ИТМО

Протоколы аутентификации

Александр Менщиков,
к.т.н., доцент ИТМО

Содержание лекции

1. Protobuf
2. HTTP/2
3. Wasm
4. gRPC
5. OAuth

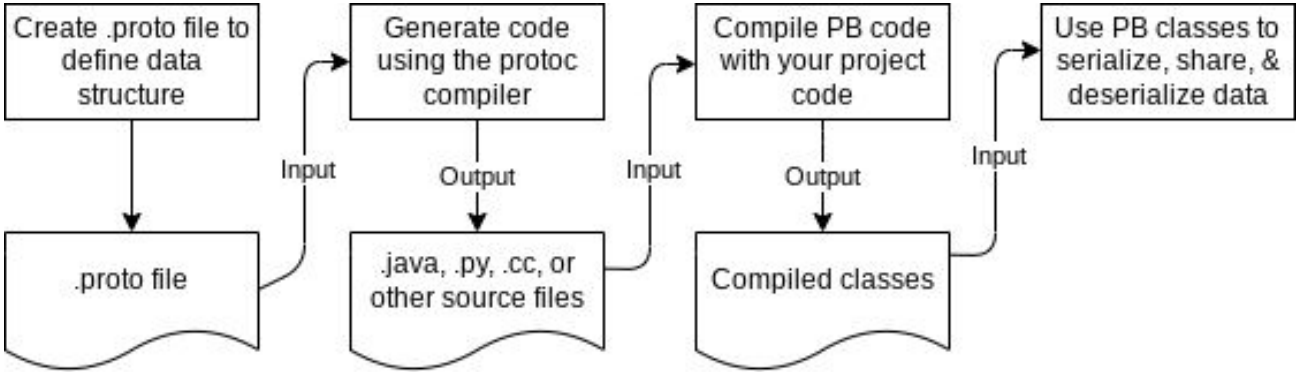
Protobuf

Protocol Buffers

Protocol Buffers — протокол сериализации структурированных данных, предложенный Google.

Протокол предоставляет способ описания данных на специальном языке (Protocol Buffer Language) и далее генерирует код сериализации и десериализации на различных языках программирования.

https://protobuf.dev/



Установка компилятора

Загружаем **protoc** под нужную архитектуру
https://github.com/protocolbuffers/protobuf/releases/tag/v26.1

```
./protoc --version
```

Устанавливаем библиотеку под нужный язык программирования (Python)

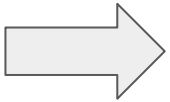
```
pip install protobuf
```

Hello, protobuf

```
message Person {
 optional string name = 1;
 optional int32 id = 2;
 optional string email = 3;
}
```

definition

example1.proto



```
DESCRIPTOR =
_descriptor_pool.Default().AddSerializedFile(b'\n\x0e\x65xample
1.proto\"1\n\x06Person\x12\x0c\n\x04name\x18\x01
\x01(\t\x12\n\n\x02id\x18\x02
\x01(\x05\x12\r\n\x05\x65mail\x18\x03 \x01(\t')

_globals = globals()
_builder.BuildMessageAndEnumDescriptors(DESCRIPTOR, _globals)
_builder.BuildTopDescriptorsAndMessages(DESCRIPTOR,
'example1_pb2', _globals)
if not _descriptor._USE_C_DESCRIPTORS:
 DESCRIPTOR._loaded_options = None
 _globals['_PERSON']._serialized_start=18
 _globals['_PERSON']._serialized_end=67
# @@protoc_insertion_point(module_scope)
```

```
./protoc --python_out=. -I. example1.proto
./protoc --java_out=. -I. example1.proto
./protoc --cpp_out=. -I. example1.proto
```

Использование сгенерированного класса

<object>.**SerializeToString**()

<object>.**ParseFromString**(***serialized_str***)

```python
import example1_pb2

person = example1_pb2.Person()
person.id = 1234
person.name = "John Doe"
person.email = "jdoe@example.com"

print(person.SerializeToString())
```

```
→  l8 python3 usage.py
serialized_value=b'\n\x08John Doe\x10\xd3\t\x1a\x10jdoe@example.com'
original_value=name: "John Doe"
id: 1235
email: "jdoe@example.com"
```

Декодируем значения

```
pip3 install grpcio-tools

echo <...> | base64 -d |
python3 -m grpc_tools.protoc
--decode_raw
```

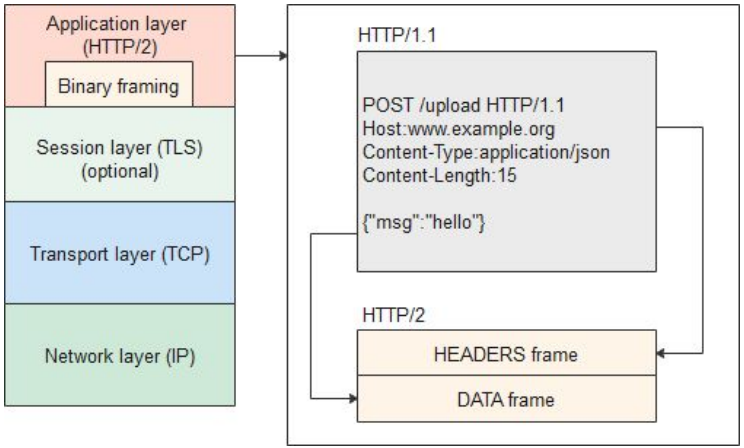
```
1: "John Doe"
2: 1234
3: "jdoe@example.com"
4 {
  1: "555-4321"
  2: 2
}
```

