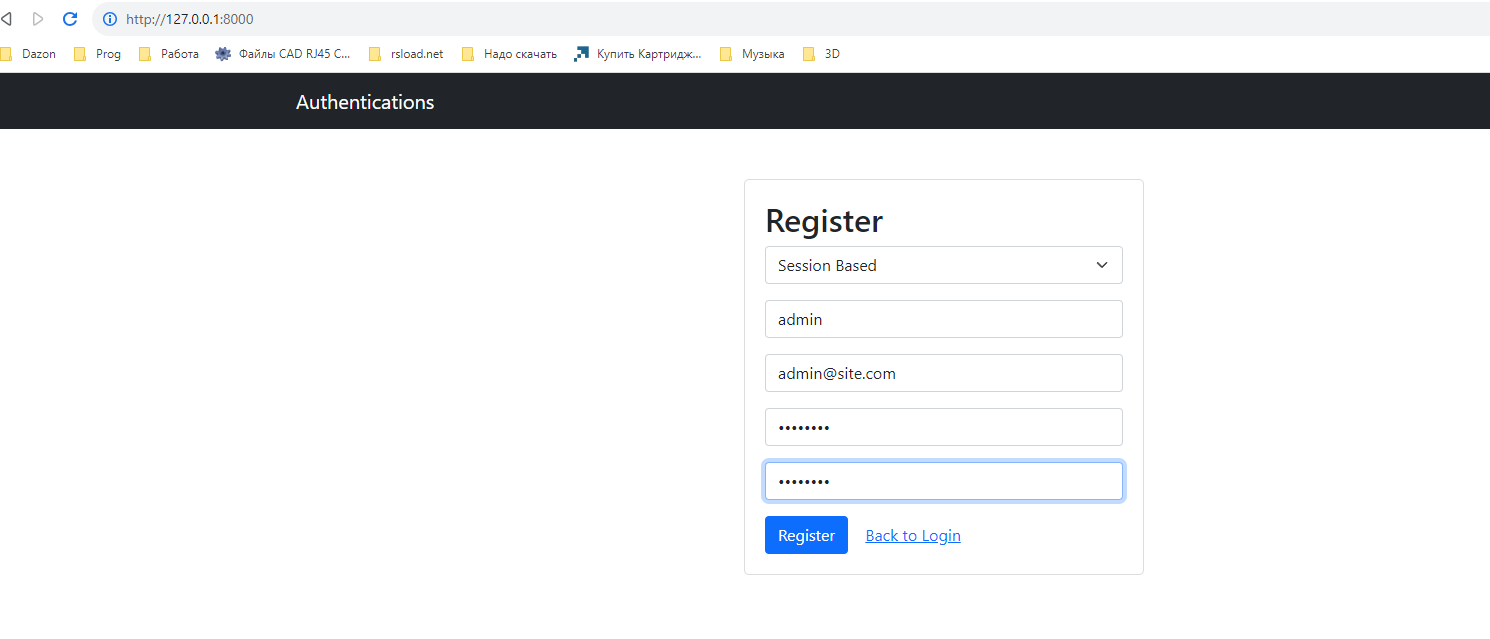
# Анализ методов аутентификации

Проведем анализ каждого из выбранных методов:

## Session-based authentication

Session-based authentication - это классический и надёжный метод аутентификации.

Для тестирования данного метода создадим нового пользователя admin. Процесс ввода регистрационных данных показан на Рисунке 4:

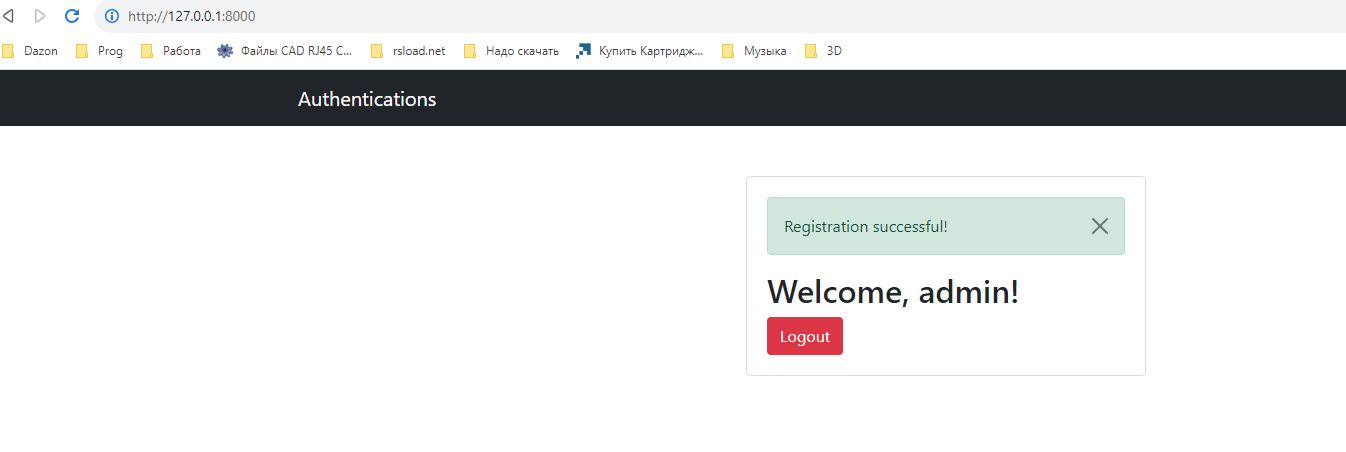


*Рисунок 4 Форма регистрации с введенными учетными данными*

Принцип работы данного метода построен на процессе аутентификации, при котором:

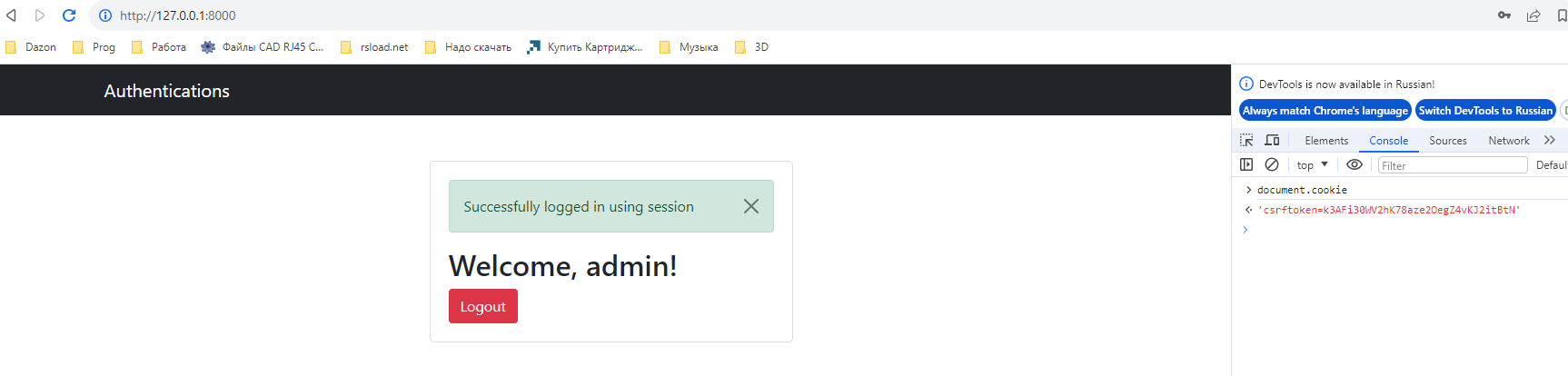
* Пользователь вводит логин и пароль;
* Сервер проверяет учетные данные;
* При успехе создаётся сессия с уникальным ID;
* ID сессии сохраняется в cookie браузера.

После отправке данных на сервер и успешной проверки на совпадения пользователь попадает в свой профиль, раздел или заданную администратором область сайта (приложения). Пример входа показан на Рисунке 5.



*Рисунок 5 Успешная аутентификация*

При реализации данного метода сессионные данные хранятся на сервере, в cookie хранится только ID сессии, так же можно хранить дополнительные данные пользователя (например, пользовательские настройки, состояние корзины и прочее). Пример cookie показан на Рисунке 6.

 *Рисунок 6 Пример сохраняемых cookie*

К преимуществам данного метода можно отнести:

* Простоту реализации;
* Доступность «из коробки», так как он является встроенным в Django по умолчанию;
* Легкость в отзыве сессии;
* Хорошая безопасность так как имеет защиту от XSS и CSRF атак.

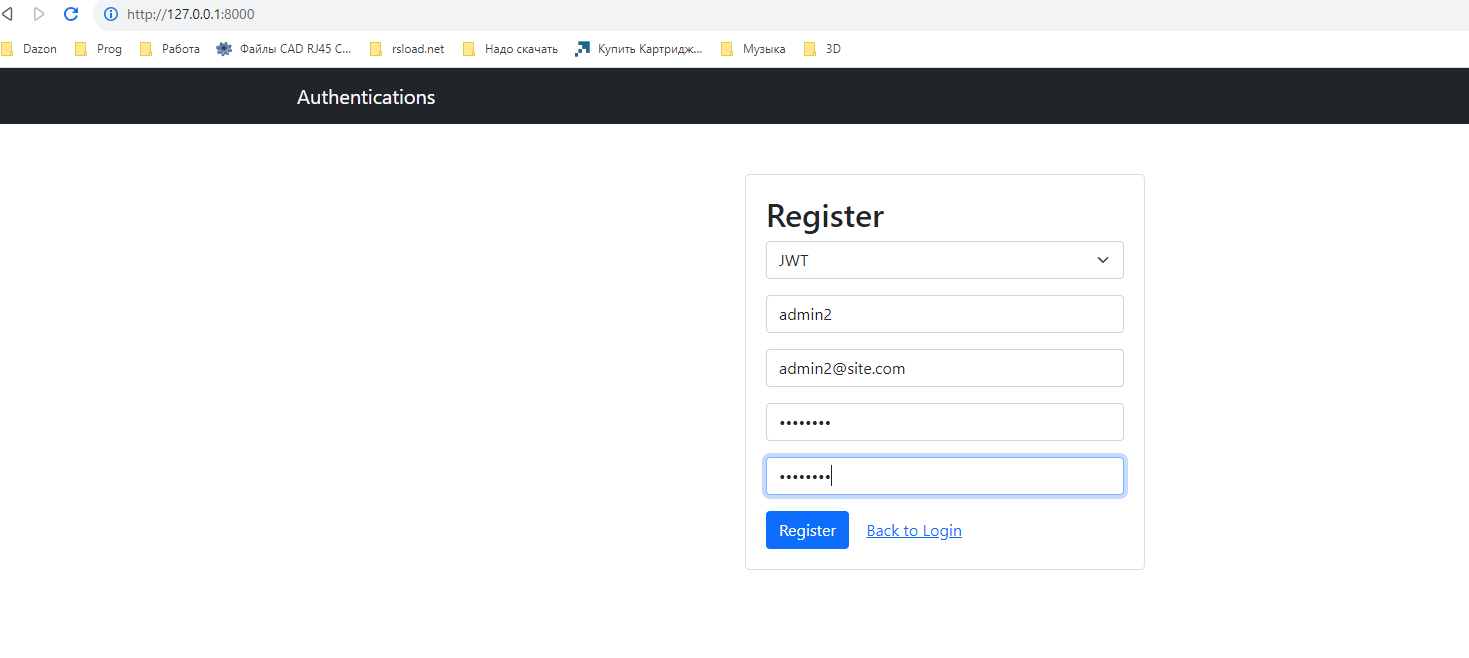
К недостаткам можно отнести:

