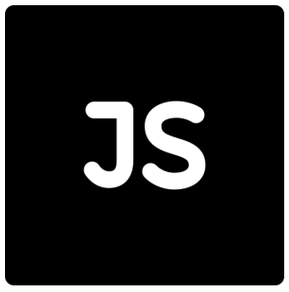
CORS

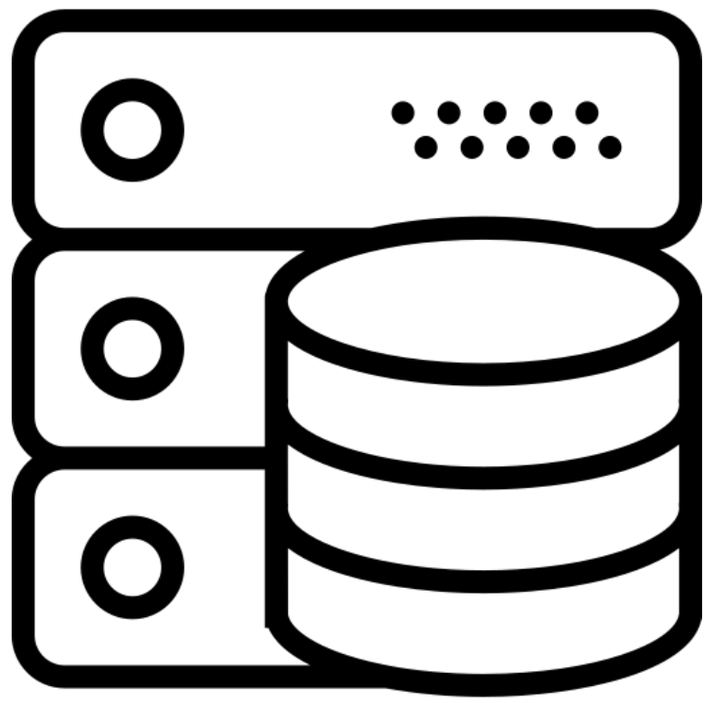
Cross-Origin Resource Sharing

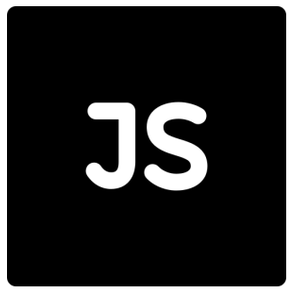
```
const url = 'http://api.ya.ru';

fetch(url);
```

```
const url = 'http://api.ya.ru';

fetch(url);
```

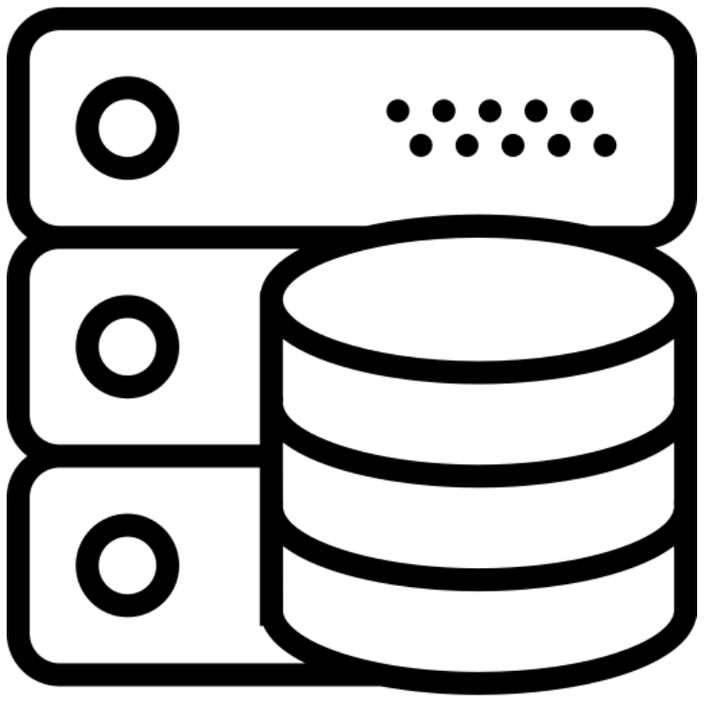
```
Access to fetch at 'http://api.ya.ru/' from origin 'http://localhost:3000' has localhost/:1
been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the
requested resource. If an opaque response serves your needs, set the request's mode to 'no-
cors' to fetch the resource with CORS disabled.
Uncaught (in promise) TypeError: Failed to fetch                                localhost/:1
```

