Web услуги. GET и POST заявки.

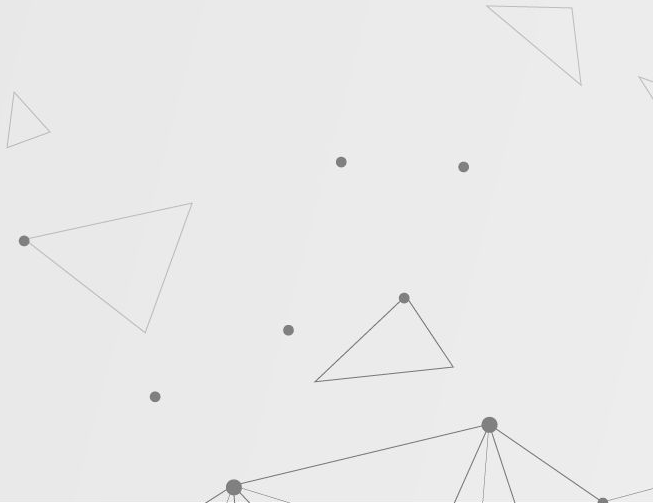
Иван Арабаджийски

Client-server архитектура.

HTTP Request.

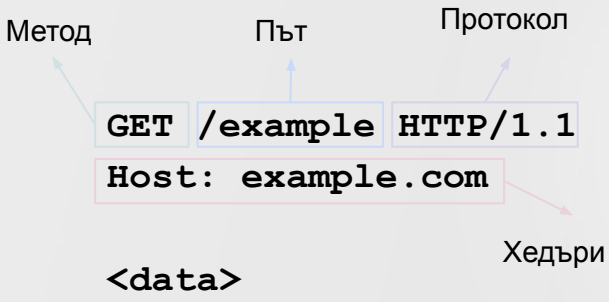
HTTP Request Method.

API

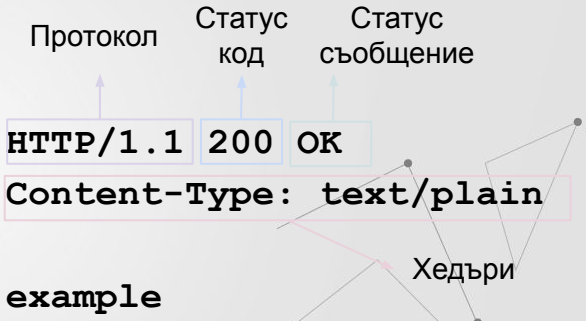



gargasz.info
Google
Network process to application
7. Application
Data representation and encryption
6. Presentation
Interhost communication
5. Session
End-to-end connections and reliability
4. Transport
Path determination and logical addressing
3. Network
Physical addressing
2. Data link
Media, signal and binary transmission
1. Physical
MAC
00-0C-F1-56-98-AD
router MAC
00-14-C3-A4-AB
DNS , WWW/HTTP
P2P,EMAIL/POP,SMTP
Telnet, FTP
7. Application
recognizing data
HTML, DOC, JPEG, MP3, AVI
6. Presentation
Sockets. Session establishment
in TCP, SIP, RTP
RPC - Named pipes,
5. Session
TCP, UDP, SCTP, SSL, TLS
4. Transport
IP, IPsec, ICMP, IGMP, OSPF
3. Network
Ethernet, 802.11, MAC/LLC,VLAN, ATM, HDP
Fibre Channel Frame Relay, HDLC,
PPP, Q.921, Token Ring, ARP
2. Data link
RS-232, RJ45, V.34, 100BASE-TX, SDH, DSL, 802.11
1. Physical

Request



Response



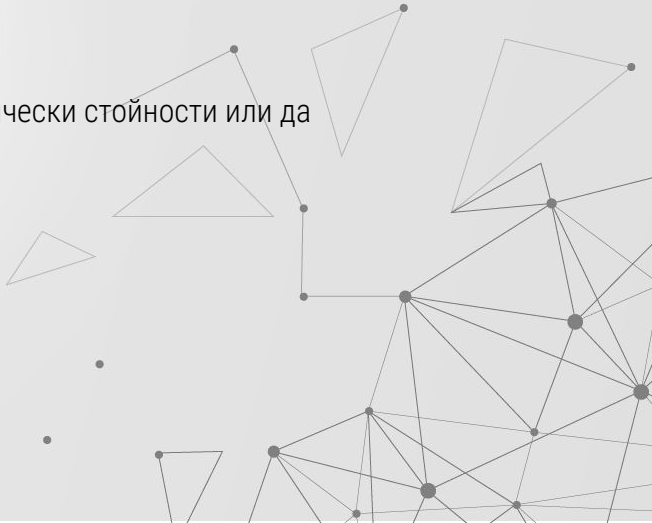
Вметки

Какво е безопасен метод?

HTTP методите се считат за безопасни, ако не променят състоянието на сървъра. Така че безопасните методи могат да се използват само за операции за четене. HTTP RFC определя следните методи като безопасни: GET, HEAD, OPTIONS и TRACE.

На практика често не е възможно да се реализират безопасни методи по начин, по който те не променят състоянието на сървъра.

Например, заявка GET може да създаде лог съобщения, да актуализира статистически стойности или да предизвика обновяване на кеша на сървъра.



Идемпотентността означава, че няколко идентични заявки ще имат един и същ резултат. Така че няма значение дали дадена заявка е изпратена веднъж или няколко пъти. Следните HTTP методи са идемпотентни: GET, HEAD, OPTIONS, TRACE, PUT и DELETE. Всички безопасни HTTP методи са идемопотентни, но PUT и DELETE са идемопотентни, но не са безопасни.

Имайте предвид, че идемпотентността не означава, че сървърът трябва да отговаря по един и същи начин на всяка заявка.

