Дипломная работа по теме: **Анализ и сравнение различных методов аутентификации и авторизации в веб-приложениях: OAuth, JWT и session-based authentication**.

Автор: Трошин Константин Владимирович

**Оглавление дипломной работы:**

Contents

1. Введение. 6

Обоснование выбора темы: 6

Определение цели и задач исследования: 6

2. Основные понятия и определения. 8

3. Методы и подходы к разработке. 10

4. Обзор использованных фреймворков

Django

Django REST framework

Djoser

[Django OAuth Toolki](https://github.com/evonove/django-oauth-toolkit)t

[Drf-social-oauth2](https://github.com/wagnerdelima/drf-social-oauth2)

5. Проектирование приложения. 12

Планирование и анализ требований. 12

Основные требования. 12

Технические требования. 12

6. Разработка в соответствии с созданной документацией. 13

Планирование разработки. 13

Разработка. 13

8. Анализ и сравнение методов

session-based authentication.

JWT

OAuth

9. Заключение. 14

Обзор выполненной работы.. 14

Мысли по диплому

**1. Введение**

**Обоснование выбора темы:**

1. С самого начала историю, становления первых сообществ , государств обмен информацией был ключевым элементом их развития. Обладание знаниями, технологиями давало их обладателям конкурентные преимущества и потому сохранение информации и контроль ее передачи являлись и являются важными инструментами.

2. Хранение информации - по сути ограничение доступа к ней. Для сохранности и контроля доступа к информации служат два инструмента: аутентификация и авторизация

Развитие технологий авторизации и аутентификации - процесс постоянный как и рост рисков связанных с хранением.

3. Компетенции в сфере сохранности информации будут востребованы еще очень долго.

4. Мне эта тема показалось интересной и возможно полезной при работе в будущем.

Таким образом, выбор темы”**Анализ и сравнение различных методов аутентификации и авторизации**” обусловлен ее актуальностью, потребностями рынка, практической значимостью и личным интересом, что делает эту тему подходящей для проведения дипломной работы.

**Определение цели и задач работы:**

Реализовать аутентификацию и авторизацию с использованием OAuth, JWT и session-based authentication, сравнить их безопасность и удобство использования.

Задачи исследования:

1. Рассмотреть способы аутентификации авторизации.

2. Рассмотреть фреймворки для их реализации.

3. Реализовать их и продемонстрировать их применение.

4. Провести анализ существующих инструментов, выбрать наиболее подходящие под имеющуюся задачу.

5. Написать дипломной работу отражающую, выше перечисленные задачи.

Цели и задачи исследования направлены на получение практических результатов и анализа их применения.

**2. Основные понятия и определения**

**Идентификация** — это процесс, когда информационная система, например социальная сеть или интернет-магазин, определяет, существует конкретный пользователь или нет. Делает она это с помощью идентификатора.

Идентификатором может быть логин, электронная почта, номер телефона или другой признак, который есть только у одного пользователя. Идентификатор позволяет сайту или приложению отличить конкретного человека от других людей.

Пример идентификации. При входе в личный кабинет на сайт vk.com просят указать электронную почту. Это идентификатор. Он помогает системе понять, какой именно пользователь хочет зайти в личный кабинет сайта

Принцип работы любой системы идентификации: два одинаковых идентификатора не могут существовать одновременно.

**Аутентификация** — это процесс, когда пользователь вводит ключ, например пароль или пин-код, подтверждая своё право на доступ к той или иной учётной записи и хранящейся в ней информации.

Пример аутентификации. Для входа в личный кабинет мы вводим пароль, который говорит системе о том, что мы имеем право на доступ в личный аккаунт

Аутентификация бывает одно-, двух- и трёхфакторной.

Однофакторная аутентификация требует подтверждения только одним способом — например, с помощью пароля. Она встречается чаще всего.

Двухфакторная аутентификация используется в системах, которые хранят важные или личные данные. Например, в банковских приложениях или в социальных сетях. При входе в соцсеть у пользователя могут попросить не только пароль, но и другую информацию — код из СМС или биометрические данные.

В системах с повышенными требованиями к безопасности — например, в банковской сфере — встречается трёхфакторная аутентификация. Третьим фактором, позволяющим подтвердить личность, могут быть электронные ключи доступа. Электронный ключ хранится на специальном USB-накопителе и подключается в момент подтверждения доступа.

В работе будет рассматриваться только однофакторная аутентификация.

**Авторизация** — это процесс присвоения учётной записи положенных ей привилегий.

Если авторизация прошла успешно, человек попадает в личный кабинет. Он может читать уведомления, видеть список доступных ресурсов, просматривать их. Права доступа к этим действиям называются **привилегиями**.

Пример авторизации. На сайте Urban University после ввода верного логина и пароля система понимает, какие права есть у пользователя, показывает доступные ему курсы и предоставляет другую функциональность личного кабинета учащегося

Другая задача авторизации — защищать систему от изменений со стороны пользователей, которые не должны влиять на её работу. Например, в компаниях часто нельзя самостоятельно установить программное обеспечение на рабочий компьютер. Прав обычного пользователя для этого не хватит. Такими привилегиями обладает только системный администратор компании, который авторизуется под своим логином и может устанавливать дополнительный софт.

**Рассматриваемые методы**

**Session-based authentication** — это метод, который используется для поддержания состояния аутентификации пользователя в течение нескольких запросов в рамках одной сессии.

**JWT** или **JSON Web Token** представляет собой надежный и безопасный метод передачи информации между сторонами в виде JSON-объектов. Не просто так этот метод один из наиболее популярных в мире.

[**OAuth 2.0**](http://tools.ietf.org/html/draft-ietf-oauth-v2) — протокол авторизации, 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.

**3. Методы и подходы к разработке**

Поскольку основной задачей работы является изучение, использование и анализ методов аутентификации и авторизации а не создание полнофункционального приложения, то ее схема видится следующим образом:

1. Создается база данных User

2. В ней регистрируются пользователи

3. Их аутентификация и авторизация проходит по очереди при помощи 3 методов, указанных в задании.

Таким образом используется одно приложение со своей локальной базой данных, а весь код, описывающий методы аутентификации и авторизации, прописывается в файлах settings.py b urls.py

**4. Обзор использованных фреймворков**

**Django** —основной фреймворк для веб-разработки на Python.

Основные возможности:

· Встроенная страница администрирования для управления данными.

· ORM (Object-Relational Mapping) для работы с базами данных.

· Поддержка аутентификации пользователей и управления доступом.

· Мощная система маршрутизации URL.

· Шаблонизатор для создания HTML-шаблонов.

Особенности:

Идеально подходит для больших и сложных проектов благодаря своей полной экосистеме.

**Django REST framework** (DRF) — это мощный набор инструментов для создания веб-сервисов и API на основе фреймворка Django. Он является одним из наиболее широко используемых для создания RESTful API в экосистеме Django.

**Django** и **Django REST Framework** – два разных инструмента для разработки веб-приложений, имеющие существенные отличия. Остановлюсь на главных, решающих в реализации этого проекта:

* Аутентификация и авторизация.

**Django** предлагает свои собственные механизмы аутентификации и авторизации, такие как аутентификация с использованием сессий и системы разрешений на основе ролей.

**Rest Django** расширяет эти механизмы и предоставляет дополнительные методы аутентификации, такие как аутентификация с использованием токенов, OAuth и JWT. Он также предоставляет более гибкие и мощные средства для управления правами доступа к API.

* Представления и маршрутизация.

**Django** имеет собственный механизм представлений (views) и маршрутизации URL. Он использует функции или классы в качестве представлений и использует URL-шаблоны для маршрутизации запросов к соответствующим представлениям.

**Rest Django** имеет свои типы представлений, такие как представления на основе классов, представления REST API и представления для работы с моделями. Он также предлагает простой и гибкий механизм для маршрутизации запросов к соответствующим представлениям.

Можно сказать, **Rest Django** является надстройкой над Django, специально предназначенной для разработки RESTful API, и предоставляет множество компонентов и инструментов, которые сделали его популярным выбором для создания API в экосистеме Django.

**Djoser** - это простая библиотека аутентификации для Django. Он используется для генерации токенов для аутентификации; этот сгенерированный токен генерируется путем ввода трех полей: **имя пользователя, адрес электронной почты** и **пароль.**

**REST**-реализация системы аутентификации **Django**. Библиотека **djoser** предоставляет набор представлений **Django Rest Framework** для обработки базовых действий, таких как регистрация, вход в систему, выход из системы, сброс пароля и активация учётной записи. Она работает с **пользовательской моделью.**

Вместо повторного использования кода **Django** (например, PasswordResetForm), были разработаны некоторые элементы, чтобы они лучше вписывались в архитектуру одностраничного приложения.

Пакет [**Django OAuth Toolkit**](https://github.com/evonove/django-oauth-toolkit) обеспечивает поддержку OAuth 2.0 и работает с Python 3.4+. Пакет поддерживается [jazzband](https://github.com/jazzband/) и использует превосходную [OAuthLib](https://github.com/idan/oauthlib). Пакет хорошо документирован и поддерживается и в настоящее время является нашим рекомендуемым пакетом для поддержки OAuth 2.0.

[**Drf-social-oauth2**](https://github.com/wagnerdelima/drf-social-oauth2) — это фреймворк, который помогает проходить аутентификацию с помощью основных поставщиков социальных сервисов oauth2, таких как Facebook, Google, Twitter, Orcid и т. д. Он генерирует токены в формате JWT с помощью простой настройки.

[**Django OAuth Toolki**](https://github.com/evonove/django-oauth-toolkit)**t** и [**Drf-social-oauth2**](https://github.com/wagnerdelima/drf-social-oauth2)используются совместно для реализации OAuth2 метода

**5. Проектирование приложения**

**Планирование и анализ требований**

Выбор фреймворка и инструментов: Определение наиболее подходящего фреймворка и инструментов для разработки.

Определение структуры приложения: Разработка схемы архитектуры, включающей фронтенд, бэкенд, базу данных

Разработка прототипа

Реализация основного функционала: Создание базовой версии приложения с минимально необходимым функционалом для тестирования и демонстрации.

Разработка ключевого функционала

Создание API для взаимодействия с фронтендом: Разработка REST API для обработки запросов от пользовательского интерфейса.

Реализация пользовательского интерфейса: Создание форм и страниц для загрузки взаимодействия с пользователем.

**Основные требования:**

Аутентификация пользователя: Пользователь должен иметь возможность войти в систему

Авторизация пользователя: Пользователь должен иметь доступ к закрытому разделу базы данных

**Технические требования:**

Бэкенд: Python с использованием фреймворков Django.

Фронтенд: Django Rest Framework

База данных: Использование базы данных для хранения данных о пользователях

API: Разработка API для взаимодействия между фронтендом и бэкендом.

**6. Разработка в соответствии с созданной документацией**

**Планирование разработки**

Разработка была разделена на несколько основных этапов: создание базы данных,

интеграция модулей аутентификации и авторизации пользователей и их испытание.

**Разработка**

1. **Бэкенд-разработка:**

Создано два приложения **user\_app** и **profiles\_app** для работы с пользователями и редакцией их профилей соответственно.

**user\_app**

**models.py**

Создан модель User на основе класса AbstractBaseUser из встроенной базовой модели пользователя base\_user, а также класса PermissionsMixin для реализации ограничений доступа

**managers.py**

Создан менеджер пользователей на основе класса BaseUserManager из встроенной базовой модели пользователя base\_user, создающий пользователя и суперпользователя

**backends.py**

Создано представление на основе снове класса ModelBackend из встроенной базовой модели бэкенда (проверки) пользователя для его аутентификации и предоставления прав доступа

**profiles\_app**

**models.py**

Создан модель Profile на основе встроенного класса models

**signals.py**

Создан декоратор к функции создание пользователя из приложения user\_app для создания его профиля (имя, фамилия)

1. **Интеграция модулей в проект** **django\_auth\_cor**

**settings**

Добавлены следующие параметры:

в список INSTALLED\_APPS кроме приложений добавлены модули:

'djoser',

'rest\_framework'

'oauth2\_provider',

'social\_django',

'rest\_framework\_social\_oauth2',

отдельный блоки:

AUTHENTICATION\_BACKENDS = (

'django.contrib.auth.backends.ModelBackend',

'users\_app.backends.AuthBackend',)

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': (

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

'rest\_framework.authentication.BasicAuthentication',

'rest\_framework.authentication.SessionAuthentication',

'oauth2\_provider.contrib.rest\_framework.OAuth2Authentication',

'rest\_framework\_social\_oauth2.authentication.SocialAuthentication',

'rest\_framework.authentication.TokenAuthentication', ),

'DEFAULT\_RENDERER\_CLASSES': [

'rest\_framework.renderers.JSONRenderer',

'rest\_framework.renderers.BrowsableAPIRenderer',

],

}

и блок настройки SIMPLE\_JVT (не стал его расписывать. смотреть в проекте)

**urls.py**

Были прописаны пути к методам аутентификации:

**path('auth/', include('djoser.urls')),**

Обычная аутентификация на основе djoser, предлагает регистрацию в случае отсутствия таковой. При ее выборе предлагает посмотреть список пользователей из базы данных. Это будет служить нам индикатором того что пользователь аутентифицирован и авторизован - иначе список закрыт.

