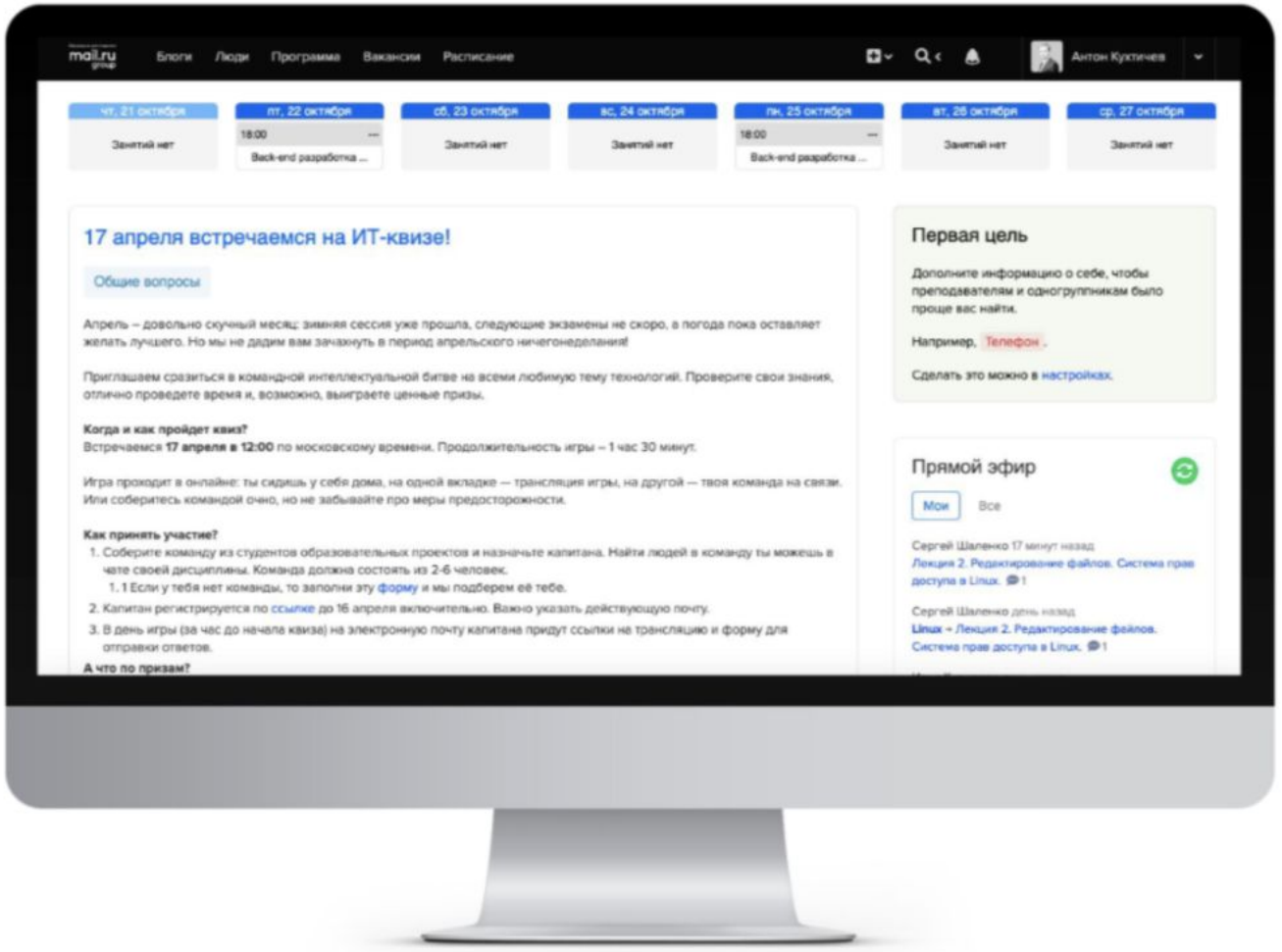
Бэкенд-разработка на Python. Лекция №8. Авторизация и аутентификация в веб-приложениях

Алена Елизарова

образование

Напоминание отметиться на портале



План занятия

1. Авторизация и аутентификация
2. Виды авторизации
3. Cookies
4. Middleware
5. OAuth2
6. Social django

Авторизация

Авторизация и аутентификация в веб-приложениях

Аутентификация и авторизация

Аутентификация - предоставление доказательств, что вы на самом деле есть тот, кем идентифицировались (от слова “authentic” — истинный, подлинный).

