Лекция 6
Авторизация. Сессии

Проектирование систем и продуктовая веб-разработка

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 8-12 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.

Оформление и ГОСТ (не записываем)

- 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.

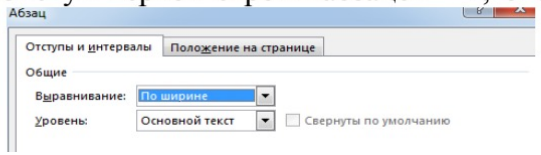
Ниже показаны примеры оформления текста РПЗ и различных структурных элементов.

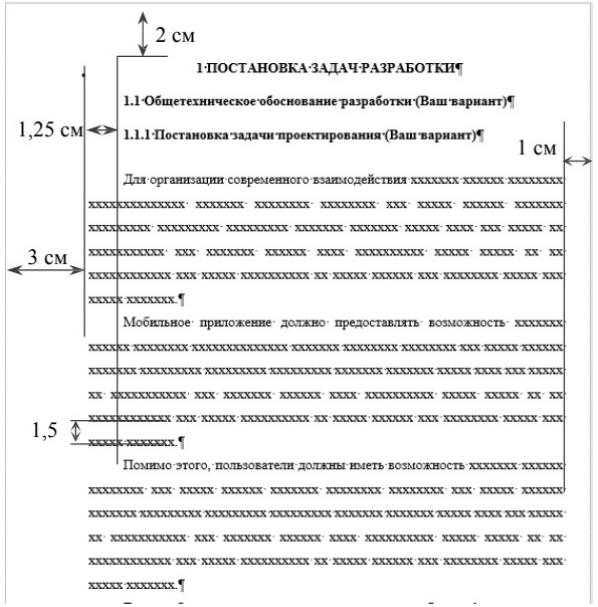
В РПЗ необходимо соблюдать следующие размеры полей страницы: левое – 3 см, правое – 1 см, нижнее – 2 см, верхнее – 2 см.

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.





https://drive.google.com/file/d/1bmQMZeqCY9WmmqQviDvA0rju7a1RgMh_/view

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;

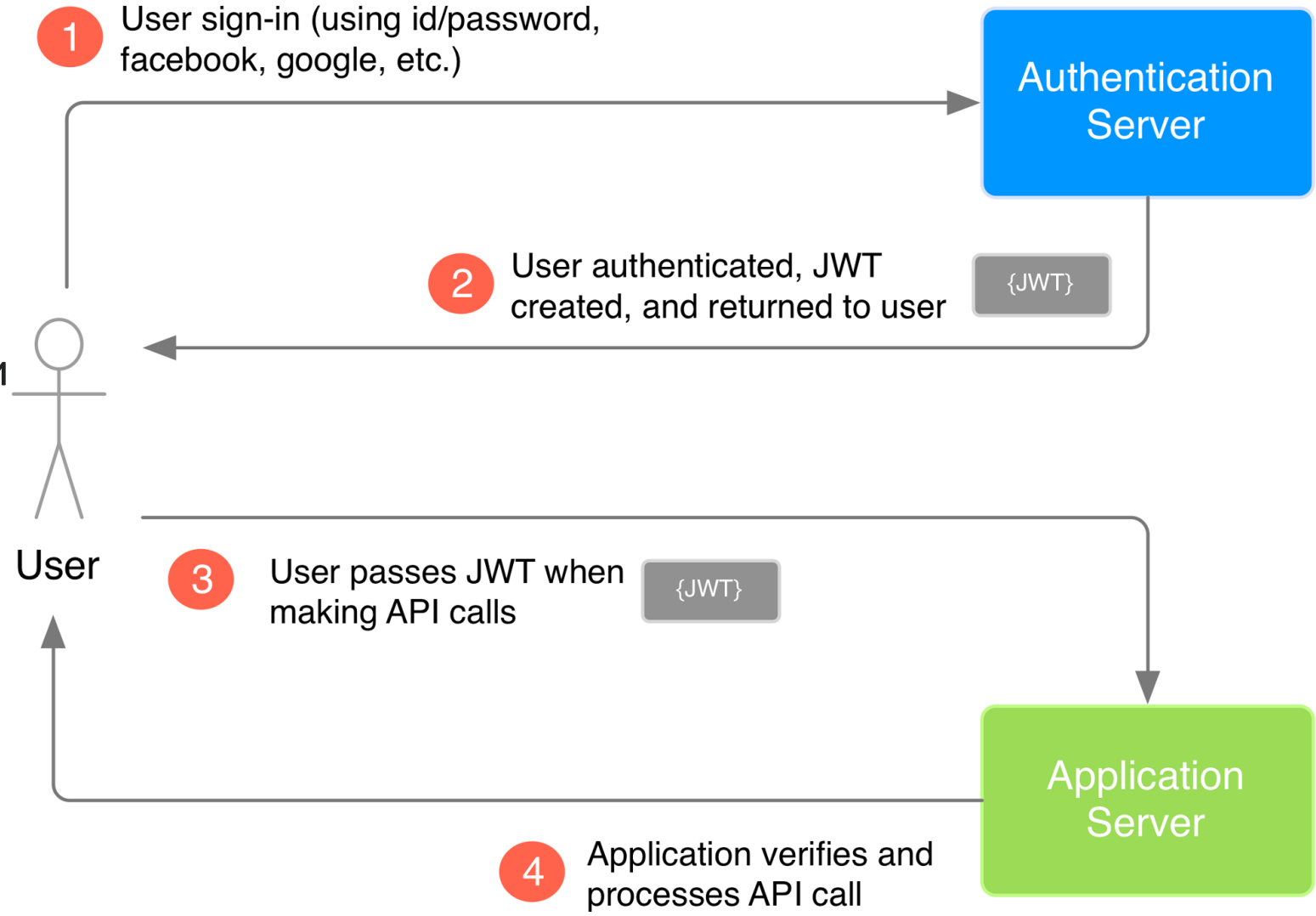
Идентификация — процедура, в результате выполнения которой для субъекта идентификации выявляется его идентификатор, однозначно определяющий этого субъекта в информационной системе (**Определяем** пользователя для заявки).

