

# Лекция 6

# Авторизация. Сессии

Разработка интернет приложений

Канев Антон Игоревич

# Техническое задание

- Состав **ТЗ** на итоговую систему (использовать сплошную нумерацию внутри пунктов):

## 1. цель

## 2. назначение - краткое описание для чего, кто работает в системе

## 3. задачи - 12 задач (8 лабораторных-задач по бекенду/фронтенду, нативное приложение, демо в Pages, документация, репозиторий Git) с указанием ваших технологий по каждой задаче

## 4. функциональные требования - список HTTP методов, далее список окон и какие действия для каких групп пользователей доступны. Указать, какие методы бэкенда при этом вызываются. Всего 14 подпунктов - HTTP методы, меню и 12 страниц приложения:

- гость: регистрация, аутентификация, главная, список услуг, одна услуга, 403 и 404
- создатель заявки: конструктор заявки, список заявок, личный кабинет
- модератор: список услуг таблицей, редактирование/создание услуги

## 5. требования к аппаратному обеспечению для сервера и клиента

## 6. требования к программному обеспечению с версиями для серверных компонентов и для клиента

## 1.→Цель работы¶

Реализовать систему привязки участников ВОВ к архивным документам, включающую в себя веб-сервис, веб-приложение, десктопное приложение и выделенный сервис проверки подлинности документов.¶

## 2.→Назначение¶

Система разработана для участников и архивистов архивной платформы Великой Отечественной войны. В данной платформе предусмотрен ограниченный доступ к архивным документам участников событий. Для получения разрешения на доступ к определенному документу участник обязан подать заявку через автоматизированную систему, указав выбранный документ. Система осуществляет создание, учет и отслеживание статуса заявок. Участники могут добавлять новые документы в архив, которые позже будут подтверждаться выделенным сервисом проверки подлинности. Архивисты имеют возможность принимать или отклонять заявки, а также вносить изменения в уже существующие документы и добавлять новые в архив.¶

## 3.→Задачи:¶

3.1. → Разработать дизайн приложения.¶

3.2. → Создать базу данных в PostgreSQL.¶

3.3. → Создать веб-сервис на технологии Golang 1.20.¶

3.4. → Реализовать интерфейс гостя на технологии React.¶

3.5. → Развернуть веб-приложение React на Github Pages.¶

3.6. → Добавить авторизацию и аутентификацию в веб-сервис.¶

# ТЗ. Функциональные требования

- Функциональные требования -
- Описание элементов интерфейса и доступных действий для разных ролей пользователей
- Группировка по страницам интерфейса
- При описании интерфейса либо лучше сослаться на методы, либо в методах сослаться на конкретные пункты функций

– Неавторизованному пользователю доступен только список акций.

5.1.1.4 Разработка страницы просмотра котировок и графика цены акций конкретной компании:

- На странице пользователь получает данные о компании: текущую цену акции, график и описание компании;
- Авторизованному пользователю также доступны функции покупки и продажи.

5.1.1.5 Разработка страницы добавления и редактирования компаний:

- Доступ к странице имеют только менеджеры;
- На странице можно добавлять, удалять и редактировать имеющиеся компании.

# Оформление и ГОСТ

- Документацию оформляем по ГОСТ (1.5 интервал, 1.25 см отступ, 14 пт и тд)
- Требования по оформлению, ГОСТ, **TestVKR** необходимо искать на сайте с требованиями по ВКРБ (ссылка дана в репозитории)
- Отчет по ДЗ обязательно проверить через **TestVKR**

РПЗ ВКР должна быть грамотно написана и правильно оформлена. Она должна быть распечатана на одной стороне *белого* листа бумаги формата А4 (210х297 мм) шрифтом *черного цвета Times New Roman размером 14 пунктов*, кроме фрагментов кода программ, для которых необходимо использовать шрифт *Courier New*.

При выполнении РПЗ необходимо соблюдать *равномерную плотность, контрастность и четкость изображения по всему документу*. В нем должны быть четкие, не расплывшиеся линии, буквы, цифры и знаки.

Разрешается использовать компьютерные возможности акцентирования внимания на определенных терминах, формулах, теоремах, применяя разное начертание шрифта.

Для переноса слов в тексте необходимо использовать автоматическую расстановку переносов.

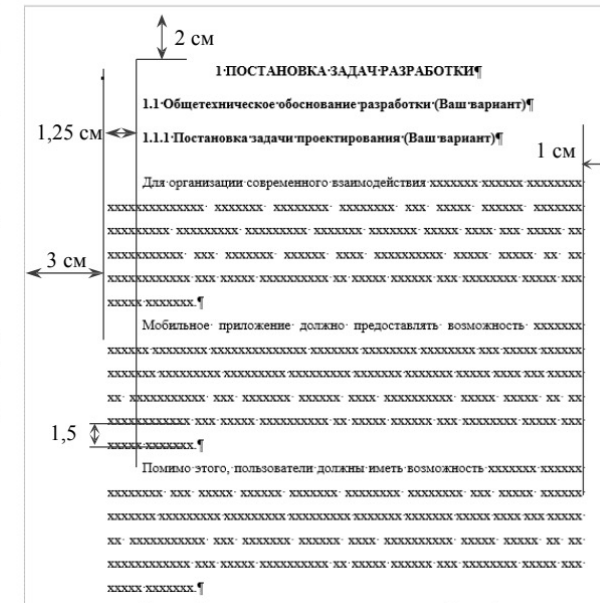
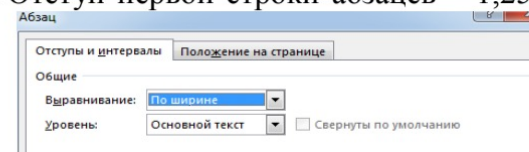
Ниже показаны примеры оформления текста РПЗ и различных структурных элементов.