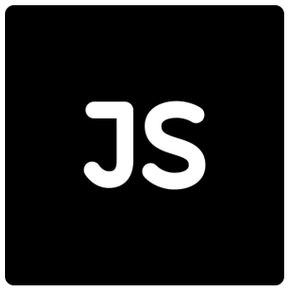

JS





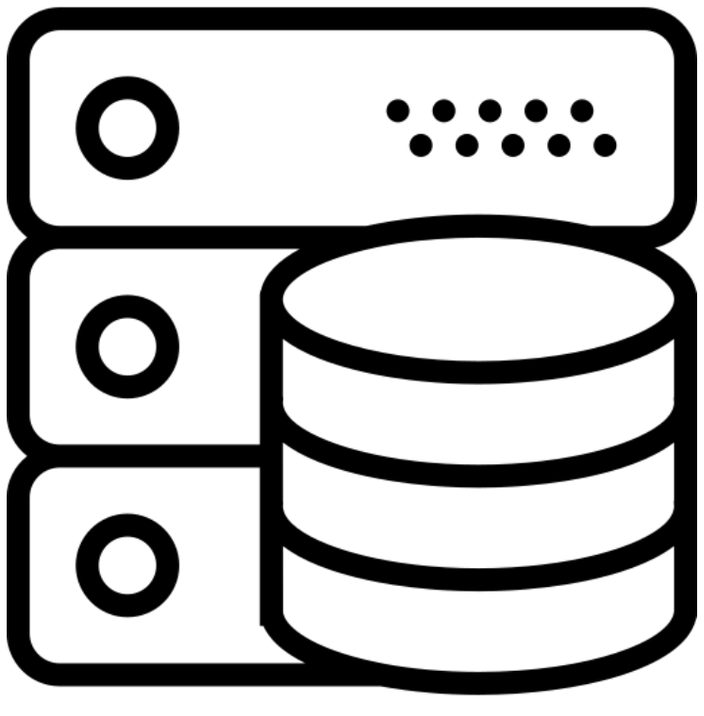
FETCH GET /data/

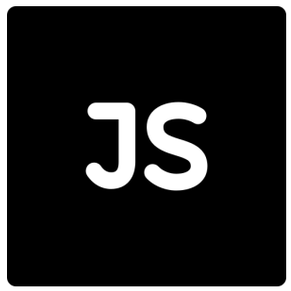




FETCH GET /data/

GET /data/
origin: http://api.ya.ru
host: google.com



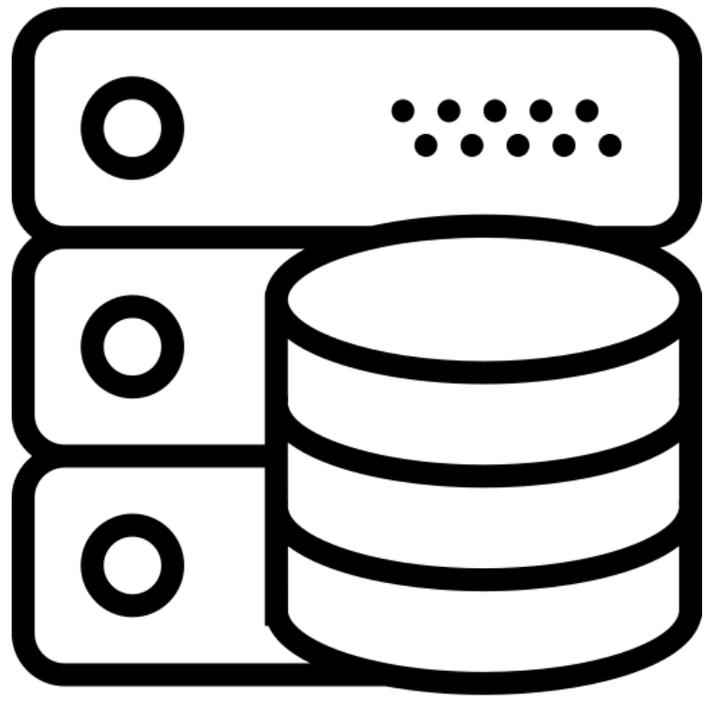


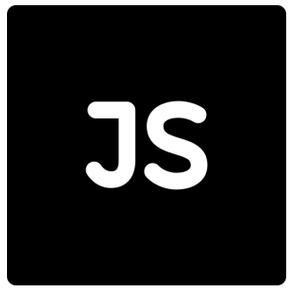
FETCH GET /data/

<ERROR>

GET /data/
origin: http://api.ya.ru
host: google.com

<RESPONSE>





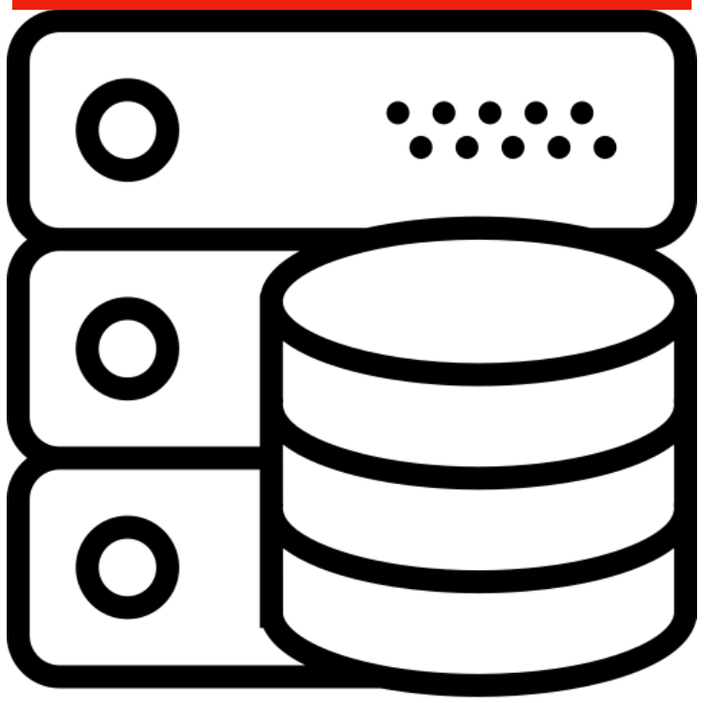
FETCH GET /data/

<ERROR>

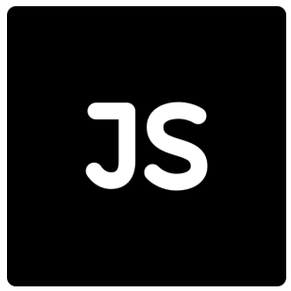
GET /data/
origin: http://api.ya.ru
host: google.com

<RESPONSE>

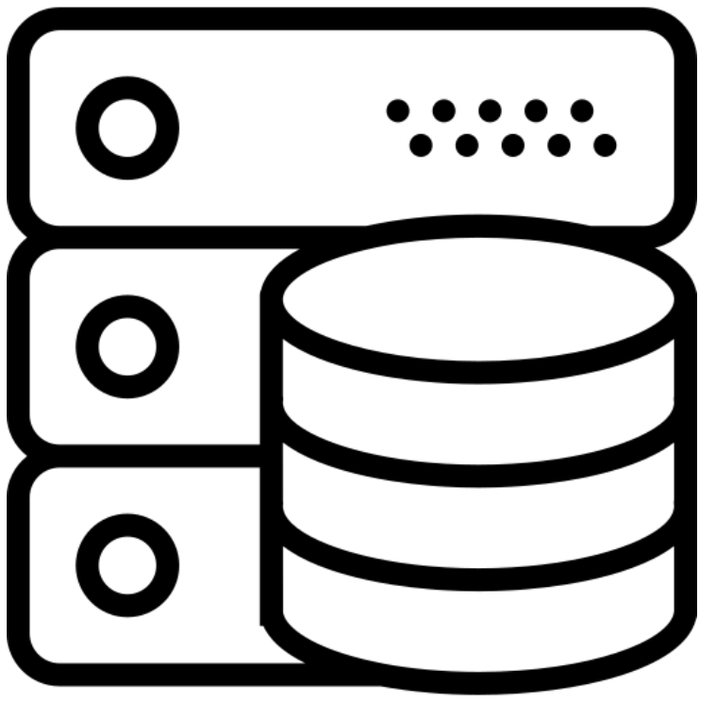
Without CORS related headers

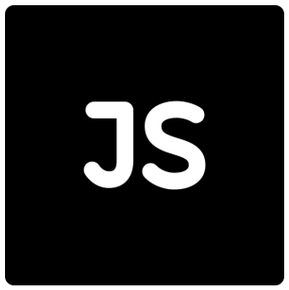


```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: http://api.ya.ru
```