Авториза́ция (*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 (**гость** смотрит товар, **покупатель** добавляет в заказ, **продавец** - редактирует).

- Авторизация производит контроль доступа к различным ресурсам системы в процессе работы легальных пользователей после успешного прохождения ими аутентификации.

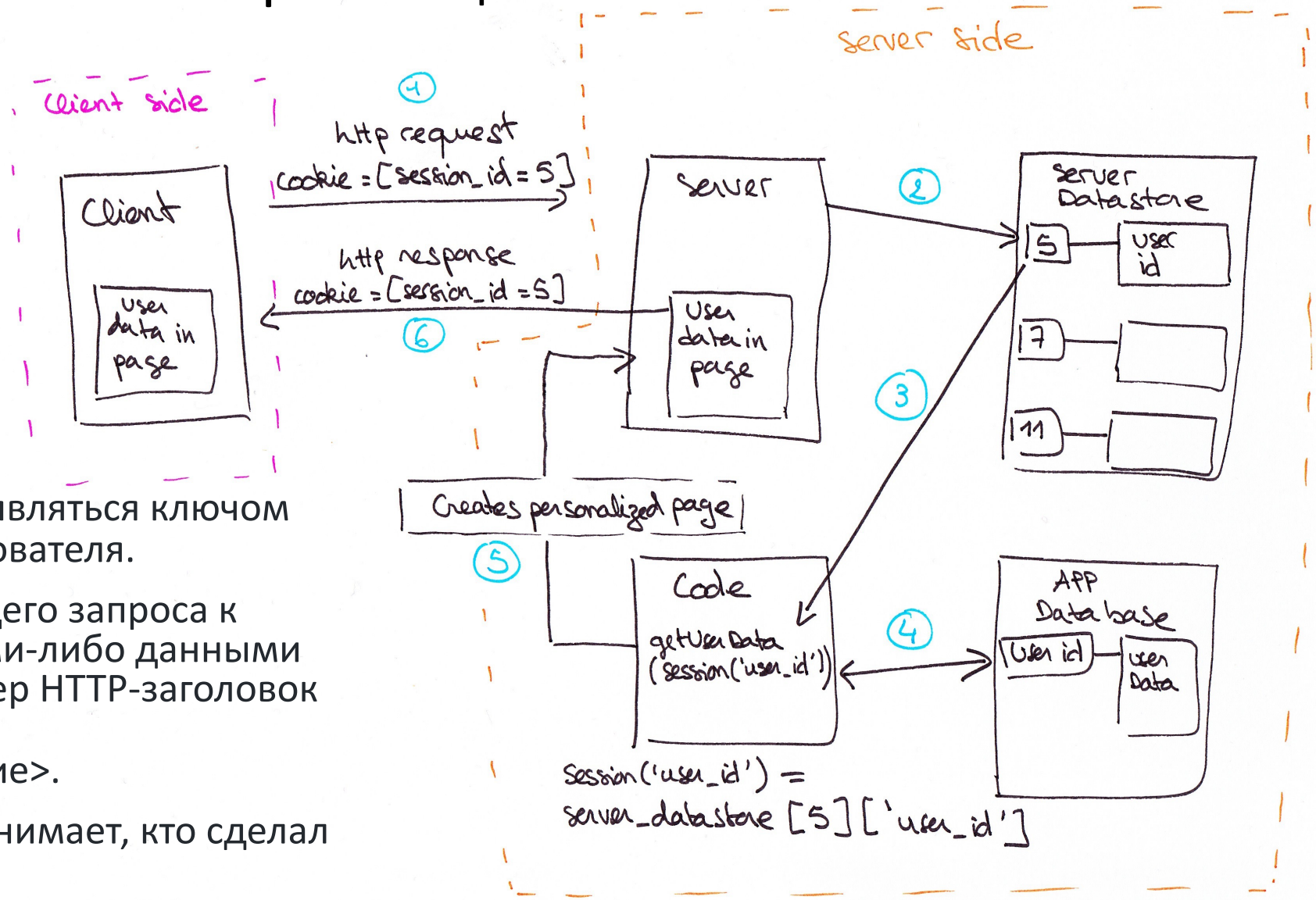
Схема аутентификации и авторизации

- Схема приведена для JWT, но мы заменив Authentication Server на метод аутентификации получим общую схему
- Метод аутентификации получает логин и пароль и если они правильные (таблица пользователь), возвращает обратно id сессии или JWT
- Во время авторизации каждый наш метод из 3ей лабораторной (Application Server) проверяет правильность сессии/JWT для выполнения действий



Сессии – схема авторизации

- При авторизации на сайте сервер отсылает в ответ HTTP-заголовок Set-Cookie (5), 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 (1), в которым в формате <ключ>=<значение>.
- Таким образом, сервер понимает, кто сделал запрос.