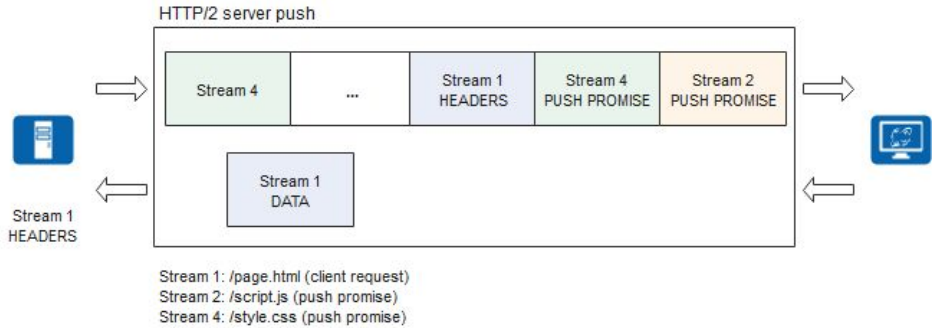
```
syntax = "proto2";

package tutorial;

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    PHONE_TYPE_UNSPECIFIED = 0;
    PHONE_TYPE_MOBILE = 1;
    PHONE_TYPE_HOME = 2;
    PHONE_TYPE_WORK = 3;
  }

  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2 [default = PHONE_TYPE_HOME];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```

HTTP/2

Предпосылки

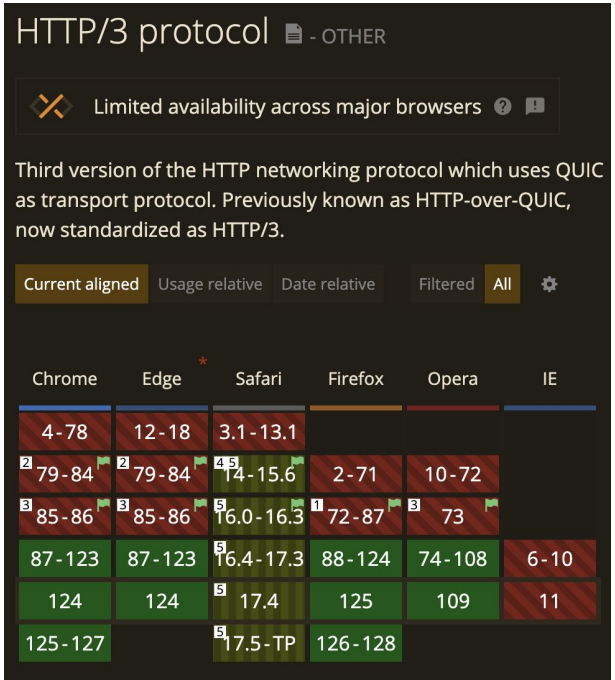
- Уменьшить задержки за счет:
  - Устранение избыточности HTTP/1.1
  - Мультиплексирование нескольких запросов в одном TCP соединении
  - Сжатие HTTP заголовков
  - Серверный push
- HTTP/2 не меняет семантику протокола (URI,статус коды и т.п.)

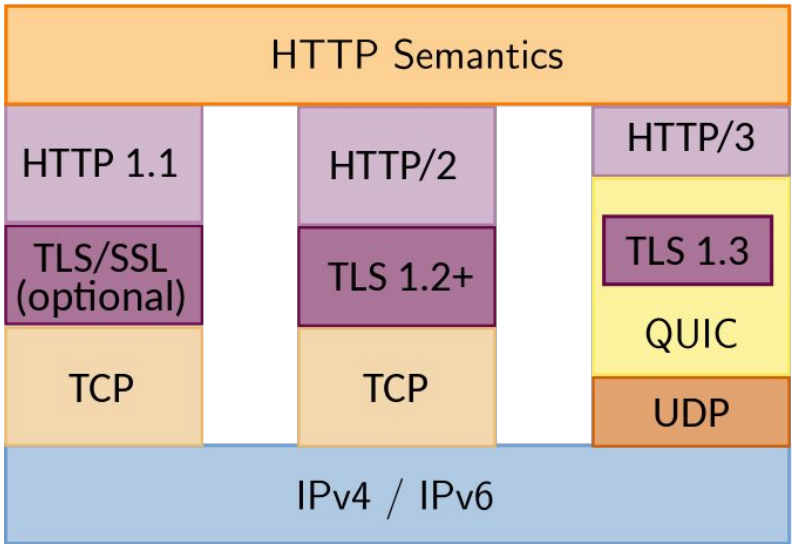




HTTP/3

- Использует UDP (QUIC протокол) вместо TCP
- Шифрование по умолчанию





Wasm

WebAssembly (Wasm)

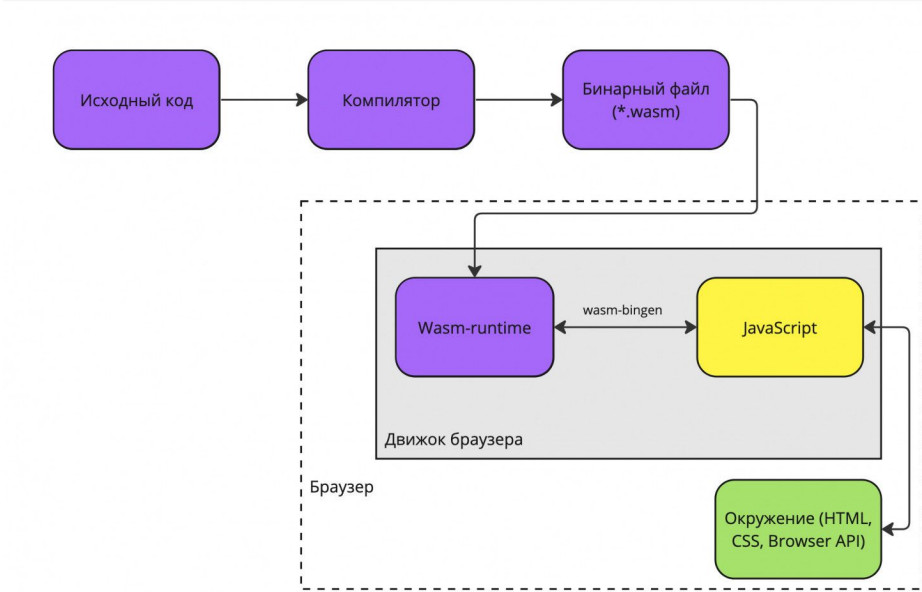
Wasm – открытый формат байт-кода, исполняемого современными браузерами. Он позволяет переносить код, написанный на таких языках как C, C++, C#, Rust, в низкоуровневые ассемблерные инструкции и использовать его в сети. Формат имеет компактные размеры, высокую производительность, близкую к нативной, и может одновременно работать с JavaScript.

WebAssembly разработан для дополнения JavaScript – используя WebAssembly JavaScript API вы можете загружать модули WebAssembly в приложения JavaScript и обеспечивать взаимодействие между ними, используя общие функции.

Преимущества и принцип работы WASM

Преимущества

- Поддерживается браузерами
- Библиотеки к современным языкам
- Доступ к API браузера и DOM (как в js)
- Скорость работы

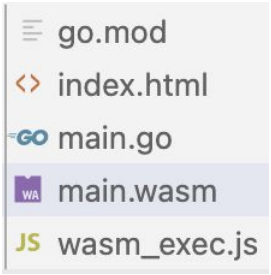


Недостатки

- Никто не использует кроме гигантов?
- Большие размеры файлов
- Сложность технологии
- Не хватает devtools для разработки и отладки

Демо

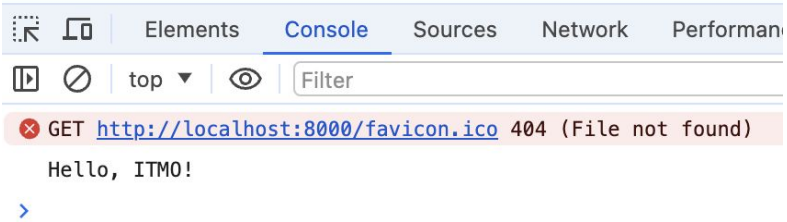
https://github.com/golang/go/wiki/WebAssembly#getting-started



http://localhost:8000/

python3 -m http.server

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, ITMO!")
}
```