В РПЗ необходимо соблюдать следующие размеры полей страницы: левое – 3 см, правое – 1 см, нижнее – 2 см, верхнее – 2 см.

Колонтитулы (верхние и нижние) во всем документе устанавливаются - 1,25.

Верхний колонтитул должен быть пустой, в нижнем должна быть только одна строка, в центре которой, симметрично тексту, должен стоять номер страницы.

Выравнивание **ВСЕГО** текста (кроме заголовков и текста в таблицах) – по ширине, без отступов и интервалов по вертикали. Междустрочный интервал – 1,5. Отступ первой строки абзацев – 1,25 см.



# Аутентификация

**Аутентифика́ция** (*authentication*) — процедура проверки подлинности, например:

- проверка подлинности пользователя путём сравнения введённого им пароля (для указанного логина) с паролем, сохранённым в базе данных пользовательских логинов;
- подтверждение подлинности электронного письма путём проверки цифровой подписи письма по открытому ключу отправителя;
- проверка контрольной суммы файла на соответствие сумме, заявленной автором этого файла.

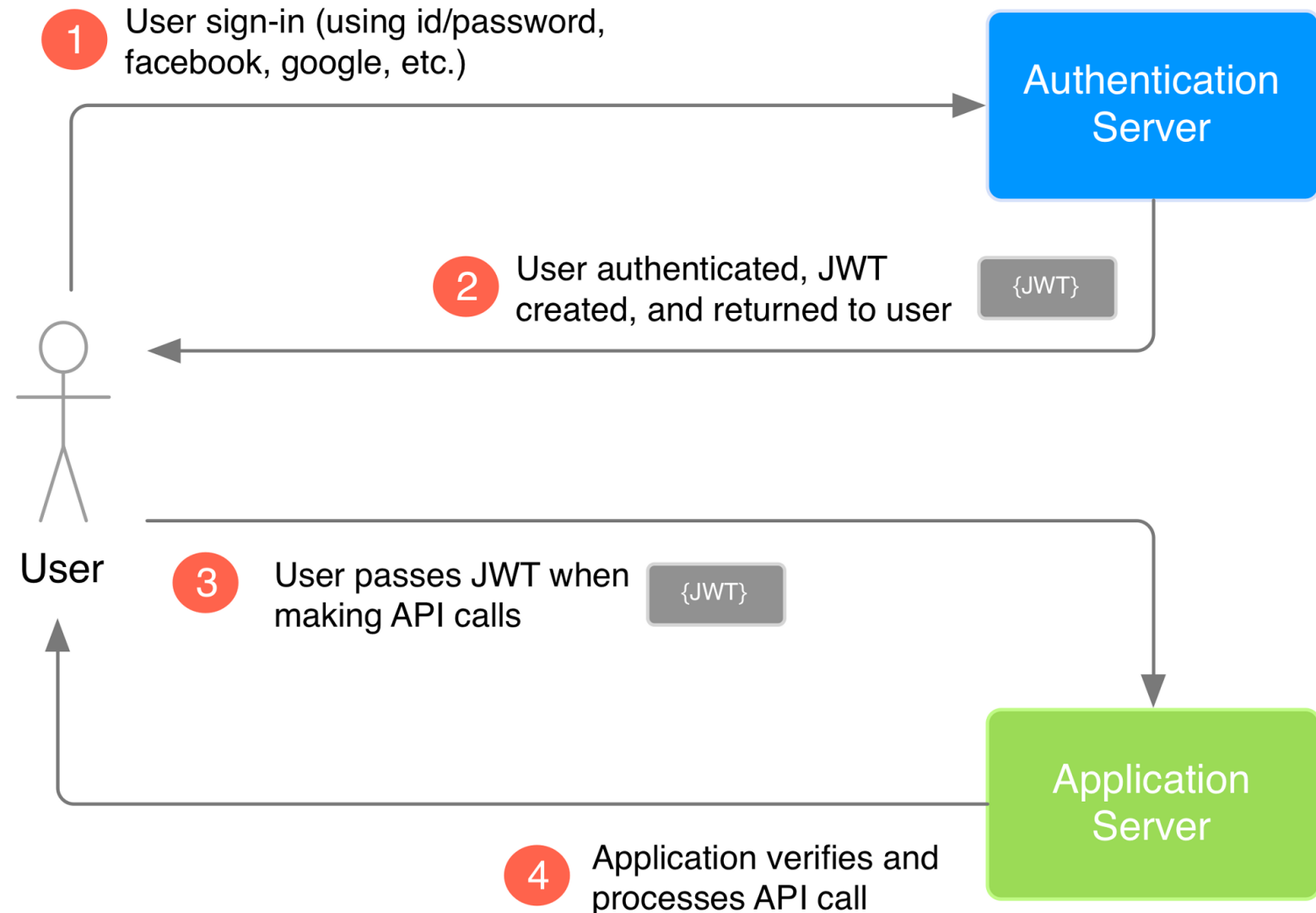
**Идентификация** — процедура, в результате выполнения которой для субъекта идентификации выявляется его идентификатор, однозначно определяющий этого субъекта в информационной системе.