# path('auth2/', include('djoser.urls.jwt')),

**path('drf-auth/', include('rest\_framework.urls')),**

Аутентификация на основе сессии **Session-Based Authentication**.

**path('api/token/', TokenObtainPairView.as\_view()),**

**path('api/token/refresh/', TokenRefreshView.as\_view()),**

**path('api/token/verify/', TokenVerifyView.as\_view()),**

Аутентификация на основе сессии **JVT**

**path('oauth/', include('social\_django.urls')),**

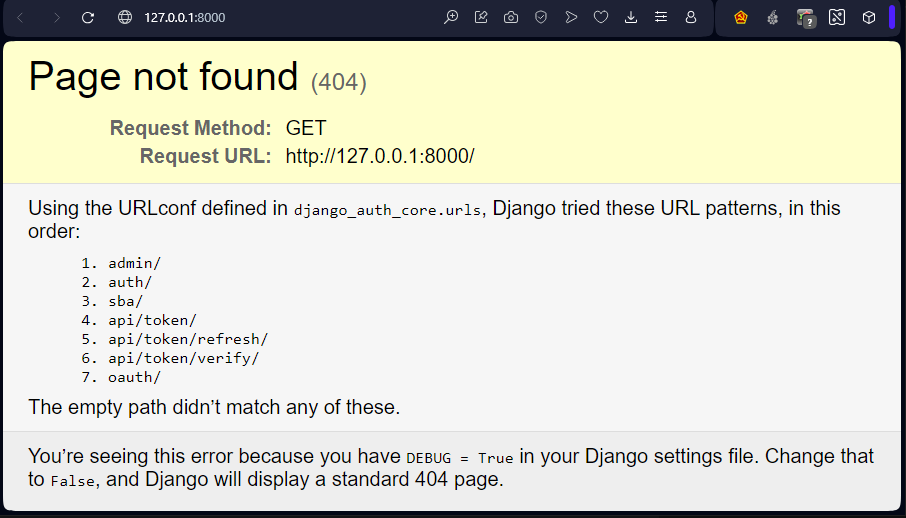
Аутентификация на основе аккаунтов соцсети **OAuth2**

**7. Запуск**

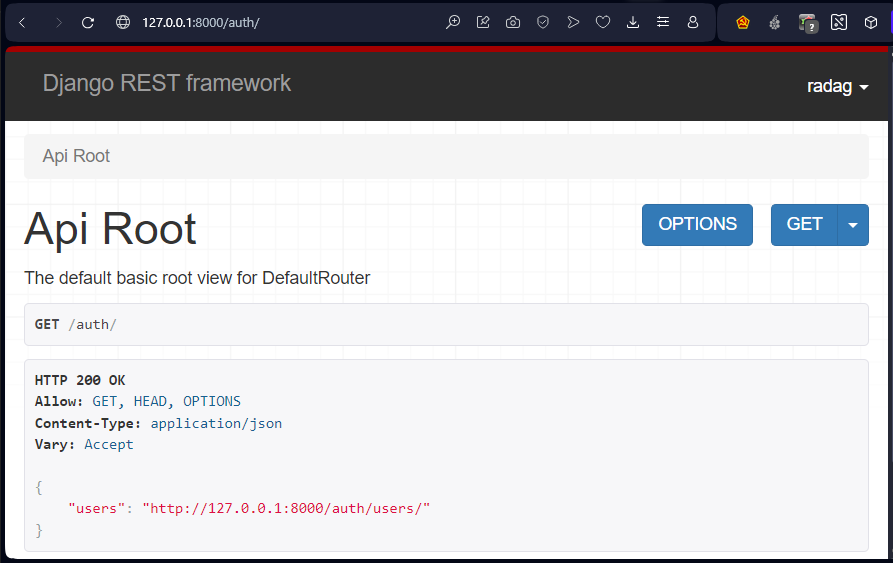
За оформление отвечает оболочка Rest Framework. Можно было бы создать шаблон страницы с кнопками для тестов, но большую часть работы все равно придется выполнять вне вэб-интерфейса. Так что, функциональность на первом месте, а оформление - по возможности.

1. **Обычная аутентификация**

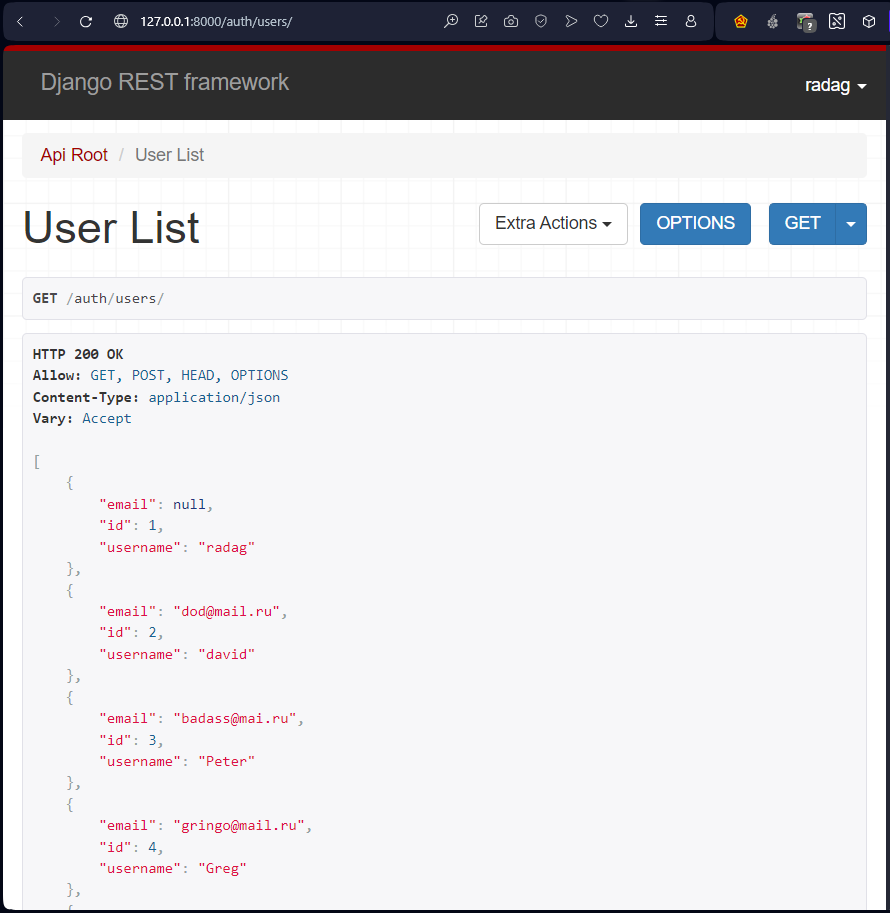
стартовая страница:



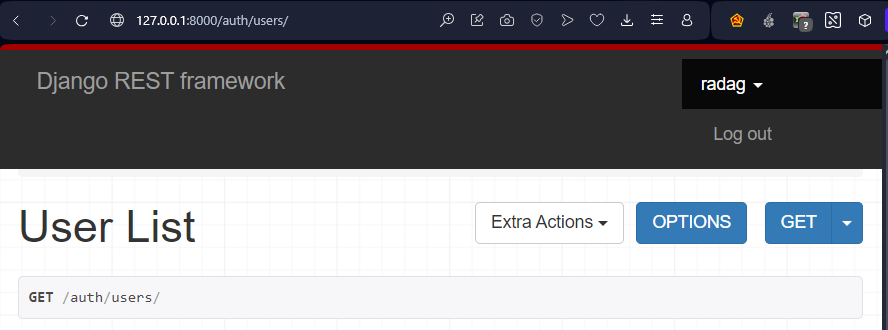
выбираем первый метод обычной аутентификации



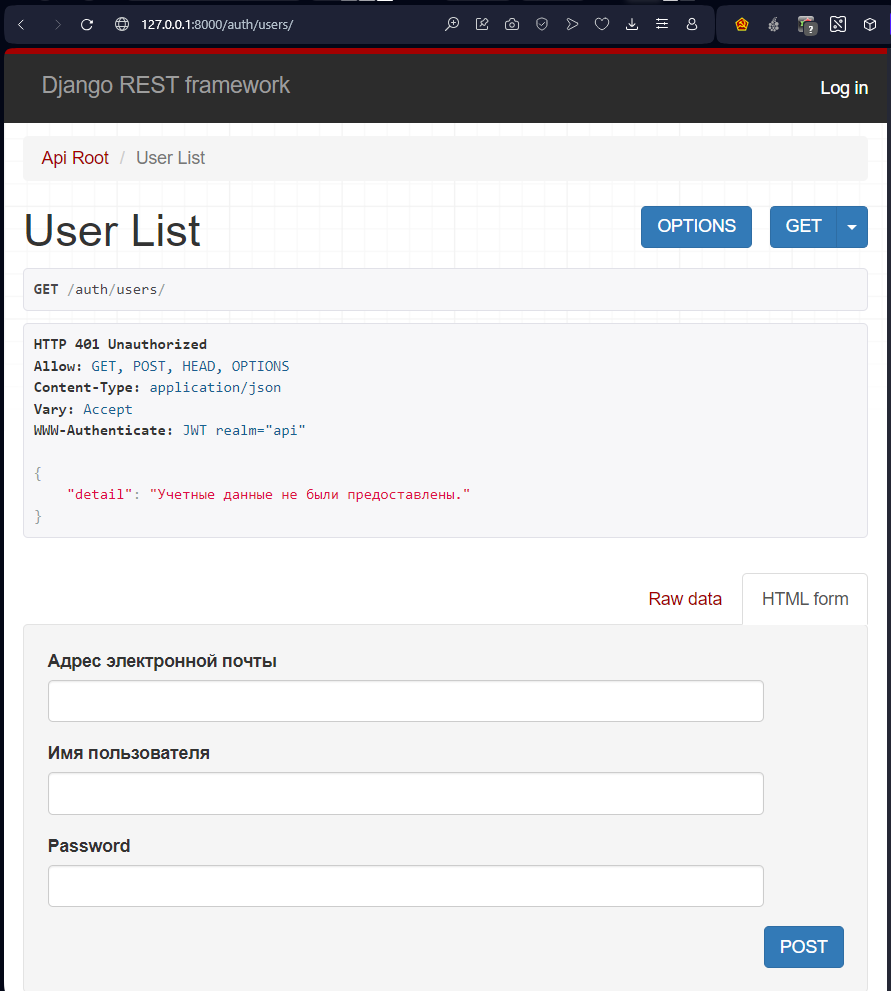
мы вошли как суперпользователь с именем radag. Переходим по предложенной ссылке



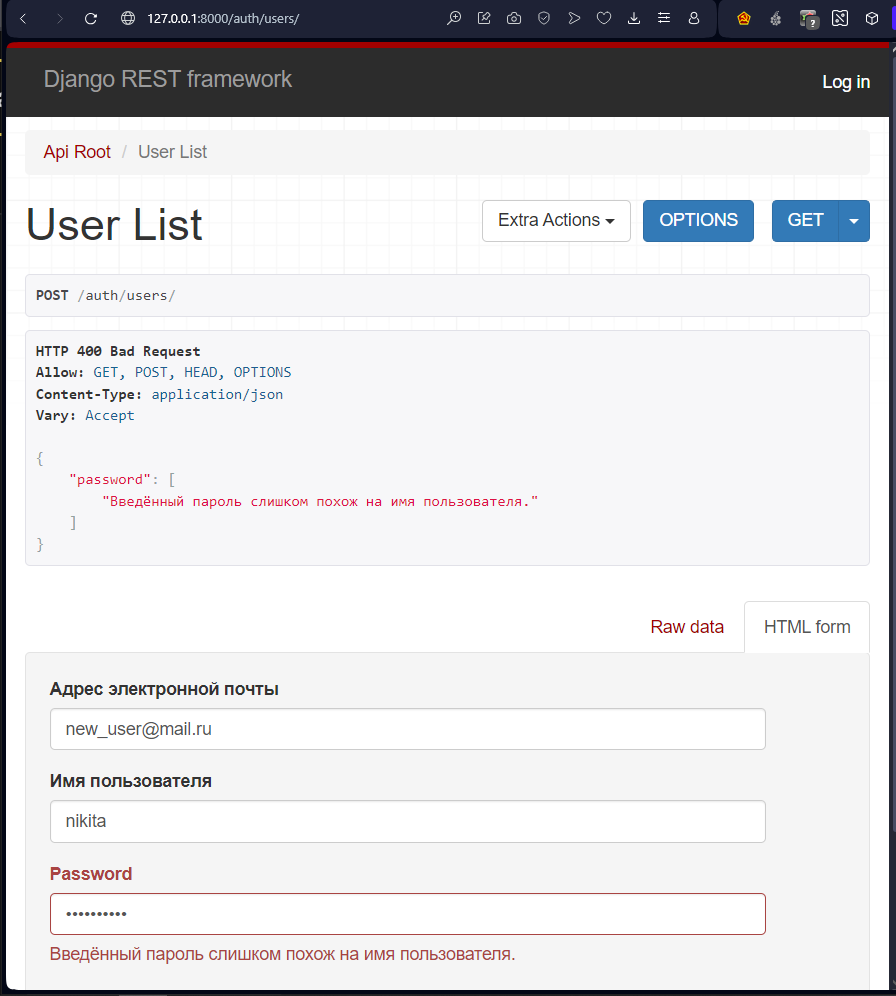
и видим, что доступ к списку пользователей имеется. Выходим из системы

