Webtech 2021

Движение на данни от клиент към сървър и обратно

Видове HTTP заявки към сървъра

- OPTIONS
- **GET**
- **POST**
- **PUT**
- DELETE
- TRACE
- CONNECT

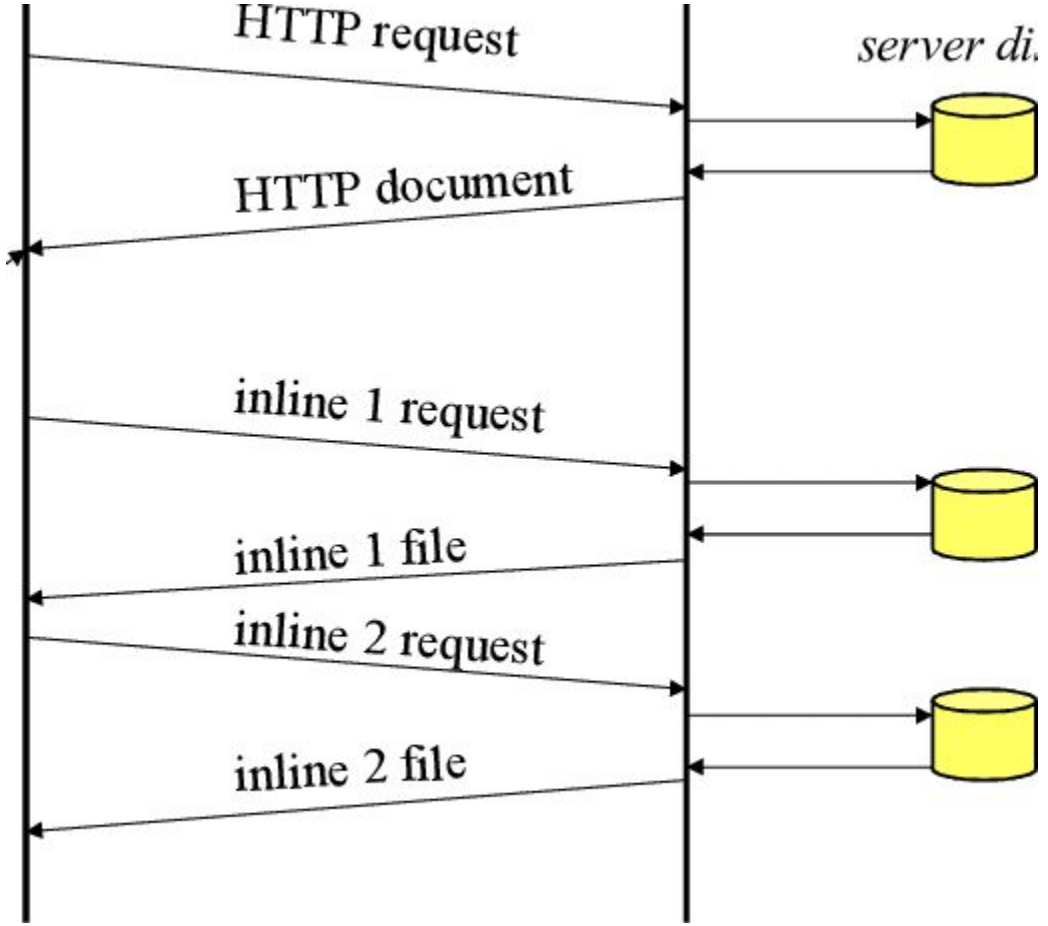
Обозначаване на резултат – status codes

- Успешни – 2xx
- Прехвърляне – 3xx (redirect)
- Грешки на протокола – 4xx
  - 404 – not found
- Сървърни грешки – 5xx
  - 500 – internal server error

Последователност при зареждане на страница с HTTP

Първата HTTP заявка зарежда HTML описанието на страницата.

Всичко останало се зарежда с последващи HTTP заявки.



Източник на картинката – неизвестен автор

Пример за статус код при зареждане на страница от Mozilla MDN

Status	Method	Domain	File	Initiator	Type	Transferred	Size
304	GET	developer.mo...	2.b0186e16.chunk.js	script	js	cached	16...
200	GET	developer.mo...	5.f4a2131e.chunk.js	runtime-main.bcb5cedd.js:1 ...	js	cached	16...
200	GET	developer.mo...	500	document	html	17.54 KB (race...	11...
304	GET	developer.mo...	bcd.json	500:1 (fetch)	json	cached	1....
200	GET	developer.mo...	favicon.ico	FaviconLoader.jsm:191 (img)	x-icon	cached	14...
200	GET	developer.mo...	favicon144.png	FaviconLoader.jsm:191 (img)	png	cached	1....
304	GET	developer.mo...	ga.js	script	js	cached	1....
⊘	GET	cdn.speedcurve.c...	lux.js?id=108906238	script		Blocked By uB...	
⊘	GET	cdn.speedcurve.c...	lux.js?id=108906238	script		Blocked By uB...	
304	GET	developer.mo...	main.e2b366ea.chur	script	js	cached	51...
304	GET	developer.mo...	runtime-main.bcb5c	script	js	cached	3....
200	GET	developer.mo...	whoami	500:1 (fetch)	json	685 B	62...
304	GET	developer.mo...	ZillaSlab-Bold.subse	font	octet...	cached	33...