* Масштабируемость, так как требуется хранение сессий на сервере, что при большом количестве пользователей будет сильно влиять на память сервера;
* Производительность, так как при каждом запросе необходимо обращаться к базе данных или хранилищу сессий;
* Безопасность, так как присутствует риск атак через перехват сессии, возможность CSRF атак без правильной защиты, необходимость правильной конфигурации cookies.

## JWT (JSON Web Token)

JWT - это JSON объект, который определен в открытом стандарте RFC 7519.

Для тестирования данного метода создадим нового пользователя admin2 и укажем метод аутентификации JWT как это показано на Рисунке 7.

 *Рисунок 7 Создаем пользователя для тестирования метода JWT*

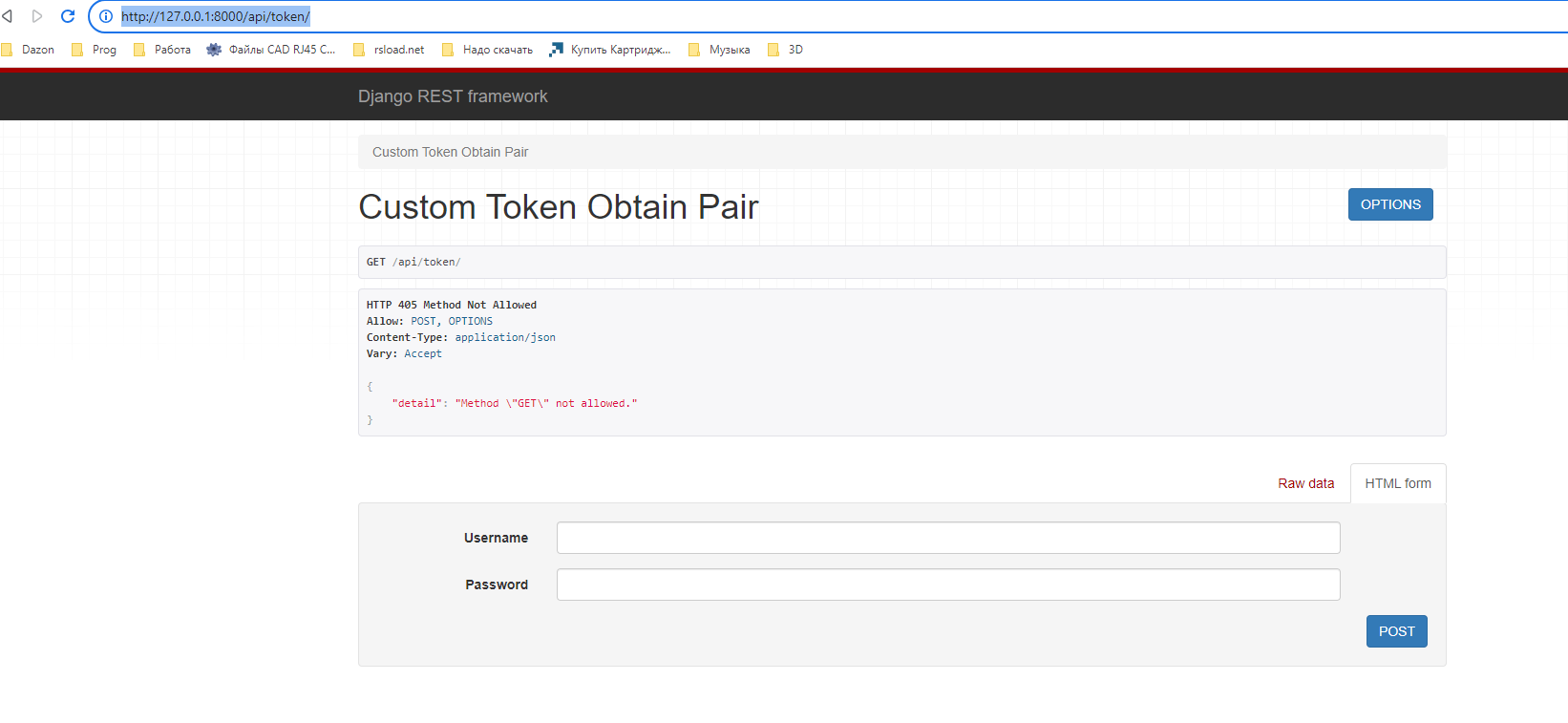
Принцип работы данного метода построен на процессе аутентификации, при котором:

* Сперва пользователь заходит на сервер аутентификации с помощью аутентификационного ключа, которым может быть пара логин/пароль, либо Google ключ, либо ключ от учетной записи стороннего сервиса;
* Сервер генерирует JWT токен (строка в формате header.payload.signature) и отправляет его пользователю. Header - содержит информацию о том, как должна вычисляться JWT подпись. Payload - это полезные данные, которые хранятся внутри JWT (например, user\_id). Signature – электронная подпись, которая формируется особым образом;
* Когда пользователь делает запрос к API приложения, он добавляет к нему полученный ранее JWT;
* Когда пользователь делает API запрос, приложение может проверить по переданному с запросом JWT является ли пользователь тем, за кого себя выдает.