****

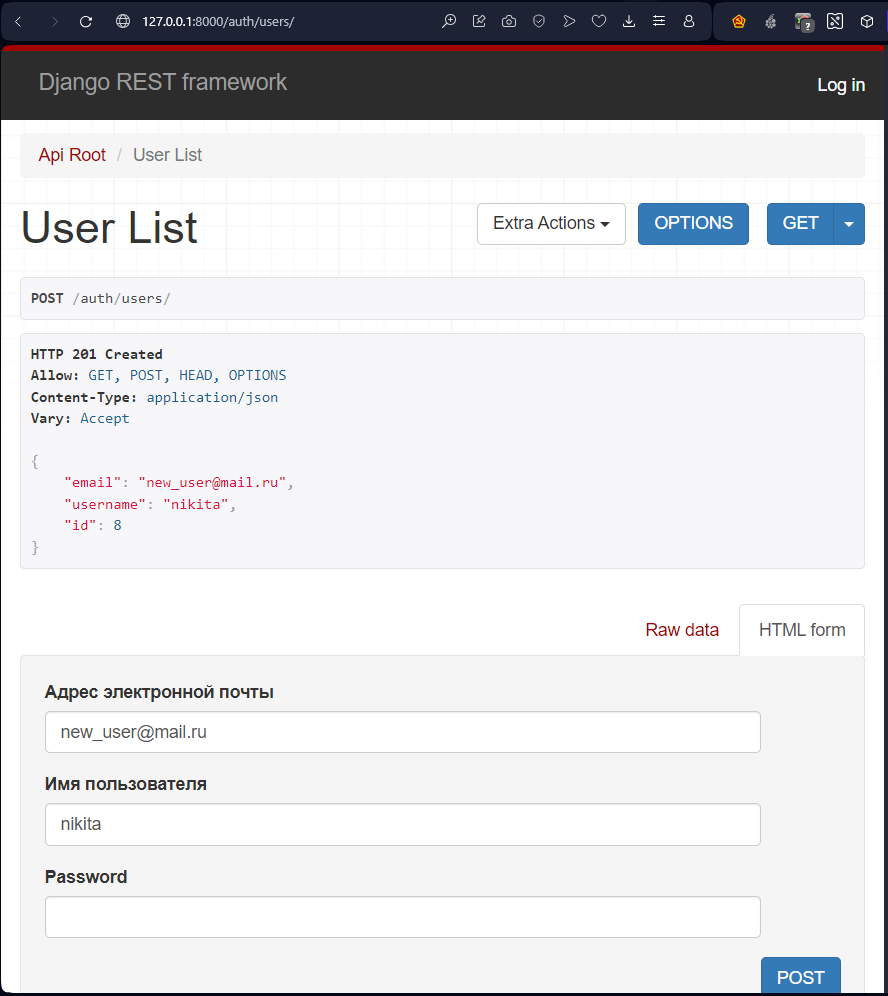
и теряем доступ к базе данных



предлагается либо вход, либо регистрация. Выбираем регистрацию



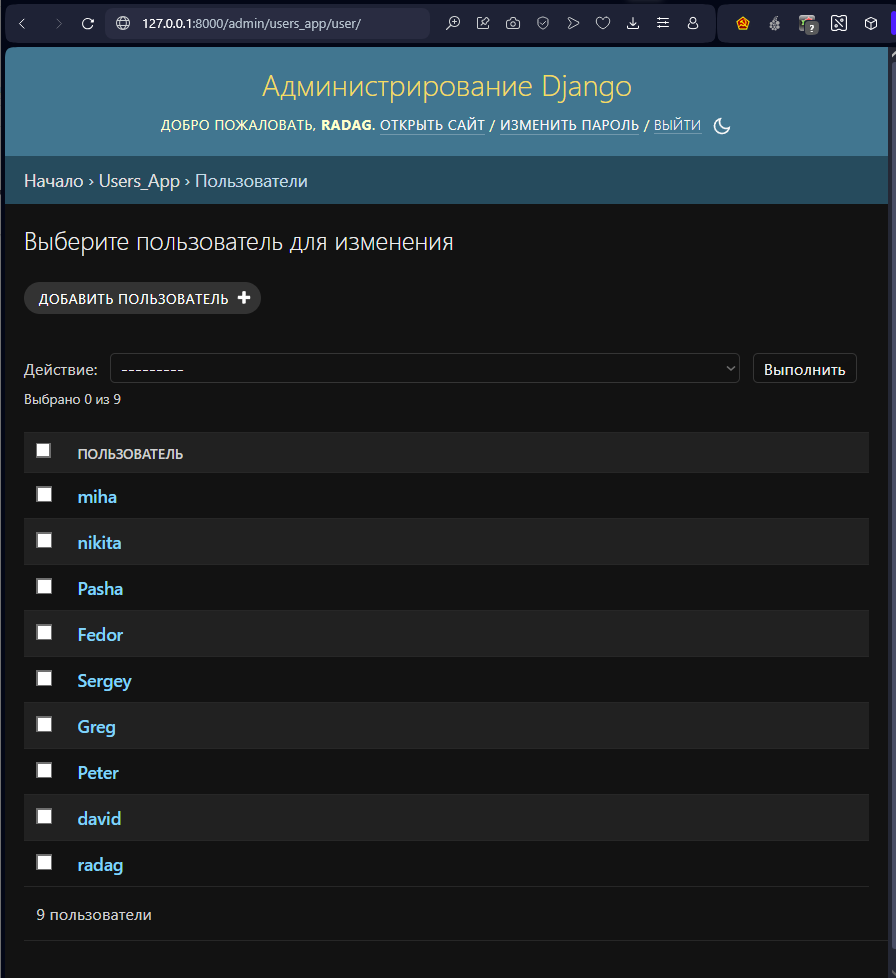
Как видим, в базовой модели пользователя вшиты проверки, которые не были описаны в коде. Что очень удобно!

****

теперь администратор должен подтвердить регистрацию нового пользователя. Идем на страницу администрирования предварительно войдя в систему как администратор



внизу страницы видим нашу базу данных users\_app. На ней видим пользователей, последние два из которых зарегистрировались недавно и ожидают окончания регистрации.



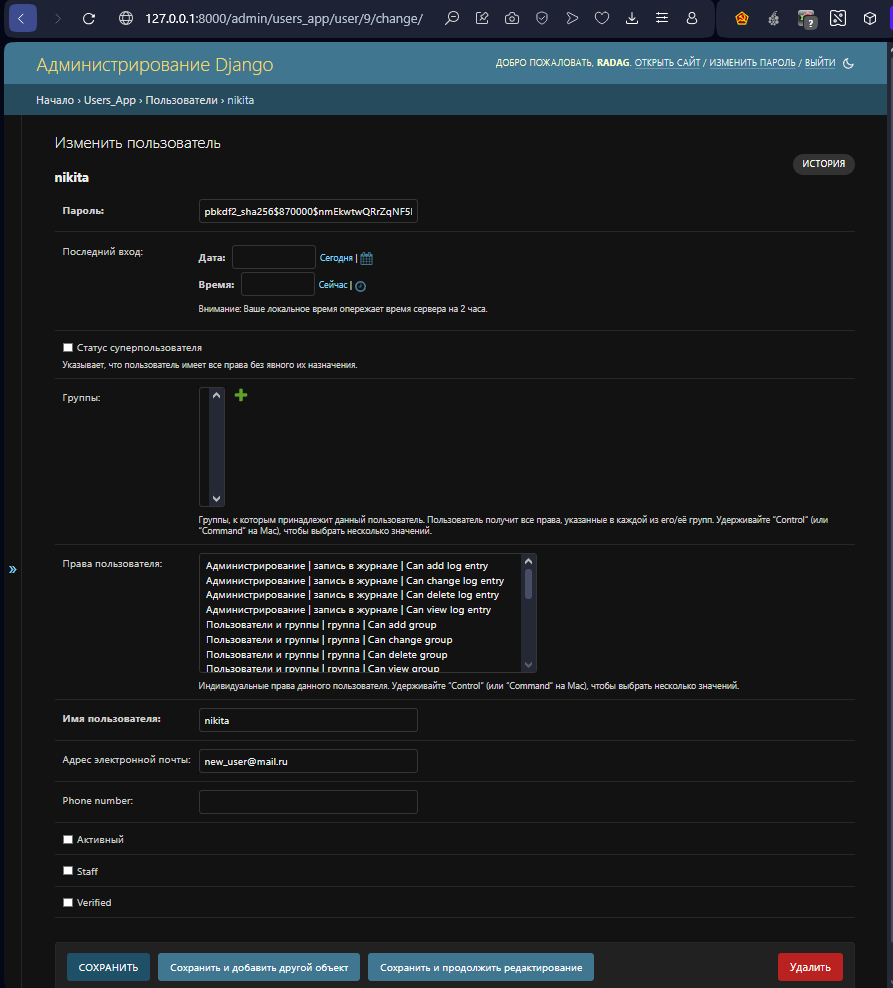
Делаем ее,активируя параметры:

Активный (Активация)

Staff (статус сотрудника)

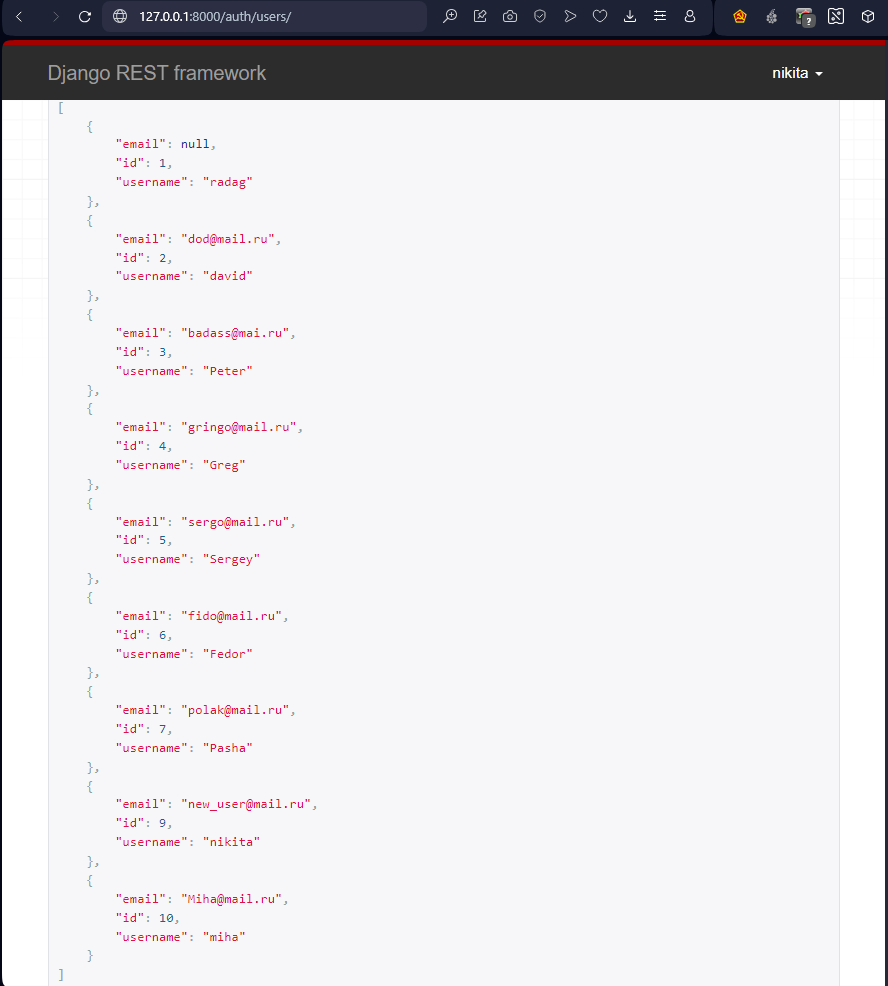
Verified (подтвержденный)

Первый параметр является обязательным для взаимодействия пользователя с системой. Остальные расширяют диапазон его действия.



“Сообщаем” новому пользователю, что он успешно зарегистрирован. И теперь заходим

в систему от его имени.



Список предоставлен и в его конце видим добавленных новых пользователей. Мы идентифицированы и успешно авторизованы.