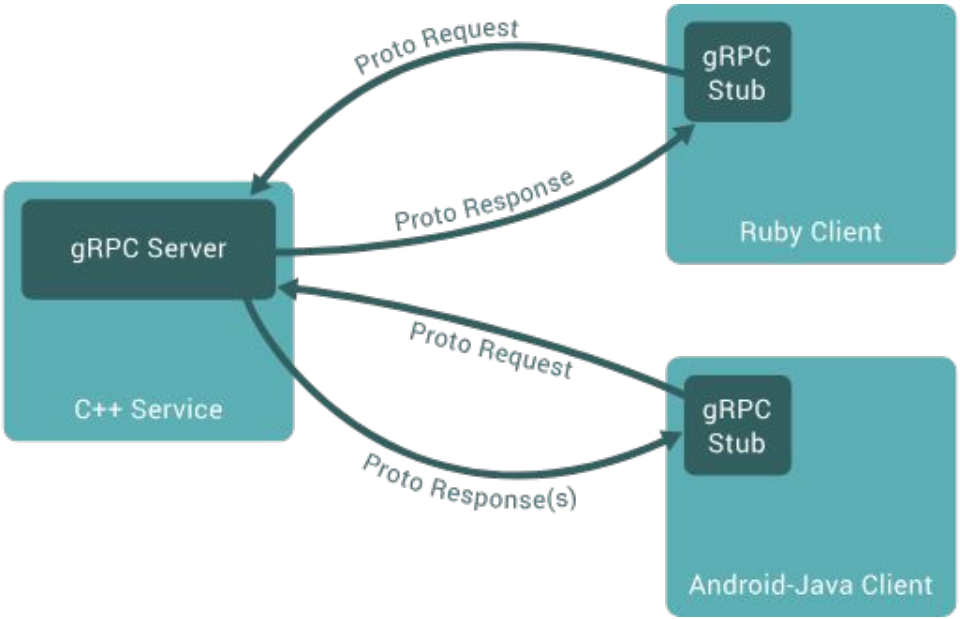



```html
<html>

<head>
    <meta charset="utf-8" />
    <script src="wasm_exec.js"></script>
    <script>
        const go = new Go();

WebAssembly.instantiateStreaming(fetch("main.wasm"),
go.importObject).then((result) => {
            go.run(result.instance);
        });
    </script>
</head>
<body></body>
</html>
```

gRPC

gRPC Remote Procedure Calls (gRPC)

gRPC — это опенсорсный фреймворк для удаленного вызова процедур. В качестве транспорта используется HTTP/2, в качестве языка описания интерфейса — Protocol Buffers.



gRPC – зачем?

- Писать клиенты на разных языках
- Возможность обрабатывать таймауты запросов клиентов
- Поддержка синхронхронных и асинхронных запросов одновременно
- Оптимальное бинарное кодирование сообщений запросов и ответов

Установка

```
python -m pip install grpcio grpcio-tools
```

https://grpc.io/docs/languages/python/quickstart/

helloworld.proto

- HelloRequest
- HelloReply
- Greeter
  - SayHello

```proto
syntax = "proto3";

package helloworld;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}

  rpc SayHelloStreamReply (HelloRequest) returns (stream HelloReply) {}

  rpc SayHelloBidiStream (stream HelloRequest) returns (stream HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Пишем клиент

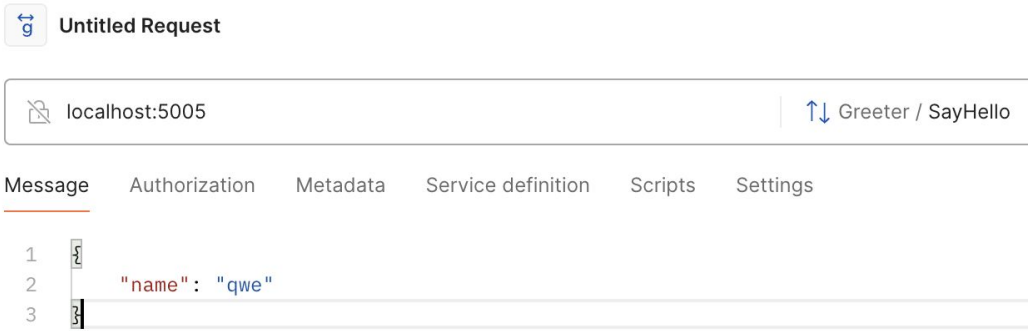
Генерация де- и сериализатора: helloworld_pb2.py и grpc классов: helloworld_pb2_grpc.py

```
python3 -m grpc_tools.protoc -I. --python_out=.
--grpc_python_out=. helloworld.proto
```

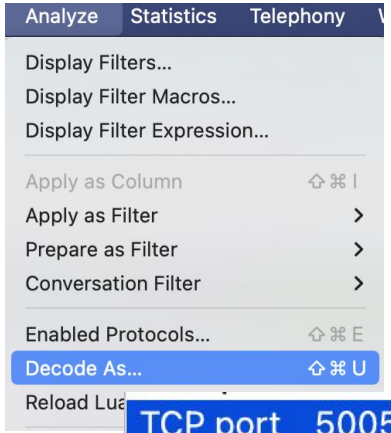
```
import helloworld_pb2
import helloworld_pb2_grpc
with grpc.insecure_channel("localhost:5005") as channel:
        stub = helloworld_pb2_grpc.GreeterStub(channel)
        response = stub.SayHello(helloworld_pb2.HelloRequest(name="admin"))
    print("Greeter client received: " + response.message)
```

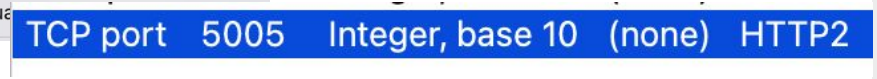
Взаимодействуем с сервером

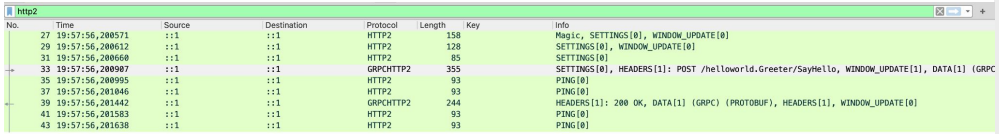
Postman

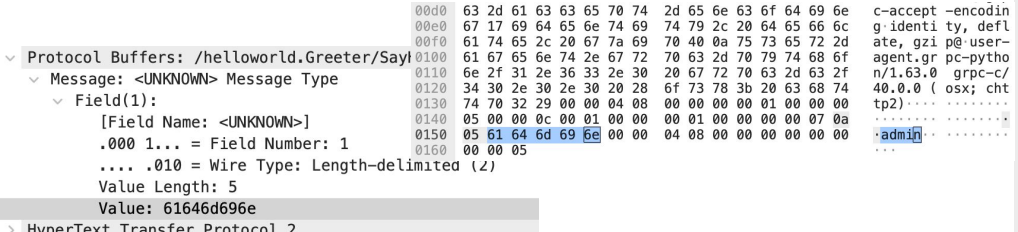


Клиент - сервер трафик