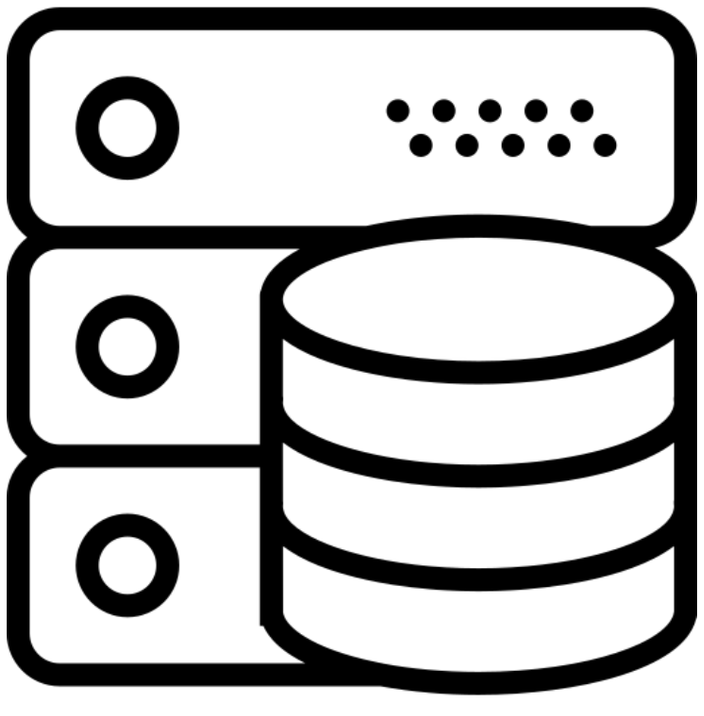
FETCH GET /data/

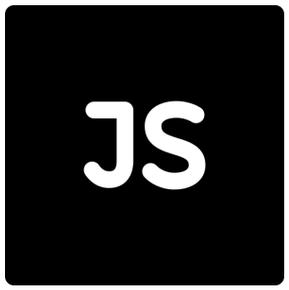




FETCH GET /data/

GET /data/
origin: http://api.ya.ru
host: google.com



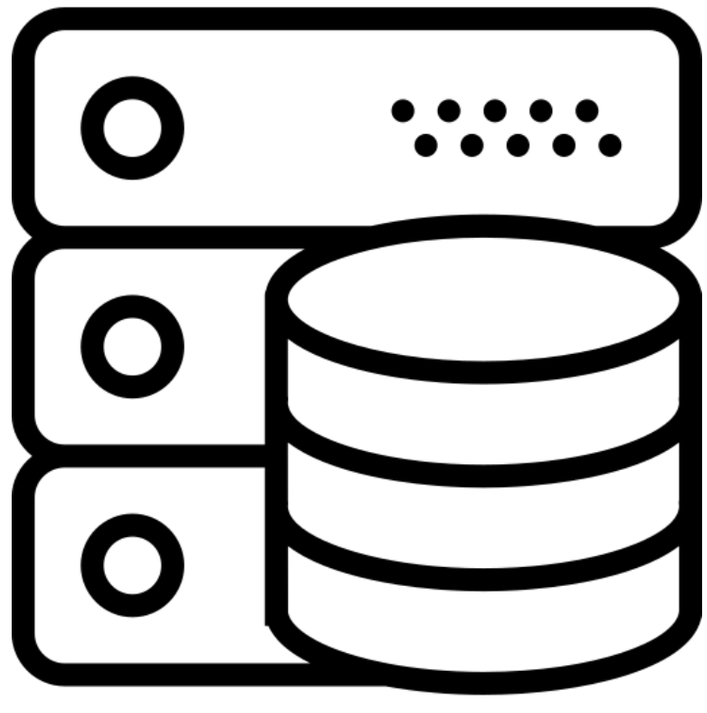


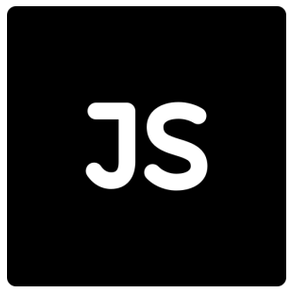
FETCH GET /data/

<CONTENT>

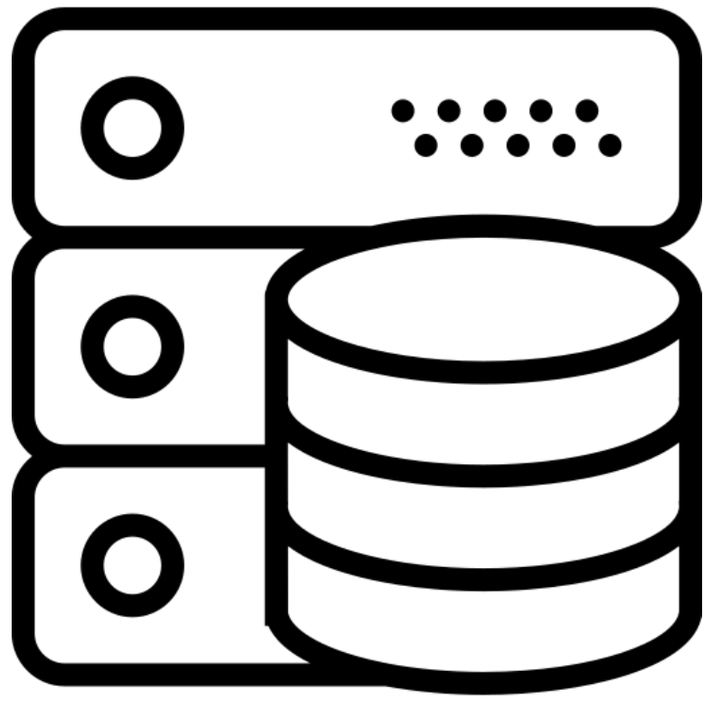
GET /data/
origin: http://api.ya.ru
host: google.com