Например, да предположим, че искаме да изтрием проект по ID, като използваме заявка DELETE:

DELETE /projects/123 HTTP/1.1

Като отговор може да получим HTTP 200 код на състоянието, който показва, че проектът е изтрит успешно. Ако изпратим тази заявка DELETE отново, може да получим HTTP 404 като отговор, тъй като проектът вече е бил изтрит. Втората заявка не е променила състоянието на сървъра, така че операцията DELETE е idempotent, дори ако получим различен отговор.

Идемотентността е положителна характеристика на API, тъй като може да направи API по-устойчив на грешки. Да предположим, че има проблем при клиента и заявките се изпращат многократно. Стига да се използват идентични операции, това няма да доведе до проблеми от страна на сървъра.

HTTP Методи

Метод	Безопасен?	Идемпотентен?
GET	✓	✓
POST	✗	✗
PUT	✗	✓
PATCH	✗	✓
DELETE	✗	✓
HEAD	✓	✓
OPTIONS	✓	✓

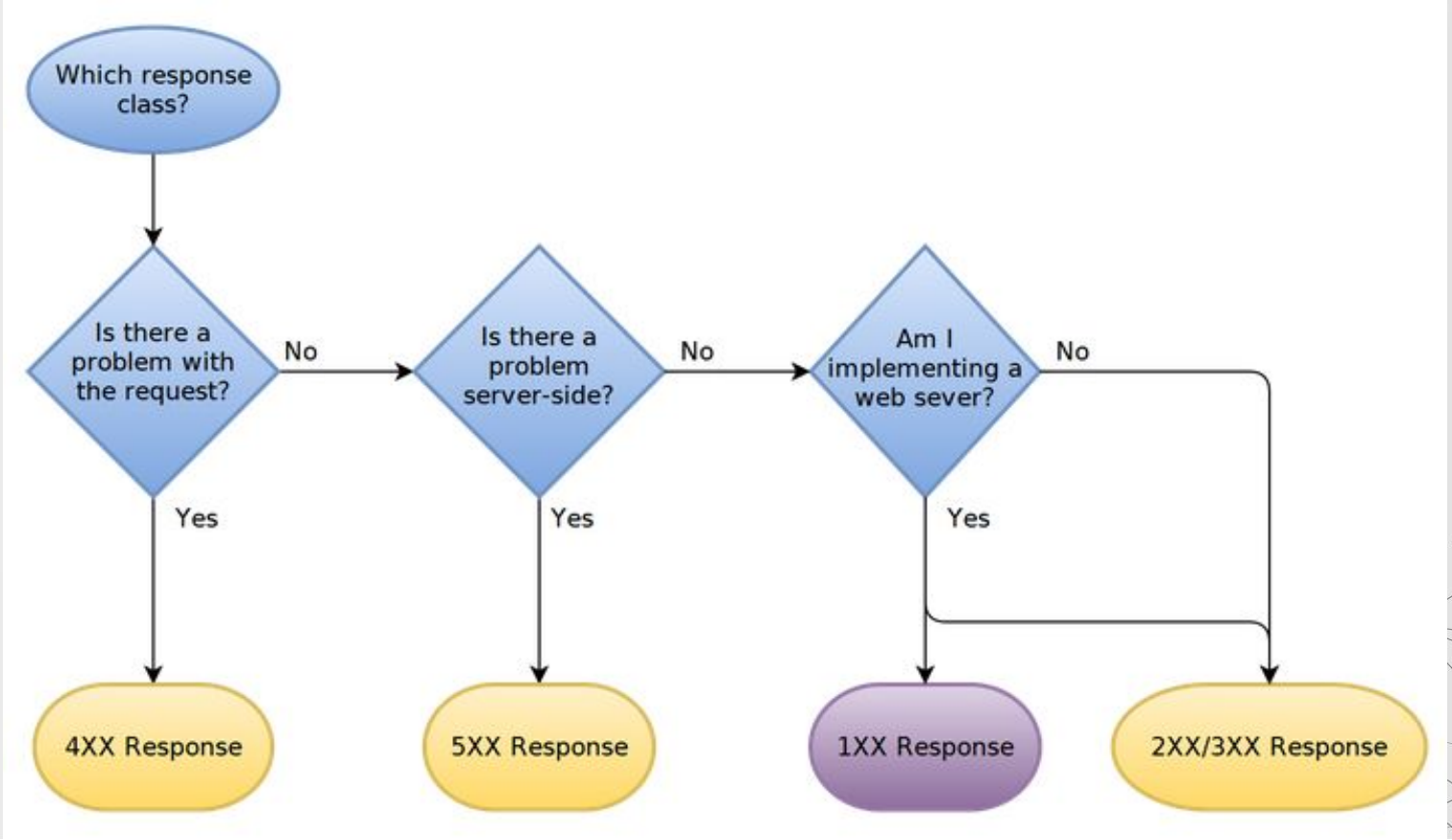
Из интернет:

A `PATCH` is not necessarily idempotent, although it can be. Contrast this with `PUT`; which is always idempotent. The word "idempotent" means that any number of repeated, identical requests will leave the resource in the same state. For example if an auto-incrementing counter field is an integral part of the resource, then a `PUT` will naturally overwrite it (since it overwrites everything), but not necessarily so for `PATCH`.

`PATCH` (like `POST`) *may* have side-effects on other resources.

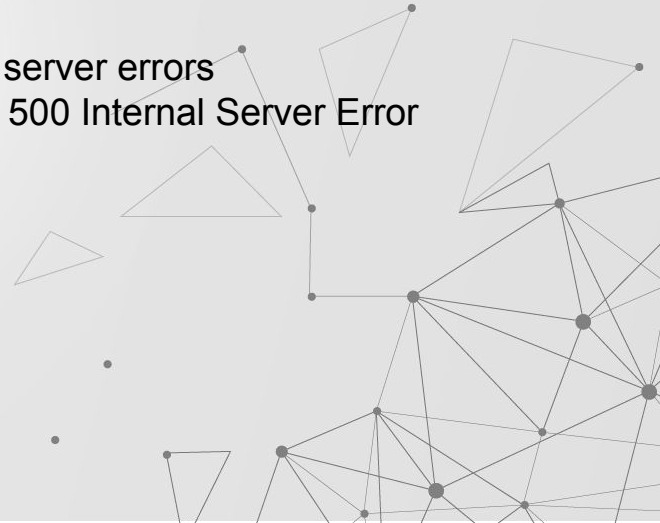
To find out whether a server supports `PATCH`, a server can advertise its support by adding it to the list in the `Allow` or `Access-Control-Allow-Methods` (for CORS) response headers.