После аутентификации пользователь видит доступный ему раздел или область сайта (приложения) как это показано на Рисунке 8.

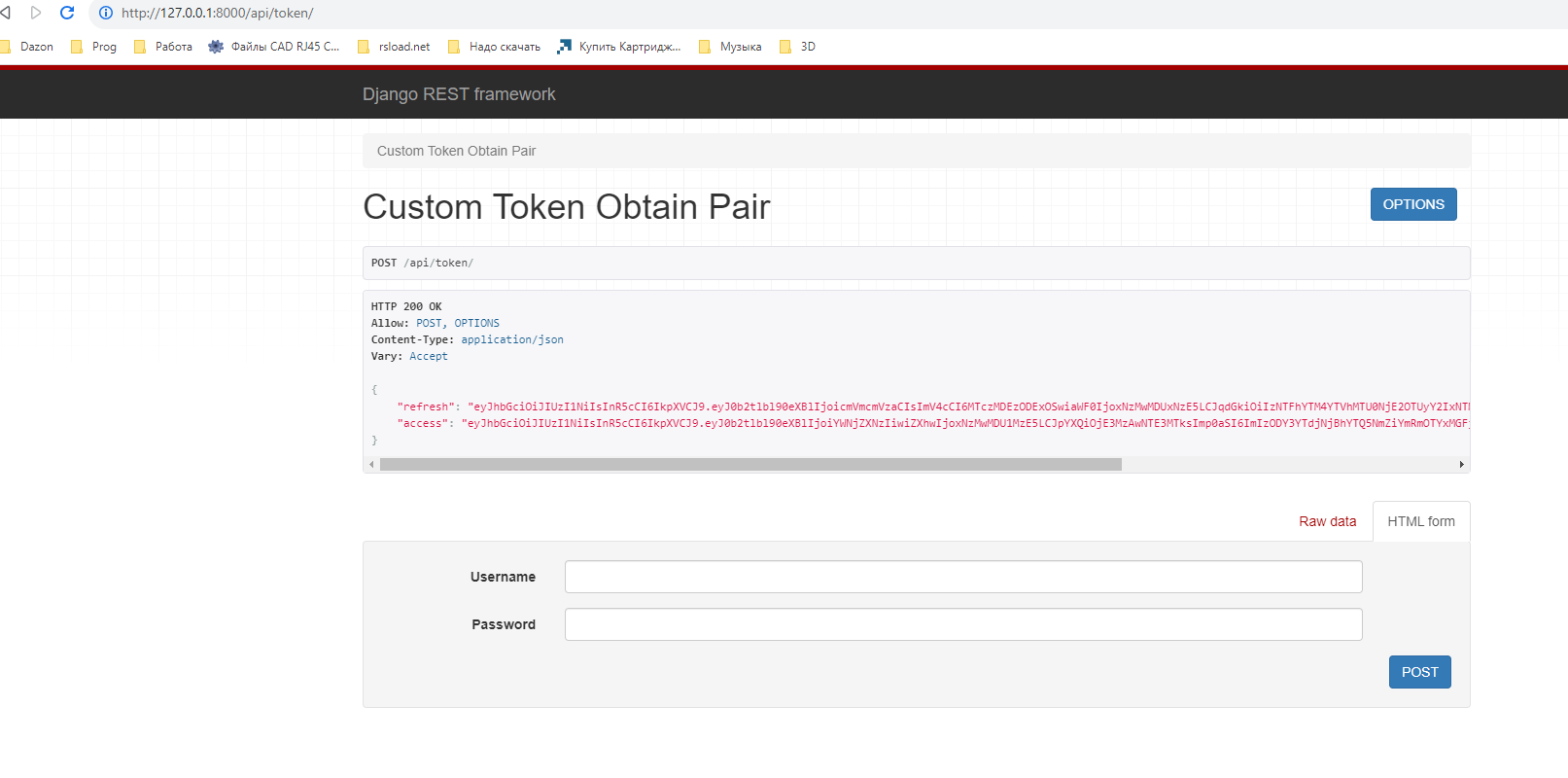
 *Рисунок 8 Пользователь успешно вошел*

Чтобы показать как работает данный метод воспользуемся предусмотренным в REST framework API. Для этого перейдем по ссылке <http://127.0.0.1:8000/api/token/>. Результат показан на Рисунке 9.