# Авторизация

- **Авторизация** (*authorization* «разрешение; уполномочивание») — предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.
- Авторизация производит контроль доступа к различным ресурсам системы в процессе работы легальных пользователей после успешного прохождения ими аутентификации.

# JWT

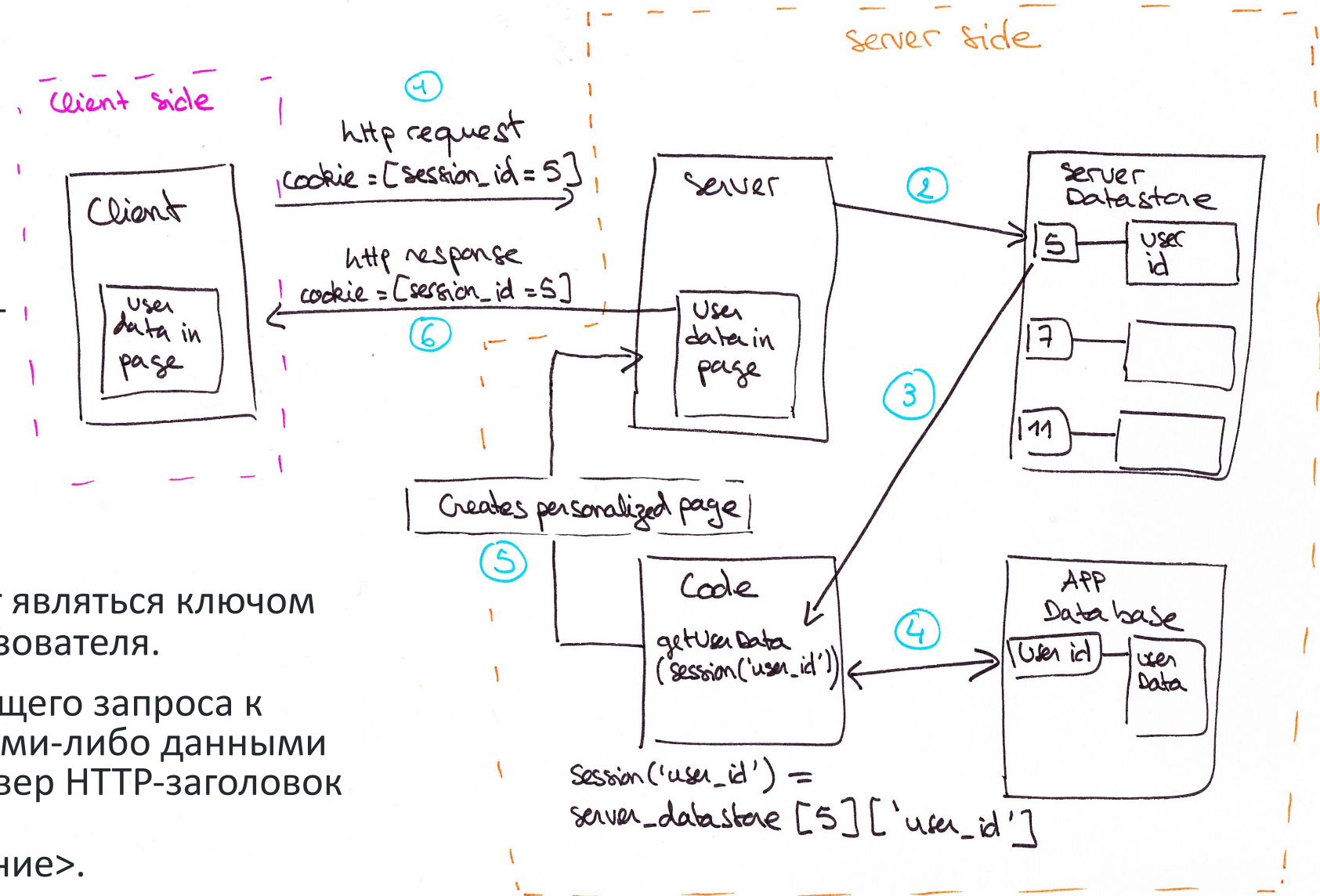
- **JSON Web Token**
- Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.
- Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения подлинности аккаунта.





# Сессии

- При авторизации на сайте сервер отправляет в ответ HTTP-заголовок Set-Cookie, чтобы сохранить куки в браузере с уникальным идентификатором сессии («session identifier»).
- Это идентификатор будет являться ключом уникальным сессии пользователя.
- Во время любого следующего запроса к этому же серверу за какими-либо данными браузер посылает на сервер HTTP-заголовок Cookie, в котором в формате <ключ>=<значение>.
- Таким образом, сервер понимает, кто сделал запрос.