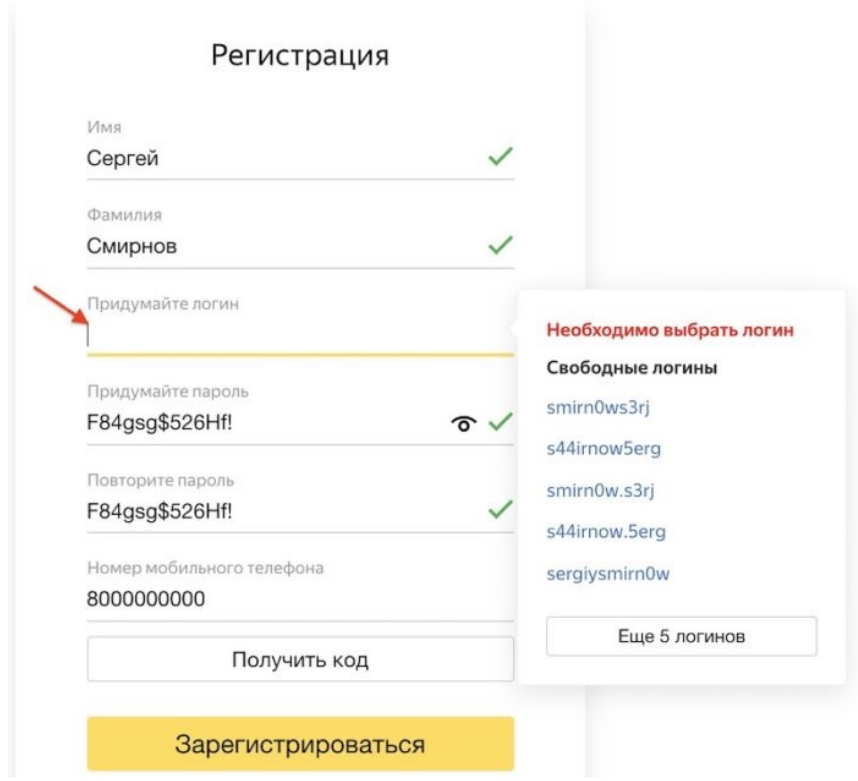


Куки

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

Пользователи

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

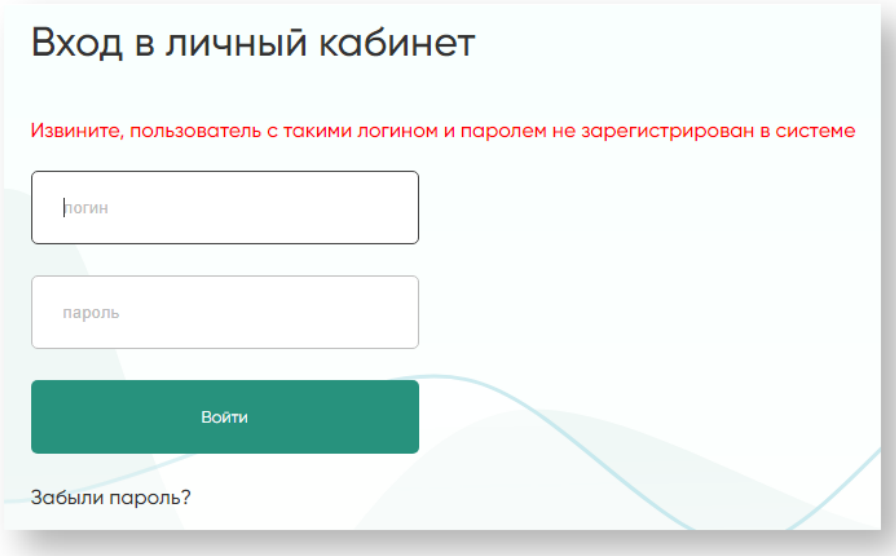



```python
from django.contrib.auth import models as user_models
from django.contrib.auth.models import PermissionsMixin

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

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.




```python
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

```python
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)
```

- Если у нас один вид пользователей – одна роль, нам достаточно просто проверить аутентифицирован ли он

```python
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)
```

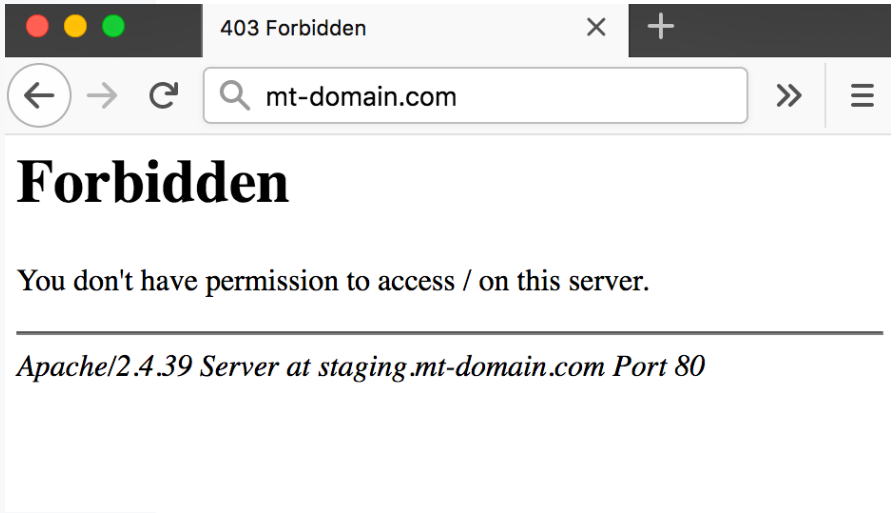
Авторизация. Ограничения на бэкенде

- Чтобы ограничить неавторизованным пользователем доступ к контенту, создадим view и добавим туда authentication_classes и permission_classes
- Получаем код 403 (у пользователя нет прав), 401 (пользователь не аутентифицирован)

```python
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)
```



Авторизация. 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

```python
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)
```

