NodeJS
środowisko i technologia ServerSide

PAWEŁ ŁUKASZUK

http://ww

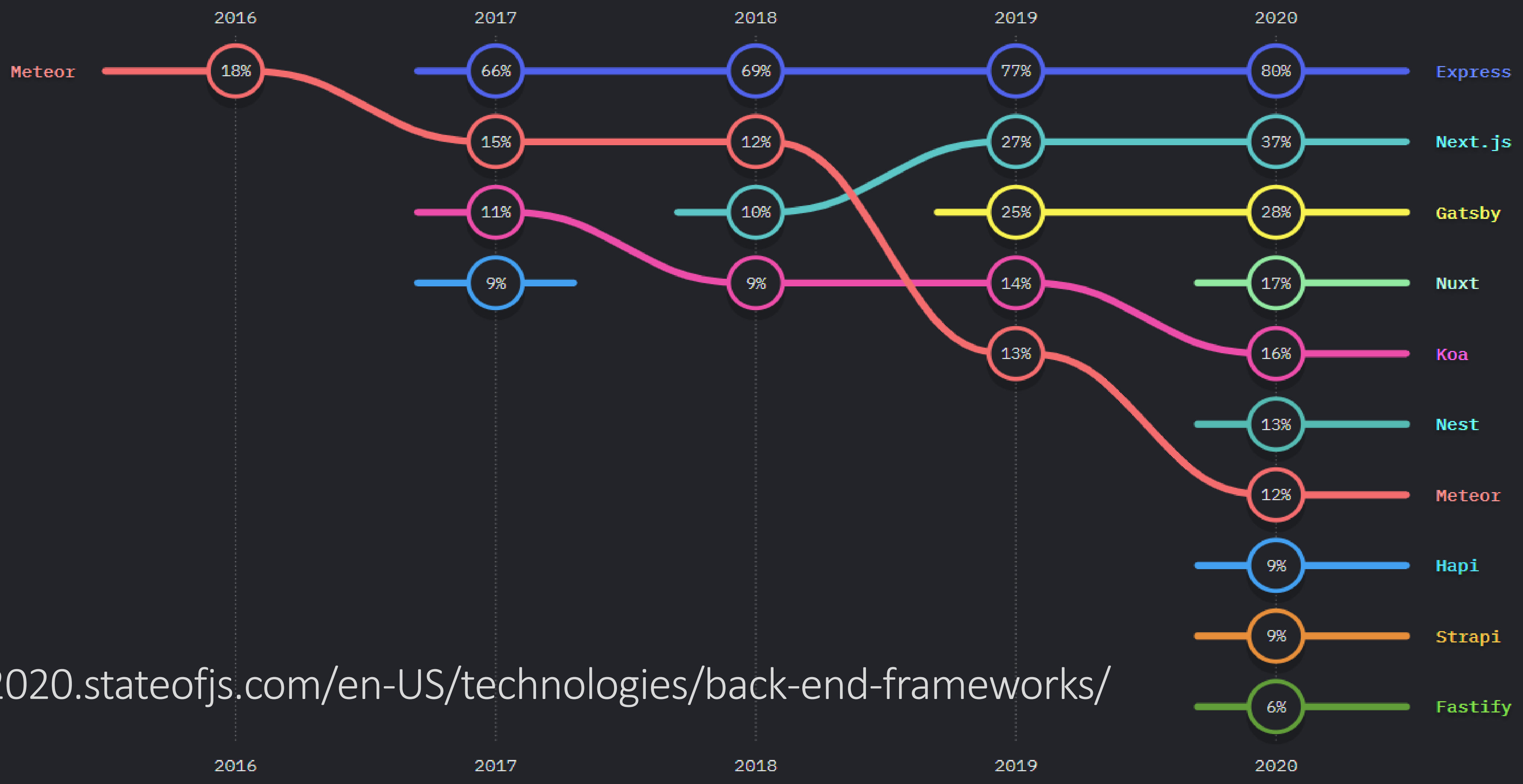
ICE
704
DB
„Ich blüh auf –
mit Hanf!"
STILL

Rankings

Export

Satisfaction | Interest | [✓ Usage] | Awareness

Satisfaction, interest, usage, and awareness ratio rankings.



https://2020.stateofjs.com/en-US/technologies/back-end-frameworks/

Express.js

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

https://expressjs.com

Why Express.js

- faster development
- much easier and powerful declaring routing rules
- convenient to use middleware
- integration with template engines
- flexible and uniwersal

Express.js - basics

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('hello world!');
});

app.listen(4700, () => console.log('server started'));
```

Express.js / Node.js

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('hello world!');
});

app.listen(4700,
 () => console.log('server started’));
```

```
const http = require("http");

const app = http.createServer((req, res) => {
    response.end("Hello World");
});

app.listen(4700);
```

Routing

Routing is determining how an application responds to a client request to a specific endpoint and specific HTTP request method (GET, POST, etc.).