‹ HTTP response status codes

▶ Table of contents

500 Internal Server Error

The HyperText Transfer Protocol (HTTP) `500 Internal Server Error` server error response code indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

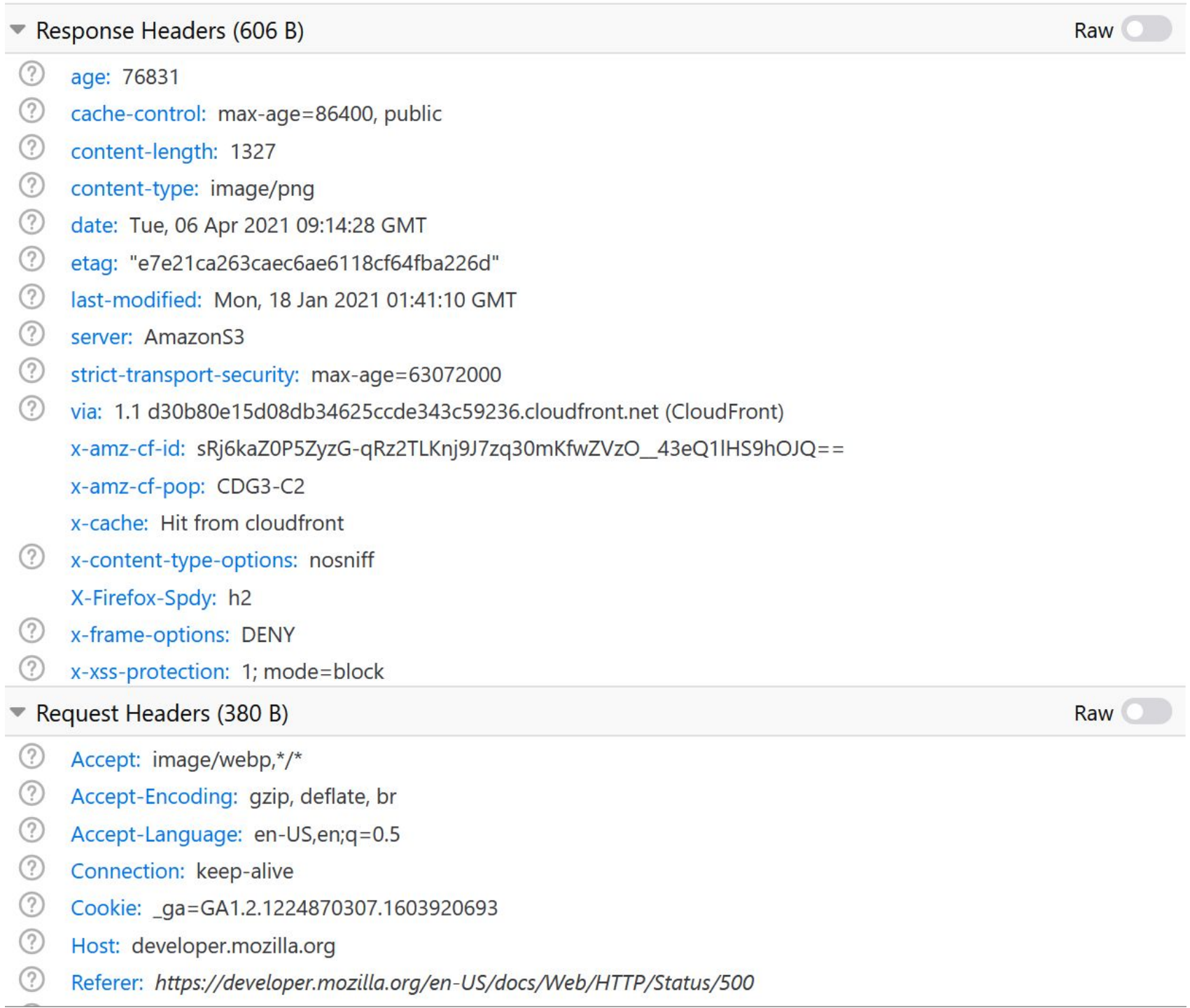
This error response is a generic "catch-all" response. Usually, this indicates the server cannot find a better 5xx error code to response. Sometimes, server administrators log error responses like the 500 status code with more details about the request to prevent the error from happening again in the future.

Status

```
500 Internal Server Error
```

И заявката и отговора от сървъра съдържат заглавна част с ключове/стойности, които формират параметрите на заявка/отговор.

Те са различни, макар и структурата да е подобна.



