

# Авторизация в Web- приложениях

# Аутентификация и авторизация

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

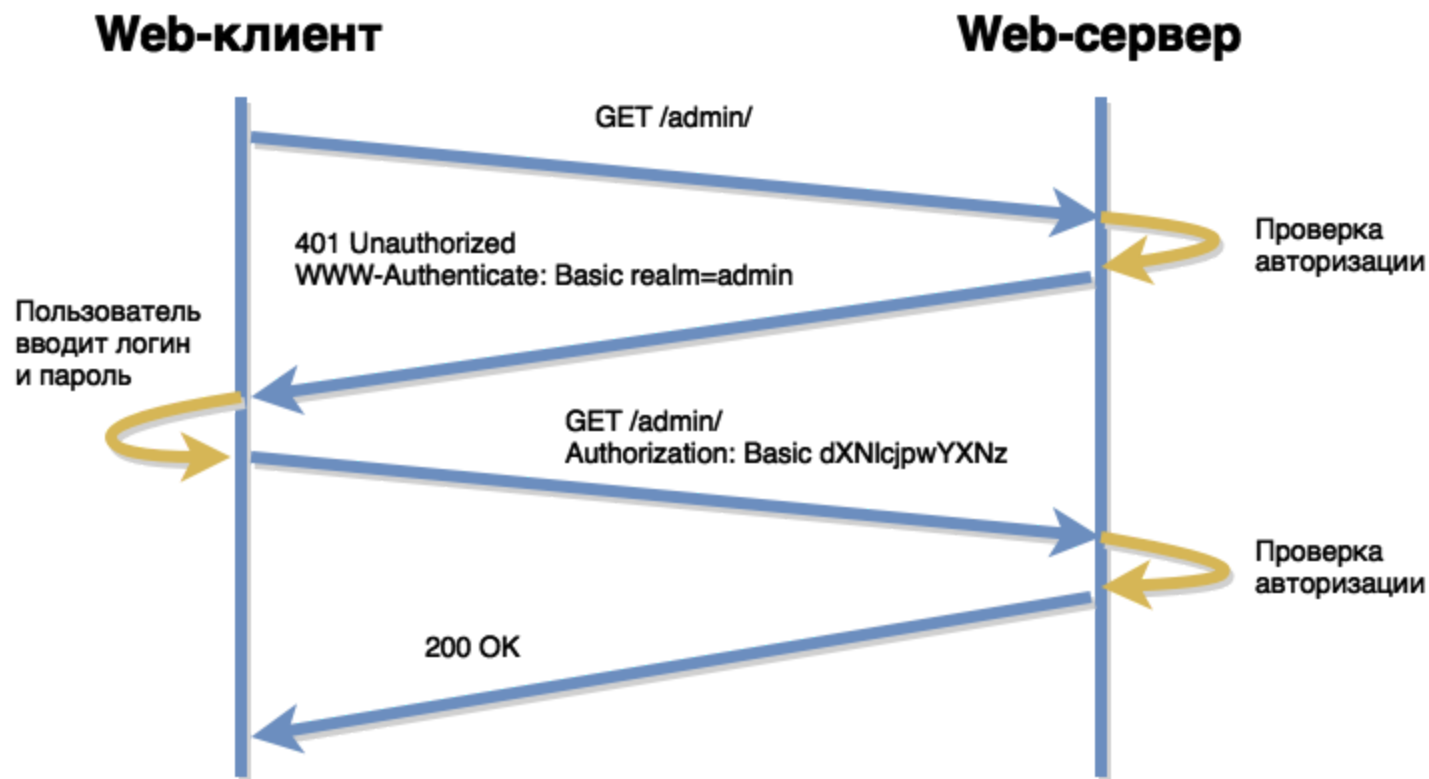
Авторизация - проверка, что вам разрешен доступ к запрашиваемому ресурсу.