Авторизация. Permissions

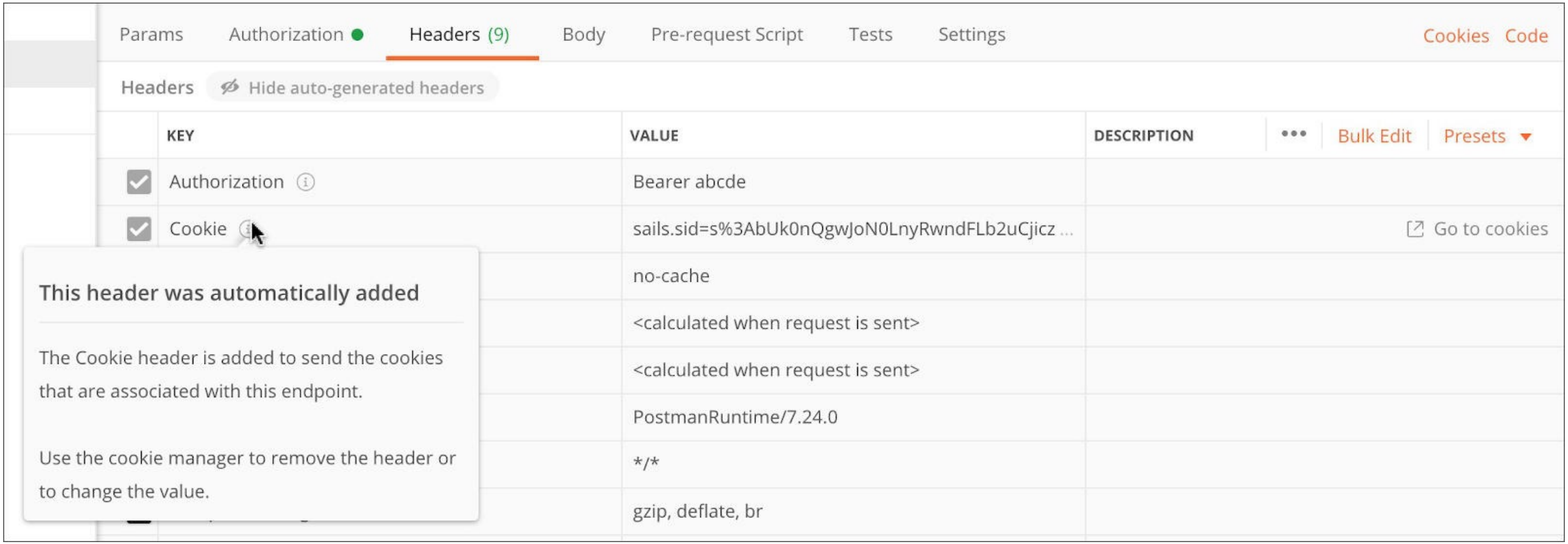
- Когда мы создали Классы прав доступа, их можно использовать в контроллерах

- 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

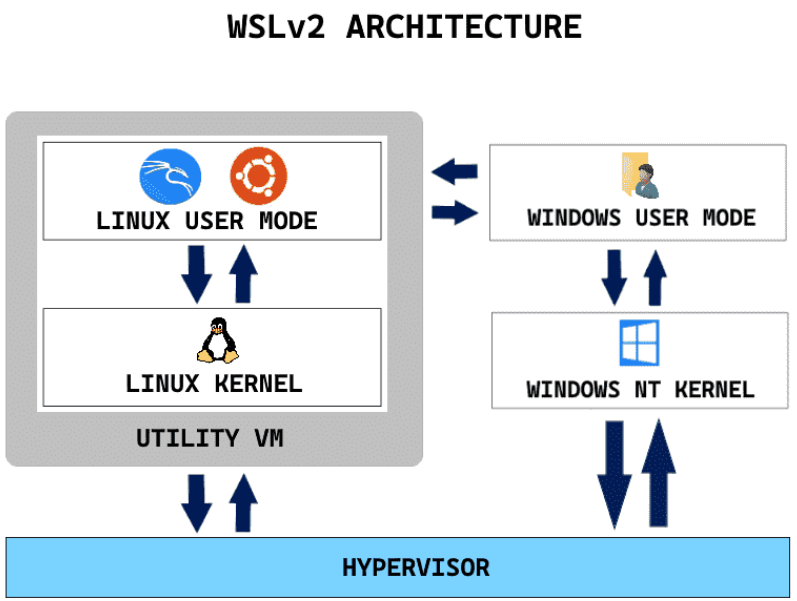


WSL - для Redis под Windows

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

Он понадобится чтобы запустить на Windows:

- Docker
- Kafka
- Redis



Redis

REmote **DI**ctionary **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

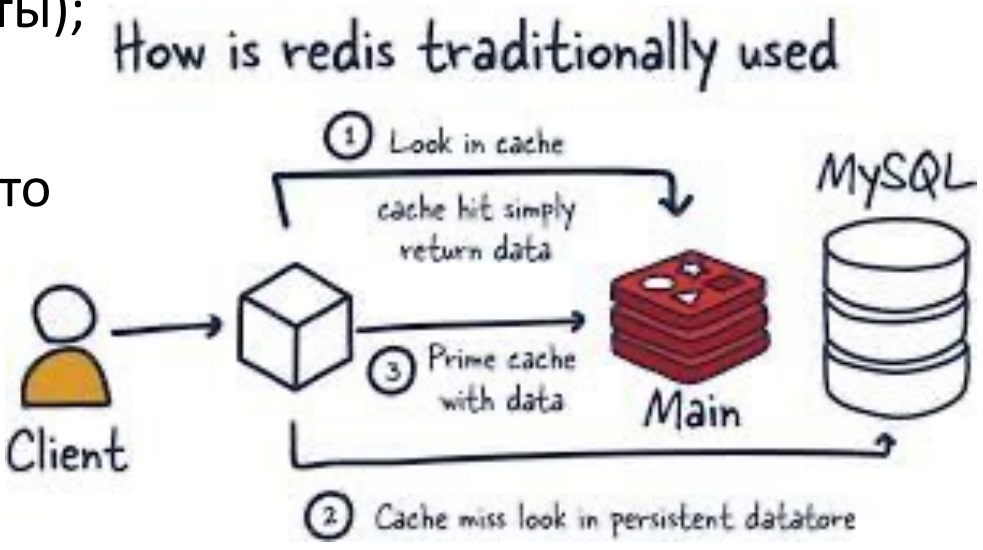
От реляционных баз Redis отличается:

- **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
```

- 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
```

- Можем получить все ключи из Redis

```
keys *
```

- Или написать целый скрипт на языке Lua для получения активных сессий пользователей

```
EVAL "local key_name = 'example_key'; for iterated_value=0,4 do redis.call('hmset', KEYS[1],
```