```
00000000  50 52 49 20 2a 20 48 54  54 50 2f 32 2e 30 0d 0a   PRI * HT TP/2.0..
00000010  0d 0a 53 4d 0d 0a 0d 0a  00 00 24 04 00 00 00 00   ..SM.... ..$.....
00000020  00 00 02 00 00 00 00 00  03 7f ff ff ff 00 04 00   ........ ........
00000030  40 00 00 00 05 00 40 00  00 00 06 00 00 40 00 fe   @.....@. .....@..
00000040  03 00 00 00 01 00 00 04  08 00 00 00 00 00 3f      ........ .......?
00000050  00 01                                             ..
    00000000  00 00 1e 04 00 00 00 00  00 00 03 7f ff ff ff 00   ........ ........
    00000010  04 00 40 00 00 00 05 00  40 00 00 00 06 00 00 40   ..@..... @......@
    00000020  00 fe 03 00 00 00 01 00  00 04 08 00 00 00 00 00   ........ ........
    00000030  00 3f 00 01                                       .?..
    00000034  00 00 00 04 01 00 00 00  00                       ........ .
00000052  00 00 00 04 01 00 00 00  00 00 00 d6 01 04 00 00   ........ ........
00000062  00 01 40 05 3a 70 61 74  68 1c 2f 68 65 6c 6c 6f   ..@.:pat h./hello
00000072  77 6f 72 6c 64 2e 47 72  65 65 74 65 72 2f 53 61   world.Gr eeter/Sa
00000082  79 48 65 6c 6c 6f 40 0a  3a 61 75 74 68 6f 72 69   yHello@. :authori
00000092  74 79 0e 6c 6f 63 61 6c  68 6f 73 74 3a 35 30 30   ty.local host:500
000000A2  35 83 86 40 0c 63 6f 6e  74 65 6e 74 2d 74 79 70   5..@.con tent-typ
000000B2  65 10 61 70 70 6c 69 63  61 74 69 6f 6e 2f 67 72   e.applic ation/gr
000000C2  70 63 40 02 74 65 08 74  72 61 69 6c 65 72 73 40   pc@.te.t railers@
000000D2  14 67 72 70 63 2d 61 63  63 65 70 74 2d 65 6e 63   .grpc-ac cept-enc
000000E2  6f 64 69 6e 67 17 69 64  65 6e 74 69 74 79 2c 20   oding.id entity,
000000F2  64 65 66 6c 61 74 65 2c  20 67 7a 69 70 40 0a 75   deflate,  gzip@.u
00000102  73 65 72 2d 61 67 65 6e  74 2e 67 72 70 63 2d 70   ser-agen t.grpc-p
00000112  79 74 68 6f 6e 2f 31 2e  36 33 2e 30 20 67 72 70   ython/1. 63.0 grp
00000122  63 2d 63 2f 34 30 2e 30  2e 30 20 28 6f 73 78 3b   c-c/40.0 .0 (osx;
00000132  20 63 68 74 74 70 32 29  00 00 04 08 00 00 00 00    chttp2) ........
00000142  01 00 00 00 05 00 00 0c  00 01 00 00 00 01 00 00   ........ ........
00000152  00 00 07 0a 05 61 64 6d  69 6e 00 00 04 08 00 00   .....adm in......
00000162  00 00 00 00 00 00 05                               .......
    0000003D  00 00 08 06 00 00 00 00  00 44 e7 cc 9d 14 d4 99   ........ .D......
    0000004D  3e                                              >
00000169  00 00 08 06 01 00 00 00  00 44 e7 cc 9d 14 d4 99   ........ .D......
00000179  3e                                                  >
    0000004E  00 00 4e 01 04 00 00 00  01 88 40 0c 63 6f 6e 74   ..N..... ..@.cont
    0000005E  65 6e 74 2d 74 79 70 65  10 61 70 70 6c 69 63 61   ent-type .applica
    0000006E  74 69 6f 6e 2f 67 72 70  63 40 14 67 72 70 63 2d   tion/grp c@.grpc-
    0000007E  61 63 63 65 70 74 2d 65  6e 63 6f 64 69 6e 67 17   accept-e ncoding.
    0000008E  69 64 65 6e 74 69 74 79  2c 20 64 65 66 6c 61 74   identity , deflat
    0000009E  65 2c 20 67 7a 69 70 00  00 14 00 00 00 00 00 01   e, gzip. ........
    000000AE  00 00 00 00 0f 0a 0d 48  65 6c 6c 6f 2c 20 61 64   .......H ello, ad
    000000BE  6d 69 6e 21 00 00 1e 01  05 00 00 00 01 40 0b 67   min!.... .....@.g
    000000CE  72 70 63 2d 73 74 61 74  75 73 01 30 00 0c 67 72   rpc-stat us.0..gr