1. **Session - Based Authentication**

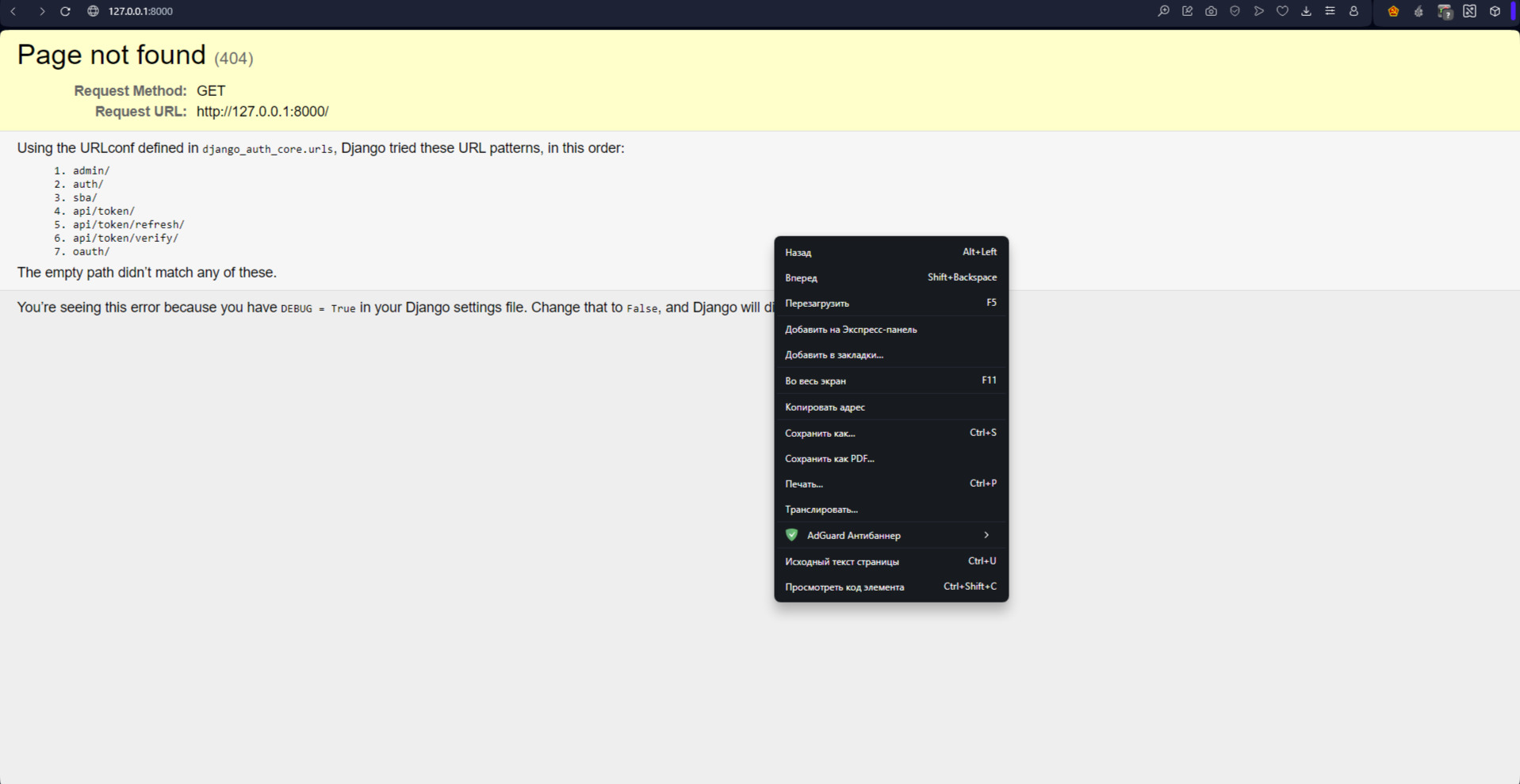
Теперь проверим как работает аутентификация на базе сессий. **(SBA)**

В двух словах, в чем идея и как это работает.

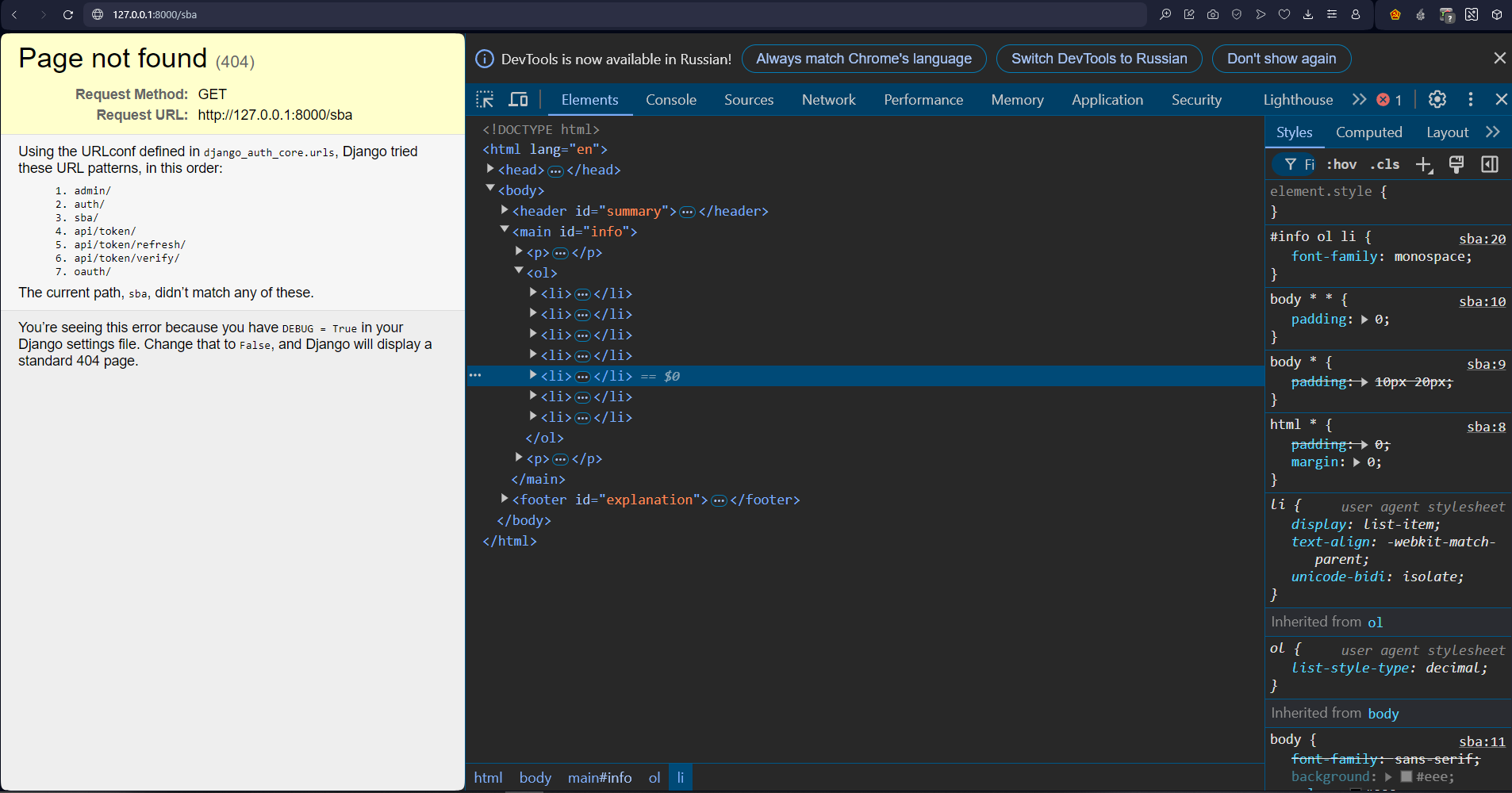
Когда пользователь входит в систему, сервер создает сессию и сохраняет соответствующие данные пользователя (например, идентификатор пользователя и роли) в памяти или в базе данных. Затем сервер отправляет уникальный идентификатор (обычно через cookie) клиенту. Этот идентификатор используется для идентификации пользователя в последующих запросах. Т.е. при обновлении страницы при случайном закрытии и открытии или открытии новой вкладки одновременно с уже открытой нет необходимости делать вход заново.

Нам надо найти это код и убедиться что он не изменился при повторном запросе.

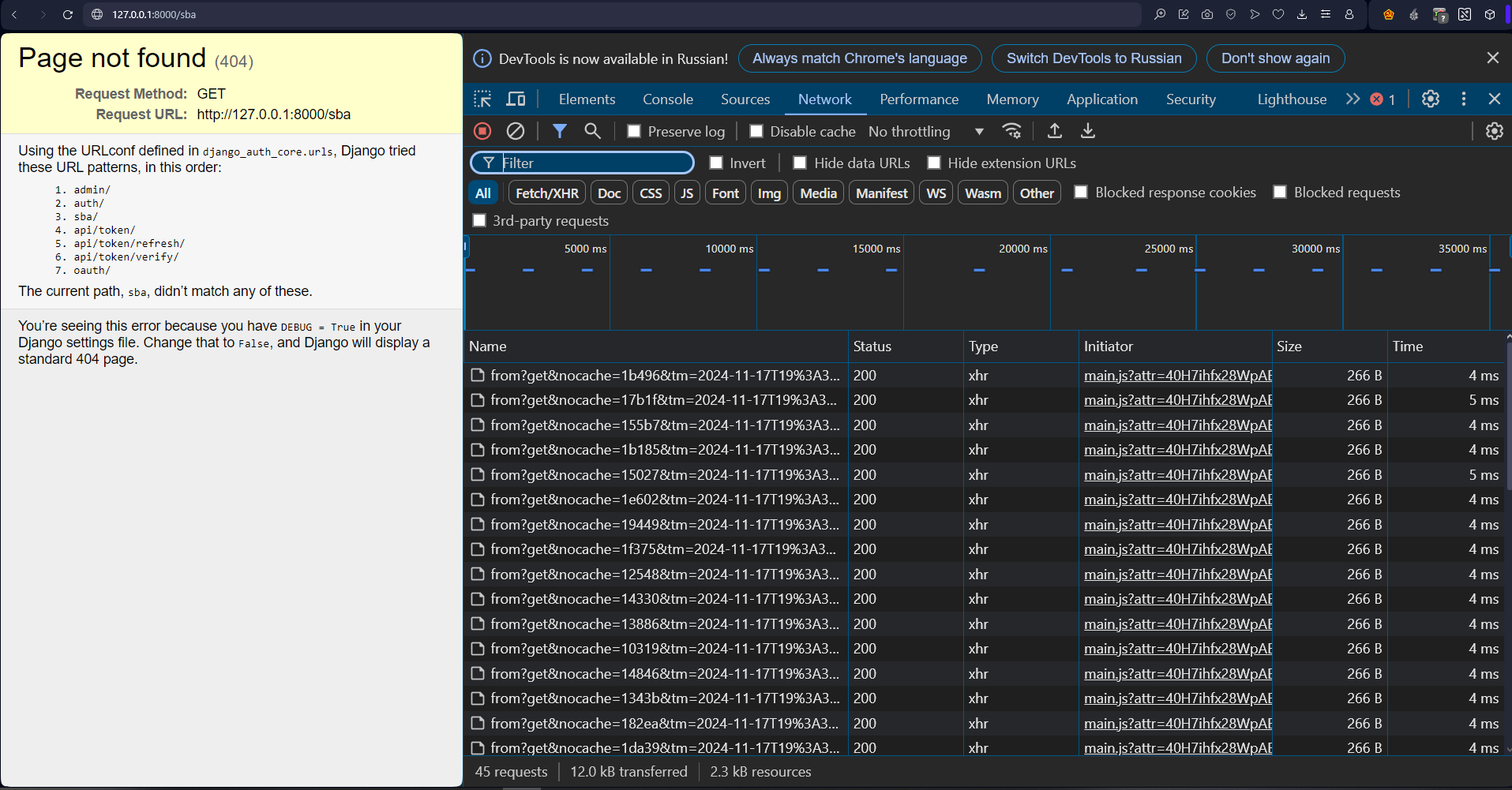
Возвращаемся на стартовую страницу . Выбираем пункт ‘sba/’. Переходим и видим, что ничего не произошло. Нам надо открыть код страницы (правой кнопкой мыши и вниз выпадающего меню)