HTTP Статус кодове



HTTP Статус кодове

- 1xx - informational
  - 100 Continue
- 2xx - success
  - 200 OK
  - 201 Created
  - 204 No Content
- 3xx - redirects
  - 301 Moved Permanently
  - 302 Moved Temporarily
  - 304 Not Modified
- 4xx - client errors
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
  - 418 I’m A Teapot
- 5xx - server errors
  - 500 Internal Server Error



401 vs 403

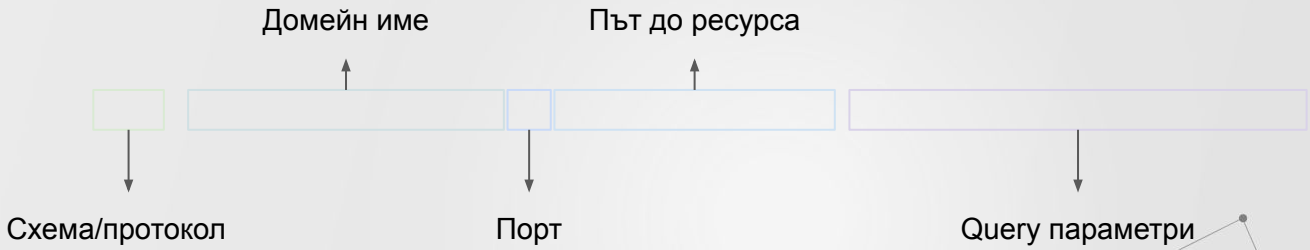
В категория 4xx особено внимание заслужават **401** и **403**. Статус кодът "**401 Unauthorized**" показва, че заявката няма валидни удостоверителни данни. От друга страна, статус кодът "**403 Forbidden**" означава, че сървърът разбира заявката, но отказва да я изпълни. Тези два етапа в контрола на достъпа обикновено се разглеждат като автентикация и оторизация.

HTTP Хедъри

Клиент / Request	Сървър / Response
Content-Type, Content-Length	
Cookie	Set-Cookie
Authorization	WWW-Authenticate
Origin, Access-Control-Request-*	Access-Control-Allow-*
	Cache-Control, Expires
If-Modified-Since, If-Unmodified-Since	Last-Modified
If-None-Match, If-Match	ETag

URL

http://www.example.com:80/path/to/resource?key1=value1&key2=value2



API Примери

- Регистриране на потребител
  `POST /users`
  `Auth: None`
- Добавяне на игра от потребител
  `POST /games`
  `Auth: User`
- Промяна информацията за конкретна игра
  `PUT /games/:game_id`
  `Auth: User`
- Списък с игри
  `GET /games?page=1&pageSize=10`
  `Auth: User`
- Изтриване на игра
  `DELETE /games/:game_id`
  `Auth: User`
- Добавяне на коментар за игра
  `POST /games/:game_id/reviews`
  `Auth: User`

HTTP Примери

Request

```
POST /games HTTP/1.1
Host: localhost:3000
Content-Type: application/json

{ “name”: “Dixit”,
“averageDuration”: “15
minutes” }
```

Response

```
HTTP/1.1 401 Unauthorized
```

HTTP Примери

Request

```
POST /games HTTP/1.1
Host: localhost:3000
Authorization: Basic Xha5dsfj
Content-Type: application/json

{ “name”: “Dixit”,
“averageDuration”: “15
minutes” }
```

Response

```
HTTP/1.1 201 Created
Etag: foobar
Content-Type: application/json

{ “id”: 1, “name”: “Dixit”,
“averageDuration”: “15 minutes”
}
```

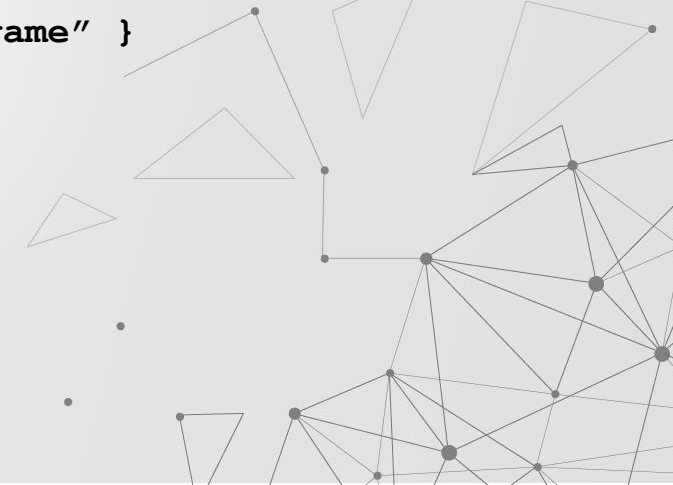
HTTP Примери

Request

```
DELETE /games HTTP/1.1
Host: localhost:3000
Authorization: Basic jlksDF032
```

Response

```
HTTP/1.1 403 Unauthorized
Content-Type: application/json

{ “error”: “Only the owner can
delete a game” }
```



HTTP Примери

Request

```
GET /games/1 HTTP/1.1
Host: localhost:3000
Authorization: Basic Xha5dsfj
If-None-Match: foobar
```

Response

```
HTTP/1.1 200 OK
Etag: bazmaz
Content-Type: application/json

{ “id”: 1, “name”: “Dixit”,
“averageDuration”: “15 minutes”
}
```

HTTP Примери

Request

```
GET /games/1 HTTP/1.1
Host: localhost:3000
Authorization: Basic Xha5dsfj
If-None-Match: bazmaz
```