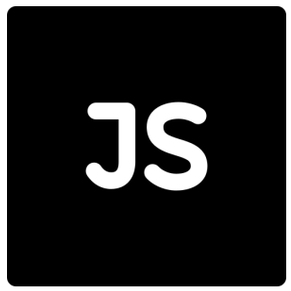
<CONTENT>
Access-Control-Allow-Origin: *





FETCH POST /data/

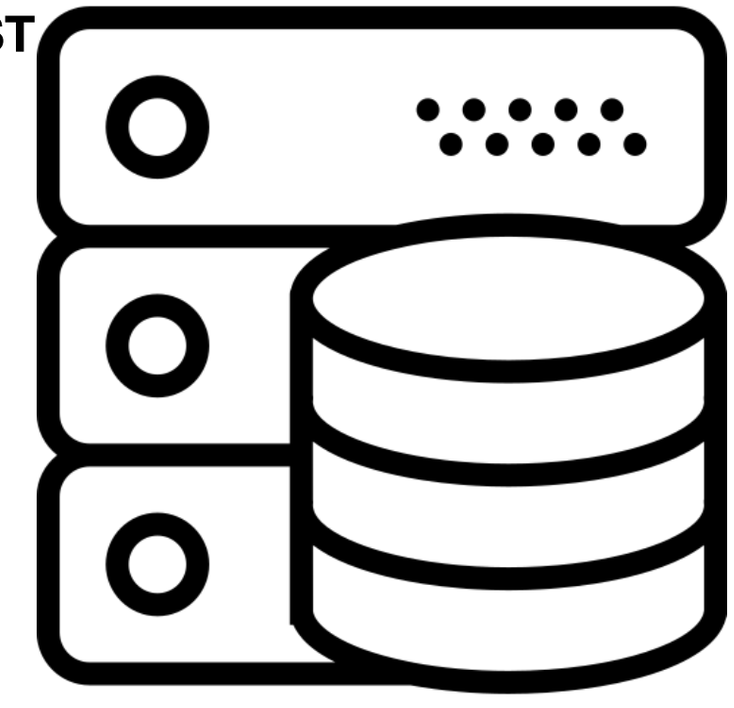


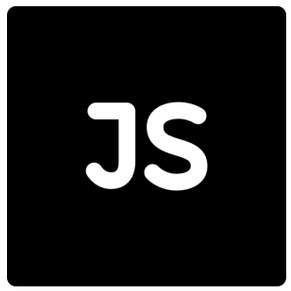


FETCH POST /data/

OPTIONS /data/

Access-Control-Request-Methods: POST



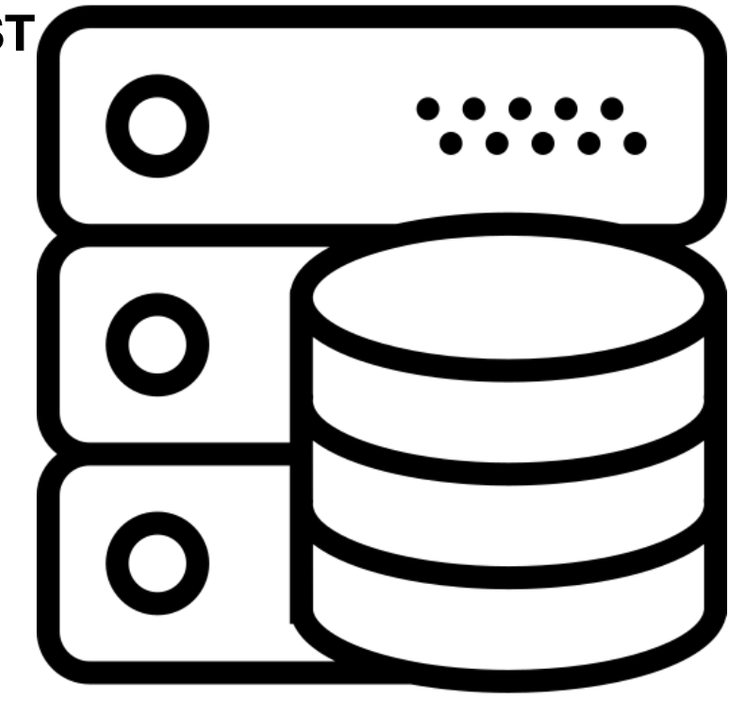


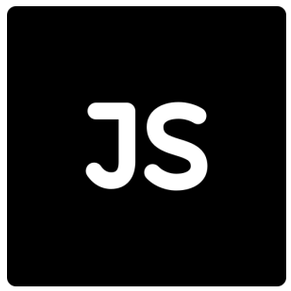
FETCH POST /data/

OPTIONS /data/

Access-Control-Request-Methods: POST

0_o???!!??!