# Куки

- **Ку́ки** (*cookie*, букв. — «печенье») — небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя.
- Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса.
- Применяется для сохранения данных на стороне пользователя



# Пользователи

- Концептуально сущность пользователя должна содержать его личные данные, такие данные:
- номер телефона
- почта
- ИМЯ
- никнейм
- и тд...

```
from django.contrib.auth import models as user_models
from django.contrib.auth.models import PermissionsMixin

class User(user_models.AbstractBaseUser, PermissionsMixin):
    username = models.CharField(max_length=150, unique=True)
    ...
```

The screenshot shows a registration form with the following fields and values:

- Имя: Сергей ✓
- Фамилия: Смирнов ✓
- Придумайте логин: (empty) - A red arrow points to this field, and a tooltip is visible.
- Придумайте пароль: F84gsg\$526Hf! ✓
- Повторите пароль: F84gsg\$526Hf! ✓
- Номер мобильного телефона: 8000000000

Buttons: "Получить код", "Зарегистрироваться", and "Еще 5 логинов".

**Tooltip:** Необходимо выбрать логин. Свободные логины: smirn0ws3rj, s44irnow5erg, smirn0w.s3rj, s44irnow.5erg, sergiysmirn0w.

# DRF аутентификация

- Создадим view для авторизации пользователей
- Чтобы зарегистрировать пользователя в системе используйте `login()`. Он принимает объект `HttpRequest` и объект `User`.
- `login()` сохраняет идентификатор пользователя в сессии, используя фреймворк сессий Django.

## Вход в личный кабинет

Извините, пользователь с такими логином и паролем не зарегистрирован в системе

Войти

[Забыли пароль?](#)

```
from django.contrib.auth import authenticate, login
from django.http import HttpResponse

def auth_view(request):
    username = request.POST["username"] # допустим передали username и password
    password = request.POST["password"]
    user = authenticate(request, username=username, password=password)
    if user is not None:
        login(request, user)
        return HttpResponse({"status": 'ok'})
    else:
        return HttpResponse({"status": 'error', 'error': 'login failed'})
```

# Авторизация

- Чтобы предоставить доступ пользователю в приложении и наделить его правами, мы реализуем авторизацию
- Если у нас один вид пользователей – одна роль, нам достаточно просто проверить аутентифицирован ли он

```
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework.views import APIView
```

```
class ExampleView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, format=None):
        content = {
            'status': 'Запрос разрешен'
        }
        return Response(content)
```

```
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
```

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def example_view(request, format=None):
    content = {
        'status': 'Запрос разрешен'
    }
    return Response(content)
```

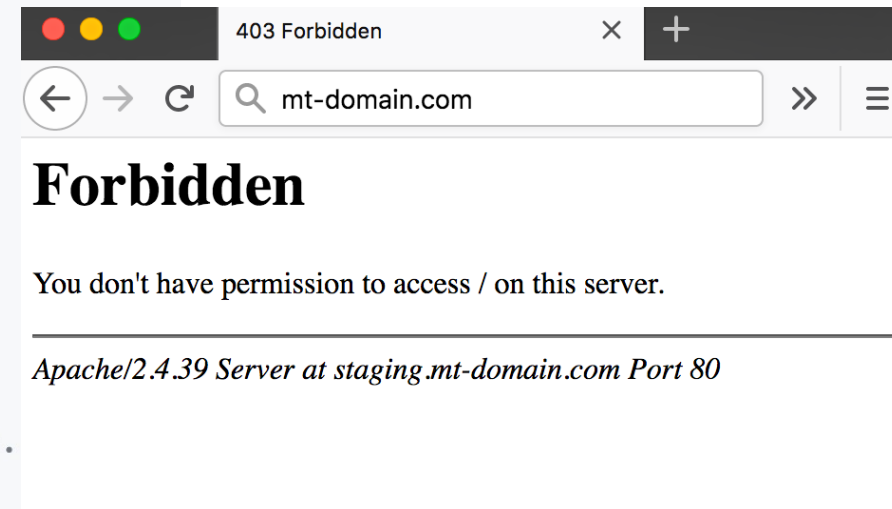
# Ограничения на бэкенде

- Чтобы ограничить неавторизованным пользователем доступ к контенту, создадим view и добавим туда authentication\_classes и permission\_classes

```
from rest_framework.authentication import SessionAuthentication, BasicAuthentication
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework.views import APIView

class ExampleView(APIView):
    authentication_classes = [SessionAuthentication, BasicAuthentication]
    permission_classes = [IsAuthenticated]

    def get(self, request, format=None):
        content = {
            'user': str(request.user), # `django.contrib.auth.User` instance.
            'auth': str(request.auth), # None
        }
        return Response(content)
```



# Ролевая модель

- Для каждого пользователя в нашей БД мы указываем его роль
- Это может быть поле в таблице пользователей, отдельная таблица или набор таблиц. Так мы можем разделить функционал по отдельным ролям
- Чтобы разделять права пользователей в приложении нам требуются написать Классы прав доступа