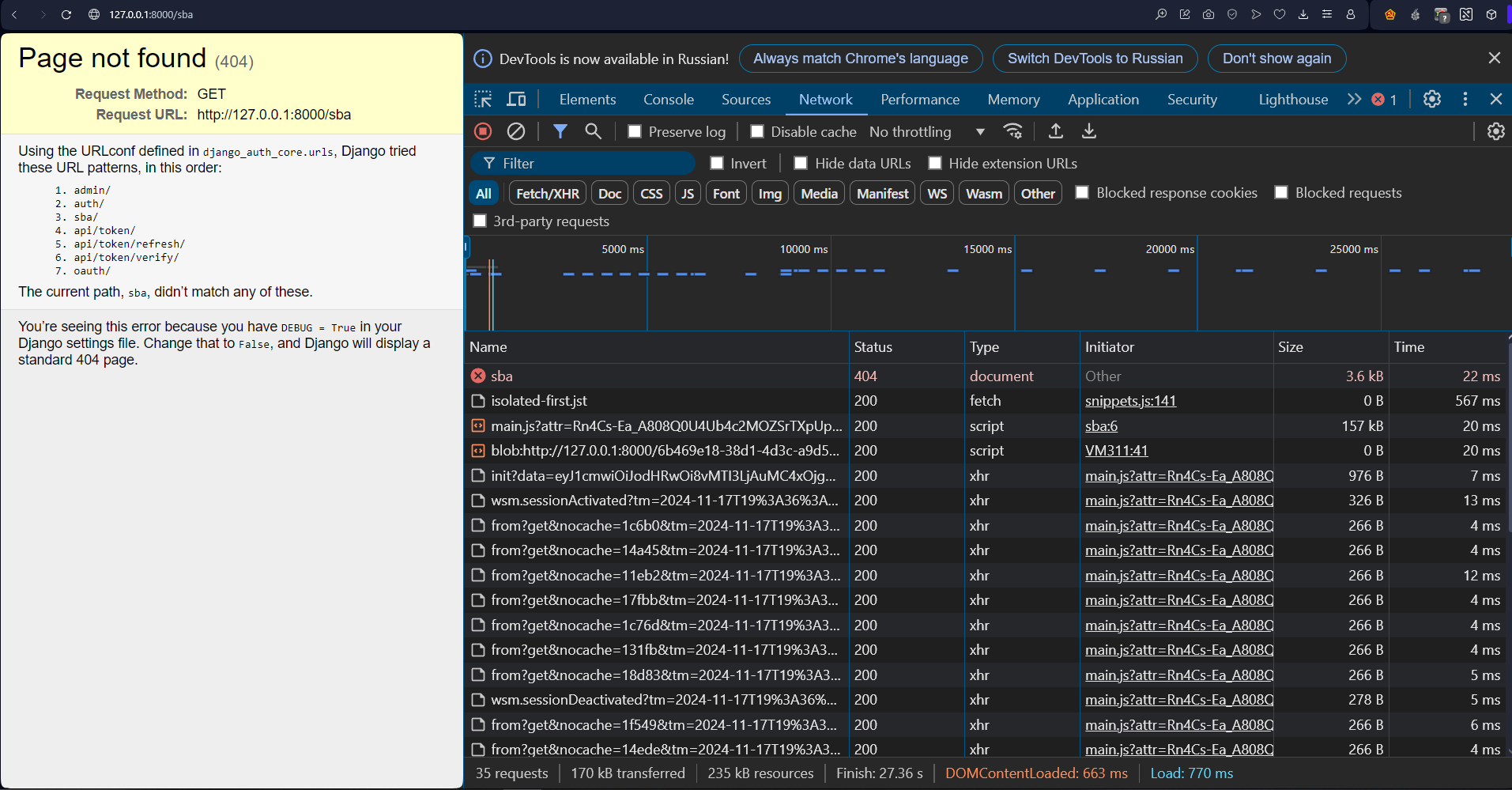
получаем

****

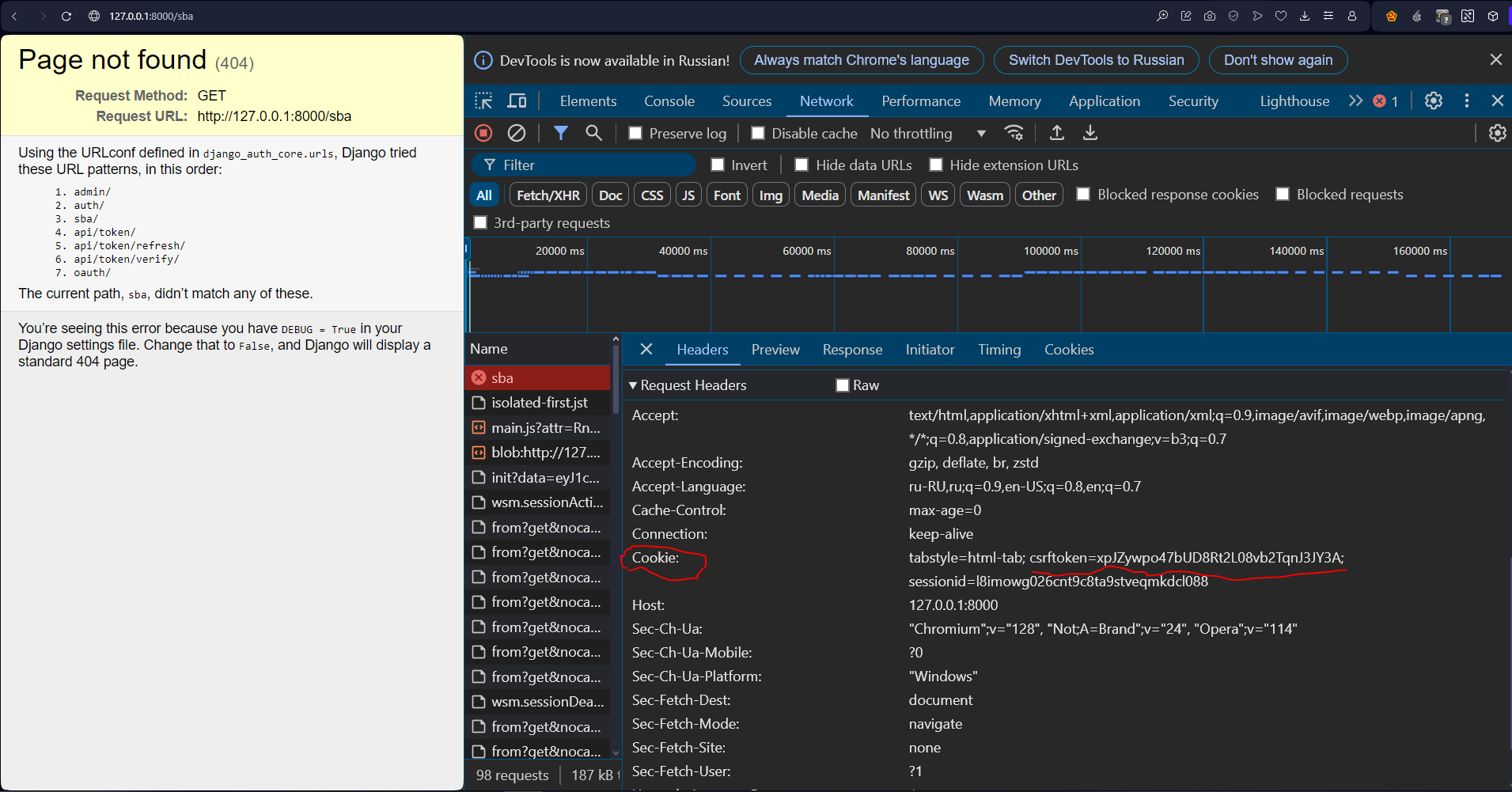
в верхней панели выбираем **Network**.



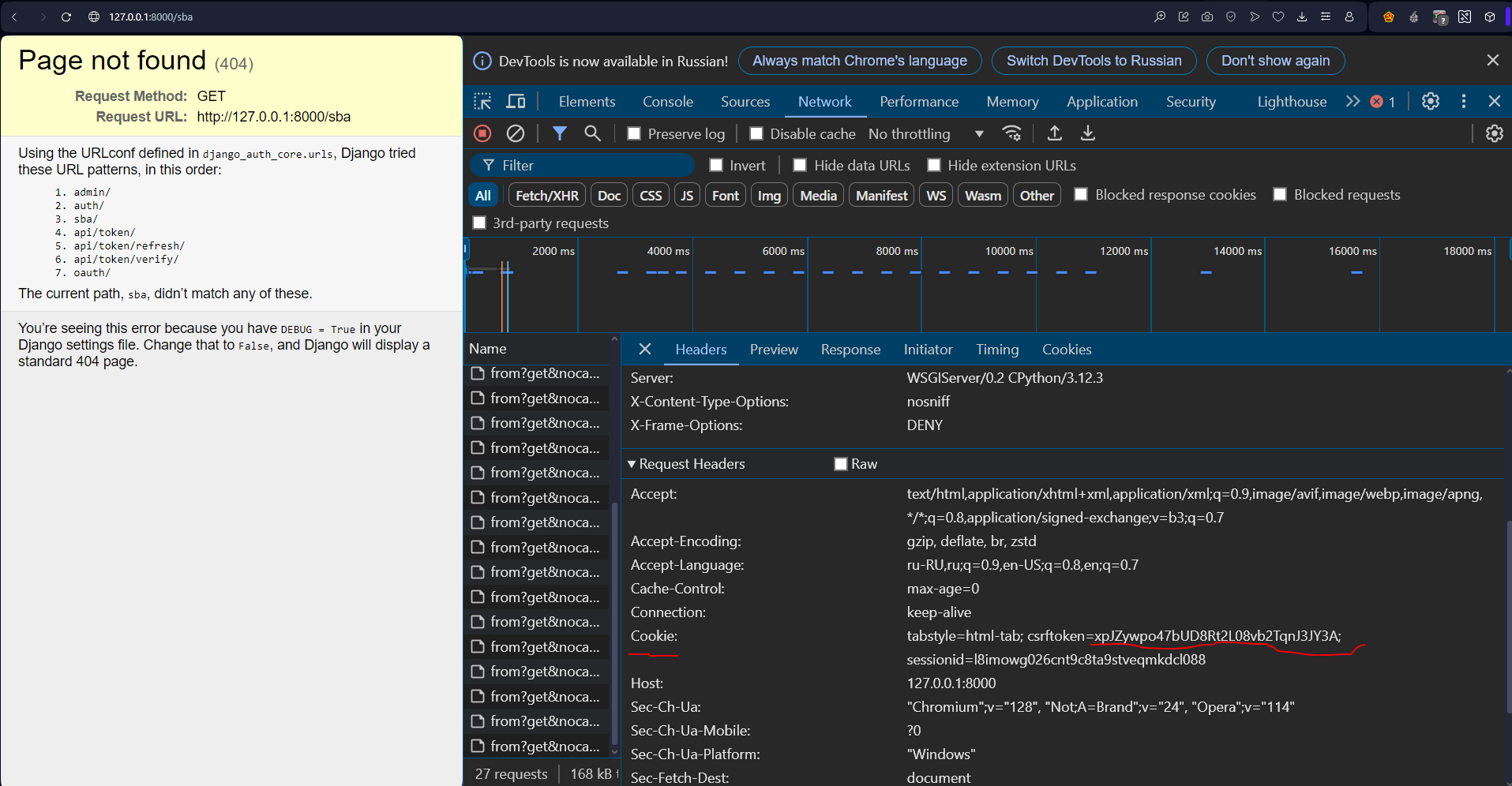
обновляем страницу и выбираем самый первый пункт - это наша аутентификация **sba**



в открывшемся окне первый параметр на панели **Headers** пролистываем до параметра **Cookie** нам нужен токен **csrftoken=xpJZywpo47bUD8Rt2L08vb2TqnJ3JY3A**



обновляем страницу, и смотрим значение Cookie. Токен не изменился



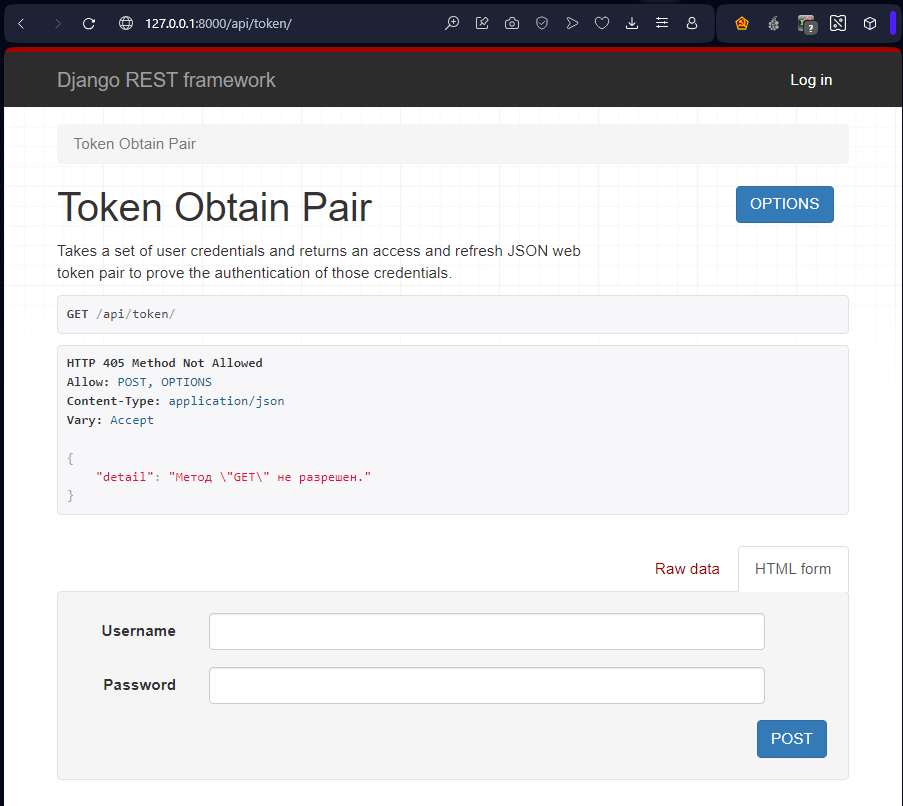
**csrftoken=xpJZywpo47bUD8Rt2L08vb2TqnJ3JY3A**