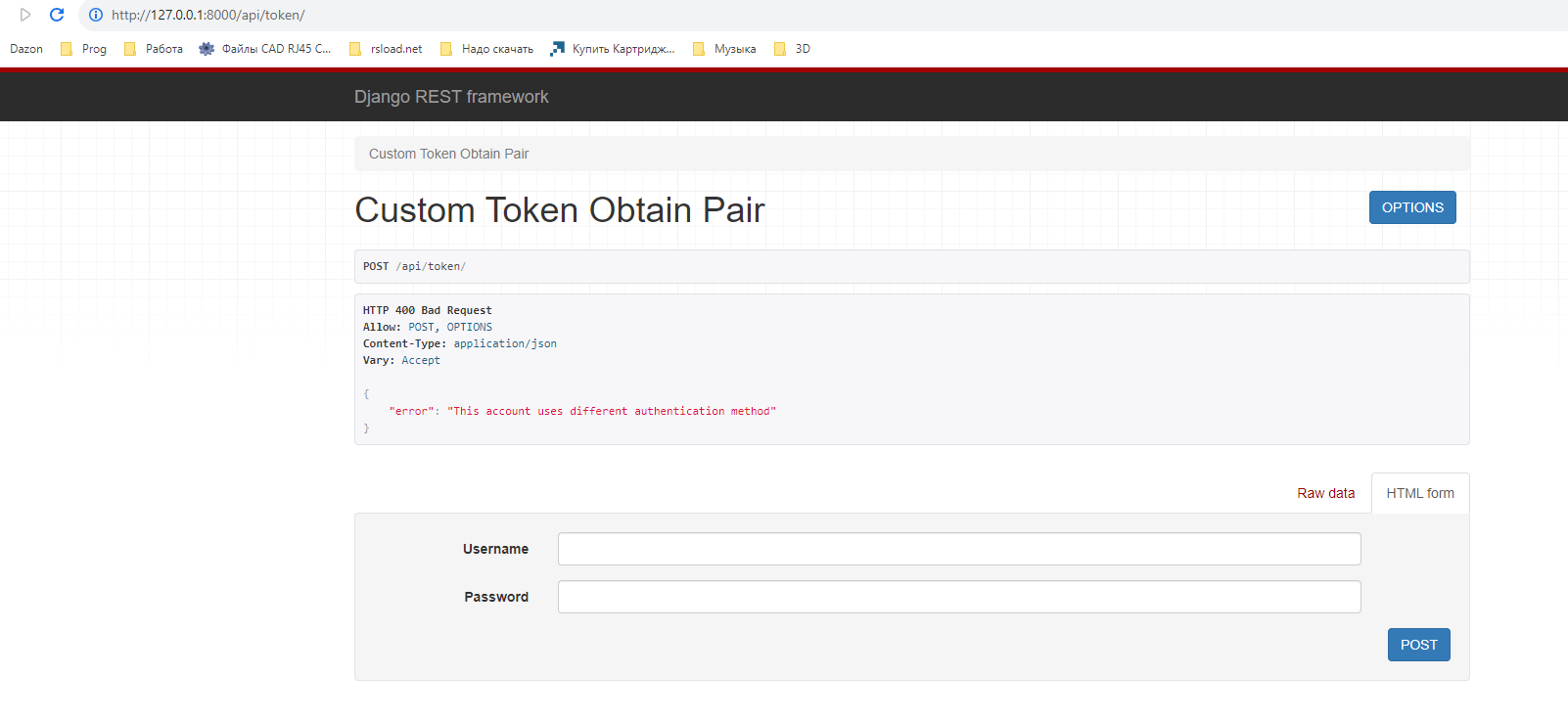


*Рисунок 9 REST framework API*

Для получения JWT токена отправим POST запрос с именем пользователя и паролем, в случае ошибки получим предусмотренные нами сообщения. Результат представлен на Рисунках 10 и 11.

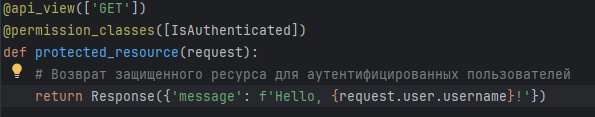


*Рисунок 10 JWT токен*



*Рисунок 11 Результат запроса с неправильными данными*

Теперь для проверки аутентификационных данных воспользуемся утилитой CURL и отправим GET запрос на сервер используя access token. В результате мы должны будем получить доступ к участку кода, показанному на Рисунке 12:

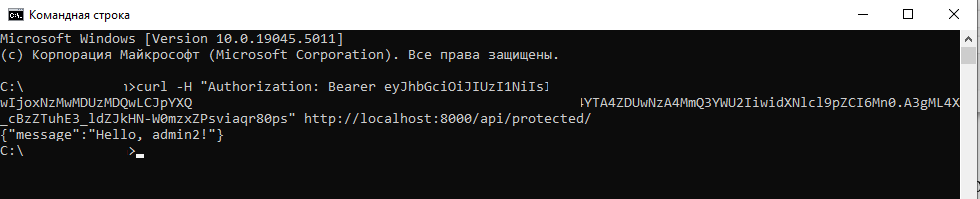


*Рисунок 12 Защищенный участок приложения, доступный только с токеном*

Наш запрос будет иметь вид:

*curl -H “Authorization: Bearer «access\_token»* [*http://localhost:8000/api/protected/*](http://localhost:8000/api/protected/)

В результате мы получим ответ в формате строки со следующим содержанием «Hello, admin2!». Выполнение запроса показано на Рисунке 13.

 *Рисунок 13 Результат запроса защищенной области используя JWT токен*

К преимуществам данного метода можно отнести:

* Не требует хранения на сервере;
* Отлично масштабируется;
* Работает на разных доменах;
* Идеален для REST API.

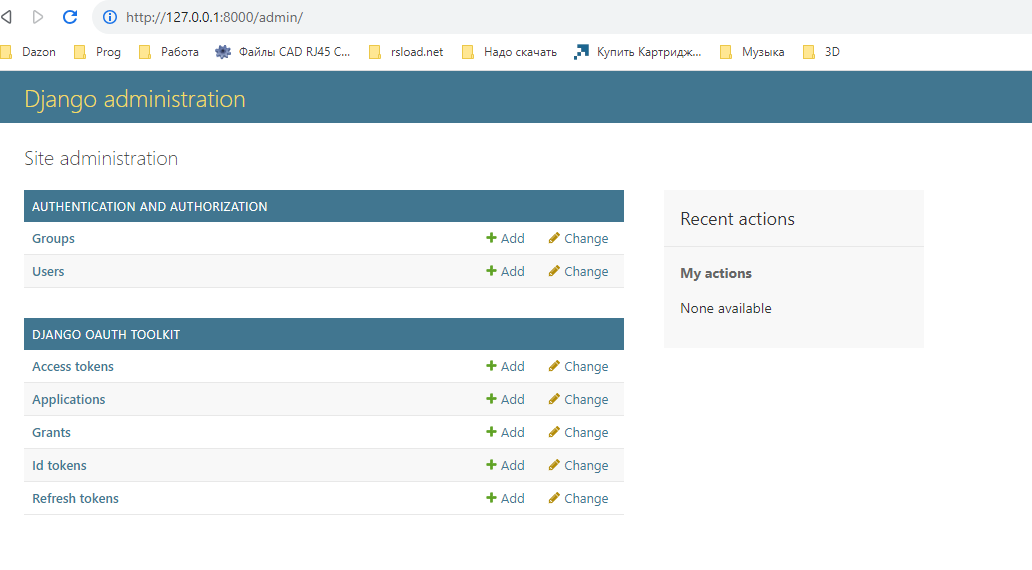
Из недостатков данного метода можно выделить:

* Объем передаваемых данных. Чем больше данных будет указано в payload, тем больше будет размер ключа;
* Низкая гарантия безопасности. Так как при создании JWT токена применяется только кодирование и подписание, но отсутствует шифрование – такой подход не дает гарантии защиты чувствительных данных;
* Нельзя отозвать отдельный токен до истечения срока;
* Токен хранится на клиенте.

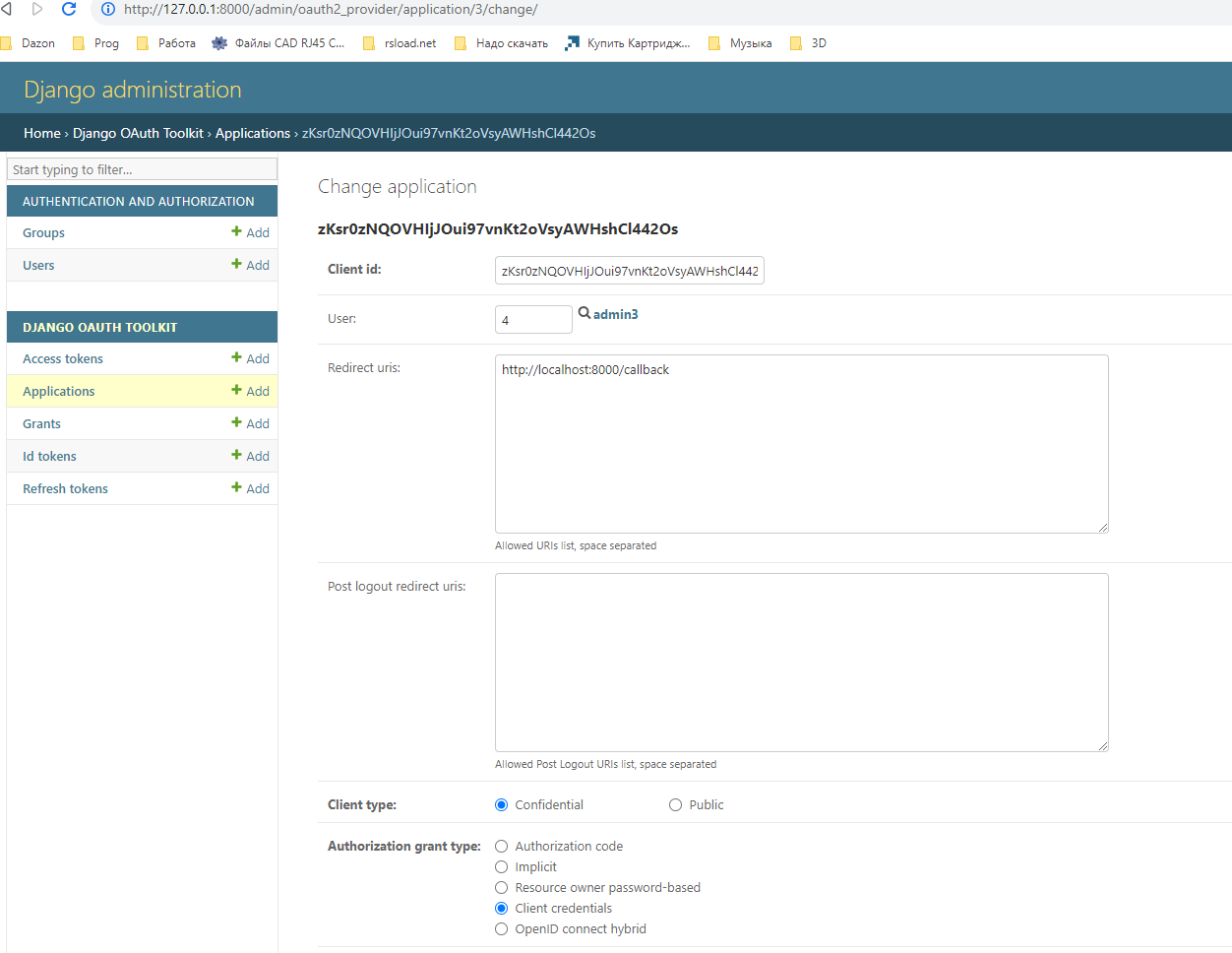
## OAuth 2

OAuth 2 - протокол авторизации (**не аутентификации**), позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя на другом сервисе. Протокол избавляет от необходимости доверять приложению **логин** и **пароль**, а также позволяет выдавать **ограниченный набор прав**, а не все сразу.