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

```python
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

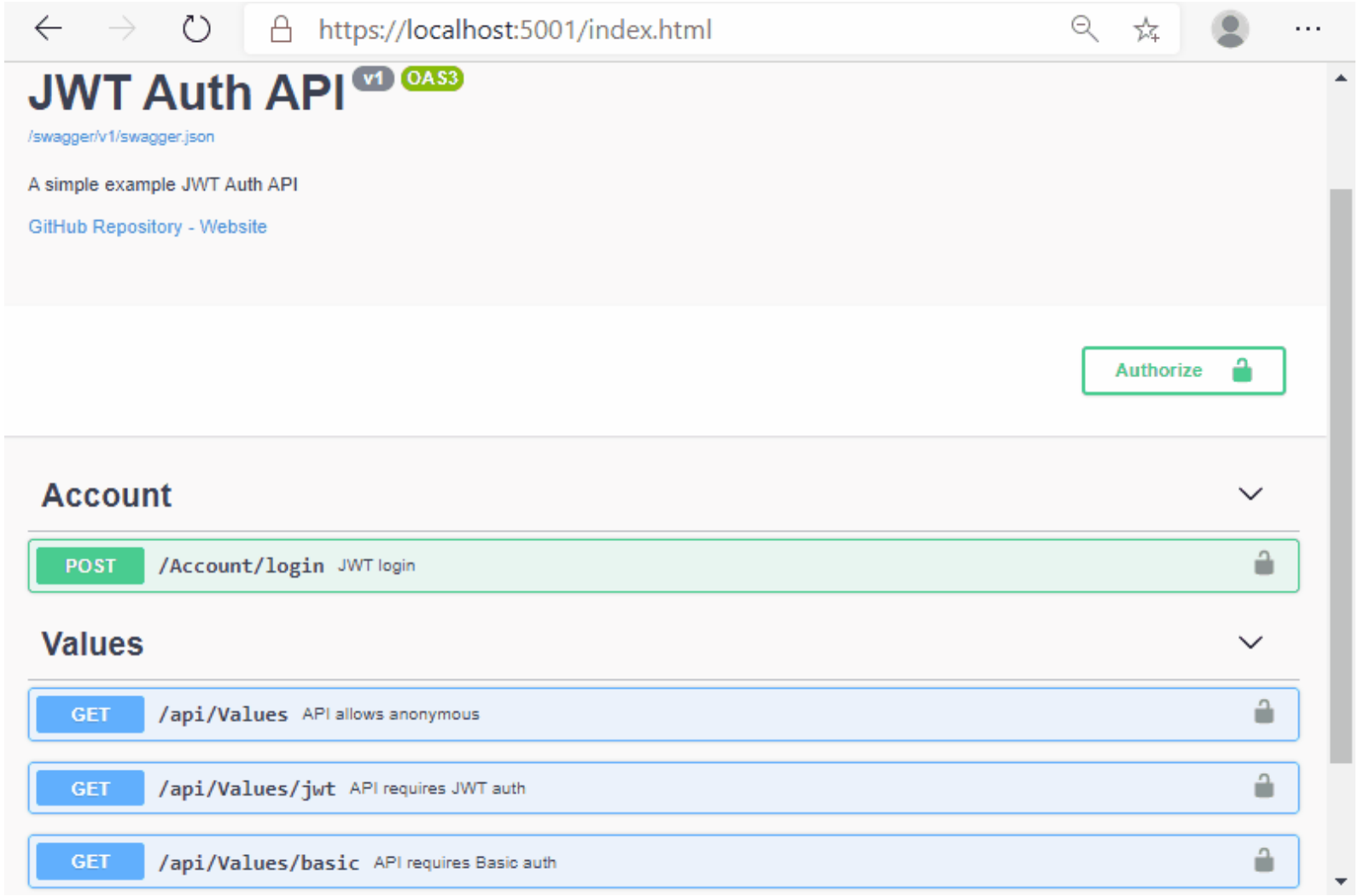
Соответственно в методах, в которых нужно проверить имеет ли пользователя доступ к запрашиваемой информации, мы должны:

- 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)

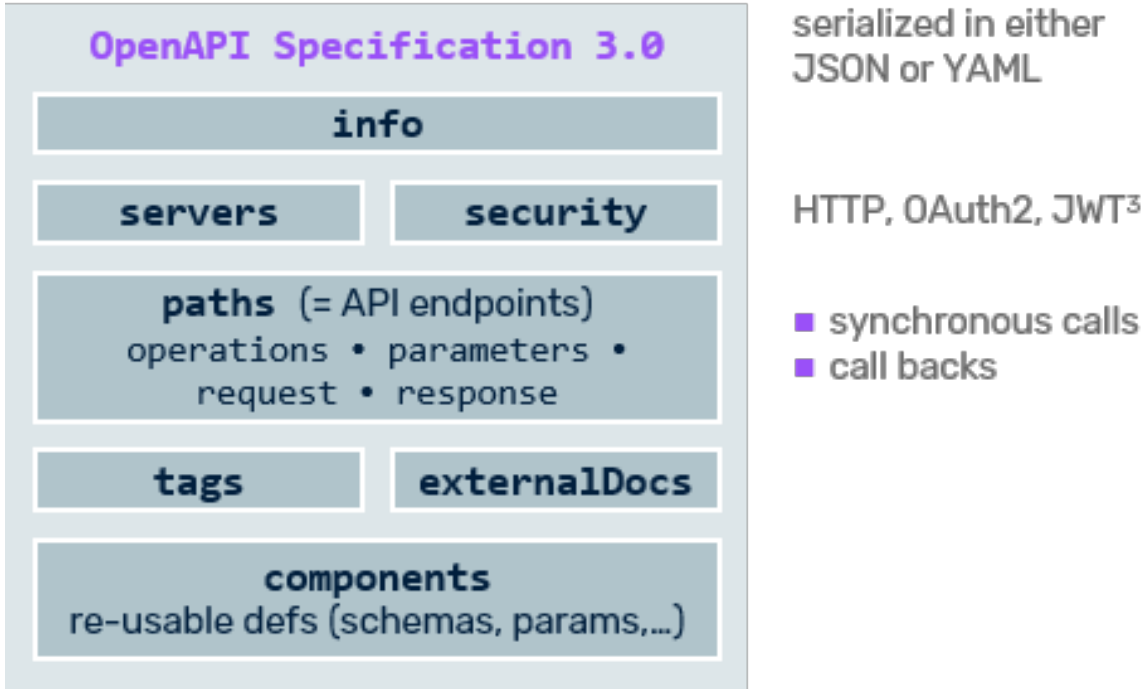
- После аутентификации обратно во фронтенд необходимо отправить признак модератора, чтобы изменить представление приложения

Swagger

- Swagger – это фреймворк для спецификации RESTful API.
- Swagger UI дает возможность интерактивно просматривать спецификацию и отправлять запросы



OpenAPI



- **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/