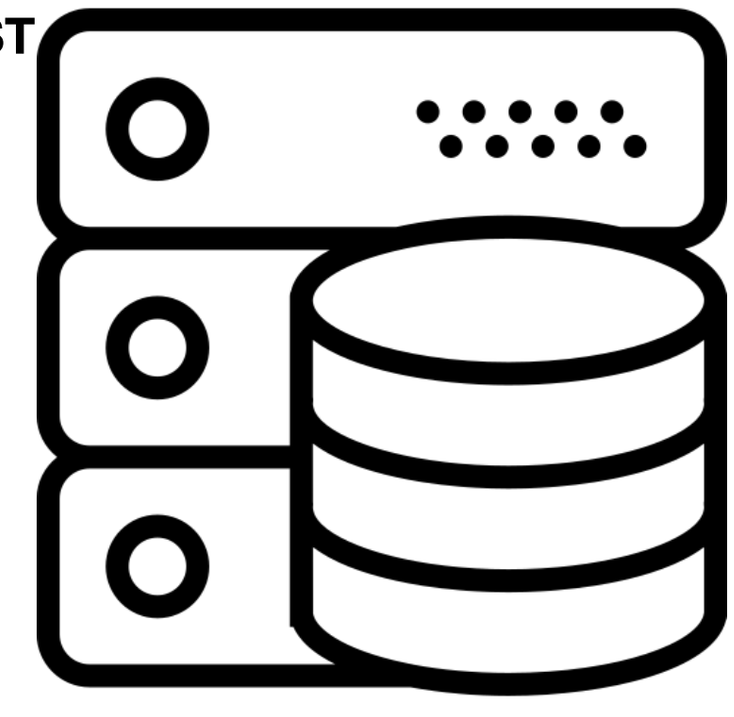
FETCH POST /data/

<ERROR>

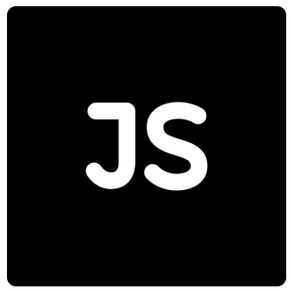
OPTIONS /data/

Access-Control-Request-Methods: POST

0_o???!!??!



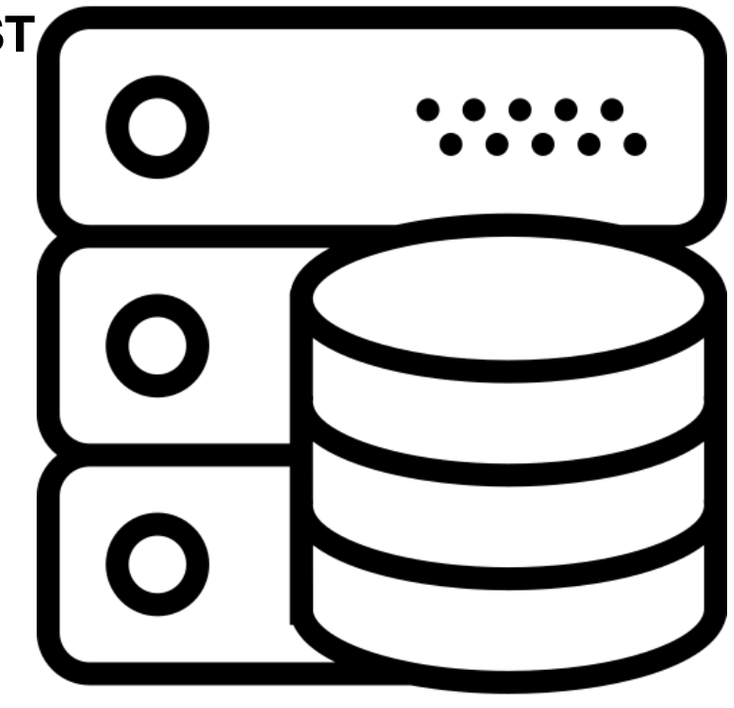
```
Access-Control-Allow-Methods: POST
Access-Control-Allow-Methods: POST, PUT, DELETE
```