```
from rest_framework import permissions

class IsManager(permissions.BasePermission):
    def has_permission(self, request, view):
        return bool(request.user and (request.user.is_staff or request.user.is_superuser))

class IsAdmin(permissions.BasePermission):
    def has_permission(self, request, view):
        return bool(request.user and request.user.is_superuser)
```



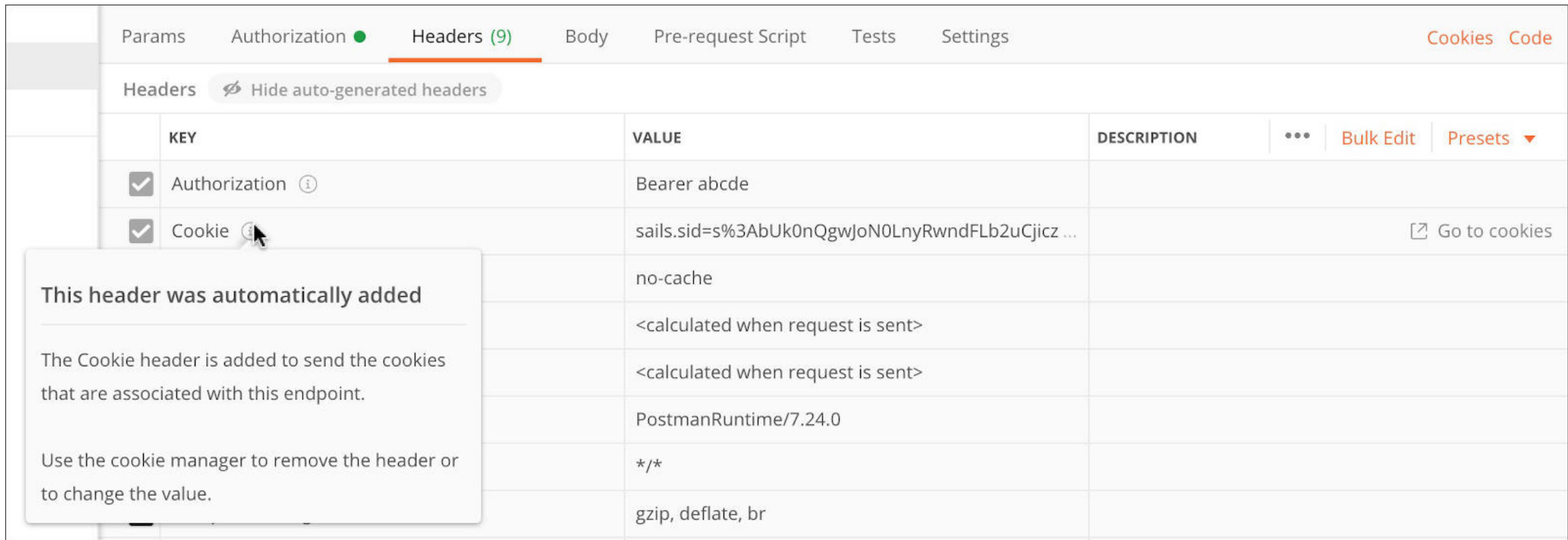
# Permissions

- Когда мы создали Классы прав доступа, их можно использовать в контроллерах
- В наших функциях указываем конкретные разрешения, которые требуются пользователям для выполнения действий