Авторизация - проверка, что вам разрешен доступ к запрашиваемому ресурсу.

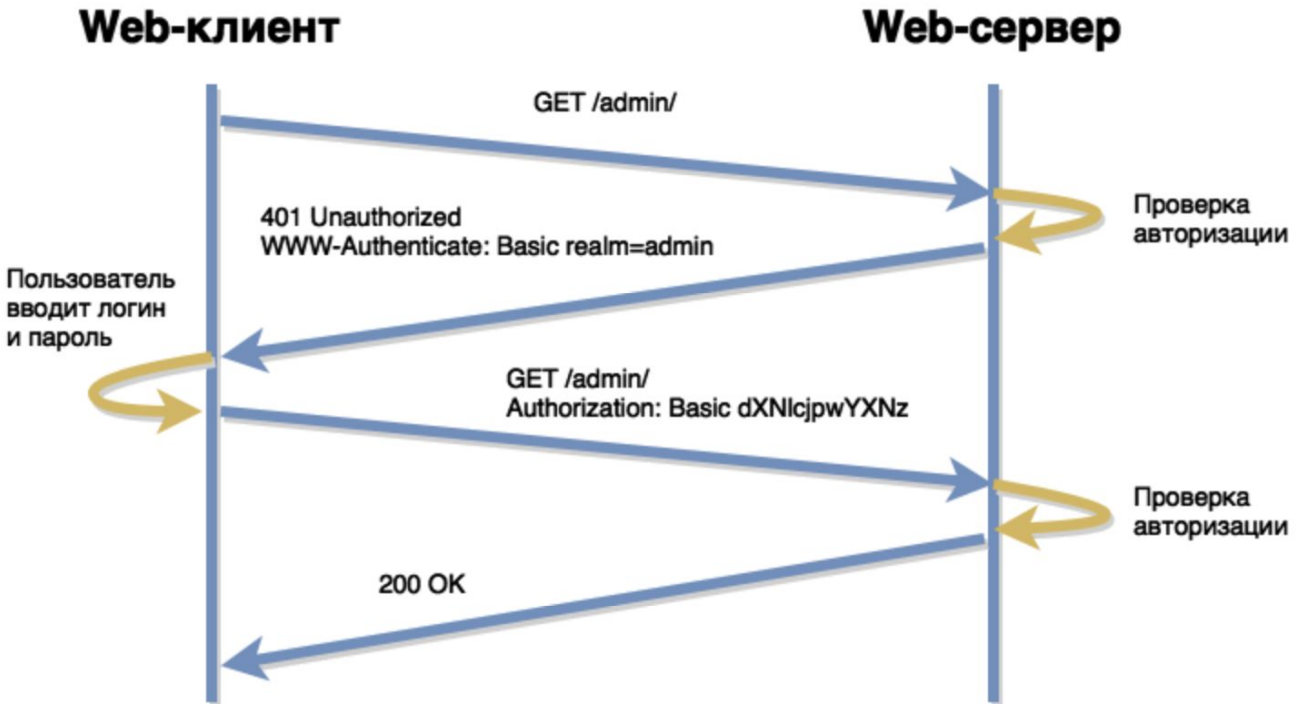
Авторизация в веб-приложениях

HTTP - **stateless** протокол, т.е. не предполагает поддержания соединения между клиентом и сервером. Это значит, что сервер не может связать информацию о пользователе с конкретным соединением и вынужден загружать ее при каждом запросе.

Виды авторизации

- Basic-авторизация
- Авторизация, основанная на куках (cookie-based)
- Авторизация через соц.сети (OAuth2)

Basic HTTP Authorization



Заголовки и коды ответа

`401 Unauthorized` - для доступа к ресурсу нужна авторизация

`WWW-Authenticate: Basic realm="admin"` - запрос логина/пароля для раздела admin

`Authorization: Basic Z2l2aTpkZXJwYXJvbA==` - передача логина/пароля в виде base64(login + ':' + password)

`403 Forbidden` - логин/пароль не подходят

`REMOTE_USER` - CGI переменная с именем авторизованного пользователя

Достоинства и недостатки

➕ Простота и надежность

➕ Готовые модули для web-серверов

➕ Не требует написания кода

➖ Логин/пароль передаются в открытом виде - нужен https

➖ Невозможно изменить дизайн формы входа

➖ Невозможно «сбросить» авторизацию

Cookies

Cookies

Cookies - небольшие фрагменты данных, которые браузер хранит на стороне клиента и передает на сервер при каждом запросе.