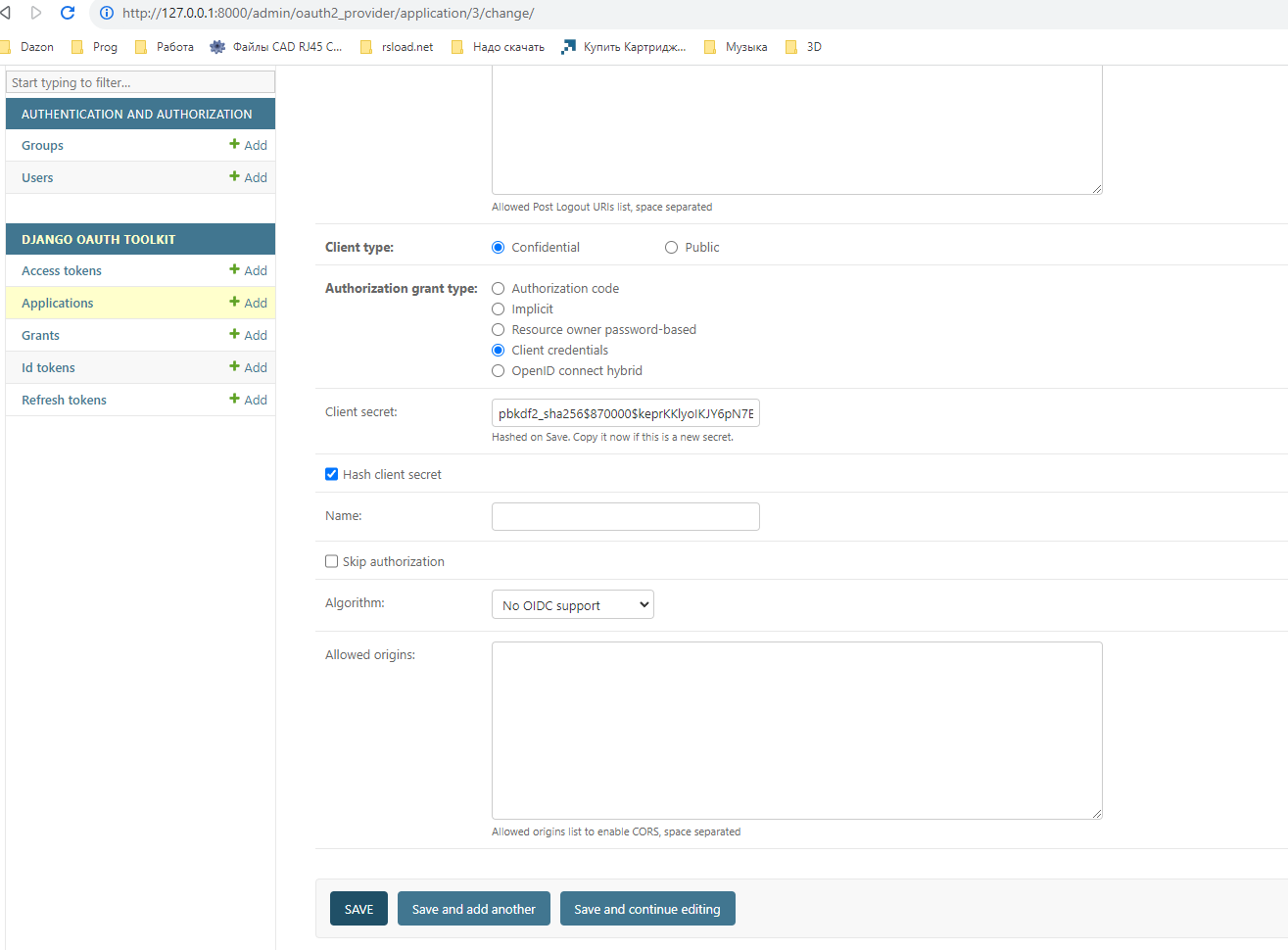
Для тестирования данного метода создадим нового пользователя admin3 как это было показано ранее. Далее для проверки метода авторизации OAuth2 перейдем в admin-панель Django, как это показано на Рисунке 14.

 *Рисунок 14 Администраторская панель Django*

Для проверки метода необходимо создать клиентское приложение, получить токен и отправить запрос на сервер, используя токен. В результате пользователь получит доступ к защищенной области вебсайта (приложения). Для этого создадим запись в Applications (Django OAuth Toolkit) как это показано на Рисунках 15.1 и 15.2.

**

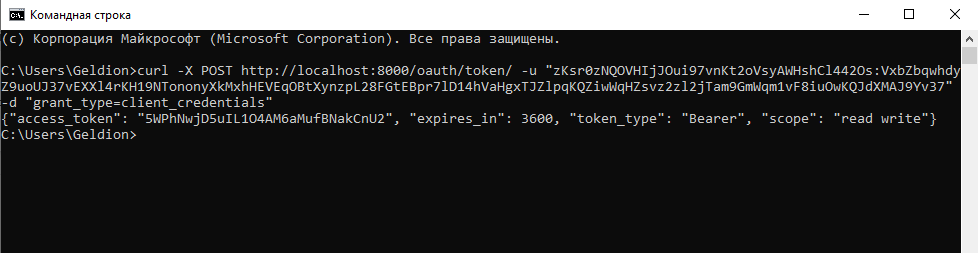
*Рисунок 15.1 Настройка клиентского приложения (начало)*

** *Рисунок 15.2 Настройка клиентского приложения (продолжение)*

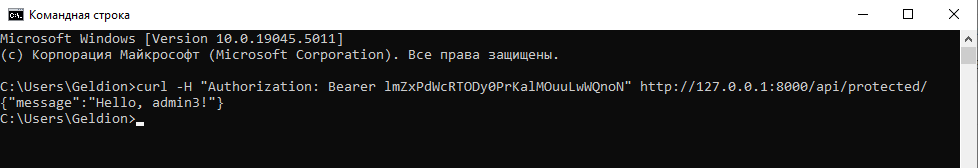
Далее для получаем токен. Для этого отправляем POST запрос используя CURL:

curl -X POST http://localhost:8000/oauth/token/ -u "client\_id:client\_secret" -d "grant\_type=client\_credentials"

И в результате, который отображен на Рисунке 16, видим токен.