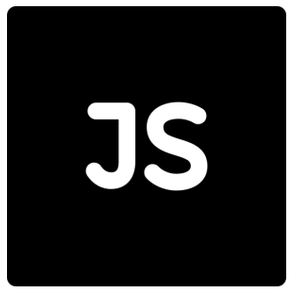


FETCH POST /data/

OPTIONS /data/

Access-Control-Request-Methods: POST



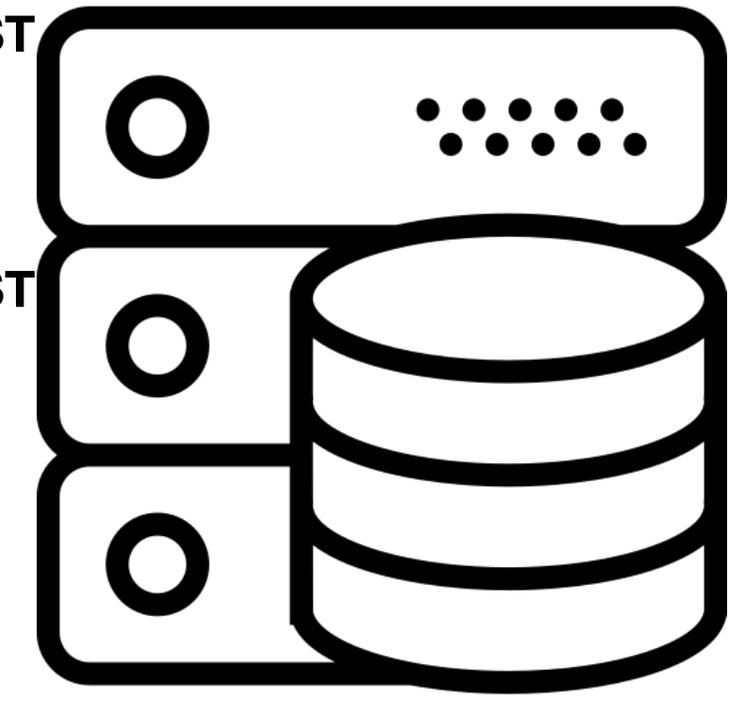


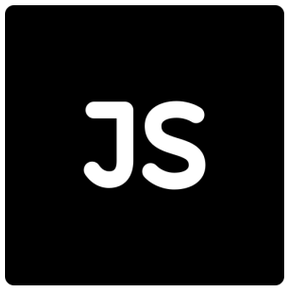
FETCH POST /data/

OPTIONS /data/

Access-Control-Request-Methods: POST

Access-Control-Request-Methods: POST





FETCH POST /data/

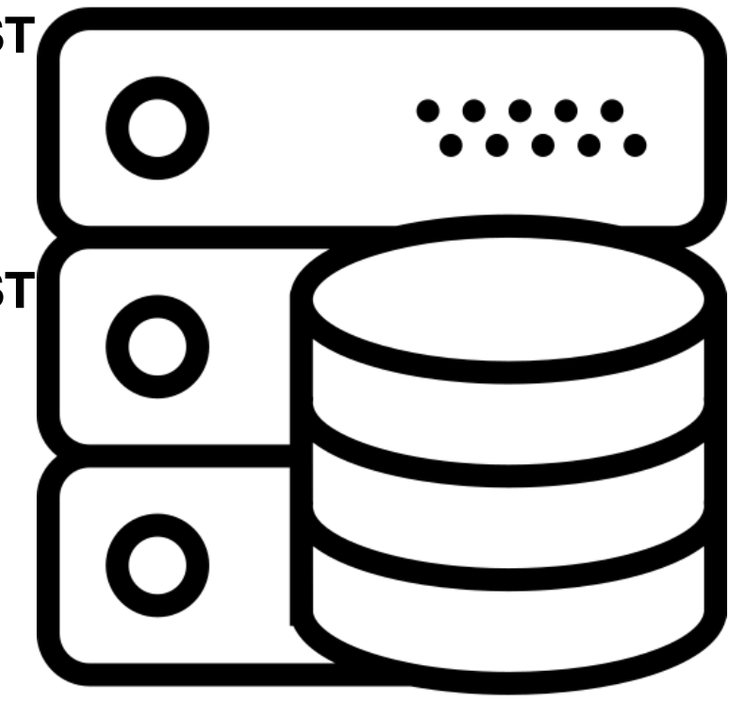
OPTIONS /data/

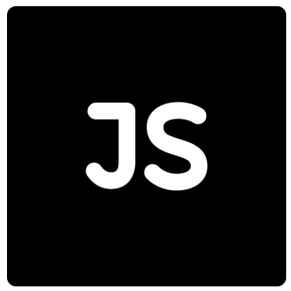
Access-Control-Request-Methods: POST

Access-Control-Request-Methods: POST

POST /data/

POST <RESULT>





FETCH POST /data/

<RESULT>

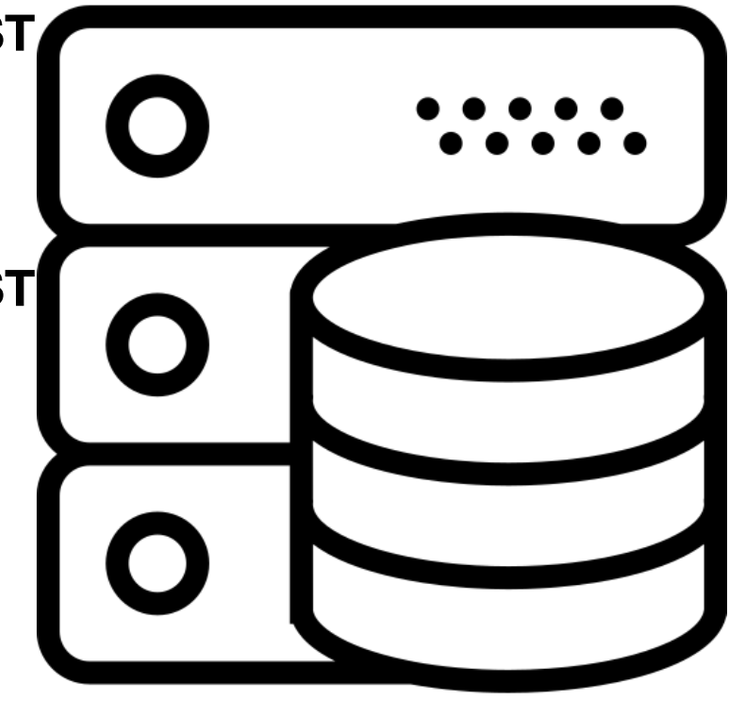
OPTIONS /data/

Access-Control-Request-Methods: POST

Access-Control-Request-Methods: POST

POST /data/

POST <RESULT>



Cookies

```
fetch(url, {
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    credentials: 'same-origin', // include, *same-origin, omit
    body: JSON.stringify(data) // body data type must match "Content-Type"
header
  });
```

```
Access-Contol-Allow-Credentials: true
```

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

CSRF

Cross Site Request Forgery

```
<form action="https://vulnerable-website.com/email/change"
method="POST">
      <input type="hidden" name="email" value="pwned@evil-user.net" />
</form>
<script>
    document.forms[0].submit();
</script>
```

CSRF token

- Если авторизация реализована через cookie
- При логине пользователю проставляется сессионная cookie
- Выпускается токен, который необходимо отправлять с каждым запросом в хедере или в теле запроса

JWT

HEADER

ALGORITHM
& TOKEN TYPE

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

+

PAYLOAD

DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

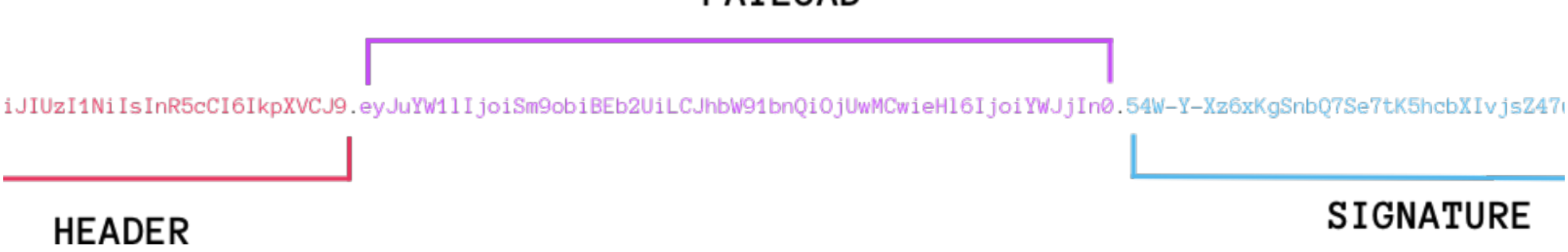
+

SIGNATURE

VERIFICATION

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),secretKey)
```

JWT TOKEN


PAYLOAD
iJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiSm9obiBEb2UiLCJhbW91bnQiOjUwMCwieHl6IjoiYWJjIn0.54W-Y-Xz6xKgSnbQ7Se7tK5hcbXIvjsZ47
HEADER
SIGNATURE

Генерируем jwt token

```
header = { "alg": "HS256", "typ": "JWT" }
encodedHeader = base64(header)

payload = { "userId": 10 }
encodedPayload = base64(payload)
```

Генерируем jwt token