Response

```
HTTP/1.1 304 Not Modified
```

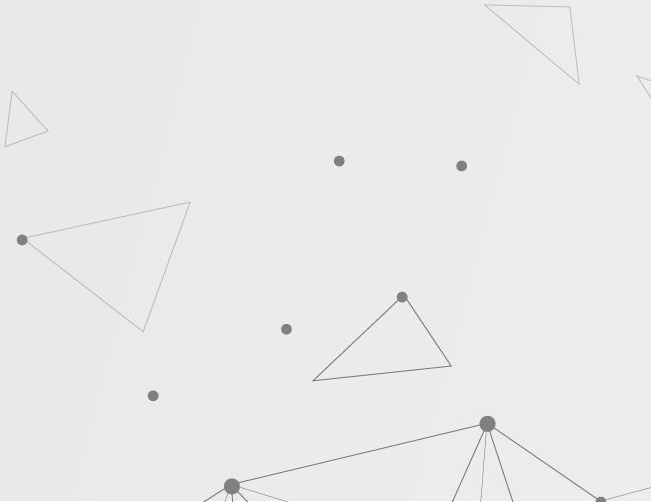
Resource-Oriented архитектури

- Сървърът е колекция от ресурси / данни
- Клиентът манипулира тези ресурси / данни
- Всички заявки, които клиентите изпращат към сървъра, са под формата на CRUD операции по различни ресурси

REST услуги
REpresentational State Transfer

Софтуерен архитектурен стил, създаден от Roy Fielding през 2000 година, за осигуряване на стандарт между компютърните системи в мрежата, който улеснява комуникацията между системите. Всеки API, който следва REST шаблона за дизайн се нарича RESTful API.

По-просто казано REST API е лесен начин два компютъра да комуникират помежду си чрез HTTP по същия начин, по който комуникират клиентите и сървърите.

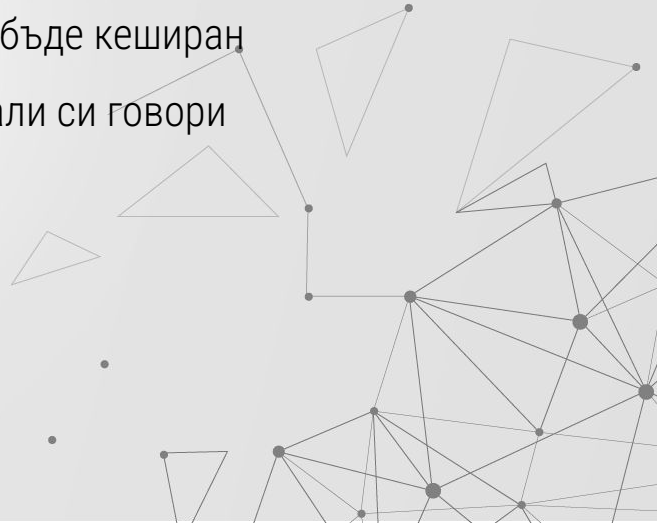


Абревиатура?

- REpresentational
- State
- Transfer

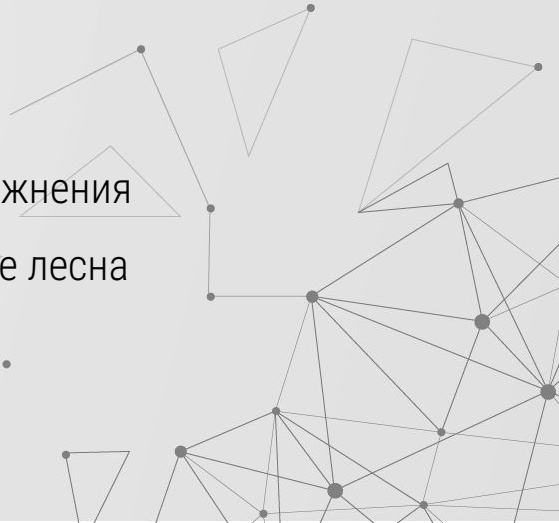
REST архитектура

- REpresentational State Transfer, вид ROA (resource-oriented architecture)
- Характеристики:
  - **Клиент-сървър архитектура**
  - **Stateless** - сървърът не пази състояние за клиентите
  - **Кеширане** - всеки ресурс определя дали може да бъде кеширан
  - **Многослойност** - клиентът не може да разбере дали си говори директно със сървър или с прокси/load balancer.
  - **Еднороден интерфейс** между клиента и сървъра



REST архитектура

- Предимства:
  - Скалируемост
  - Лесна поддръжка
  - Интуитивен интерфейс
  - Добро разделение на отговорностите
- Недостатъци
  - Изискването за stateless сървър може да създаде усложнения
  - Моделирането на бизнес логиката с ресурси не винаги е лесна задача



Какво прави едно API RESTful?

- Клиент-сървър архитектура съставена от клиенти, сървъри и ресурси, които се достъпват през HTTP.
- Stateless клиент-сървър комуникация. Това означава, че информация не се запазва между отделните get заявки и всяка заявка е отделна и несвързана с другите.

Всеки ресурс се определя по следния начин:

- Основен URI, например `http://api.example.com/`
- Стандартен HTTP метод - `GET, POST, PUT, and DELETE`
- Тип на ресурса

Добри практики

Използваме JSON формат за изпращане и получаване на данни

JavaScript има вече готови начини методи за работа с JSON обекти (fetch API), защото те са основно направени за нея. Естествено всички други програмни езици като Python и PHP също имат библиотеки и методи за за JSON данни.

За да сме сигурни, че клиентите интерпретират JSON данните правилно трябва да сложим `Content-Type: application/json`

в рекуест хедъра

От страната на сървъра доста фреймъурци вече правят това автоматично. Например `express.json() middleware`

Използваме съществителни вместо глаголи в имената на endpoint-ите

Това е така, защото всеки ресурс се определя от път и метод. Методите вече са с имена глаголи - `GET, POST, PUT, PATCH, and DELETE`

Това са основните CRUD (create, read, update, delete) операции.