Все верно

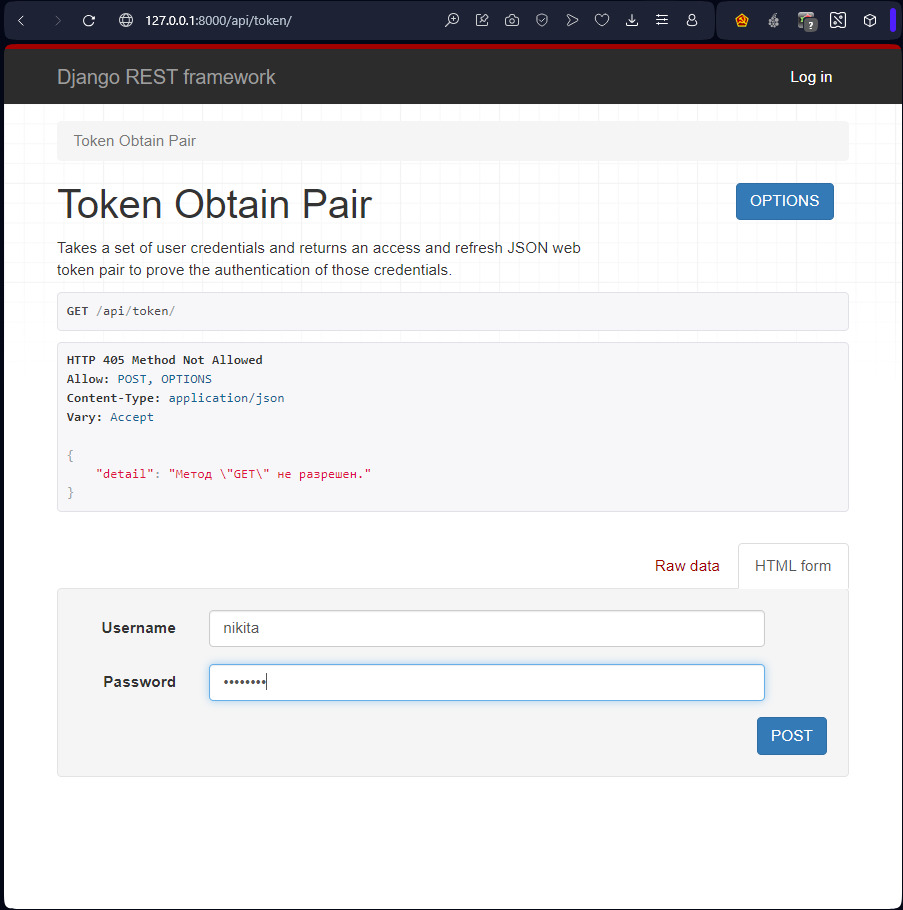
1. **Аутентификация на базе JWT токенов (JSON Web Token)**

Принцип работы такой: пользователь посылает в запросе свои данные(логин и пароль), система на их основе создает токены (2 шт). Они хранятся и в системе и у пользователя. Первый токен access token живет 5 минут. Потом он подменяется вторым токеном refresh token, срок жизни которого от нескольких часов до месяца (бывает и дольше).

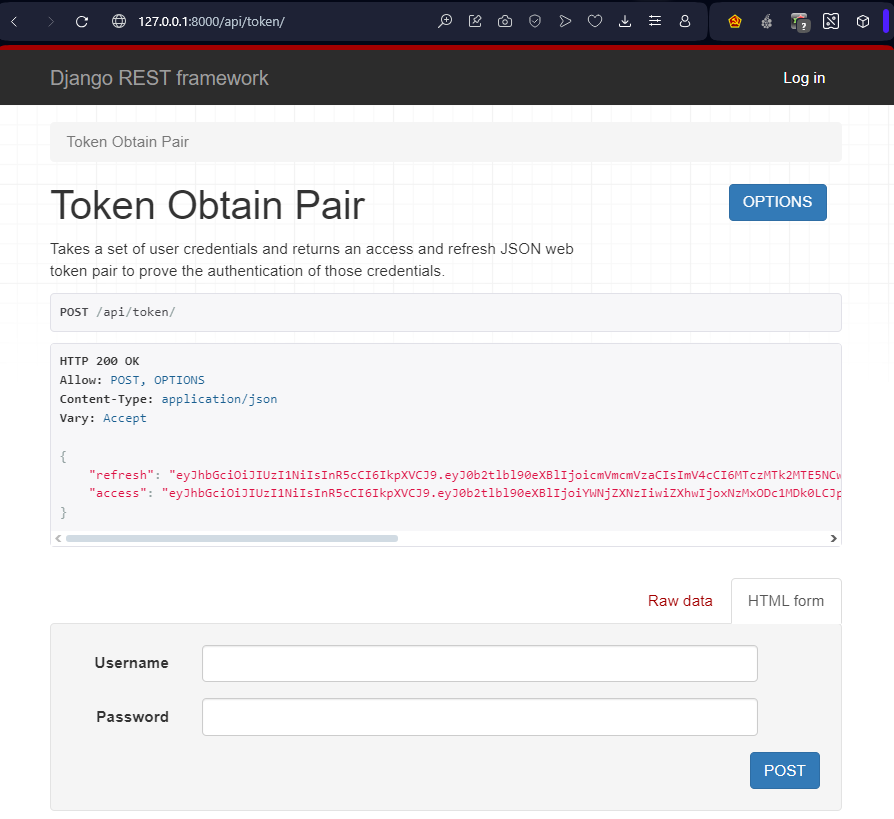
Получаем токены просто перейдя по четвертой ссылке **'api/token/’**

****

заполняем



получаем токены



копируем нижний access token, идем на сайт jwt.io, где можно рассмотреть какую информацию он содержит.

алгоритм шифрования - H256

вид токен - JWT

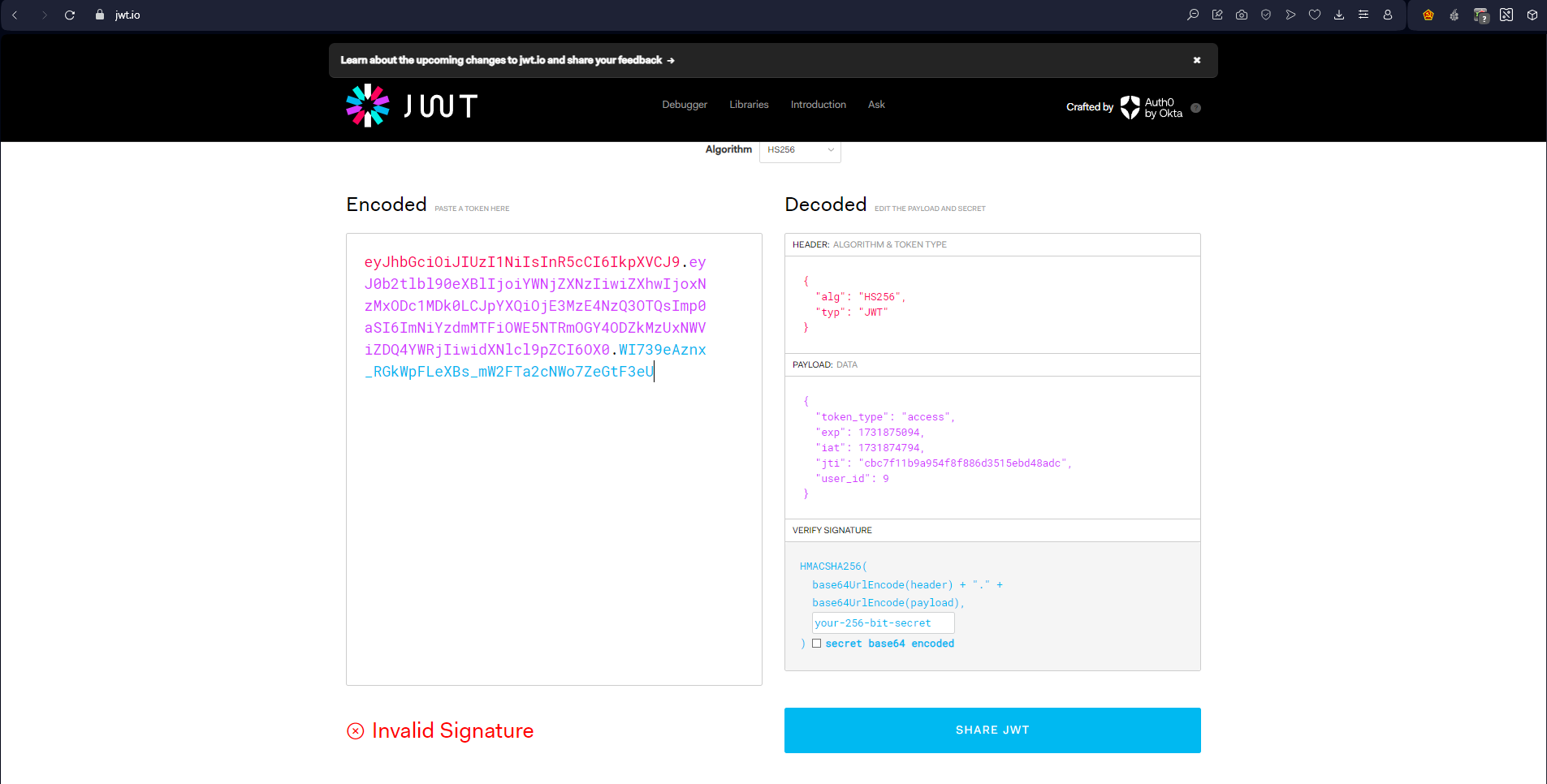
Тип токена - access

начало и конец существования в сумме 700 сек

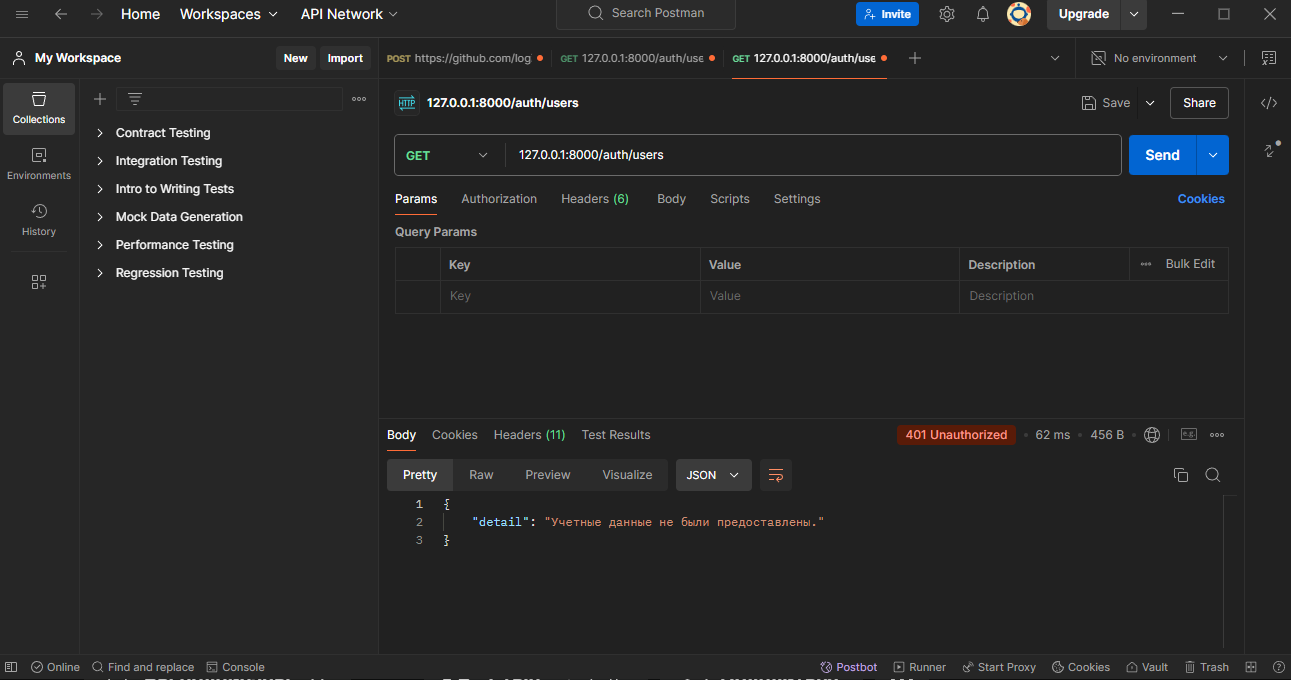
метка токена(имя) для работы с базой данных

Id пользователя этого токена - 9 ( nikita )

Теперь обратимся к программе Postmen, в которой сымитируем обращение пользователя к базе данных .