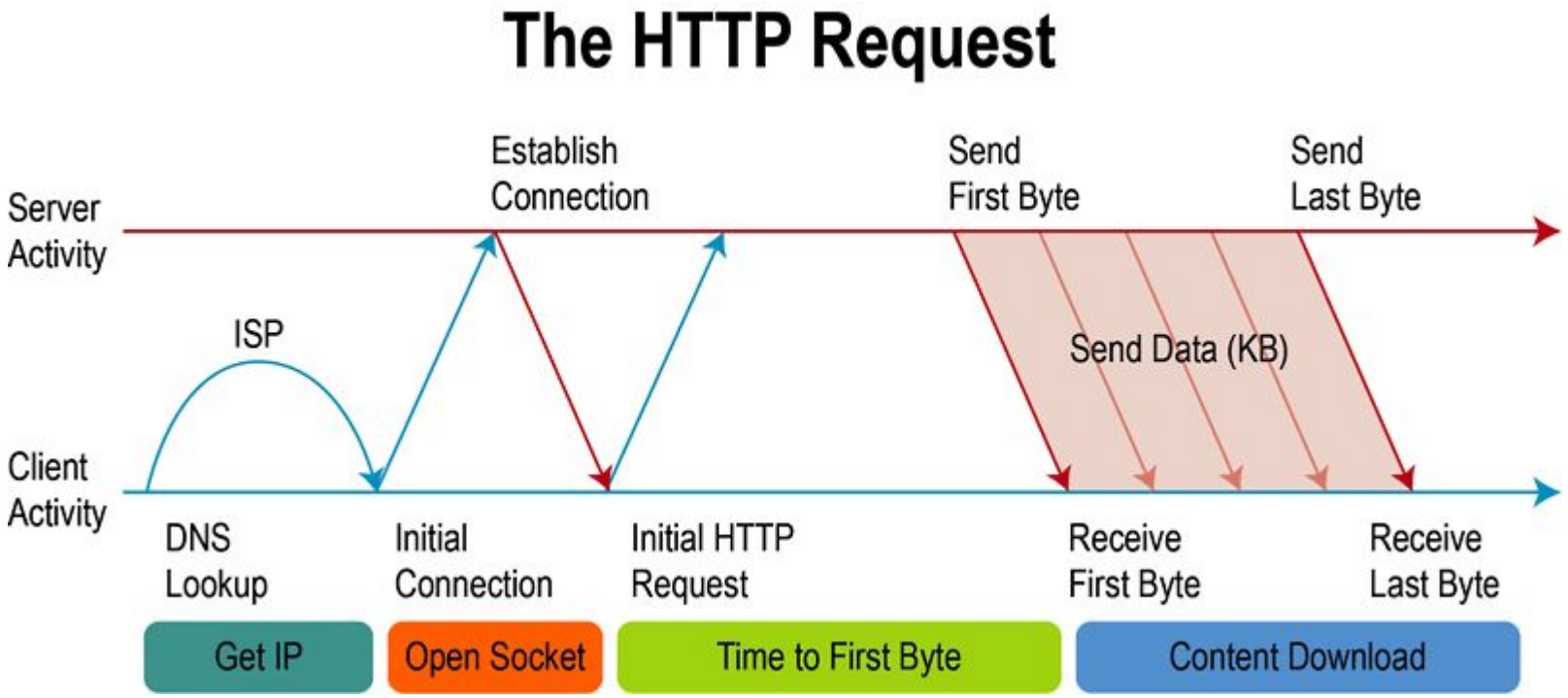
Важни характеристики

Тези попадат в HTTP header (в заглавието)

- Content-type
  - Какъв тип съдържание очакваме
- Content-encoding
  - Как сме окей да бъде представено то
- Media-type
  - Определяне на вида файл
- Multipart ? – за POST заявките

Последователност от действия в HTTP

Източник
[Anatomy of an HTTP request and correlation to Pagetest legend (websiteoptimization.com)]



URL – що е то?

Universal

Resource

Locator

Известно още като – точка на връзка (endpoint)

Източник: Anatomy of a URL (noisylittlemonkey.com)