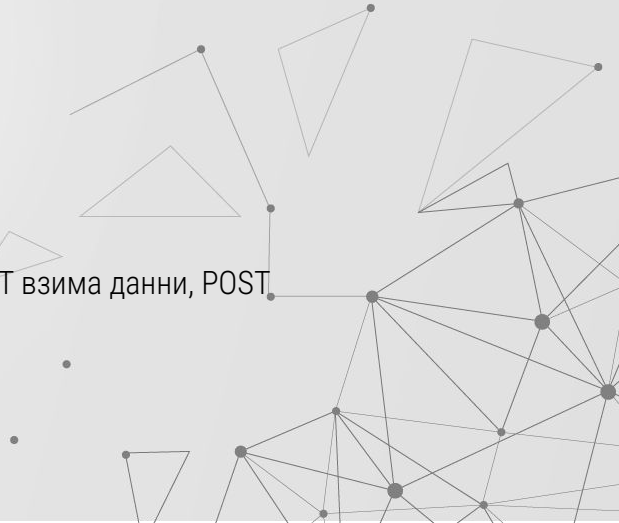
Пример: един endpoint не трябва да изглежда така

`` `https://mysite.com/getPosts` or `https://mysite.com/createPost` ``

А по-скоро така:

`` `https://mysite.com/posts` ``

Накратно, трябва да оставим HTTP методите да описват какво прави endpoint-а. GET взима данни, POST създава данни, PUT променя, DELETE изтрива.



Имената на колекциите да са в множествено число

```
https://mysite.com/post/123 -> https://mysite.com/posts/123
```

Правилни статус кодове в обработка на грешки

STATUS CODE RANGE	MEANING
100-199	Informational Responses For example, 102 indicates the resource is being processed
300-399	Redirects For example, 301 means Moved permanently
400-499	Client-side errors 400 means bad request and 404 means resource not found
500-599	Server-side errors For example, 500 means an internal server error

Използване на вложени пътища, за да покажем релация

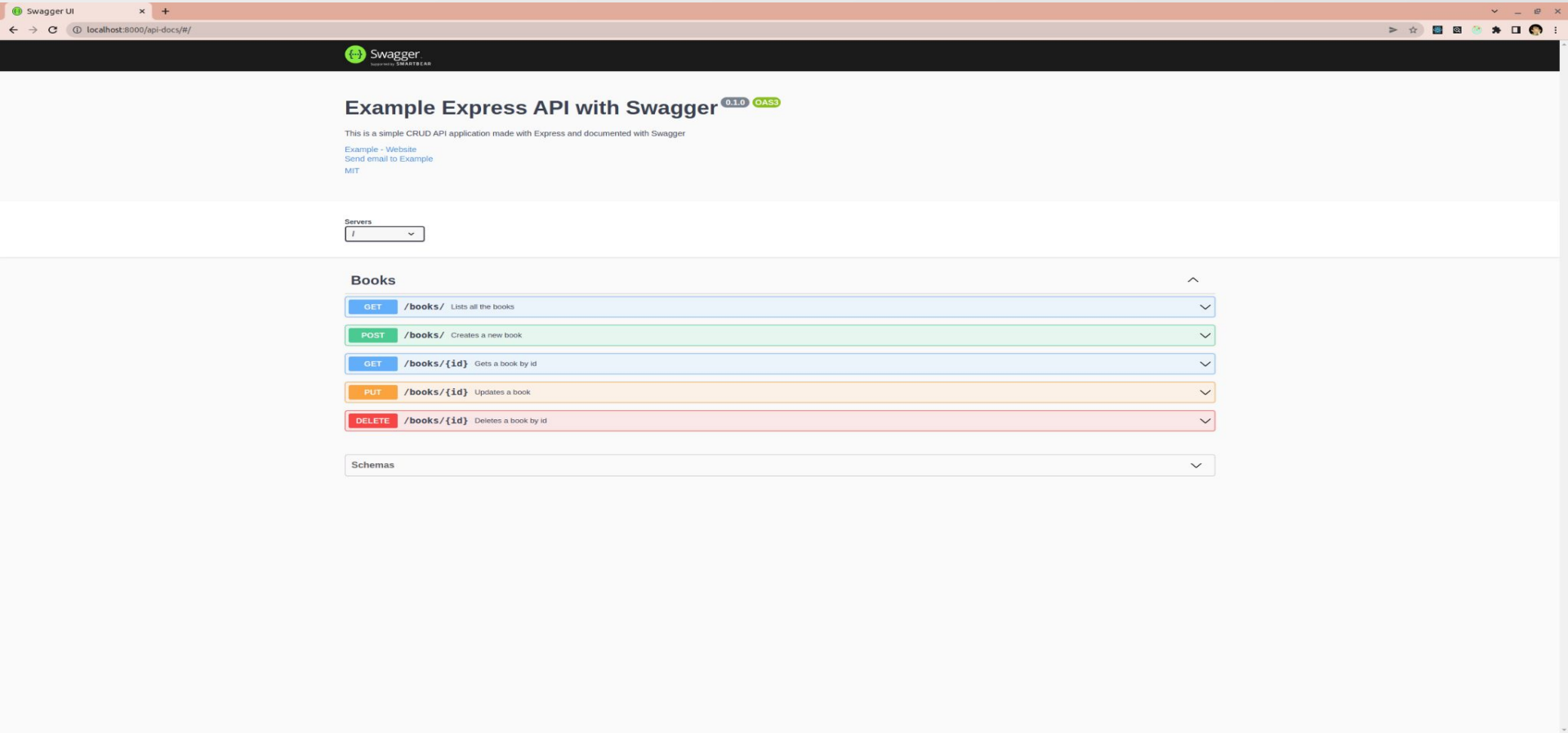
`https://mysite.com/posts/author` - Би било валидно влагане.