# Авторизация в Web-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.

# Basic HTTP Authorization

# 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.

# Атрибуты Cookie

- `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 cookie в Django

*# установка*

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

*# удаление*

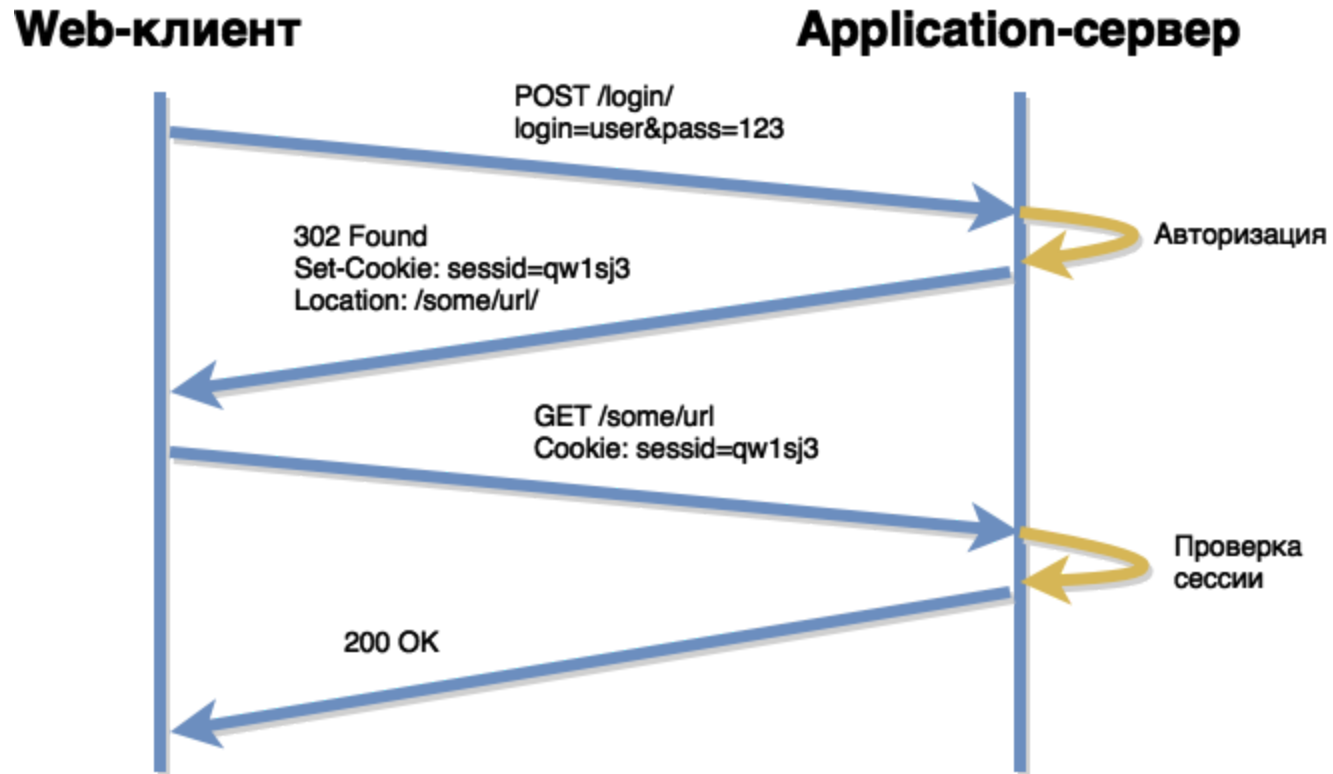
```
resp.delete_cookie('another')
```

*# получение*

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

# Cookie-based авторизация

# Cookie-based авторизация



# Middleware в Django

**Middleware** - это Python класс, в котором есть один из указанных ниже методов. Список всех активных middleware указан в настройке `MIDDLEWARE`.