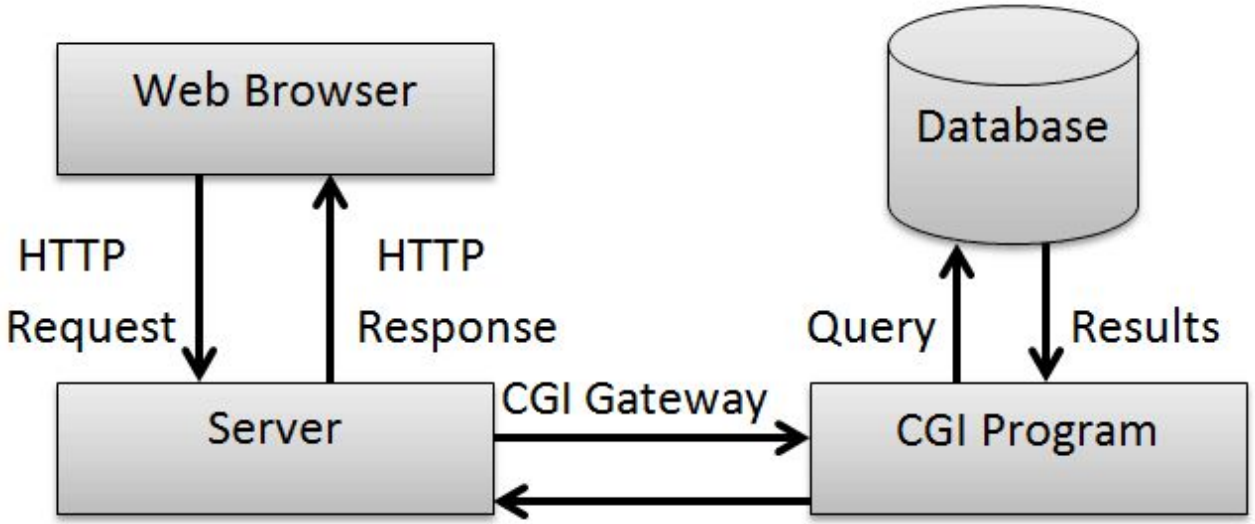
Common Gateway Interface

Класическият механизъм за доставяне на динамично съдържание предполагаше, че някаква програма стои зад определен адрес (URL път) и би следвало да се стартира всеки път, когато се изпрати заявка към този адрес.

През 2022 като цяло ...

...не се използва.

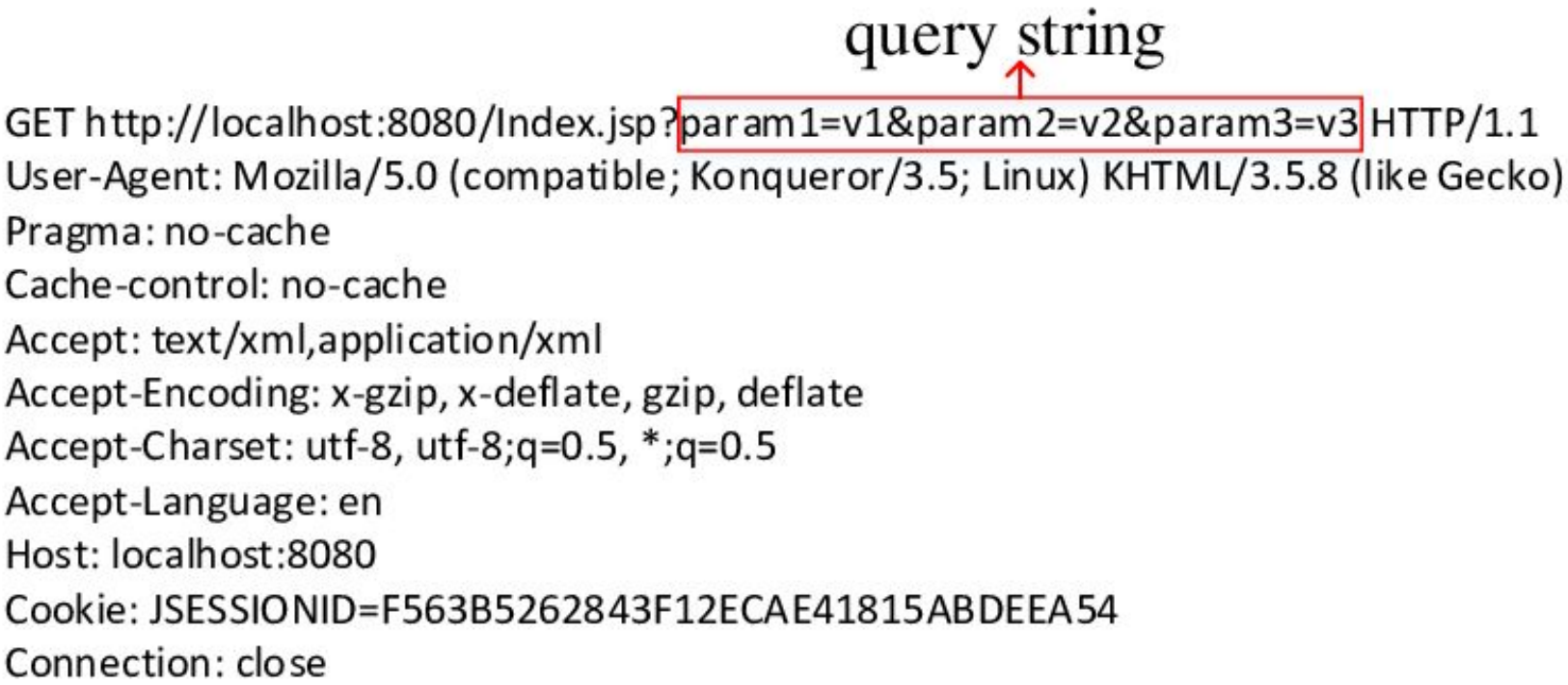
Вече програмата Е де факто уеб сървъра или програмата *живее* в уеб сървъра по някакъв начин



Малко повече за GET

- Аргументите се подават през URL текст
  - след ? в URL – това, което евентуално се ползва от приложението
- Заглавна част с параметри на заявката
  - HTTP header
- Въздесъща – винаги на разположение
- Библиотеки опростяват нещата до три-четири реда
- В общия случай приложенията и страниците са с GET
- Някой нещо да добави може би... ?