```
"""api endpoint для просмотра и редактирования списка книг"""  
(...)  
def get_permissions(self):  
    if self.action in ['list']:  
        permission_classes = [IsAuthenticatedOrReadOnly]  
    elif self.action in ['post', 'destroy']:  
        permission_classes = [IsManager]  
    else:  
        permission_classes = [IsAdmin]  
    return [permission() for permission in permission_classes]
```

# Postman

- При тестировании наши куки (или токен JWT) указываем в заголовках запроса



The screenshot shows the Postman interface with the 'Headers' tab selected. The 'Headers' section is active, and a tooltip is displayed over the 'Cookie' header entry. The tooltip contains the following text:

**This header was automatically added**

The Cookie header is added to send the cookies that are associated with this endpoint.

Use the cookie manager to remove the header or to change the value.

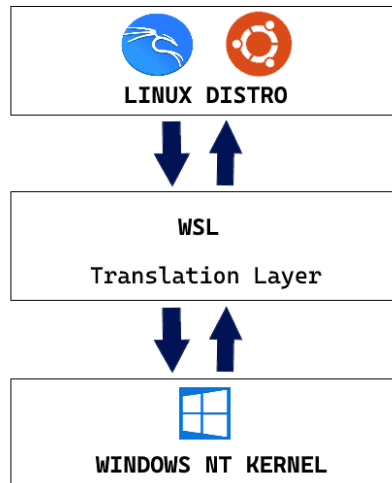
The background interface shows the following headers:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Authorization ⓘ	Bearer abcde	
<input checked="" type="checkbox"/> Cookie ⓘ	sails.sid=s%3AbUk0nQgwJoN0LnyRwndFLb2uCjicz ...	<a href="#">Go to cookies</a>
	no-cache	
	<calculated when request is sent>	
	<calculated when request is sent>	
	PostmanRuntime/7.24.0	
	*/*	
	gzip, deflate, br	

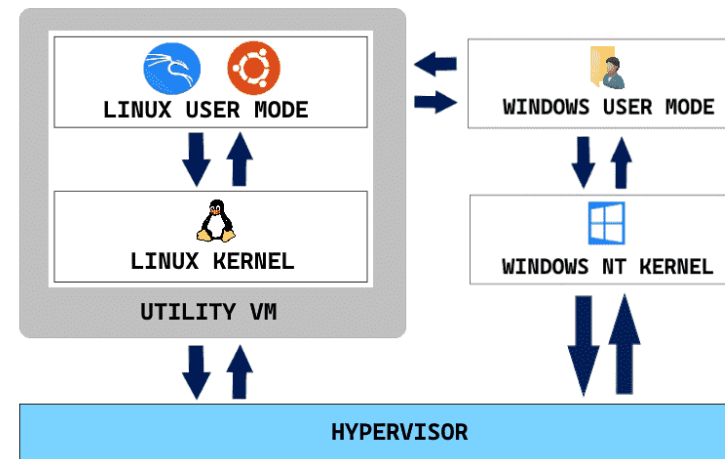
# WSL- для Redis под Windows

- **Windows Subsystem for Linux (WSL)** — слой совместимости для запуска Linux-приложений (двоичных исполняемых файлов в формате ELF) в ОС Windows

WSLv1 ARCHITECTURE



WSLv2 ARCHITECTURE



# Redis

- **RE**mote **D**ictionary **S**erver, «удалённый серверный словарь»
- Резидентная система управления базами данных
- Данные размещаются в оперативной памяти
- Механизмы снимков на дисках для постоянного хранения
- Структура данных ключ-значение, словаря
- Максимальная производительность на атомарных операциях
- Механизм подписок не гарантирует, что сообщение будет доставлено

# Redis. Отличия от реляционных

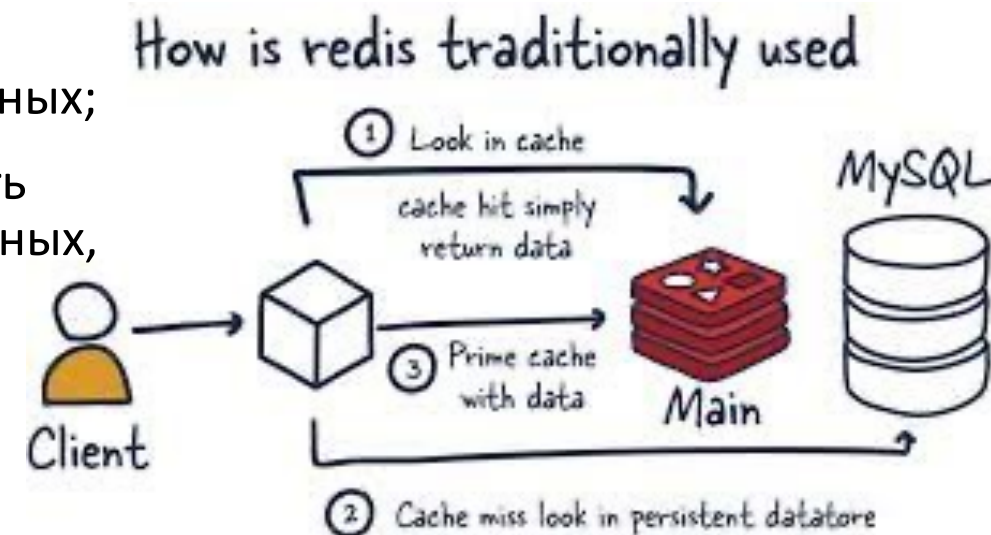
От реляционных баз Redis отличается:

- **более высокой производительностью** (благодаря хранению данных в оперативной памяти сервера, значительно увеличивается число выполняемых операций);
- **отсутствием языка SQL** (Lua-скрипты как альтернатива);
- **гибкостью** (данные находятся не в жёстких структурах (таблицах), а в более удобных (строки, списки, хеши, множества, сортированные множества), что облегчает работу программисту;
- **лучшей масштабируемостью.**

Однако Redis редко используется как основное хранилище в крупных системах, так как не удовлетворяет требованиям ACID, то есть не обеспечивает 100%-ной целостности данных.

# Redis. Применение

- для хранения пользовательских сессий (HTML-фрагменты веб-страниц или товары корзины интернет-магазина);
- для хранения промежуточных данных (поток сообщений на стене, голосовалки, таблицы результатов);
- как брокер сообщений (стратегия «издатель-подписчик» позволяет создавать новостные ленты, групповые чаты);
- как СУБД для небольших приложений, блогов;
- для кэширования данных из основного хранилища, что значительно снижает нагрузку на реляционную базу данных;
- для хранения «быстрых» данных — когда важны скорость и критичны задержки передачи (аналитика и анализ данных, финансовые и торговые сервисы).





# Redis. Пример

- HSET — сохраняет значение по ключу
- создали объект person1 с двумя полями (name и age) и соответствующими значениями.