    000000DE  70 63 2d 6d 65 73 73 61  67 65 00 00 00 04 08 00   pc-messa ge......
    000000EE  00 00 00 00 00 00 00 0c                           ........
0000017A  00 00 08 06 00 00 00 00  00 fb cb d7 b6 33 c9 70   ........ .....3.p
0000018A  92                                                  .
    000000F6  00 00 08 06 01 00 00 00  00 fb cb d7 b6 33 c9 70   ........ .....3.p
    00000106  92                                              .
```

Протоколы аутентификации и авторизации

Аутентификация vs Авторизация

- Аутентификация — это процесс проверки личности пользователя
- Авторизация — это процесс проверки того, к чему у пользователя есть доступ

Аутентификация	**Авторизация**
Просит пользователя подтвердить учетные данные (пароль, биометрия, …)	Проверяет, разрешен ли доступ с помощью политик и правил
Обычно выполняется до авторизации	Обычно выполняется после успешной аутентификации
Обычно передает информацию через токен идентификатора (ID)	Обычно передает информацию через токен доступа (Access Token)
Обычно регулируется протоколом OpenID Connect	Обычно регулируется протоколом OAuth 2.0.

JSON Web Token (JWT)

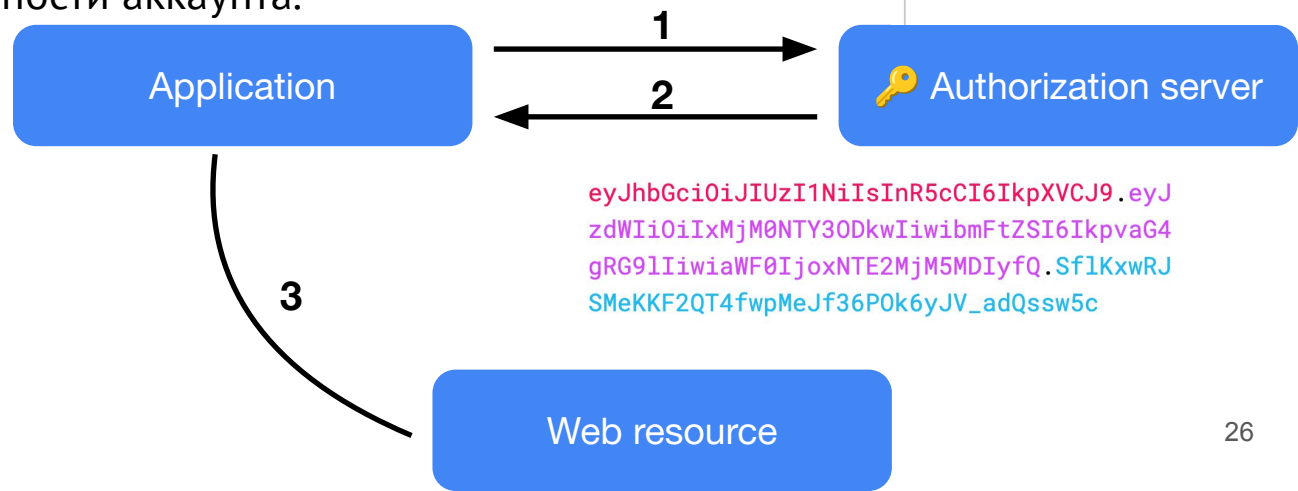
JWT — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.
Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует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

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```



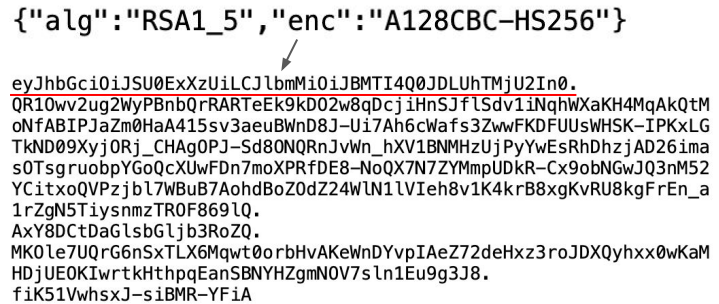

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJ
SMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

https://jwt.io/
RFC 7519

JWT, JWS, JWE, JWK, JWA

https://www.loginradius.com/blog/engineering/guest-post/what-are-jwt-jws-jwe-jwk-jwa/

JWE – JSON Web Encryption
https://jwcrypto.readthedocs.io/en/latest/jwe.html
https://github.com/panva/jose?tab=readme-ov-file
https://www.rfc-editor.org/rfc/rfc7519

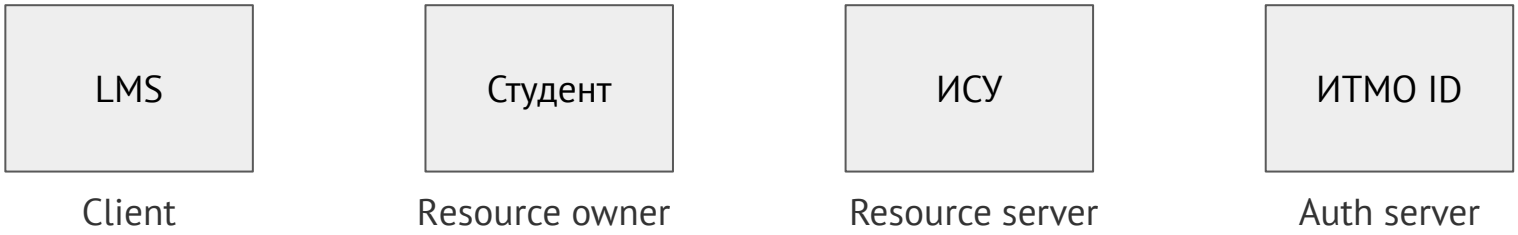


OAuth 2.0

LMS хочет анализировать ваши оценки в ИСУ по различным предметам, чтобы на их основе предлагать пройти дополнительные онлайн курсы

Как дать доступ?

- Попросить у вас логин и пароль от ИСУ? 😱
- Договориться с разработчиками ИСУ, чтобы они внесли в интерфейс галочку «разрешать доступ для LMS»? 🫣



1 шаг – зарегистрировать клиента на Auth Server

Подаем официальную заявку в ИТМО на подключение нового ресурса к ИТМО ID.

- Название приложения – «LMS2.0»
- Адрес сайта – lms.itmo.xyz
- Логотип – 
- Redirect URL – https://lms.itmo.xyz/oauth/callback

После регистрации получаем от сервиса:

- client_id
- client_secret

2 шаг – получить access_token

Основные виды грантов (grants):

- Client Credentials – когда мы запрашиваем доступ к своим ресурсам по client_id и client_secret
- Authorization Code – типовая схема для Веб приложений, в рамках которого пользователь соглашается на предоставление доступа

Устаревший:

- Implicit flow – аналог Authorization Code, но без подтверждения клиентом получения кода

Client Credentials

1. Регистрируем в ИТМО ID свое приложение (LMS2.0). Получаем от ИТМО ID случайные **client_id** и **client_secret**
2. Используем для аутентификации своего приложения в ИТМО ID сервисе
3. Например, меняем аватарку нашего приложения или получаем список пользователей, которые авторизовались у нас