Източник - https://www.researchgate.net/figure/a-request-parameter-in-an-HTTP-header_fig1_329115783



Заявка на страница с GET

```
const https = require('https')

const options = {
  hostname: 'mediapool.bg‘,
  port: 443,
  path: '/‘,
//    path: 'https://www.mediapool.bg/‘,
  method: 'GET‘
}

const reqeust = https.request(function (opts, res) {
  console.log(`statusCode: ${res.statusCode}`)

  res.on('data', d ⇒ {
    process.stdout.write(d)
  })
})

request.on('error', error ⇒ {
  console.error(error)
})

request.end()
```

< зареждане на библиотека
< параметри на заявка

< създаване на обект-“заявка“
< задаване на продължение във функция

< получаване и изчитане на данните
< извеждане на стандартен изход

< (възможна) обработка на грешки

< край на настройките
< т.е.изпращане на заявката 😃

Web service – уеб услуга

- Крайна точка, зад която се крие програмна логика
- Връща резултат в структуриран – машинно-четим вид
- Най-често е XML, JSON, CSV и т.н.
- Преди – SOAP/RPC, сега REST или rest-о-подобни неща
- Една страница може да *си говори* с много на брой услуги
- Обикновено дадена услуга извършва една... дейност

XML/RPC и SOAP се ползваха активно в края на 90те, до към 2010.

Допълнителното усилие за сглабяне и обработка на XML и по-късно повсеместното възприемане на JavaScript като де-факто език за WEB доведоха до възприемане на JSON като предпочитан контейнер/формат за пренос на данни м/у клиент и сървър.

Идеологията на XML/RPC се претвори в REST.

XML-RPC call

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
   <methodName>sample.sum</methodName>
   <params>
      <param>
         <value><int>17</int></value>
      </param>

      <param>
         <value><int>13</int></value>
      </param>
   </params>
</methodCall>
```

XML RPC Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
   <params>
      <param>
         <value><int>30</int></value>
      </param>
   </params>
</methodResponse>
```

REST - **Representational State Transfer**

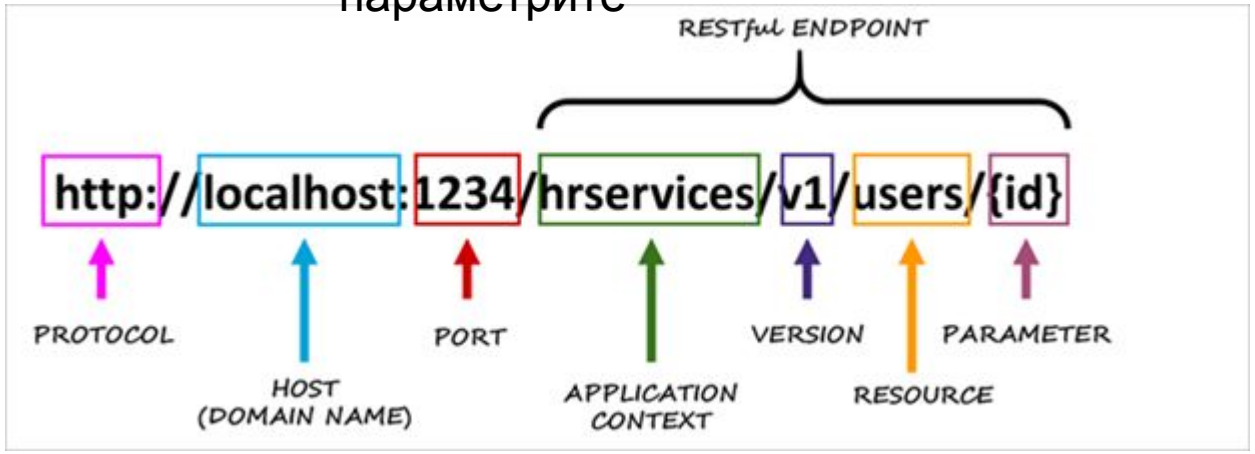
Из Wikipedia…

Representational state transfer (REST) is a software architectural style which uses a subset of HTTP. It is commonly used to create ***interactive applications*** *that use* ***Web services****. A Web service that follows these guidelines is called RESTful.*

Such a Web service must provide its Web resources in a textual representation and allow them to be read and modified with a ***stateless protocol*** *and a* ***predefined set of operations****…*

REST is an alternative to, for example, SOAP as way to access a Web service.