Each route can have one or more handler functions that are executed after the path is matched.

Routing structure

```
app.method(PATH, HANDLER)
```

app - an instance of our server

method - HTTP request method (lowercase)

PATH - the path on the server

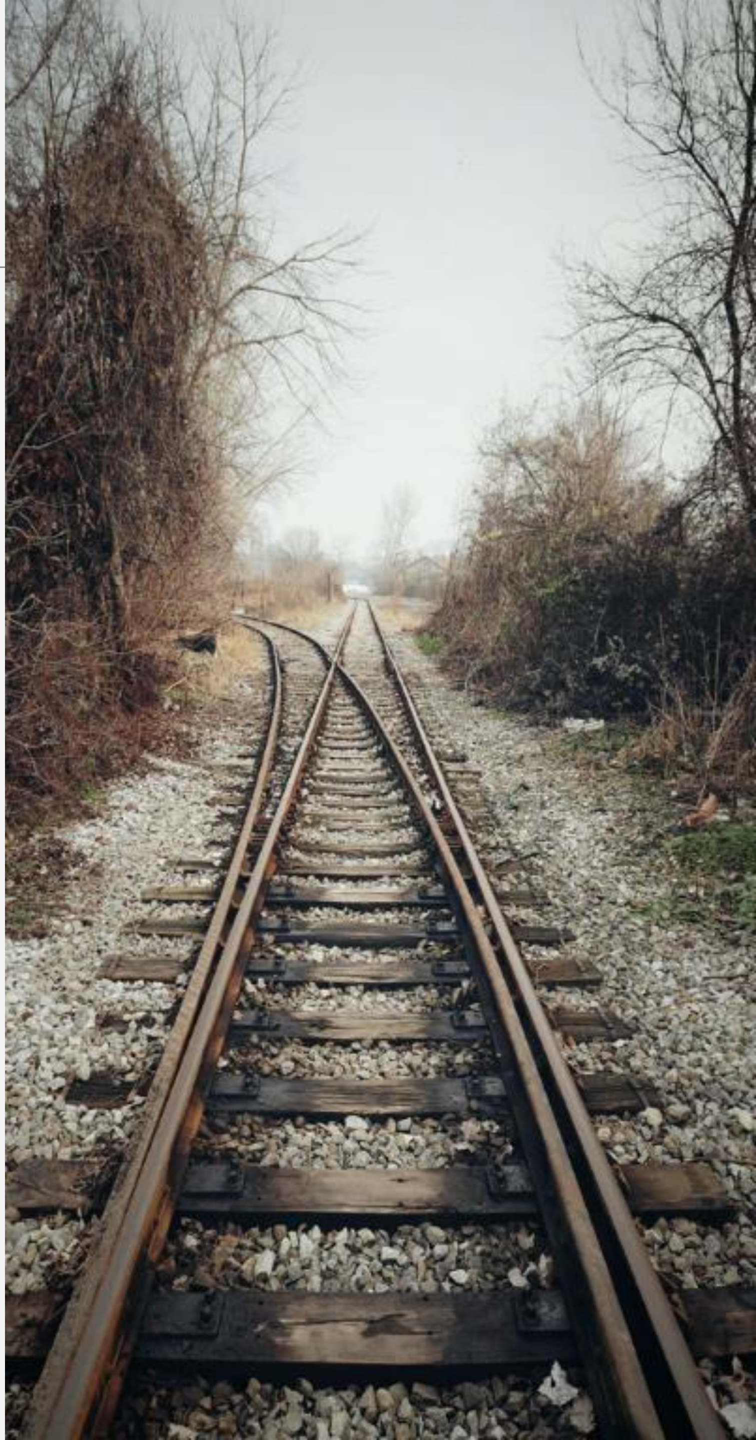
HANDLER - function executed after path matching

Routing example

```
app.get('/', (req, res) => {
    res.send('hello world!');
});

app.post('/', (req, res) => {
    res.send('Got a POST request');
});

app.all('/', (req, res) => {
    res.send('Any of HTTP method');
});
// rules are applied from top to bottom!
```



Routing paths

Paths, together with the request method, define endpoints.