- `__init__(get_response)`
- `__call__(request)`
- `process_view(request, func, args, kwargs)`
- `process_exception(request, exception)`
- `process_template_response(request, response)`



# 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

## авторизация

# OAuth(2.0) авторизация

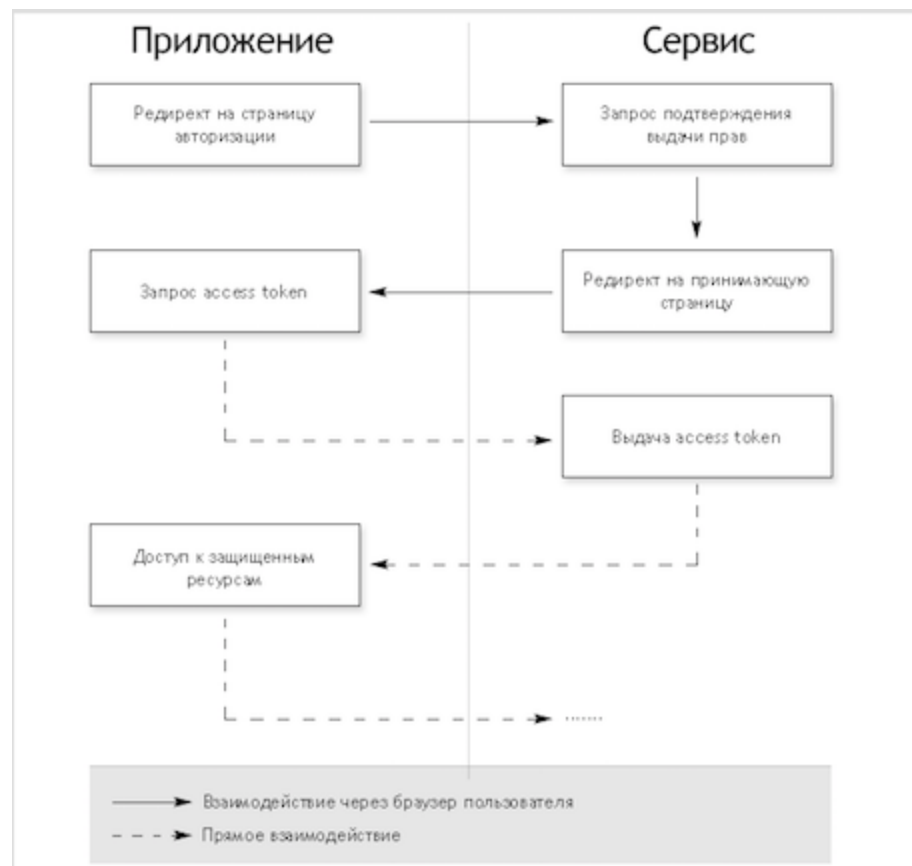
OAuth 2.0 - протокол авторизации, позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя на другом сервисе. Протокол избавляет от необходимости доверять приложению логин и пароль, а также позволяет выдавать ограниченный набор прав, а не все сразу.

# OAuth(2.0) авторизация

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.

# Варианты OAuth авторизации

- авторизация для приложений, имеющих серверную часть
- авторизация для полностью клиентских приложений (мобильные и desktop-приложения)
- авторизация по логину и паролю
- восстановление предыдущей авторизации





# JSON Web Tokens (JWT)

# JSON Web Tokens (JWT)

**JSON Web Token (JWT)** — открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных авторизации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

# Структура

**Хедер** содержит информацию о том, как должна вычисляться JWT подпись. Хедер — это тоже JSON объект, который выглядит следующим образом:

```
header = { "alg": "HS256", "typ": "JWT" }
```

**Payload** — это полезные данные, которые хранятся внутри JWT. Эти данные также называют JWT-claims (заявки)

```
payload = { "userId": "b08f86af-35da-48f2-8fab-cef3904660bd" }
```

# Структура

**Signature** - подпись вычисляется с использованием следующего

псевдо-кода:

```
const SECRET_KEY = 'cAtwa1kkEy'  
const unsignedToken = base64urlEncode(header) + '.' + base64urlEncode(payload)  
const signature = HMAC-SHA256(unsignedToken, SECRET_KEY)
```

# Создание токена

```
const token = encodeBase64Url(header) + '.' + encodeBase64Url(payload) + '.' + encodeBase64Url(signature)
```