`https://mysite.com/posts/postId/comments` също

Използване на филтриране, сортиране и номериране на страници за да вземем необходимите данни

```
`https://mysite.com/posts?tags=javascript`
```

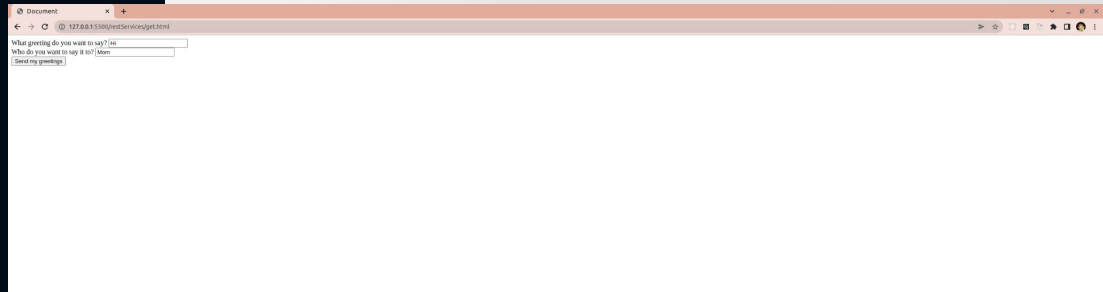
Добра документация



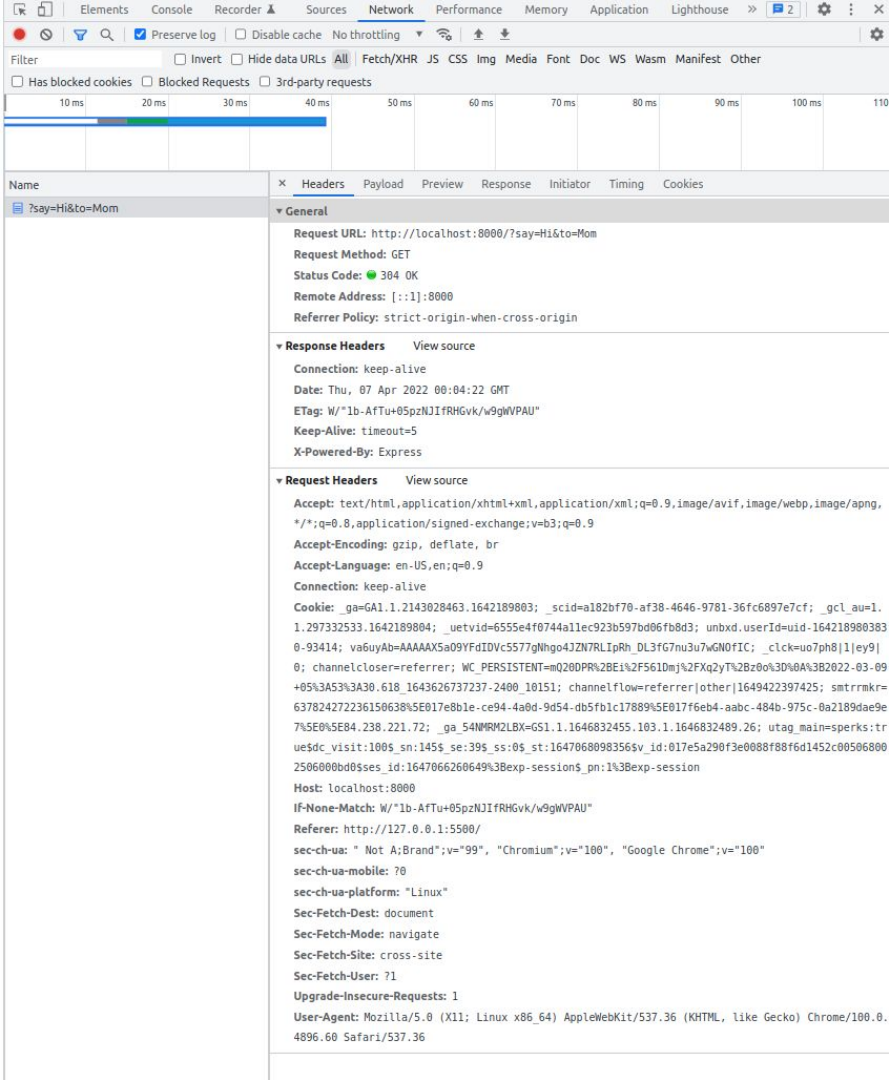
Изпращане на данни чрез HTML форми

GET

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <form action="http://localhost:8000/" method="GET">
      <div>
        <label for="say">What greeting do you want to say?</label>
        <input name="say" id="say" value="Hi" />
      </div>
      <div>
        <label for="to">Who do you want to say it to?</label>
        <input name="to" id="to" value="Mom" />
      </div>
      <div>
        <button>Send my greetings</button>
      </div>
    </form>
  </body>
</html>
```

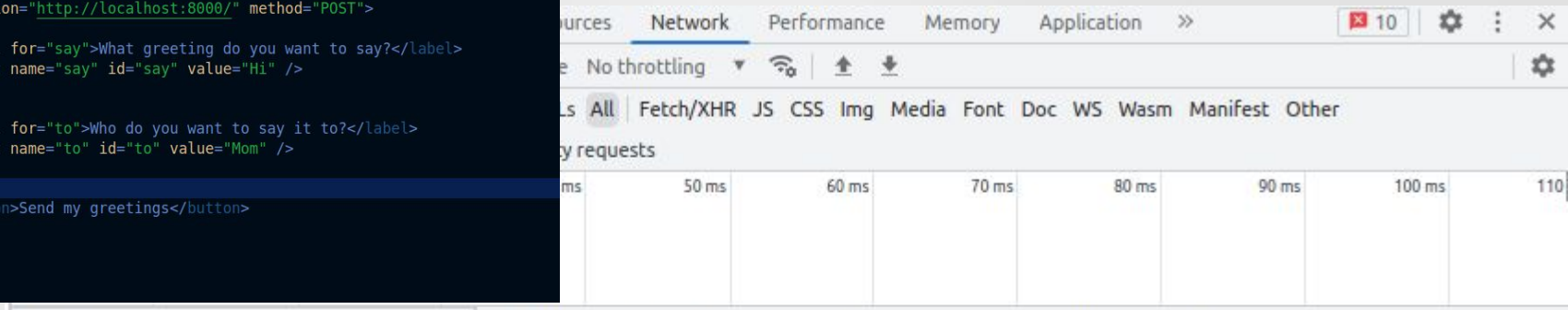


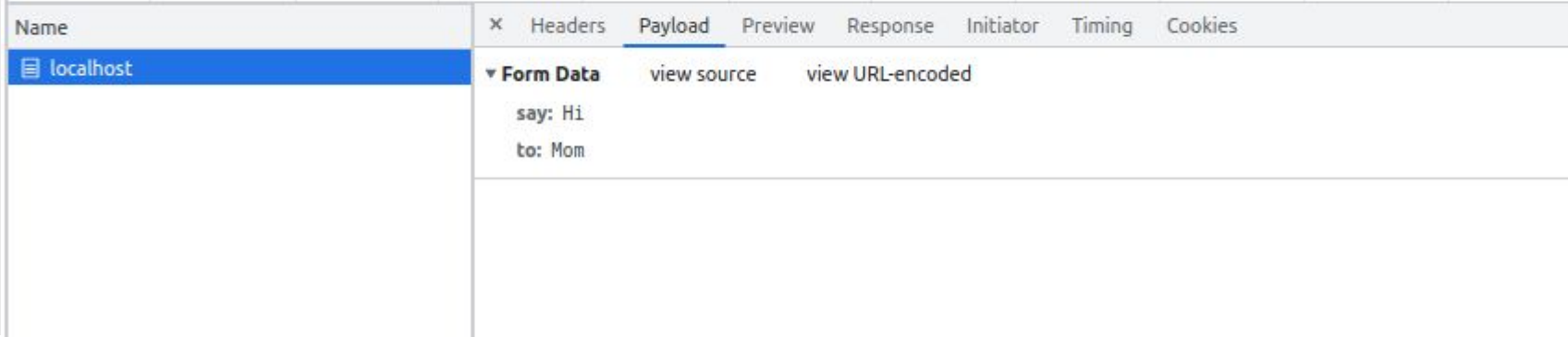
А така изглежда когато се изпрати заявка



POST заявката изпраща и тяло

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <form action="http://localhost:8000/" method="POST">
      <div>
        <label for="say">What greeting do you want to say?</label>
        <input name="say" id="say" value="Hi" />