Адресът на услугата съдържа и параметрите



```
...

app.get('/users', (req, res) => {
  return res.send('GET HTTP method on user resource');
});

app.post('/users', (req, res) => {
  return res.send('POST HTTP method on user resource');
});

app.put('/users/:userId', (req, res) => {
  return res.send(
    `PUT HTTP method on user/${req.params.userId} resource`,
  );
});

app.delete('/users/:userId', (req, res) => {
  return res.send(
    `DELETE HTTP method on user/${req.params.userId} resource`,
  );
});

...
```

С REST използваме GET/POST/PUT/DELETE като всяка операция вече има има конкретен семантичен смисъл:

POST - създаване на обект
GET - четене на обект
PUT - промяна на съществуващ
DELETE – изтриване

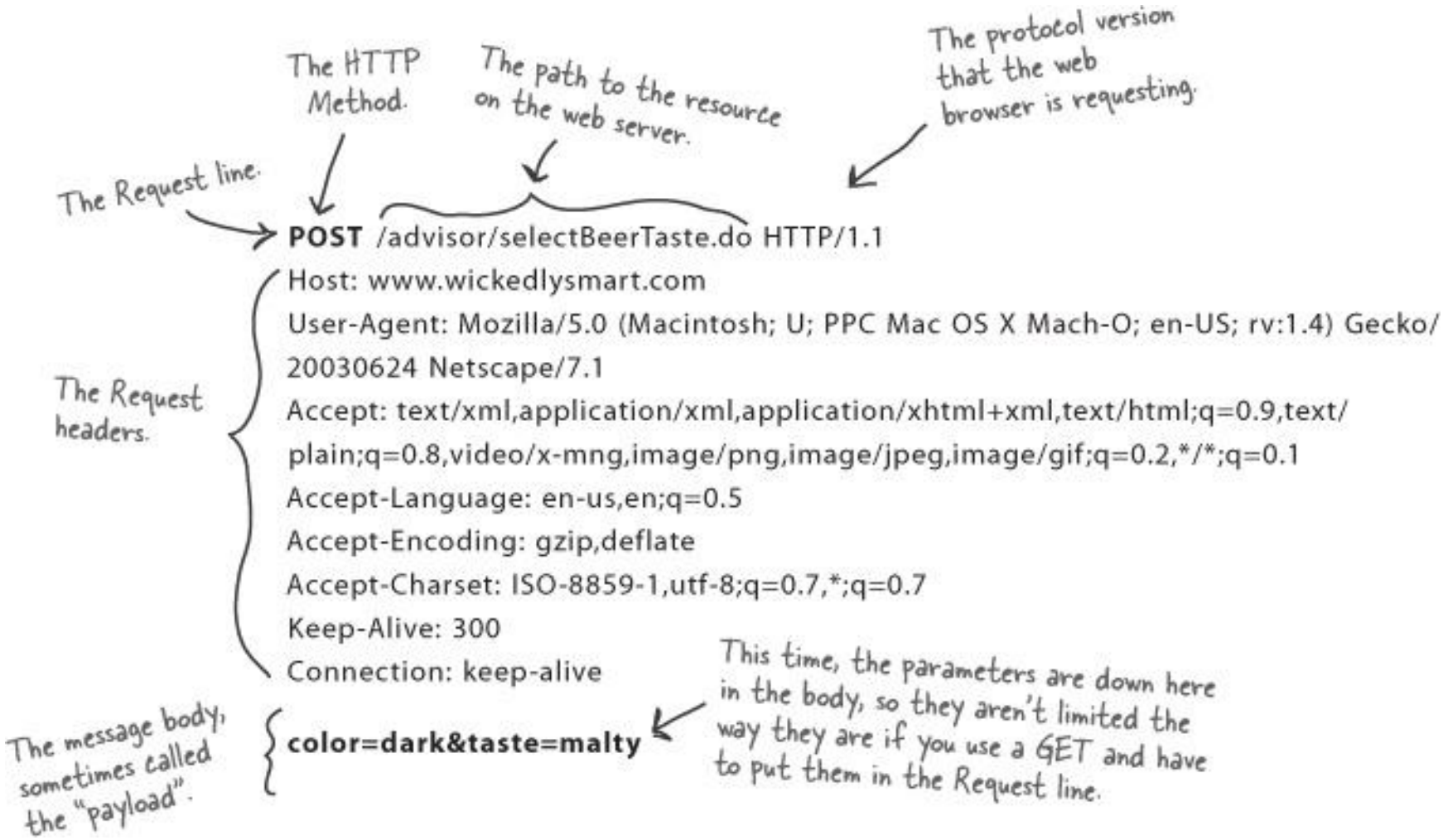
Библиотеки като express ни позволяват да обработим лесно тези HTTP заявки.