 *Рисунок 16 Получаем OAut2h токен*

Теперь используя полученный токен, можно обратимся к защищенной области веб-сайта (приложения). Пример ответа сервера показан на Рисунке 17.

 *Рисунок 17 Ответ от сервера на запрос с использованием токена*

К преимуществам данного метода можно отнести:

* Упрощенная интеграция;
* Повышенная безопасность;
* Гибкая настройка прав;
* Поддержка мобильных приложений;
* Возможность обновления токенов.

Из недостатков данного метода можно выделить:

* Сложность реализации. Так как метод требует тщательного планирования архитектуры, правильной настройки безопасности и имеет сложную обработка ошибок;
* Требует отдельного сервера авторизации;
* Поскольку данный метод еще развивается, отсутствует устоявшаяся спецификация в следствии чего при выпуске очередного обновления может сильно меняться, что приводит усложнению технической поддержки;
* Безопасность. Так как метод во многом основана на SSL, это сильно упрощает жизнь разработчикам, но требует дополнительных вычислительных ресурсов и администрирования. Это может быть существенным вопросом в высоко нагруженных проектах.

# Заключение

Основываясь на представленном анализе выбранных методов аутентификации и авторизации, можно сделать следующий вывод:

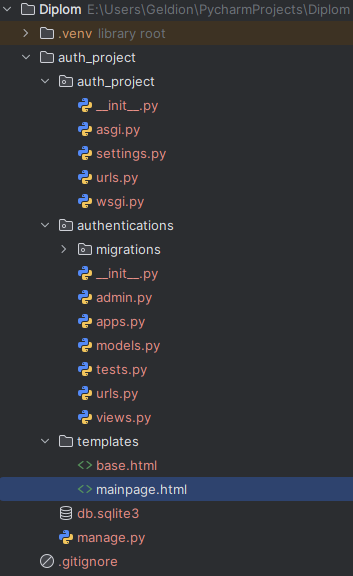
* Session-based: идеален для классических веб-приложений;
* JWT: отличный выбор для современных API и мобильных приложений;
* OAuth2: лучшее решение для сложных систем с множественной интеграцией.

# Идеи по модернизации

В качестве модернизации и улучшения функционала приложения можно выделить некоторые позиции, например:

* Добавить отображении cookie в профиле пользователя, использующего метод Session-based authentication;
* В форме аутентификации добавить кнопки авторизации сторонними ресурсами, такими как Google, GitHub;
* Добавить форму и механизм восстановления пароля учетной записи путем генерации уникальной ссылки;
* Добавить в профили пользователей функционал управления cookie (например, удаление), генерацию и перевыпуск токенов для методов JWT и OAuth2.

# Приложение 1. Пример файловой структуры проекта



# Приложение 2. Список необходимых библиотек

asgiref==3.8.1  
certifi==2024.8.30  
cffi==1.17.1  
charset-normalizer==3.4.0  
cryptography==43.0.3  
Django==5.1.2  
django-oauth-toolkit==3.0.1  
djangorestframework==3.15.2  
djangorestframework-simplejwt==5.3.1  
idna==3.10  
jwcrypto==1.5.6  
oauthlib==3.2.2  
pycparser==2.22  
PyJWT==2.9.0  
requests==2.32.3  
sqlparse==0.5.1  
typing\_extensions==4.12.2  
tzdata==2024.2  
urllib3==2.2.3

# Приложение 3. Ссылки на используемую документацию

1. [Django REST framework](https://www.django-rest-framework.org/api-guide/authentication/);
2. [Django Oauth2 toolkit documentation](https://django-oauth-toolkit.readthedocs.io/en/latest/);
3. [Simple JWT](https://django-rest-framework-simplejwt.readthedocs.io/en/latest/getting_started.html);
4. Сообщество IT специалистов [Хабр](https://habr.com/):
   1. [Пять простых шагов для понимания JSON Web Tokens (JWT)](https://habr.com/ru/articles/340146/);
   2. [OAuth 2.0 простым и понятным языком](https://habr.com/ru/companies/vk/articles/115163/);
   3. [Настройка аутентификации JWT в новом проекте Django](https://habr.com/ru/articles/538040/);
   4. [Безопасность в деталях: исследование cистемы защиты от CSRF](https://habr.com/ru/companies/banki/articles/759058/).
5. [Сравнение аутентификации на основе токенов и аутентификации на основе сессий](https://blog.logto.io/ru/token-based-authentication-vs-session-based-authentication);
6. [REST framework JWT Auth](https://jpadilla.github.io/django-rest-framework-jwt/);
7. [How to Use JWT Authentication with Django REST Framework](https://simpleisbetterthancomplex.com/tutorial/2018/12/19/how-to-use-jwt-authentication-with-django-rest-framework.html);
8. Сообщество [Reddit](https://www.reddit.com/);
9. Сообщество [Stackoverflow](https://stackoverflow.com/);
10. [Django-oauth2-provider’s documentation](https://django-oauth2-provider.readthedocs.io/en/latest/);
11. [Pypi.org](https://pypi.org/).