      </div>
      <div>
        <label for="to">Who do you want to say it to?</label>
        <input name="to" id="to" value="Mom" />
      </div>
      <div>
        <button>Send my greetings</button>
      </div>
    </form>
  </body>
</html>
```





И как създавам REST API?

Expressjs

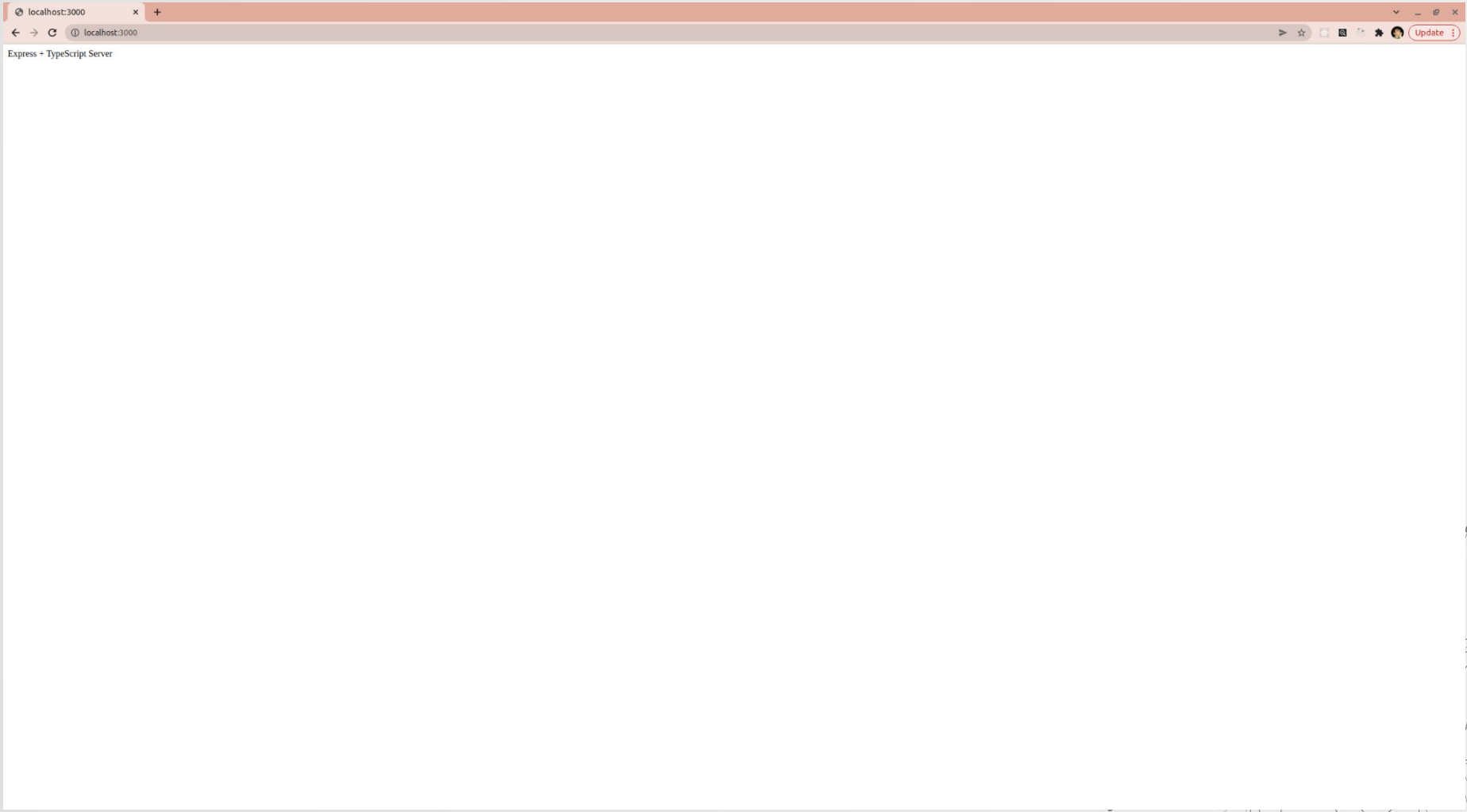
```
iarabadzhiyski@Astea-20158:~$ node -v
v14.18.0
iarabadzhiyski@Astea-20158:~$ npm -v
6.14.15
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm init -y
Wrote to /home/iarabadzhiyski/js/lectures/restServices/expressExample/package.json:

{
  "name": "expressexample",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm install express

added 50 packages, and audited 51 packages in 2s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
1   const express = require("express");
2
3   const app = express();
4   const port = 3000;
5
6   app.get("/", (req, res) => {
7     res.send("Express + TypeScript Server");
8   });
9
10  app.listen(port, () => {
11    console.log(`[server]: Server is running at https://localhost:${port}`);
12  });
13
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ node index.js
[server]: Server is running at https://localhost:3000
```


