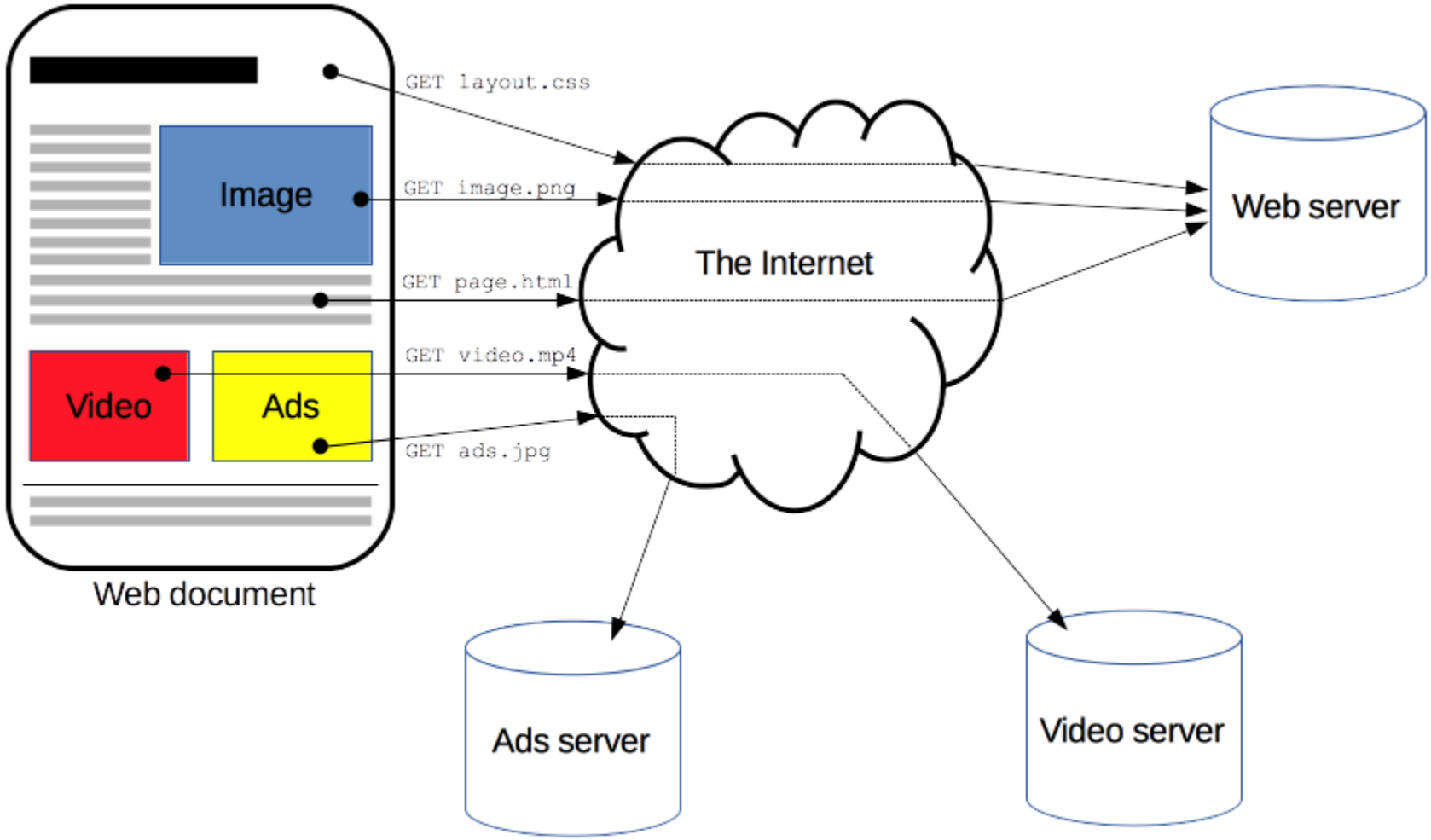
Протокол HTTP

и выполнение AJAX из браузера


GET layout.css
Image
GET image.png
GET page.html
Video
Ads
GET video.mp4
GET ads.jpg
Web document
The Internet
Web server
Ads server
Video server

Клиент-серверное взаимодействие

Request

NETWORK

Response

Clients:

- Browser
- Crawler
- Bot
- Server
- etc...

Servers:

- IIS
- Apache
- Nginx
- Tomcat
- etc…

Место HTTP в стеке TCP/IP

Application Layer: **HTTP**

Transport Layer: **TCP** (по умолчанию порт 80, но можно любой)

Internet Layer: **IP**

Network Access Layer: **Ethernet, WLAN**

Есть отличная шутка про UDP, но не факт, что она до вас дойдет.