```
127.0.0.1:6379> HSET person1 name "Aleksey"  
(integer) 1  
127.0.0.1:6379> HSET person1 age 25  
(integer) 1
```

# Redis. Пример

- HGET — получение значения по ключу (для определённого поля)
- Получили значение поля name у ключа person1

```
127.0.0.1:6379> HGET person1 name  
"Aleksey"
```

# Redis

- Установить

```
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyrings/redis-archive-keyring.gpg  
echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis.io/deb $(lsb_release  
sudo apt-get update  
sudo apt-get install redis
```

- Запустить

```
sudo service redis-server start
```

- Использовать

```
redis-cli  
127.0.0.1:6379> ping  
PONG
```

# Redis с Django

- Зайдем в файл settings.py и пропишем туда сокет запущенной БД:

```
REDIS_HOST = 'localhost'  
REDIS_PORT = 6379
```

- Далее создадим библиотечный инстанс нашего хранилища сессий в файле views.py:

```
from django.conf import settings  
import redis  
  
# Connect to our Redis instance  
session_storage = redis.StrictRedis(host=settings.REDIS_HOST, port=settings.REDIS_PORT)
```

# Аутентификация с Redis

```
from django.contrib.auth import authenticate, login
from django.http import HttpResponse
import uuid

def auth_view(request):
    username = request.POST["username"] # допустим передали username и password
    password = request.POST["password"]
    user = authenticate(request, username=username, password=password)
    if user is not None:
        random_key = uuid.uuid4()
        session_storage.set(random_key, username)

        response = HttpResponse({"status": 'ok'})
        response.set_cookie("session_id", random_key) # пусть ключем для куки будет session_id
        return response
    else:
        return HttpResponse({"status": 'error', 'error': 'login failed'})
```

# Авторизация с Redis

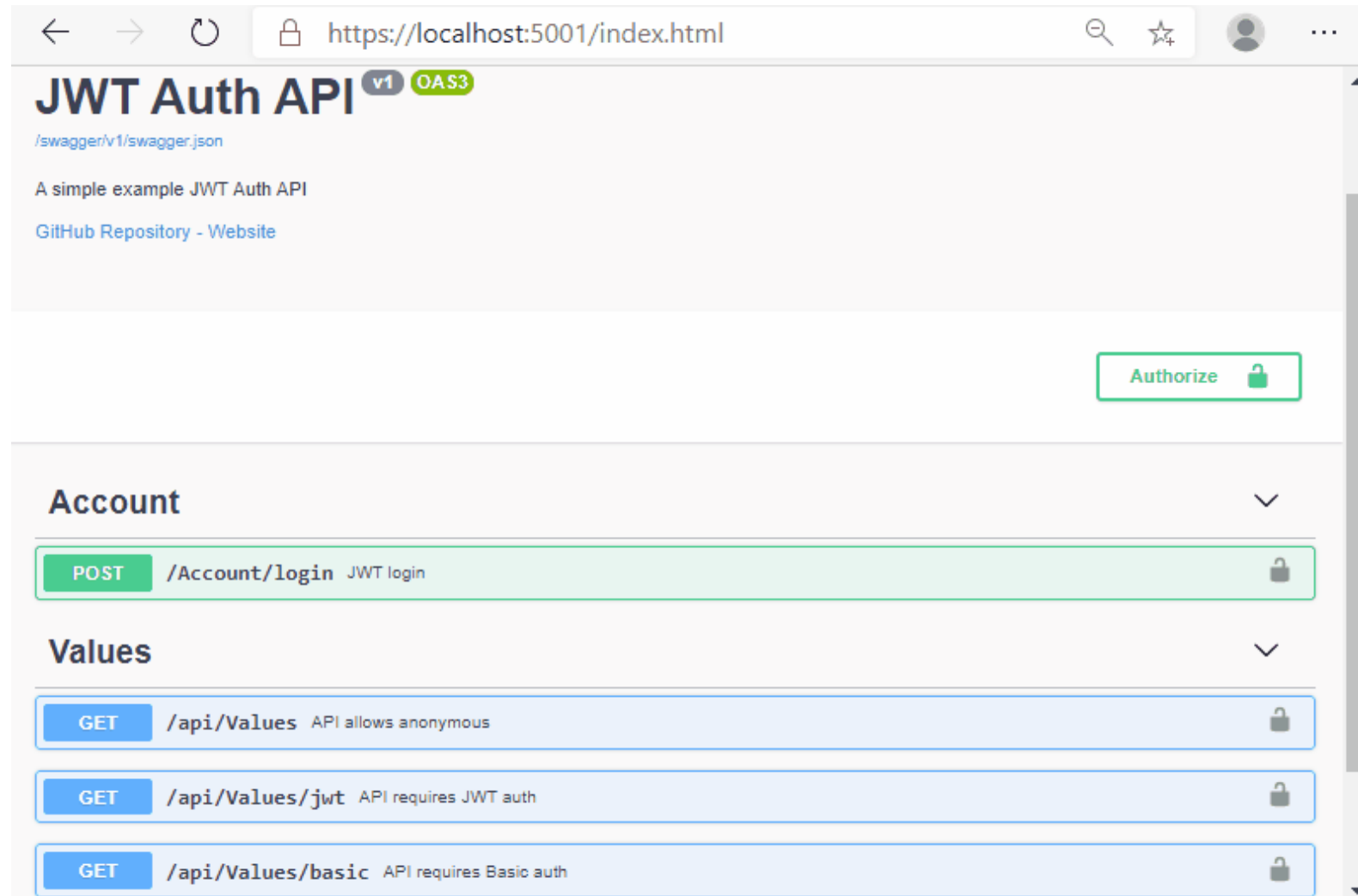
Соответственно в методах, в которых нужно проверить имеет ли пользователя доступ к запрашиваемой информации, мы должны:

- взять из запроса куки (через `ssid = request.COOKIES["session_id"]`)
- посмотреть есть ли в хранилище сессий такая запись, и достать идентификатор пользователя (`session_storage.get(ssid)`)
- проверить, можно ли данному пользователю смотреть запрошенную информацию через `permissions` (зависит от бизнес-логики вашего проекта)
- После аутентификации обратно во фронтенд необходимо отправить признак модератора, чтобы изменить представление приложения

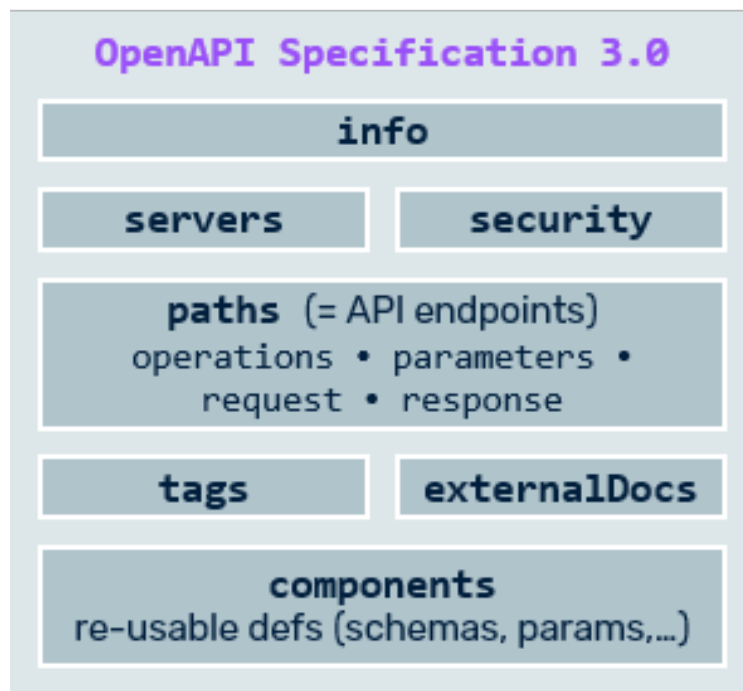


# Swagger

- Swagger – это фреймворк для спецификации RESTful API.
- Его прелесть заключается в том, что он дает возможность интерактивно просматривать спецификацию
- и отправлять запросы – так называемый Swagger UI



# OpenAPI



serialized in either  
JSON or YAML

HTTP, OAuth2, JWT<sup>3</sup>

■ synchronous calls  
■ call backs

- **The OpenAPI Specification** (изначально известная как *Swagger Specification*)
- формализованная спецификация и экосистема множества инструментов, предоставляющая интерфейс между front-end системами, кодом библиотек низкого уровня и коммерческими решениями в виде API.
- Вместе с тем, спецификация построена таким образом, что не зависит от языков программирования, и удобна в использовании как человеком, так и машиной

# Добавление Swagger

- Устанавливаем drf-yasg
- Подключаем swagger в url, обработчики появятся в swagger автоматически
- По этому адресу будет json файл. Мы будем использовать его для генерации кода фронтенда