Для этого введем url базы данных users **http://127.0.0.1:8000/auth/users**/



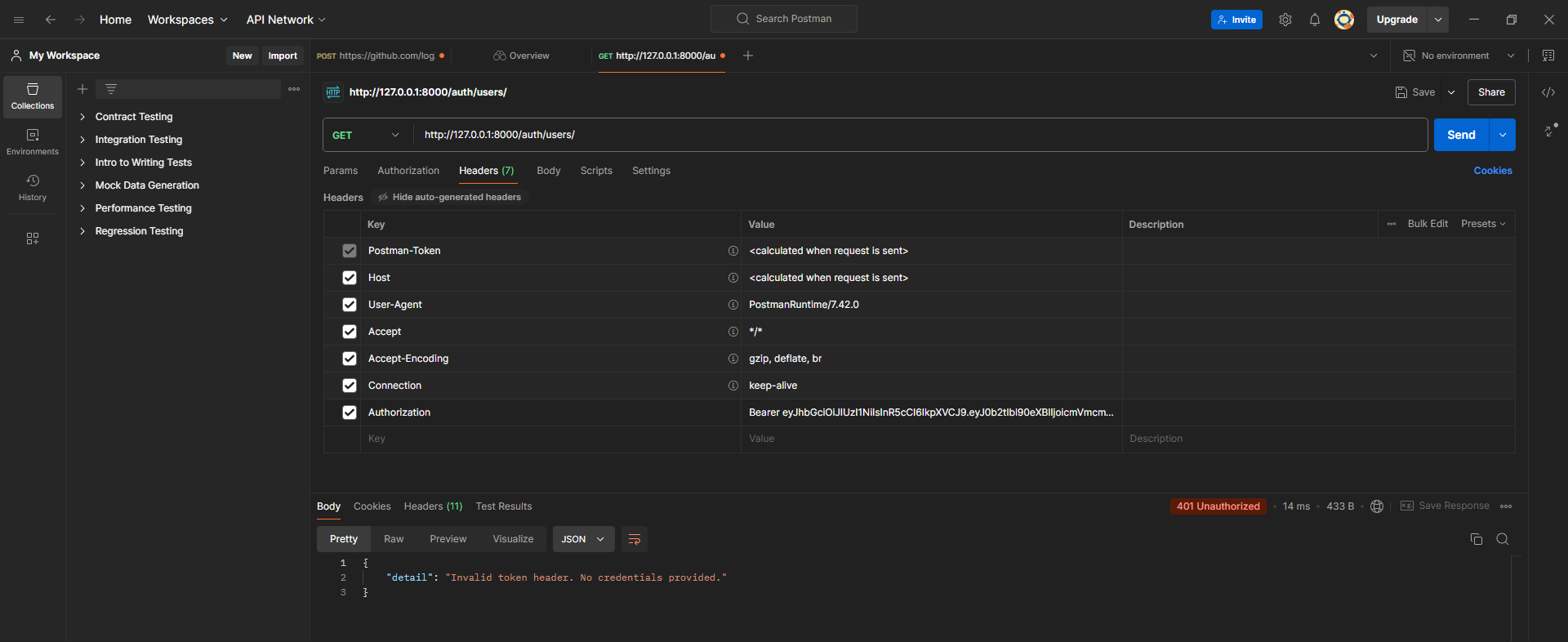
в заголовки Headers добавляем параметр Authorisation а в значение Пишем Bearer и наш токен

Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNzMxODg1ODc5LCJpYXQiOjE3MzE4ODE3OTQsImp0aSI6IjExYjgzYzZlODNlYzQ0Y2Q4NjBjMjM4YmU4ZGRhYTQzIiwidXNlcl9pZCI6MX0.hw2sB-XwXtSrklKwa6UJOVhfyMaR\_9fjKrUE8ZJNxZ4

нажимаем Send и получаем список пользователей.

(Тут у меня в момент написания текста происходит сбой в работе программы. Postmen перестал выдавать запрашиваемую информацию, сообщая: "detail": "Invalid token header. No credentials provided.".

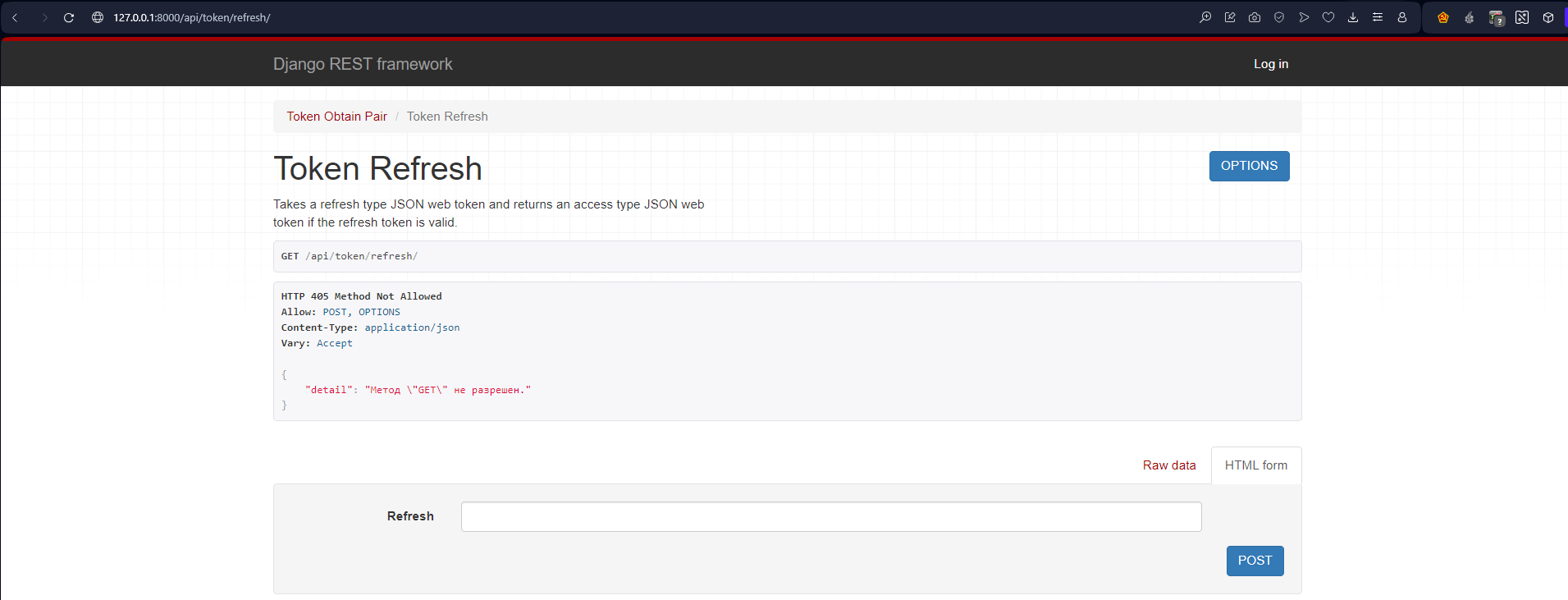
Забегая наперед, скажу что ранее эта ошибка выпадала при запросе с токеном для OAuth2 аутентификации. Последние несколько дней я никак не мог справится с ней. Но на запросы с JWT токеном все срабатывало отлично. И вот в последний вечер оформления пояснительной записки я теряю результат. Могу с полной ответственностью заявить, что не менял ничего в файле settings.py со вчерашнего дня, когда осознал, что ошибку с запросами для OAuth2 аутентификации решить уже не успею и лучше оставить все как есть пока работает.)



Вобщем, так мы получаем доступ к базе данных при помощи JWT токена

Как получить доступ по истечению срока пользования токеном?

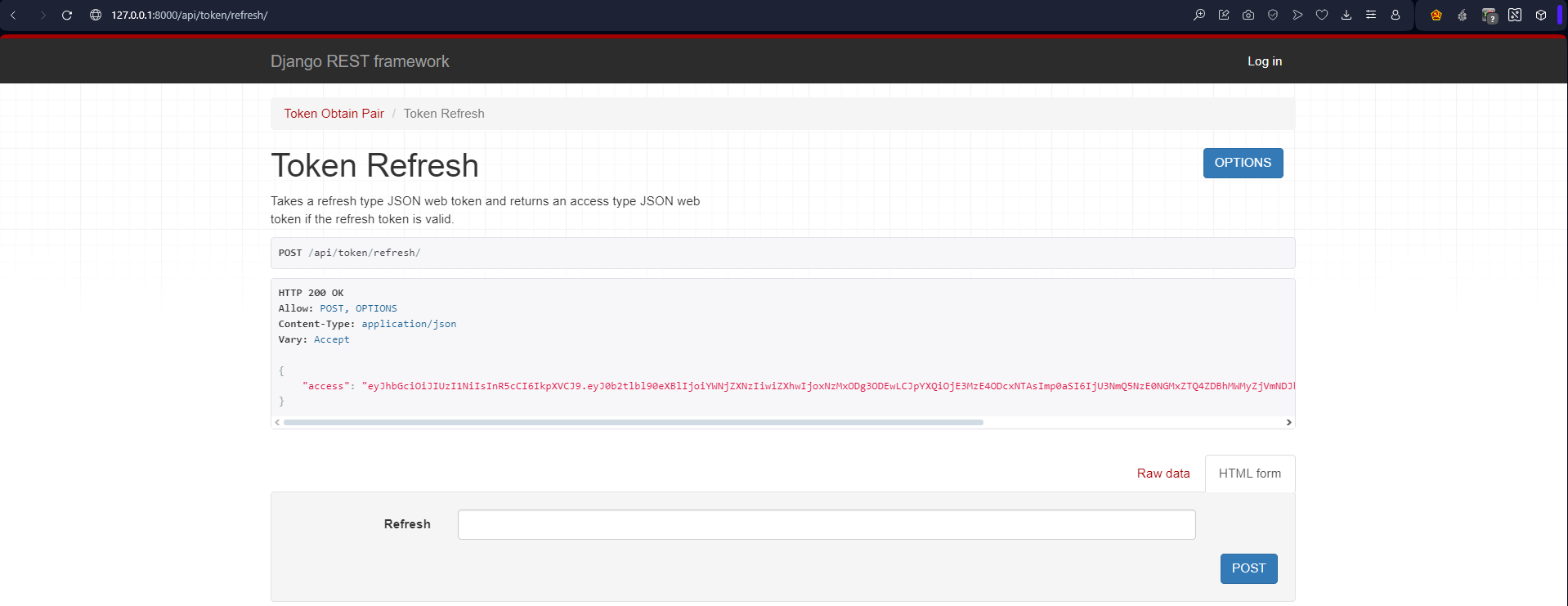
заходим уже по ссылке номер 5 **'api/token/refresh/’**



и вставляем туда второй refresh token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTczMTk3MzU1MCwiaWF0IjoxNzMxODg3MTUwLCJqdGkiOiJjMWZlMzVjMDUwNDc0MjBmYjE0NTY1MmNiNTg2MmE2YyIsInVzZXJfaWQiOjl9.MW4OK01mia0w23O8nkjnSa-MpBuL3RUZ0bR3JxDss3A

получаем обновленный access токен



1. **Аутентификация на основе аккаунтов из соцсетей OAuth2**

OAuth 2.0 — протокол авторизации, 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. Также используются токены (куда без них).

Здесь наше приложение отправляет запрос серверу социальной сети , в которой зарегистрирован пользователь. Тот в свою очередь присылает запрос пользователю подтвердить регистрацию в нашем приложении отправкой логина и пароля на основе которых создается токен для нашего сервера, он декодируется и к нам поступает данные пользователя, необходимые для регистрации.

Использовать будем учетную запись github (так как процедура описано максимально подробно в отличии от Яндекс или VK)

создаем приложение в своем профиле. **https://github.com/settings/developers**

1. Нажать на кнопку **"New OAuth App".**

Заполнить форму:

Application name: Social Auth.

Homepage URL: http://localhost:8000

Authorization callback URL: http://localhost:8000/oauth/complete/github

Нажмить "Register application".

Запомнить Client ID и Client Secret, которые будут сгенерированы для вашего приложения.

2. Формирование URL для авторизации

https://github.com/login/oauth/authorize?client\_id=YOUR\_CLIENT\_ID&redirect\_uri=YOUR\_REDIRECT\_URI&scope=YOUR\_SCOPE

Замените YOUR\_CLIENT\_ID на ваш Client ID.

Замените YOUR\_REDIRECT\_URI на URL, указанный в предыдущем шаге.

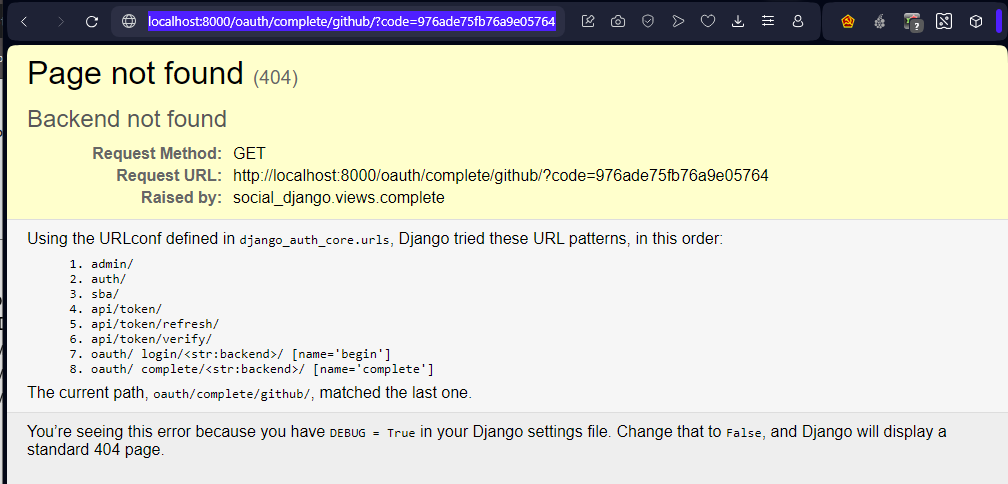
Замените YOUR\_SCOPE на необходимые разрешения (например, user, repo).

вот мой запрос:

https://github.com/login/oauth/authorize?client\_id=Ov23liA4xXaWHJ6TYcnT&redirect\_uri=http://localhost:8000/oauth/complete/github&scope=user

3. Получение кода авторизации

После успешной авторизации пользователя GitHub перенаправит его на указанный вами redirect\_uri с параметром code.