Структура HTTP запроса

1. Starting line

```
POST /api/messages HTTP/1.0
Host: mymessenger.com
```

2. Headers

```
Content-Type: application/json
User-Agent: Mozilla/5.0
```

3. Body

```
{ "content": "hello!" }
```

Некоторые другие HTTP методы:

- GET – получение данных
- POST – создание и добавление данных
- PUT – перезапись данных
- PATCH – частичное изменение данных
- DELETE – удаление данных

Структура HTTP ответа

1. Starting line

```
HTTP/1.0 200 OK
```

2. Headers

```
Content-Type: application/json
Server: Apache/2.2.11 (Win32) PHP/5.3.0
```

3. Body

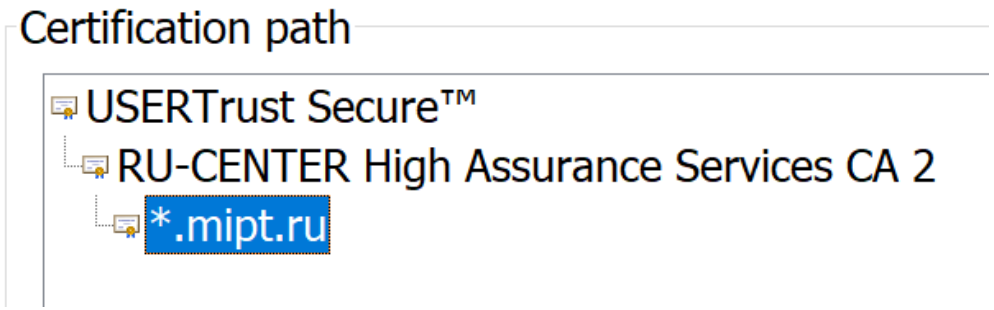
```
{ "messages": [] }
```

Другие статус коды:

- 2xx – всё хорошо
- 3xx – редирект
- 4xx – ошибка на стороне клиента
- 5xx – ошибка на стороне сервера

HTTPS

Аутентификация сервера



При помощи цепочки сертификатов

Шифрование и целостность передаваемых данных

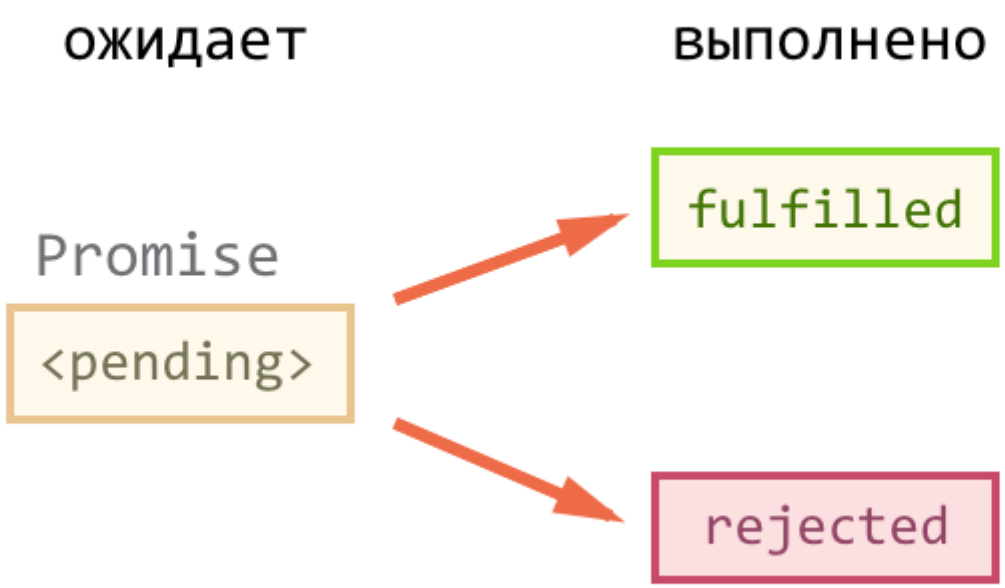
Обычно применяется протокол Диффи — Хеллмана для генерации сессионного ключа.

Но бывают и другие способы.

Promise API

способ организации асинхронного кода

Состояния Promise объекта



Синтаксис Promise

```
var promise = new Promise(function(resolve, reject) {
    // Эта функция будет вызвана автоматически.

    // В ней можно делать любые асинхронные операции,
    // А когда они завершатся — нужно вызвать одно из:
    // resolve(результат) при успешном выполнении
    // reject(ошибка) при ошибке
});
```

```
promise.then(onFulfilled, onRejected);

function onFulfilled() { /* обработчик успешного выполнения */ }
function onRejected() { /* обработчик ошибки */ }
```

Синтаксис Promise

```
let promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        // переведёт промис в состояние fulfilled с результатом "result"
        resolve("result");
    }, 1000);
});

promise.then(
        result => {
            // первая функция-обработчик - запустится при вызове resolve
            alert("Fulfilled: " + result); // result - аргумент resolve
        },
        error => {
            // вторая функция - запустится при вызове reject
            alert("Rejected: " + error); // error - аргумент reject
        });
```

Fetch API

выполнение HTTP запросов в JS коде

Синтаксис Fetch API

```
let data = { login: 'admin', password: '12345' };
let url = 'http://bank.com/login';

const responsePromise = fetch(url, {
     method: 'POST',
     headers: {
          'content-type': 'application/json'
     },
     body: JSON.stringify(data)
});
```

Цепочки Promise

```
responsePromise
     .then(function (response) {
          if (response.status >= 200 &&
               response.status <= 299) {
               // Если код ответа - 2xx,
               // то возвращаем Promise
               // с результатом в виде js объекта.
               return response.json();
          }
          else {
               // Возвращаем реджектнутый Promise.
               return Promise.reject(response.statusText);
          }
     })
     .then(function (data) {
          // do smth with data
     })
     .catch(function (error) {
          // do smth with error
     });
```

Обработчики в then могут возвращать результат, который передаётся в следующий then.

Если очередной then вернул промис, то далее по цепочке будет передан не сам этот промис, а его результат.