localhost:3000
localhost:3000
Update
Express + TypeScript Server

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm i -D typescript @types/express @types/node

added 10 packages, and audited 61 packages in 3s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

package.json

```
"devDependencies": {
  "@types/express": "^4.17.13",
  "@types/node": "^17.0.23",
  "typescript": "^4.6.3"
}
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npx tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true


You can learn more at https://aka.ms/tsconfig.json
```

```ts
import express, { Express, Request, Response } from "express";

const app: Express = express();
const port = process.env.PORT;

app.get("/", (req: Request, res: Response) => {
  res.send("Express + TypeScript Server");
});

app.listen(port, () => {
  console.log(`⚡[server]: Server is running at https://localhost:${port}`);
});
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm install -D concurrently nodemon

added 131 packages, and audited 192 packages in 10s

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
  "scripts": {
    "build": "npx tsc",
    "start": "node dist/index.js",
    "dev": "concurrently \"npx tsc --watch\" \"nodemon -q dist/index.js\""
  },
```

Какво друго можем да правим с express?

```
app.get("/", (req: Request, res: Response) => {
  console.log(req.query);
  res.send("Express + TypeScript Server");
});

app.post("/", (req: Request, res: Response) => {
  console.log(req.body);
  res.sendFile(path.join(__dirname, "htmlTemplates/index.html"));
});

app.post("/upload", upload.single("myFile"), (req: Request, res: Response) => {
  if (!req.file) {
    return res.status(401).json({ error: "Please provide an image" });
  }
  const imagePath = path.join(__dirname, `public/${req.file.originalname}`);
  createWriteStream(imagePath).write(req.file.buffer);
  res.redirect(`/media/${req.file.originalname}`);
});
```

What the middleware?

```
app.use(urlencoded({ extended: true }));
app.use(express.json());

function logMethod(req: Request, res: Response, next: NextFunction) {
  console.log("Request Type:", req.method);
  next();
}

app.use((req, res, next) => {
  console.log("Time:", Date.now());
  next();
});

app.use(logMethod);
```

Автентикация?

```
function authenticateJWT(req: Request, res: Response, next: NextFunction) {
  const authHeader = req.headers.authorization;
  if (authHeader && authHeader !== "null") {
    const token = authHeader.split(" ")[1];
    jwt.verify(token, JWT_KEY, (err: any, user: any) => {
      if (err) {
        return res
          .status(403)
          .send({ success: false, message: "Token Expired" });
      }
      // req.user = user;
      next();
    });
  } else {
    res.status(403).json({ success: false, message: "UnAuthorized" });
  }
}
```

Ние искаме RESTful API -> трябва да връщаме JSON.

```
app.get("/", (req: Request, res: Response) => {
  console.log(req.query);
  res.json({ name: "Express + TypeScript Server", ...req.query });
});

app.post("/", (req: Request, res: Response) => {
  console.log(req.body);
  res.json({ ...req.body });
});

app.post("/upload", upload.single("myFile"), (req: Request, res: Response) => {
  if (!req.file) {
    return res.status(401).json({ error: "Please provide an image" });
  }
  const imagePath = path.join(__dirname, `public/${req.file.originalname}`);
  createWriteStream(imagePath).write(req.file.buffer);
  res.json({ path: `http://localhost:8000/media/${req.file.originalname}` });
});
```

Как обикновено изглежда един RESTful API?

```ts
type Book = {
  id?: string;
  createdAt?: number;
  title: string;
  author: string;
  finished: boolean;
};

const harryPotterAndThePhilosophersStone: Book = {
  id: "0",
  title: "Harry Potter and the Philosopher's Stone",
  author: "J. K. Rowling",
  finished: true,
};

const books = [harryPotterAndThePhilosophersStone];

app.get("/books/", (req: Request, res: Response) => {
  res.json({ books });
});

app.get("/books/:id", (req: Request, res: Response) => {
  const book = books.find((book) => book.id === req.params.id);
  if (!book) {
    return res.status(404).send();
  }
  res.json({ book });
});
```

```ts
function createNewBook({
  title,
  author,
  finished,
}: Pick<Book, "title" | "author" | "finished">): Book {
  // automatically create an id, use uuid or let the db generate one
  return {
    id: Math.random().toString(),
    title,
    author,
    finished,
    createdAt: Date.now(),
  };
}

function saveItToTheDatabase(book: Book) {
  return true;
}

app.post("/books", (req: Request, res: Response) => {
  // create a new book and save it to the database
  const book = createNewBook(req.body);
  if (!book) {
    return res.status(400).send();
  }
  const saved = saveItToTheDatabase(book);
  if (!saved) {
    return res.status(400).send();
  }
  res.json({ book });
});
```

Swagger?

Въпроси?