How to create a REST API with Express.js in Node.js - RWieruch (robinwieruch.de)

Анатомия на POST заявки

Източник – O'Reiley

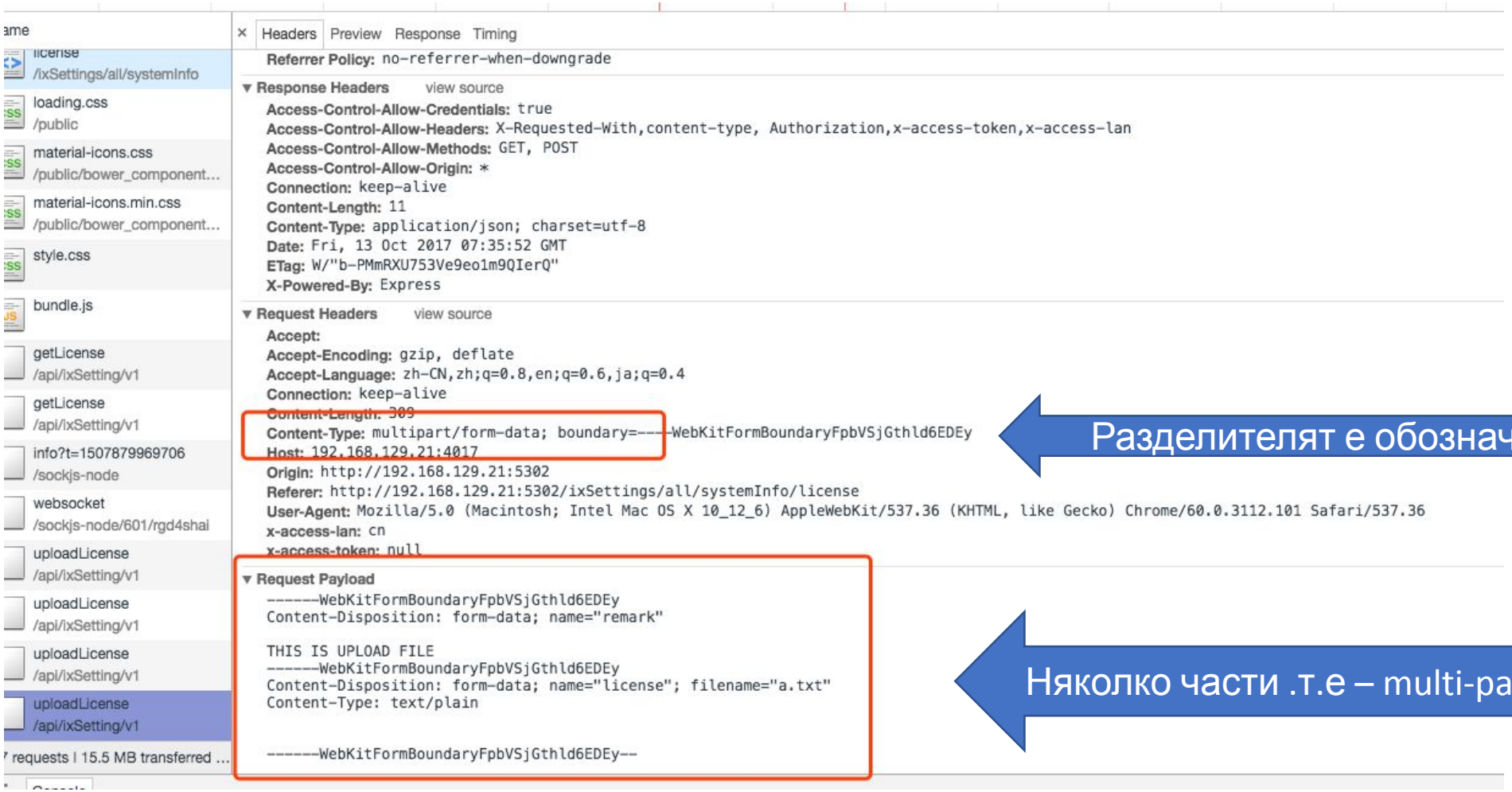
/дано не ни бият/



Проста HTML форма с качване на файл

```
<html>
<head>

    <title>upload example</title>

</head>

<body>

     <br />

     <form name ='exform'
           action="http://localhost:3000/upload?var=1000"
           method="post" enctype="multipart/form-data">

          <input type="hidden" name="myhidden" value="100">

          <input type="text"   name="sometext"><br />

          <input type="number" name="age" id="age" min="1" max="10" step="2"><br />

          <input type="file"   name="filetoupload"><br />

          <input type="submit">

     </form>

</body>
</html>
```