```
secretServerKey = "my super secret key"
unsignedToken = encodedHeader + "." + encodedPayload

signature = sha256(unsignedToken, secretKey)
```

Генерируем jwt token

```
signedToken = encodedHeader + "." + encodedPayload + "." signature
```

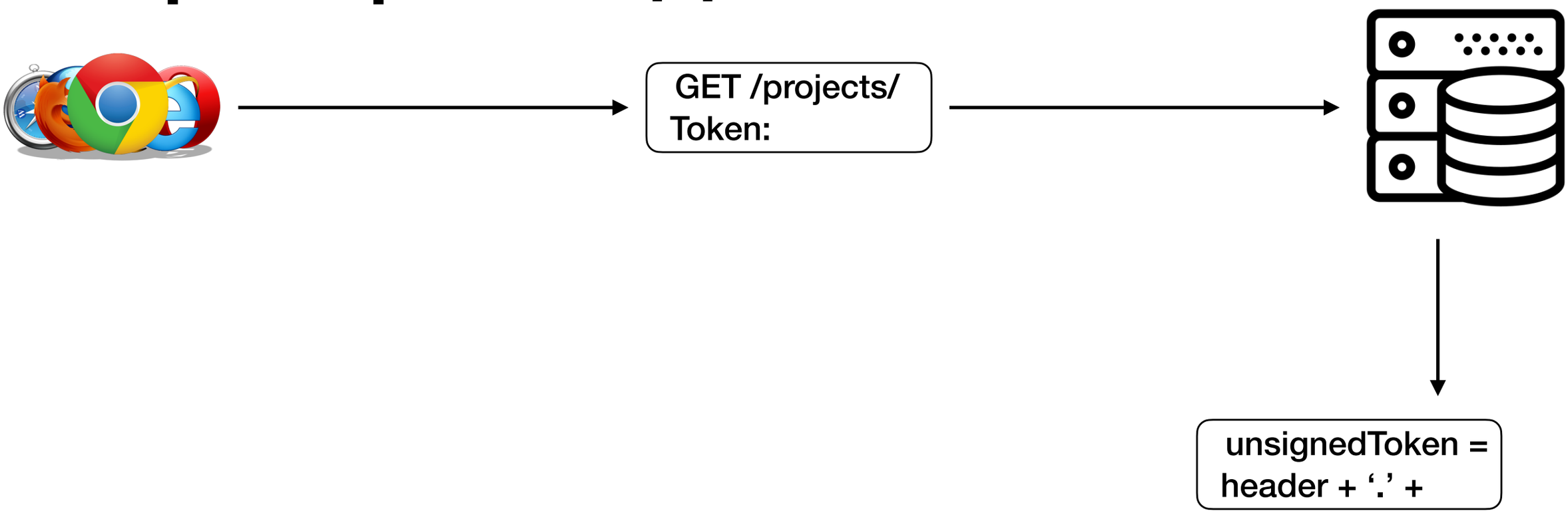
Генерируем jwt token