Cookies привязаны к доменам, поэтому при каждом запросе сервер получает только «свои» cookies. Невозможно получить доступ к cookies с другого домена.

Cookies используются для поддержания состояния (state management) в протоколе HTTP и, в частности, для авторизации.

Cookies

`name=value` - имя и значение cookie

`Expires` - время жизни cookie, по умолчанию - до закрытия окна

`Domain` - домен cookie, по умолчанию - домен текущего URL

`Path` - путь cookie, по умолчанию - путь текущего URL

`Secure` - cookie должна передаваться только по https

`HttpOnly` - cookie не доступна из JavaScript

Установка и удаление Cookies

Set-Cookie: sessid=d232rn38jd1023e1nm13r25z;
Domain=.site.com; Path=/admin/;
Expires=Sat, 15 Aug 2015 07:58:23 GMT;
Secure; HttpOnly
Set-Cookie: lang=ru

Set-Cookie: sessid=xxx;
Expires=Sun, 06 Nov 1994 08:49:37 GMT

Для удаления cookie, сервер устанавливает Expires в прошлом.

Получение Cookies

Cookie: sessid=d232rn38jd1023e1nm13r25z; lang=ru;
csrftoken=vVqoyo5vzD3hWRHQDRpIHzVmKLfBQIGD;

При каждом запросе браузер выбирает подходящие cookies и отправляет только их значения.

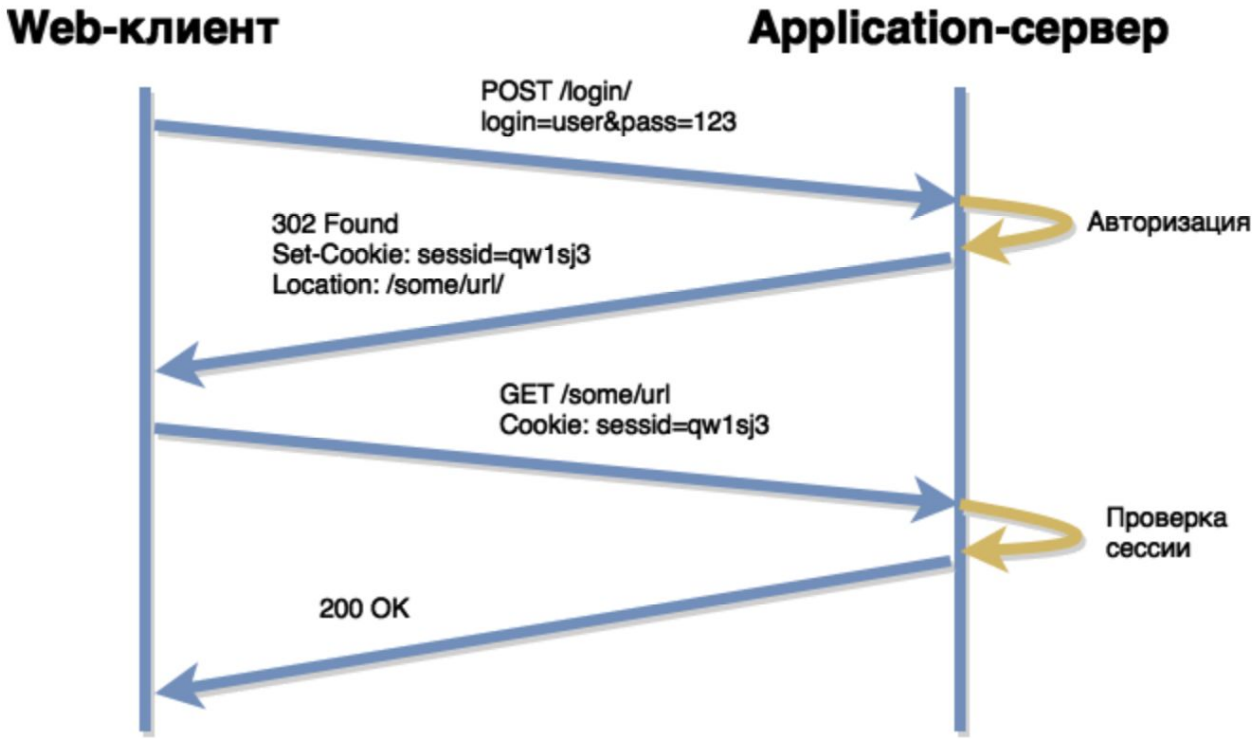
Работа с cookies в Django

```
# установка
response.set_cookie('sessid','asde132dk13d1')
response.set_cookie(
    'sessid','asde132dk13d1',
    domain='.site.com', path='/blog/',
    expires=(datetime.now() + timedelta(days=30))
)

# удаление
response.delete_cookie('another')

# получение
request.COOKIES # все cookies
request.COOKIES.get('sessid') # одна cookie
```