в адресной строке достаем code

http://localhost:8000/oauth/complete/github/?code=**6bb92bd59b2e7ed83f92**

4. Получение токена доступа

Используйте полученный code, чтобы запросить токен доступа. Сделайте POST-запрос к следующему URL:

в программе Postmen создаем вкладку с адресом <https://github.com/login/oauth/access_token>

создаем POST запрос в JSON фомате

{

"client\_id": "Ov23liA4xXaWHJ6TYcnT",

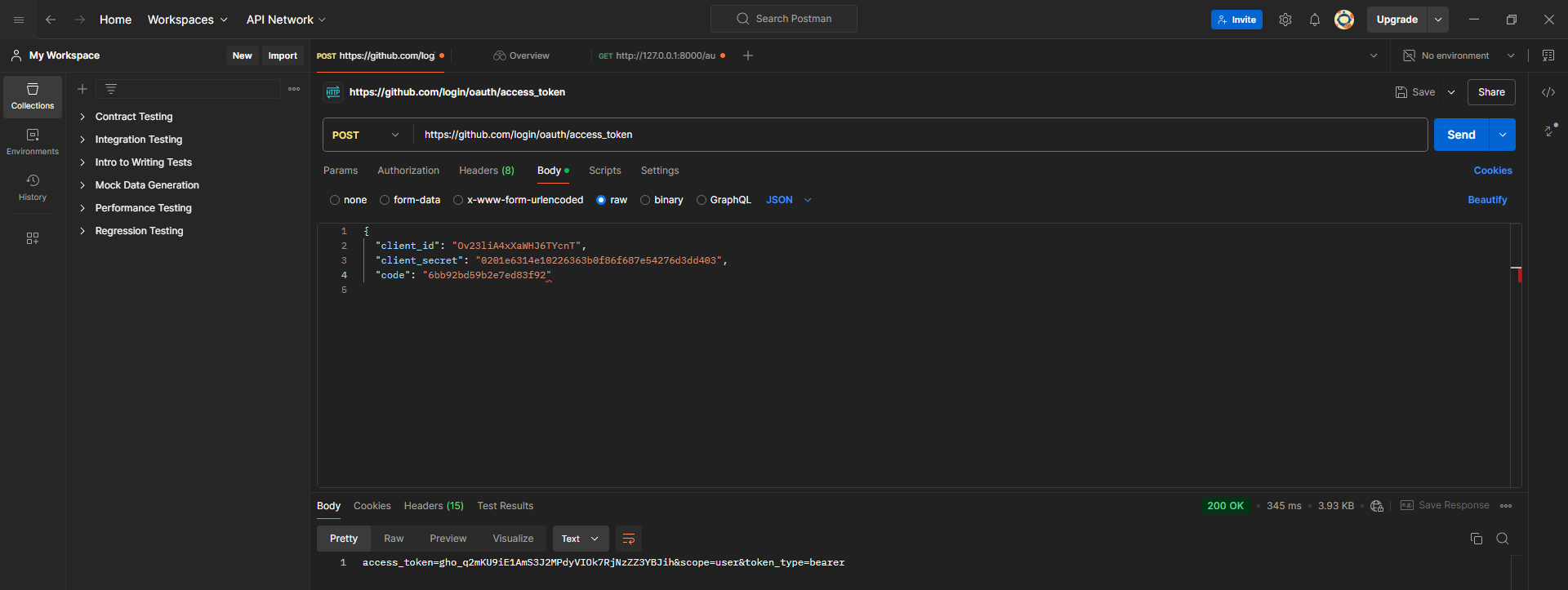
"client\_secret": "0201e6314e10226363b0f86f687e54276d3dd403",

"code": "6bb92bd59b2e7ed83f92"

}

отправляем и получаем токен доступа

access\_token=**gho\_q2mKU9iE1AmS3J2MPdyVIOk7RjNzZZ3YBJih**&scope=user&token\_type=bearer



там же создаем вкладку со знакомым адресом[**http://127.0.0.1:8000/auth/users/**](http://127.0.0.1:8000/auth/users/)

и отправляем знакомый GET запрос в **Headers** прописываем **Authorisatio**n а в значение Пишем Bearer **gho\_q2mKU9iE1AmS3J2MPdyVIOk7RjNzZZ3YBJih.**

тут уже без скрина.

Появляется доступ к базе данных и добавляется новый пользователь с ником из профиля github.

**8. Анализ и Сравнение методов**

**Session - Based Authentication**

Преимущества session-based authentication:

— сохранение состояния пользователя в течение нескольких запросов;

— простота реализации;

— возможность настройки данных сеанса в соответствии с потребностями приложения.

Недостатки session-based authentication:

— увеличение нагрузки на сервер из-за хранения данных сеанса;

— проблемы масштабируемости;

— риски безопасности.

Рекомендации по аутентификации на основе сеанса:

Используйте HTTPS: Всегда используйте безопасные подключения, чтобы предотвратить перехват сеанса.

Установите атрибуты файлов cookie: помечайте файлы cookie как HttpOnly и Secure для повышения безопасности.

Реализуйте истечение срока действия сеанса: определите соответствующие интервалы ожидания и автоматически выводите пользователей из системы после бездействия.

Повторное создание идентификатора сеанса: измените идентификатор сеанса после успешного входа в систему, чтобы защититься от атак с фиксацией сеанса.

Используйте централизованное хранилище сессий: для приложений с несколькими серверами использование общего хранилища сессий, такого как Redis, может повысить масштабируемость.

Применение: Идеален для систем состоящих из персональных компьютеров с небольшой нагрузкой (небольшие фирмы)

**Аутентификация на базе JWT токенов (JSON Web Token)**

Преимущества использования JWT в сравнении с классической схемой аутентификации, использующей сессии:

Во-первых, подход с использованием токенов позволяет не хранить информацию обо всех выданных токенах, как при классической схеме. Когда пользователь обращается к приложению, он передает ему свой токен. Приложению остается только проверить подпись и извлечь необходимые поля из полезной нагрузки.

Во-вторых, приложению вообще не обязательно заниматься выдачей и валидацией токенов самостоятельно, зачастую для этих целей используется отдельный сервис аутентификации.

В-третьих, при использовании отдельного сервиса аутентификации становится возможным организовать единую точку входа в различные сервисы с одними и теми же учетными данными. Единожды пройдя процедуру аутентификации, пользователь сможет получить доступ со своим токеном к тем ресурсам, которые доверяют этому сервису аутентификации.

В-четвертых, как было указано ранее, приложение может хранить в части полезной нагрузки практически любые данные, что при грамотной архитектуре приложения может существенно увеличить производительность.

Благодаря перечисленным факторам схема аутентификации с использованием JWT широко используется в различных корпоративных приложениях. Особенно популярна эта схема в тех приложениях, которые реализуют парадигмы микросервисной архитектуры: при таком подходе каждый сервис получает необходимые ему сведения о пользователе непосредственно из токена, а не тратит время на получение этой информации из базы данных.

Плюсы и минусы программных токенов

Как и в случае с любой методологией или техникой, при выборе программных токенов нужно учитывать их достоинства и недостатки.

Плюсы использования токенов

Эффективность — программные токены эффективны и масштабируемы. Сервер может легко создавать и проверять любое нужное количество токенов, что упрощает увеличение количества пользователей, получающих доступ к веб-сайту или веб-приложению. Важно, что они не требуют передачи пользователям физических токенов.

Гибкость — программные токены можно использовать на нескольких серверах и они могут одновременно обеспечивать аутентификацию для нескольких веб-сайтов и приложений. Они обычно используются для реализации single sign on (SSO), что удобно для пользователей и повышает безопасность.

Безопасность — токены на основе стандартов типа JWT не хранят состояния и могут проверяться только когда приватный ключ получен серверным приложением, которое использовалось для его генерации. Поэтому они считаются надёжным и безопасным способом аутентификации.

Минусы использования токенов

Скомпрометированный секретный ключ — серьёзным недостатком стандарта JWT является то, что он полагается на один ключ. Если разработчики или администраторы веб-сайта не обращаются с ключом аккуратно и он скомпрометирован нападающими, это может подвергнуть риску уязвимую информацию. Это позволит нападающим выдавать себя за пользователей и похищать пользовательские сессии, а злонамеренные действия становится сложно выявлять и сдерживать.

Излишний объём данных — размер JWT гораздо больше, чем у обычного токена сессии, и он увеличивается с объёмом сохраняемых о клиенте данных. Добавление новых данных в токен влияет на время, необходимое для установки пользовательской сессии, и в конечном итоге увеличивает время загрузки страницы.

Не подходит для аутентификации на длительный срок — системы, позволяющие пользователям оставаться залогиненными в течение долгого времени, более уязвимы. Такие токены требуют частых повторных проверок, что может раздражать пользователей. Хорошим решением этой проблемы являются токены обновления и правильное их хранение. Токены обновления позволяют пользователям оставаться аутентифицированными в течение длительного времени без повторной авторизации.

Итак, для безопасного использования JWT следует:

использовать защищенное соединение при передаче токенов;

не передавать в токенах чувствительные пользовательские данные;

ограничить время жизни JWT и использовать механизм refresh tokens;

использовать ключевые фразы большой длины;

обеспечить периодическую смену ключевой фразы;

вести на стороне приложения белый список разрешенных алгоритмов подписи;

в идеальном случае работать строго с одним алгоритмом подписи;

выбирать хорошо известные и проверенные библиотеки для работы с JWT;

всегда валидировать и санитизировать полученные от пользователя данные.

Применение: Для систем с разными устройствами и возможностью использования с нескольких устройств одним пользователем(тем кто берет работу на дом). Там где,

и учет вести надо и клиентам ассортимент показать (крупный бизнес)

**Аутентификация на основе аккаунтов из соцсетей OAuth2**

### Достоинства:

использование SSL для защиты данных,

ограничение прав доступа стороннего приложения к пользовательским ресурсам областями видимости,

обилие библиотек для реализации использования протокола во всех языках общего назначения.

### Недостатки:

различия в подходах к реализации у разных сервисов, порождающие необходимость написания отдельного кода под каждый новый сервис;

если реализована интеграция с семейством приложений, работающих в одной экосистеме (например, Google), существует риск для всех интеграций при компрометации учетных данных либо сбое на стороне сервера API;

OAuth 2.0 является новым и динамично развивающимся протоколом, в связи с чем возможны некоторые логические противоречия в его спецификации;

основанная на SSL безопасность протокола может стать проблемой в высоконагруженных проектах.

Применение: Идеально для клиентоориентированных систем (соцсети, маркетплейсы)

**9. Заключение**

**Обзор выполненной работы**

Достаточно полезный опыт изучения и испытания методов Аутентификации и Авторизации. Могу точно сказать,тема показалась интересной и я не пожалел о выборе , несмотря на неудачу с оформлением материала на финальной стадии.

Конечно же, сказать, что я изучил эту тему было бы большим преувеличением. Так. Ознакомился с основными понятиями.

Есть области, которых коснулся вскользь, но на которых просто не хватило времени. Это разрешения и ограничения доступа Permission. Сериализация вообще не была упомянута. Профили были созданы, но их использование как и Групп остались без внимания. По хорошему месяцок бы на это дело ..

**Мысли по диплому**

Увы последний метод забрал 5 дней и так и остался недоработанным. И здесь чисто по-человечески хочется выразить недовольство разработчикам фреймворков. Самые большие проблему на финальной стадии обучения и в работе над дипломным заданием возникали именно в преодолении ошибок в работе с базами данных при миграции, при создании моделей, изменении настроек в файле settings. В Python есть подсказки при неправильном оформлении, а в Django он примет написанное а ошибку покажет в своих файлах, где ничего трогать нельзя. И найти того. кто может объяснить в чем проблема можно только на форумах.На это уходит много времени. А документация написана для людей либо с большим опытом, либо получившим образование в институте. Примеров написания кода дается мало или слишком много. Одна надежда, что со временем станет легче.

# Приложение 1. Пример файловой структуры проекта

**│ db.sqlite3**

**│ manage.py**

**│ requirements. txt**

**├───django\_auth\_core**

**│ │ asgi.py**

**│ │ settings.py**

**│ │ urls.py**

**│ │ wsgi.py**

**│ │ \_\_init\_\_.py**

**│ │**

**│**

**├───profiles\_app**

**│ ├───migrations**

**│ │** | **0001\_initial.py**

| | | **0002\_initial.py**

| | | **\_\_init\_\_.py**

│ |--**\_\_init\_\_.py**

| |-**-admin.py**

| |--**apps.py**

| |--**models.py**

| |--**signals.py**

| |--**tests.py**

| |-- **viiews.py**

│

└───**users\_app**

│ ├───**migrations**

│ │ | **0001\_initial.py**

| | | \_\_**init\_\_.py**

│ |--**\_\_init\_\_.py**

| |--**admin.py**

| |--**apps.py**

| |--**backends.py**

| |--**managers.py**

| |--**models.py**

| |--**tests.py**

| |-- **viiews.py**

# 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**

**defusedxml==0.8.0rc2**

**Django==5.1.3**

**django-oauth-toolkit==3.0.1**

**django-rest-framework-social-oauth2==1.2.0**

**djangorestframework==3.15.2**

**djangorestframework-simplejwt==5.3.1**

**djoser==2.3.1**

**drf-social-oauth2==3.1.0**

**idna==3.10**

**jwcrypto==1.5.6**

**oauthlib==3.2.2**

**pycparser==2.22**

**PyJWT==2.9.0**

**python3-openid==3.2.0**

**requests==2.32.3**

**requests-oauthlib==2.0.0**

**social-auth-app-django==5.4.2**

**social-auth-core==4.5.4**

**sqlparse==0.5.1**

**typing\_extensions==4.12.2**

**tzdata==2024.2**

**urllib3==2.2.3**