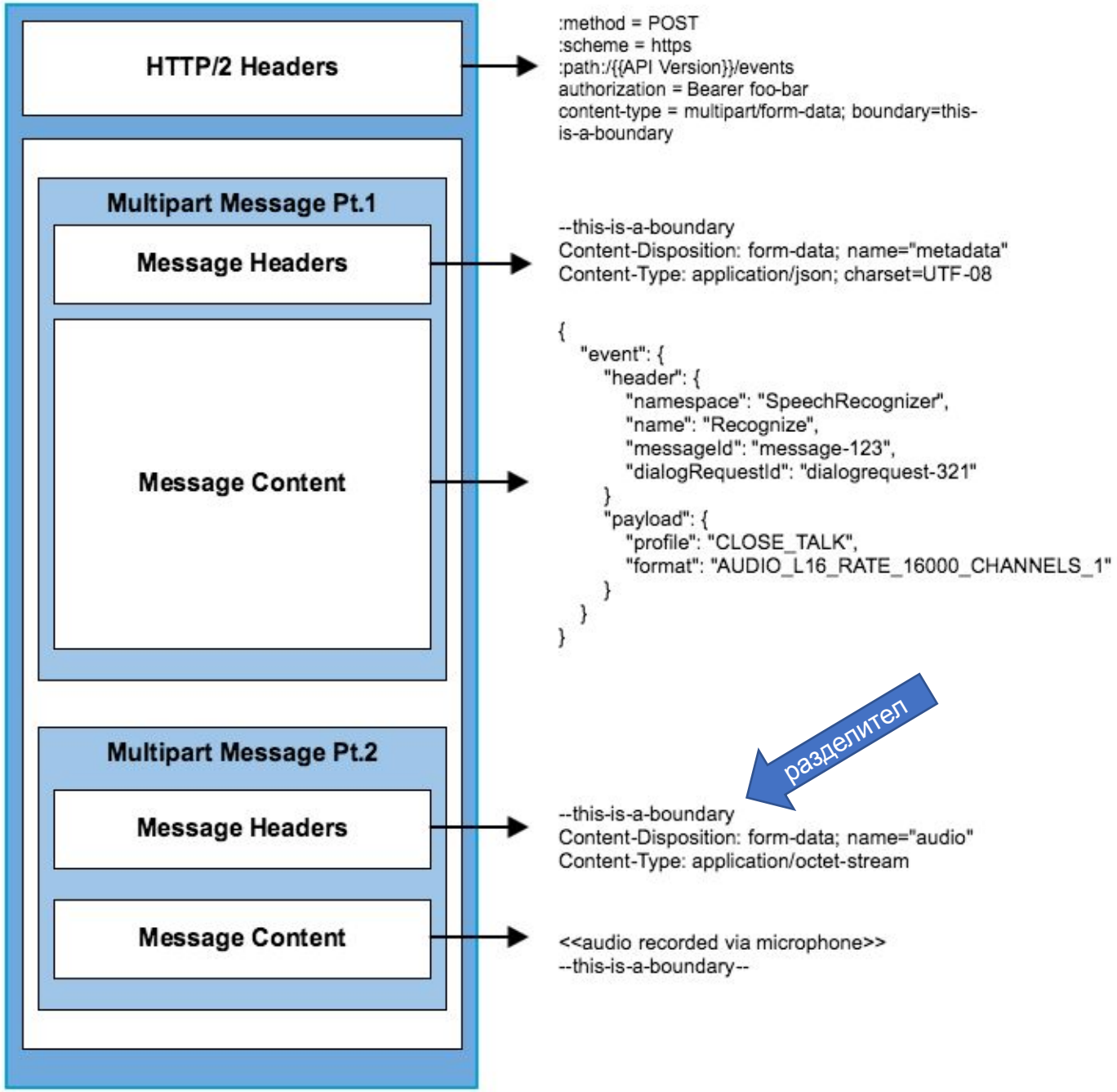
Какво ще се получи - Multipart Form Data



Източник - https://github.com/expressjs/multer/issues/532

Когато обработваме multipart/form-data ние ще трябва да имаме предвид, че съдържанието е в няколко части, с разделители, и всяка част си има заглавие и собствени характерситики.

За целта най-лесно да използваме някоя от наличните за дадения език библиотеки, които ще направят това за нас.



В този пример използваме NodeJS с express за сървър.

За да не се *борим* със съдържанието на multipart заявката, с помоща на обработващ слой (middleware), създаден въз основата на **formidable** се разбива на части съдържанието.

Така изпратеният файл и полета от формата ще попаднат съответно в атрибутите **fields** и **files** на request обекта, който получава обработващата функция.

Програмата слуша на порт 3000 и не прави нищо съществено, но има точка за прекъсване, която можете да ползвате за да изследвате този сценарии.

```
const express = require('express');
const formidable = require('express-formidable');
var app = express();

app.use(formidable());

app.post('/upload', function(req, res) {
    debugger;
//   req.fields;   // contains non-file fields
//   req.files;    // contains files
});

app.listen(3000)
```

Благодаря за вниманието

За връзка с автора:

https://linkedin.com/in/Penkov
georgicp@fmi.uni-sofia.bg

Лекцията е представена като част от курса по уеб технологии през зимния семестър на 2020/2021 учебна година във ФМИ на Софийски Университет „Св. Климент Охордски“.