```
POST /token HTTP/1.1
Host: id.itmo.ru

grant_type=client_credentials
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{"access_token":"MTQ0N<...>ZjNGZmZjI3",
  "token_type":"Bearer",
  "Expires_in":3600,
  "refresh_token":"IwOGYzY<...>YNTVk",
  "scope":"create"}
```

https://www.oauth.com/oauth2-servers/access-tokens/client-credentials/

3 шаг – использовать access_token для доступа

- **access_token** – его прикладываем к запросам
- expires_in (recommended) – после этой даты придется заново авторизоваться
- refresh_token (optional) – дополнительный токен, который живет дольше и через который можно обновить access_token проще чем заново авторизовываться
- scope – набор разрешенных операций (например, «create,update»)

```
curl -H "Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia" \
 https://api.authorization-server.com/1/me
```

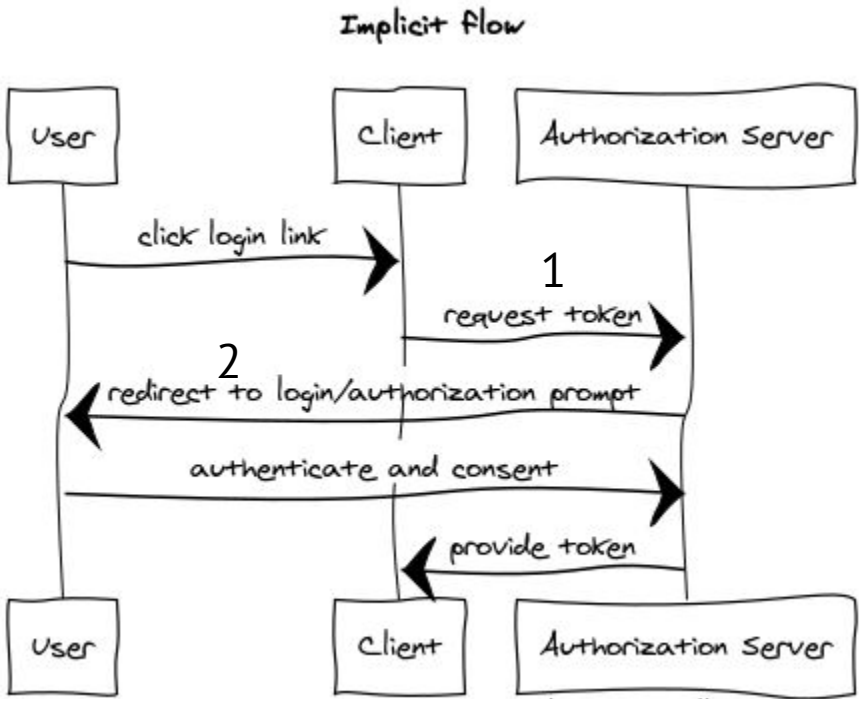
https://www.oauth.com/oauth2-servers/access-tokens/access-token-response/

Implicit flow (устарело)

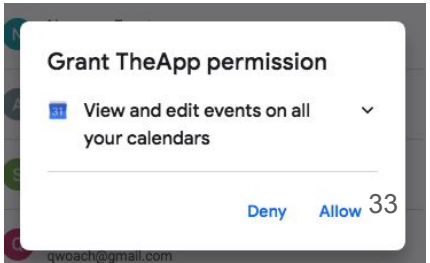
1. Формируем ссылку для логина

(+генерируем и кладем в Cookie случайный **state**)

```
https://id.isu.ru/authorize?
  response_type=token
  &scope=photo
  &state=kUcu<..random..>kN
  &client_id=eD1<...>Ddra
  &redirect_uri=https://lms.itmo.xyz/callback
```



2. У пользователя открывается окно с запросом подтверждения

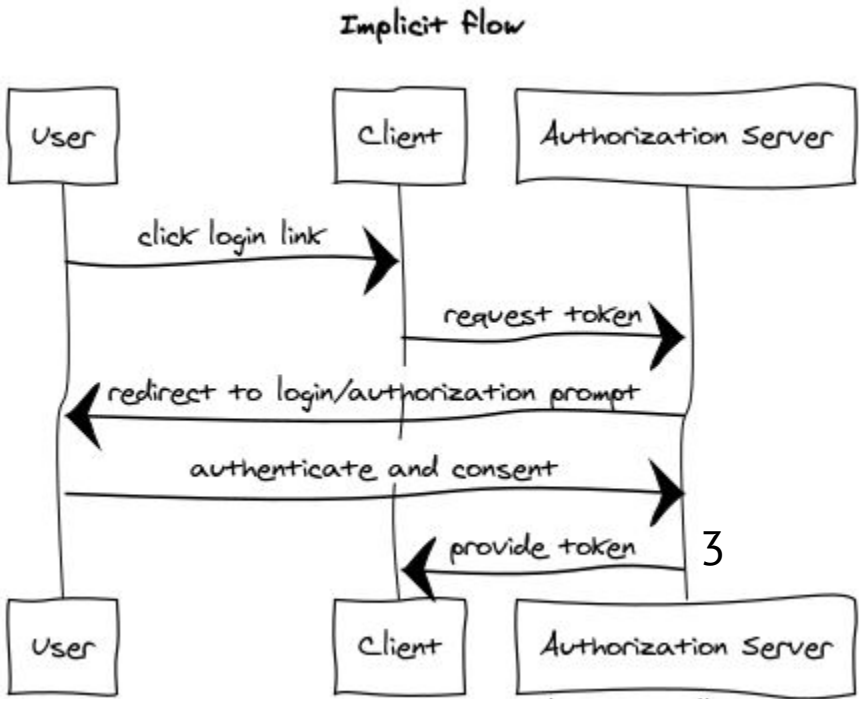


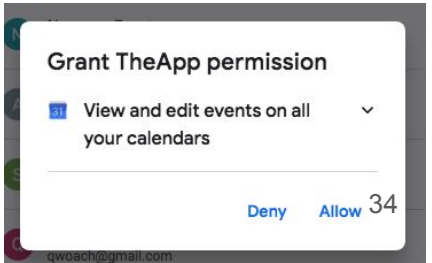
Implicit flow (устарело)

3. Приложение редиректит пользователя на redirect_uri=https://lms.itmo.xyz/callback с fragment (#)