```
signedToken = encodedHeader + "." + encodedPayload + "." signature
```

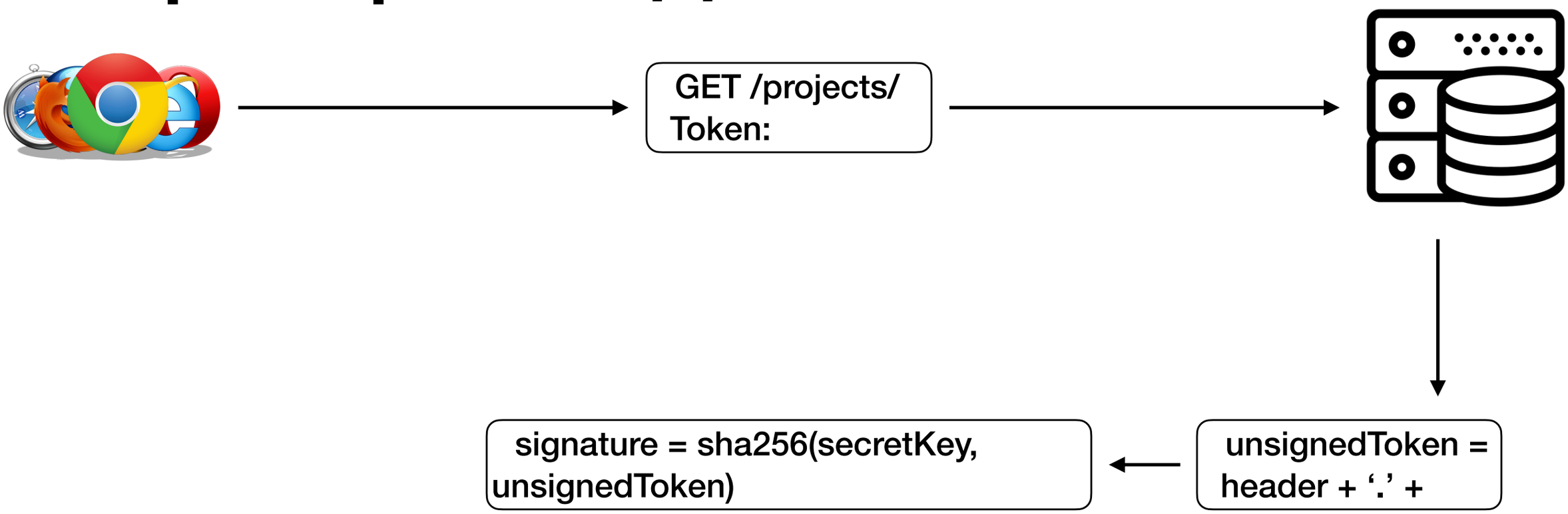
Этот токен возвращается пользователю при успешном запросе на логин. Его необходимо отправлять в качестве request header или в еле запроса в любом последующем запросе.

Проверка подлинности токена

Проверка подлинности токена

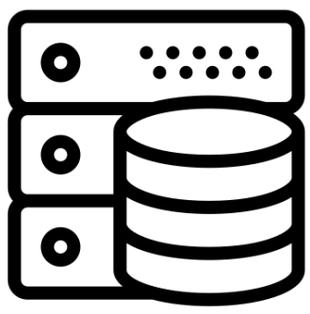


Проверка подлинности токена



Проверка подлинности токена

GET /projects/
Token:

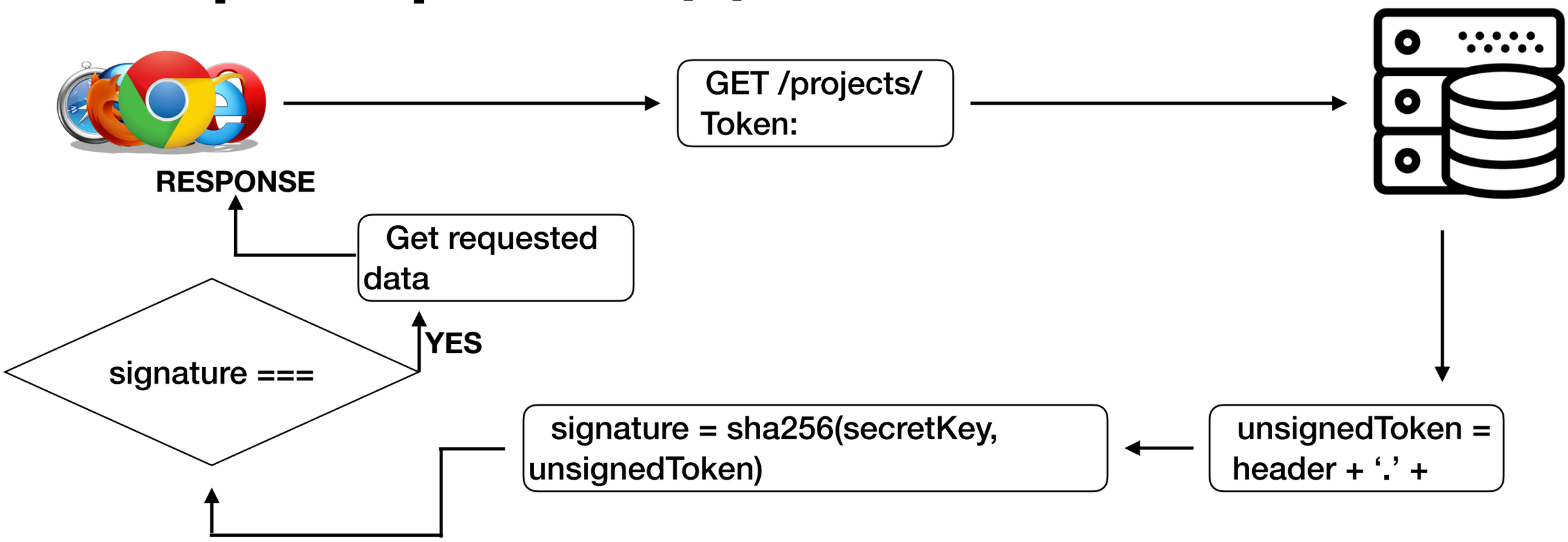


unsignedToken =
header + ‘.’ +

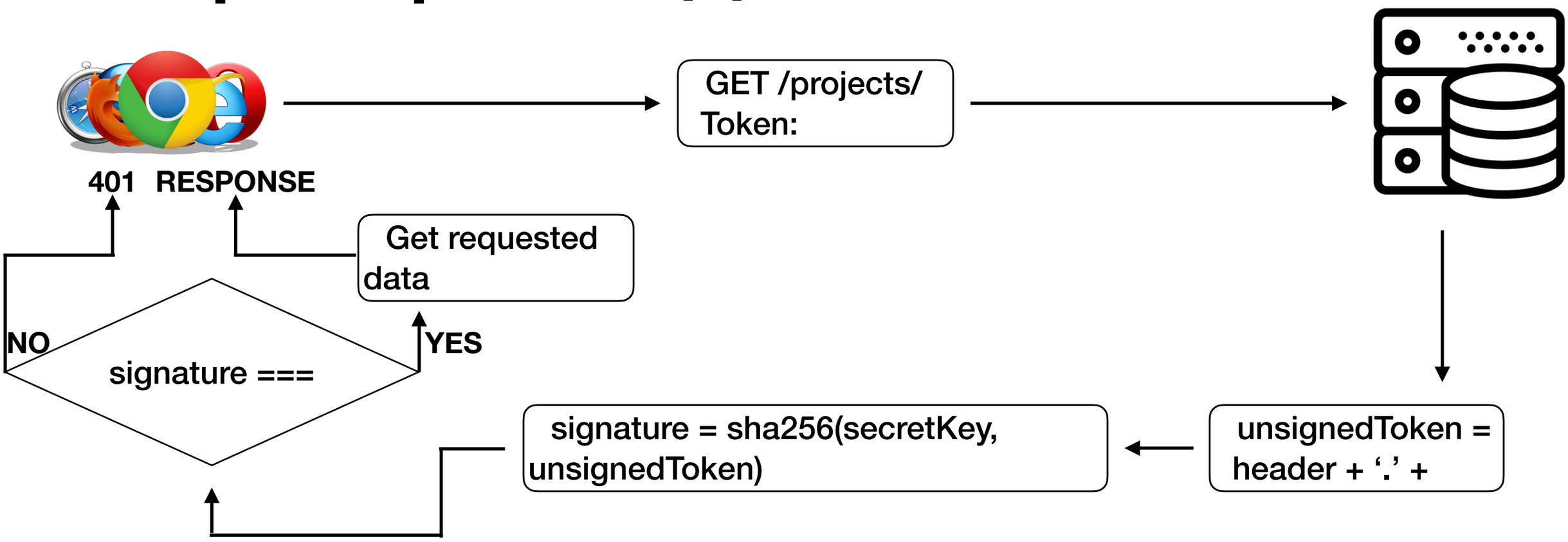
signature = sha256(secretKey,
unsignedToken)

signature ===

Проверка подлинности токена



Проверка подлинности токена



Где хранить токен после логина?

Где хранить токен после логина?

Local storage!

Где хранить токен после логина?

```
const myStorage = window.localStorage;

myStorage.setItem("nameOfTheKey", value);

const valueFromLocalStorage = myStorage.getItem("nameOfTheKey");
```