```
pip install -U drf-yasg
```

```
schema_view = get_schema_view(
    openapi.Info(
        title="Snippets API",
        default_version='v1',
        description="Test description",
        terms_of_service="https://www.google.com/policies/terms/",
        contact=openapi.Contact(email="contact@snippets.local"),
        license=openapi.License(name="BSD License"),
    ),
    public=True,
    permission_classes=(permissions.AllowAny,),
)

urlpatterns = [
    ...
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),
    ...
]
```

<http://127.0.0.1:8000/swagger/?format=openapi>

# Просмотр

- По данной ссылке доступен наш swagger
- Теперь здесь мы можем протестировать все наши методы

<http://127.0.0.1/swagger/>

The screenshot shows the Swagger UI for the 'Snippets API v1'. The top bar includes the Swagger logo, the text 'Supported by SMARTBEAR', and a search bar containing 'http://127.0.0.1:8000/swagger/?format=openapi' with an 'Explore' button. Below the header, the API title 'Snippets API v1' is displayed, along with its base URL and a link to the OpenAPI spec. A 'Test description' section contains links for 'Terms of service', 'Contact the developer', and 'BSD License'. On the right, there are 'Django Login' and 'Authorize' buttons. A 'Schemes' dropdown is set to 'HTTP'. A 'Filter by tag' input field is present. The main content area lists API endpoints under the 'stocks' tag, including GET, POST, PUT, PATCH, and DELETE methods for various stock-related actions. Each endpoint is color-coded and includes a dropdown for the operation name and a lock icon. At the bottom, a 'Models' section shows a 'Stock' model with a right-pointing arrow.

Swagger  
Supported by SMARTBEAR

http://127.0.0.1:8000/swagger/?format=openapi Explore

**Snippets API** v1  
[ Base URL: 127.0.0.1:8000/ ]  
<http://127.0.0.1:8000/swagger/?format=openapi>

Test description  
[Terms of service](#)  
[Contact the developer](#)  
BSD License

Schemes  
HTTP

Django Login Authorize

Filter by tag

**stocks**

GET /stocks/ stocks\_list

POST /stocks/ stocks\_create

GET /stocks/{id}/ stocks\_read

PUT /stocks/{id}/ stocks\_update

PATCH /stocks/{id}/ stocks\_partial\_update

DELETE /stocks/{id}/ stocks\_delete

**Models**

Stock >