Cookie-based авторизация



Middleware

```
# middleware.py

class LoginMiddleware(object):
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        return self.get_response(request)

    def process_view(self, request, view_func, view_args, view_kwargs):
        if is_authenticated(request, view_func):
            return None
        return redirect_to_login() # возвращает HttpResponse
```

Встроенная авторизация Django

`django.contrib.sessions`

Предоставляет поддержку сессий, в том числе анонимных.
Позволяет хранить в сессии произвольные данные, а не только ID пользователя. Позволяет хранить сессии в различных хранилищах, например Redis или Memcached.

```
def some_view(request):
    val = request.session['some_name']
    request.session.flush()
    request.session['some_name'] = 'val2'
```

django.contrib.auth

Предоставляет готовую модель User , готовую систему разделения прав, view для регистрации / входа / выхода. Используется другими приложениями, например django.contrib.admin

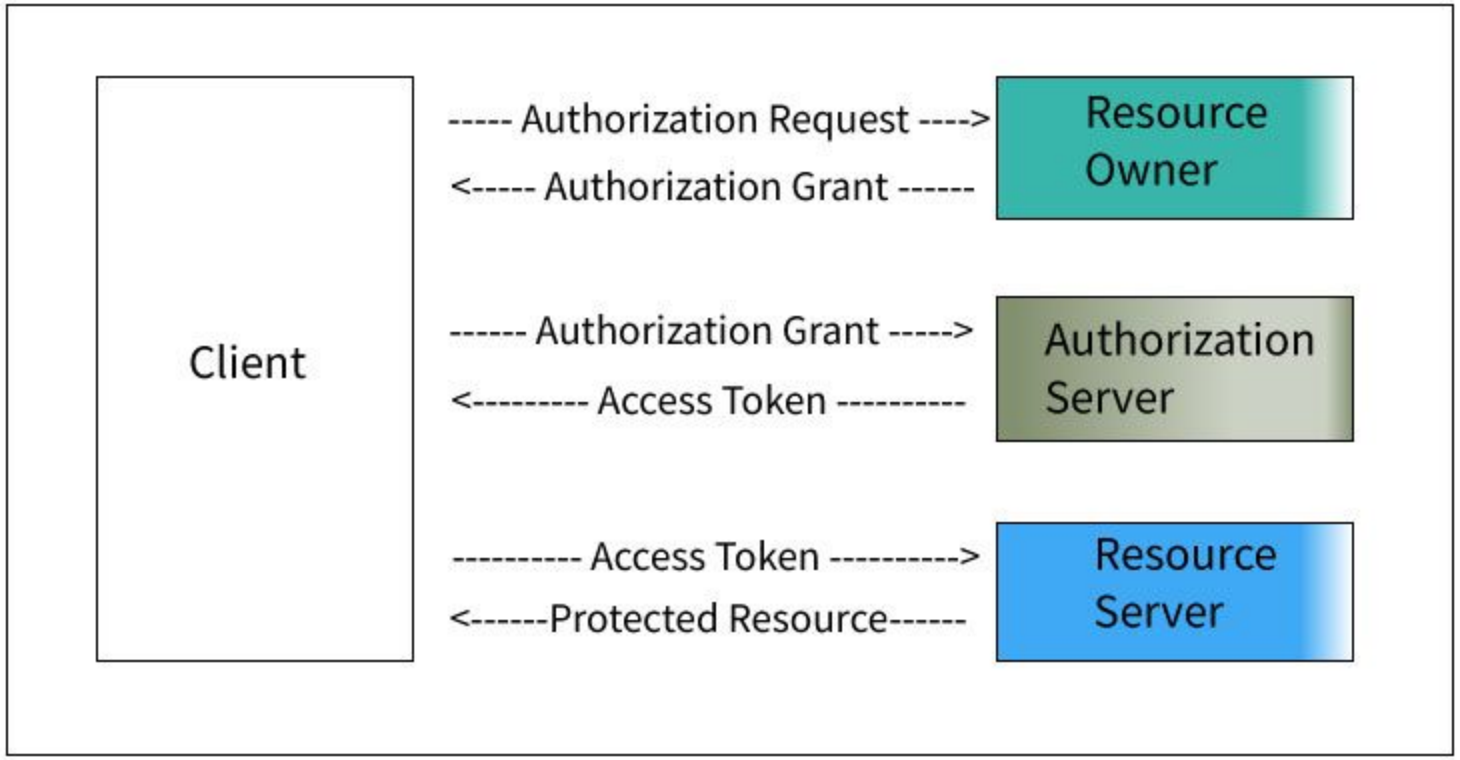
```
def some_view(request):
    user = request.user # Определено всегда!
    if user.is_authenticated():
        pass # обычный пользователь
    else:
        pass # анонимный пользователь
```