```
https://lms.itmo.xyz/callback#access_token=dQ8bZSC49B2kZnCgSzC5wYBOANCF3h-SgjDf59hhm0Y7ukkz6b4YxwTi0jQqTHwDf2_UIXaZ&token_type=Bearer&expires_in=86400&scope=photos&state=lgZFqQ8QCWp84suZ
```

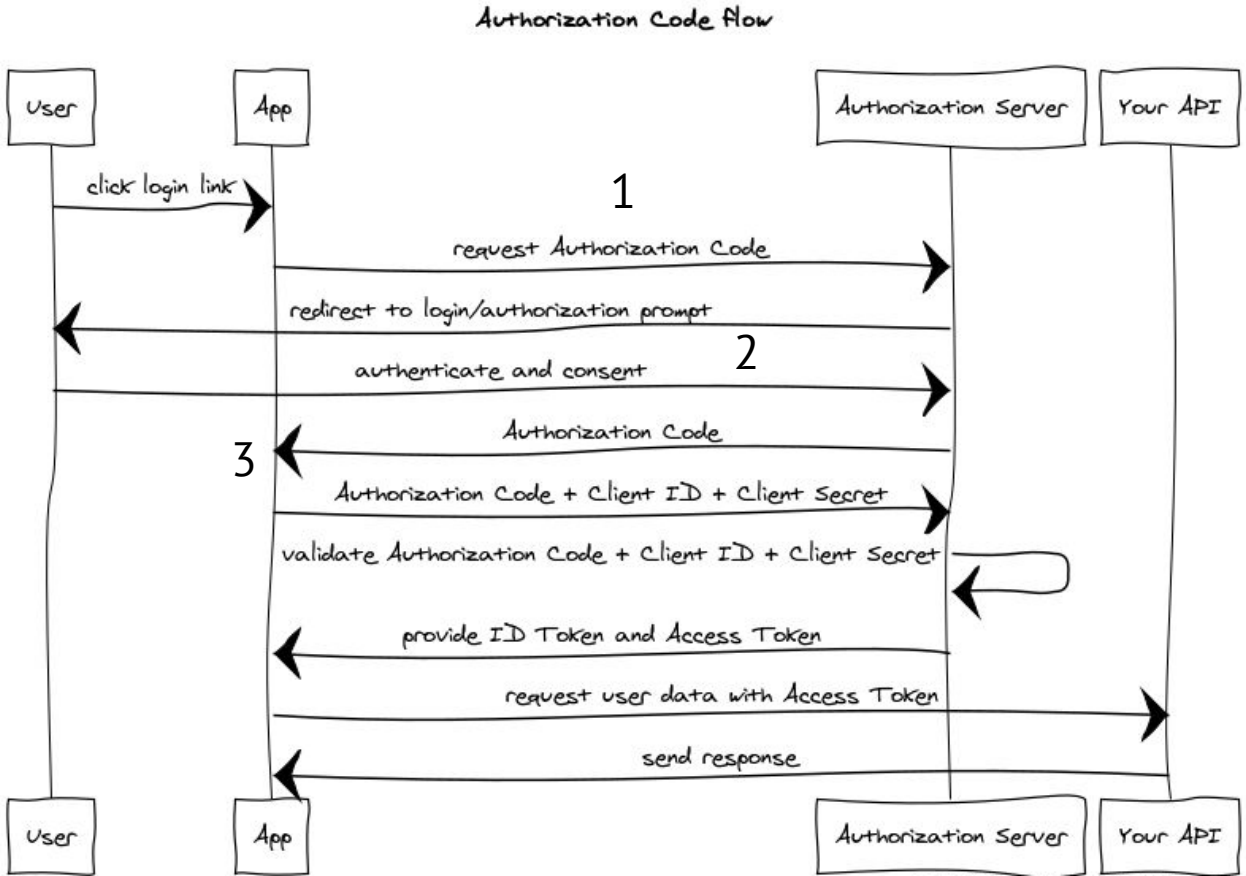
+ проверяем, что state совпадает с тем, который в Cookie (CSRF)





https://www.oauth.com/playground/implicit.html

Authorization Code flow



Authorization Code flow

1. Опять собираем ссылку для редиректа

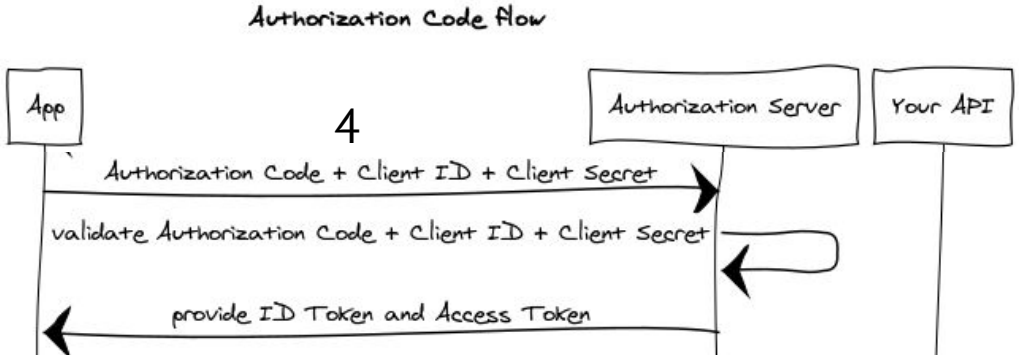
```
https://id.isu.ru/authorize?
  response_type=code
  &scope=photo+name
  &state=kUcu<..random..>kN
  &client_id=eD1<...>Ddra
  &redirect_uri=https://lms.itmo.xyz/callback
```

2. Клиент авторизуется и его редиректит к нам

Authorization Code flow

```
https://lms.itmo.xyz/callback?state=BRWQKYKKPrd1s65B&code=vp
0NF4trAH6cSKY93k379EAD9ob1FvOzQJKQMNARbojXG7X6
```

3. Проверяем state и видим **code**
4. Теперь **code** нужно обменять на access_token (уже не клиентский, а **серверный** запрос)




```
POST https://id.itmo.ru/token

grant_type=authorization_code
&client_id=eD1<...>HDdra
&client_secret=pH9LvQ<...>Tu4Zhh9
&redirect_uri=https://lms.itmo.xyz/callback
&code=vp0NF4<...>7X6
```

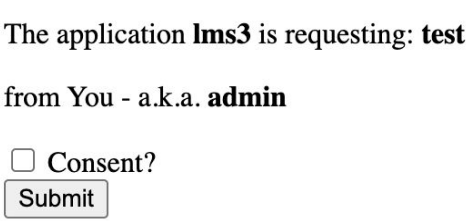
Authorization Code flow

В ответ придет access_token, refresh_token и .т.п

```
POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=refresh_token
&refresh_token=xxxxxxxxxxx
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx
```

https://www.oauth.com/oauth2-servers/access-tokens/access-token-response/

Демо cURL & Postman

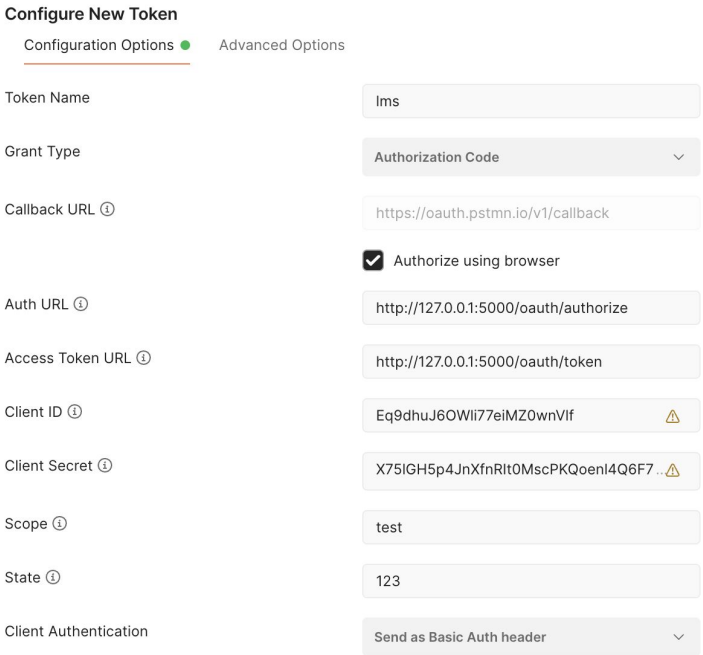


Client Info client_id: Eq9dhuJ6OWli77eiMZ0wnVlf **client_secret:** X75lGH5p4JnXfnRIt0MscPKQoenl4Q6F7woSSiy9DHRtMUSo **client_id_issued_at:** 1715626925 **client_secret_expires_at:** 0 **Client Metadata client_name:** lms3 **client_uri:** https://oauth.pstmn.io/ **grant_types:** ['authorization_code'] **redirect_uris:** ['https://oauth.pstmn.io/v1/callback'] **response_types:** ['code'] **scope:** test **token_endpoint_auth_method:** client_secret_basic

```
curl
'http://127.0.0.1:5000/oauth/authorize? response
_type=code&client_id=Eq9d<...>lf&scope=test&sta
te=123&redirect_uri=https://oauth.pstmn.io/v1/c
allback' -H 'Cookie: session=eyJpZC<...>9Xc'
```

```
curl -X POST --data "confirm=on" ... -vvv
```

Location:
https://oauth.pstmn.io/v1/callback?code=lz
IssR<...>4LjjS&state=123



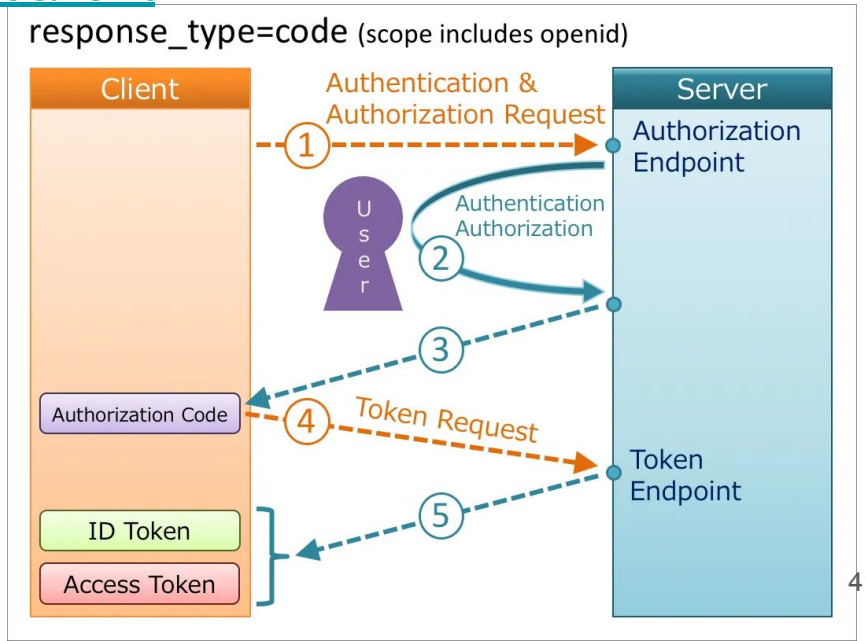
Обмениваем code на token и используем token

```
curl -u
Eq9dhuJ6OWli77eiMZ0wnVlf:X751GH5p4JnXfnRIt0MscPKQoenl4Q6F7woSSiy9
DHRtMUSo -XPOST "http://127.0.0.1:5000/oauth/token?" -F
"redirect_uri=https://oauth.pstmn.io/v1/callback" -F
grant_type=authorization_code -F scope=profile -F
code=sgzCuaHo7eWPmU8Vppy2VJJmxPwoHpVMDPckaUJR8lHKQpTL

{"access_token": "zn7yLau4zb929xF4jEeFQ8wuIoMeRAWOk95xJo211I",
"expires_in": 864000, "scope": "test", "token_type": "Bearer"}

curl -H "Authorization: Bearer
zn7yLau4zb929xF4jEeFQ8wuIoMeRAWOk95xJo211I"
http://127.0.0.1:5000/api/me
```

Аутентификация

OpenID Connect – тот же самый OAuth, только после успешного получения token, считаем пользователя авторизованным.

Демо: https://www.oauth.com/playground/oidc.html



Конец

Итоги лекции

1. Разобрались с протоколом бинарной сериализации **Protobuf**
2. Узнали о принципах работы **HTTP/2** и **Wasm**
3. Изучили **gRPC**
4. Освоили принципы аутентификации и авторизации в Веб с использованием **OAuth 2.0**

Итоги курса

1. История развития Web, принципы работы Web, HTML, CSS и стандарты
2. HTTP протокол, Cookie и инструменты для работы с веб протоколами
3. Хранилище данных в браузере, методы фингерпринтинга, Basic Auth, REST, прокси и форматы представления данных в Веб
4. GraphQL и протоколы передачи мультимедиа. HTTP Range Requests. Websockets и DNS
5. Техники эксплуатации некоторых уязвимостей: IDOR, LFI/RFI, Path Traversal, Command Injection, SSRF и SSTI
6. JS, Same Origin Policy, CORS, Content Security Policy и уязвимости CSRF, XSS
7. Браузерные API. WebDriver и Selenium. Браузерные расширения. Прогрессивные Веб приложения. Service Workers, Web Push API и Notifications API
8. Protobuf, Wasm, HTTP/2, Протоколы аутентификации: OAuth 2.0

Клиентские приложения

HTML
CSS
JS
Storage
Service worker
PWA
API
Wasm

Безопасность и клиентские технологии: **IDOR,LFI,RFI,Traversal, CInj,SSRF,SSTI** **CORS,SOP,Origin,CSP, CSRF,XSS**

API и форматы данных: **GraphQL** **XML API** **HTML** **JSON API** **Protobuf**

Протоколы: **HTTP** **Websockets** **gRPC** **WebRTC** **RTMP** **OAuth 2.0**

Сервер и веб-приложение

BasicAuth
Proxy

DNS

Selenium

Картина мира после курса Веб Технологии

Коротко про ДЗ и экзамен

- **Прогрессивные Веб приложения** –> *разработка браузерного расширения*
- Дополнительные таски
- Дедлайны добора баллов до 31 мая

ИТМО

Веб-программирование

ВЫ ИЗУЧИТЕ

REACT - самый популярный фреймворк для создания динамических интерфейсов

FLASK, DJANGO - фреймворки для создания многофункционального бекенда

GIT - решение для контроля версий и совместной разработки

SQL & NOSQL - базы данных для хранения информации

DOCKER - мощнейший инструмент контейнеризации приложений

Полезные ссылки

- Формат сериализации Protobuf – https://protobuf.dev/programming-guides/encoding/#simple
- Про HTTP/2 Performance – https://www.cloudflare.com/learning/performance/http2-vs-http1.1/
- Wasm Go getting started – https://github.com/golang/go/wiki/WebAssembly#getting-started
- Генерация gRPC Protobuf классов – https://grpc.io/docs/languages/python/basics/
- Разбор gRPC трафика в Wireshark – https://grpc.io/blog/wireshark/
- JWT – https://jwt.io/
- OAuth Implicit Flow – https://www.oauth.com/playground/implicit.html
- Code Flow – https://www.oauth.com/playground/authorization-code.html
- OpenID Connect Playground – https://www.oauth.com/playground/oidc.html
- Удобный self-hosted OAuth сервер на Python – https://github.com/authlib/example-oauth2-server

Вопросы?