OAuth(2.0) авторизация

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.

Результатом авторизации является **access token** — некий ключ, предъявление которого является пропуском к защищенным ресурсам. Обращение к ним в самом простом случае происходит по HTTPS с указанием в заголовках или в качестве одного из параметров полученного access token.

OAuth(2.0)


Client
----- Authorization Request ---->
<----- Authorization Grant ------
Resource Owner
------ Authorization Grant ----->
<--------- Access Token ----------
Authorization Server
---------- Access Token ---------->
<------Protected Resource------
Resource Server

Практика

Реализация

```
>>> pip install social-auth-app-django

#settings.py

AUTHENTICATION_BACKENDS = [
    'social_core.backends.google.GoogleOAuth2',
    'social_core.backends.linkedin.LinkedinOAuth2',
    'social_core.backends.instagram.InstagramOAuth2',
    'social_core.backends.facebook.FacebookOAuth2',
    'django.contrib.auth.backends.ModelBackend',
]

INSTALLED_APPS = [
        ....
    'social_django',
]
```

Реализация

```
#settings.py

LOGIN_URL = 'login'
LOGIN_REDIRECT_URL = 'home'
LOGOUT_URL = 'logout'
LOGOUT_REDIRECT_URL = 'login'

SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = ''  # App ID
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = ''  # App Secret


STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]

>>> ./manage.py migrate
```

Реализация

```
#settings.py

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                …

                'social_django.context_processors.backends'
            ],
        },
    },
]
```

Реализация

```
from django.contrib import admin
from django.urls import path, include
from django.contrib.auth import views as auth_views
from blog import views as views


urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('login/', views.login, name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('social-auth/', include('social_django.urls', namespace='social')),
]
```

Реализация

```
<a href="{% url 'social:begin' 'google-oauth2' %}">Login With Google</a>
```

Реализация. В приложении Google Console

```
https://console.cloud.google.com/

# Authorized JavaScript origins

http://127.0.0.1:8000
http://localhost:8000

# Authorized redirect URIs

http://localhost:8000/social-auth/complete/google-oauth2/
http://127.0.0.1:8000/social-auth/complete/google-oauth2/
```

Реализация. В приложении Google Console

- Выберите страну и примите соглашения
- Create new project, введите название проекта
- Credentials -> OAuth Client ID
- OAuth consent screen -> External
- Введите название приложения и email
- Тип приложения web

Домашнее задание

1. Реализовать OAuth2-авторизацию - логин, логаут
2. Написать декоратор, проверяющий авторизацию при вызовах API
3. Изменить запросы и код API

Не забудьте оставить отзыв на портале

Спасибо
за внимание

образование