Paths can take:

- a plain address(string)
- a pattern(string patterns)
- a regular expression(RegExp).

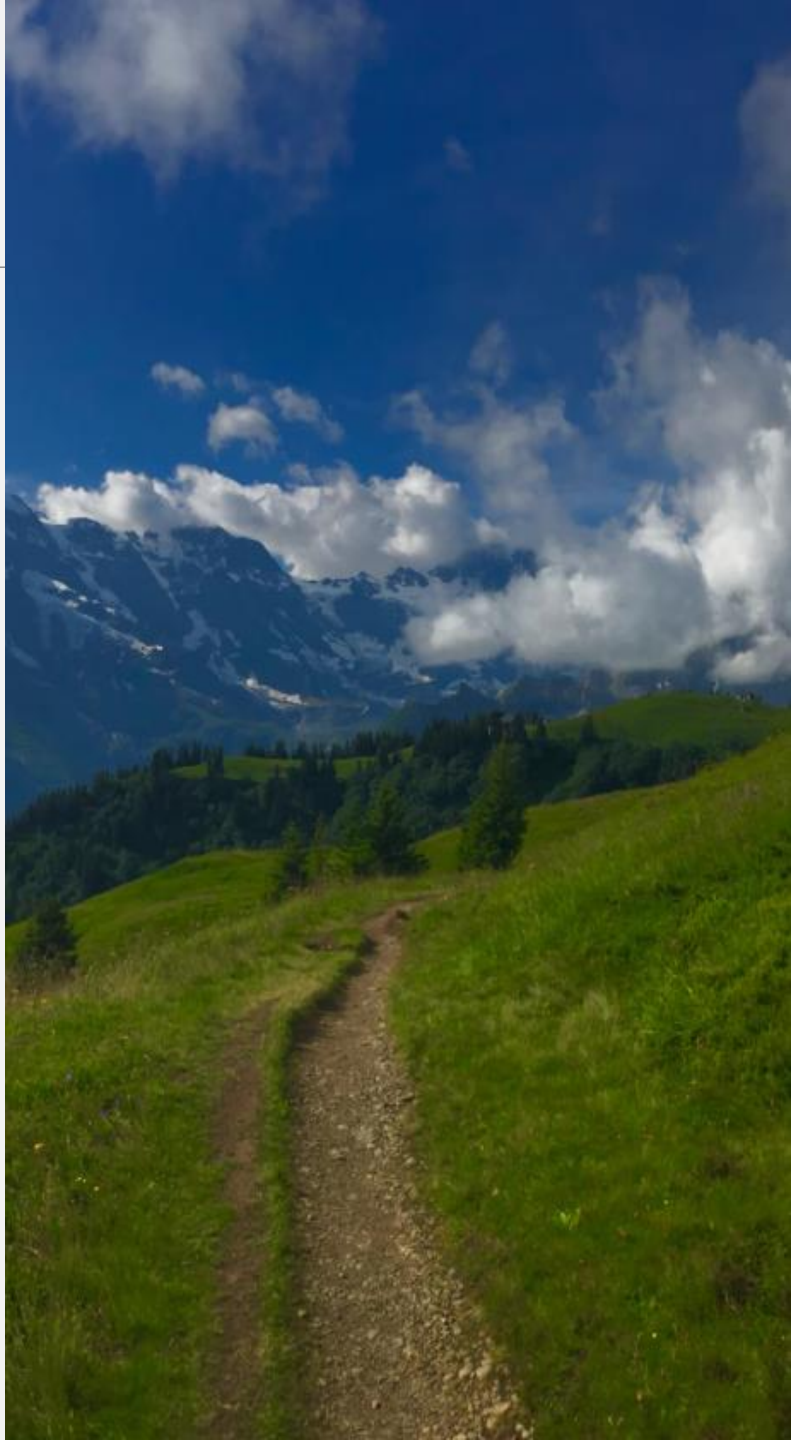
The characters ?, +, *, and () are subsets of their equivalents in regular expressions. The hyphen (-) and period (.) are interpreted literally according to string-based paths.

Routing paths - plain address

```
app.get('/users', (req, res) => {
    // ...
});

app.post('/posts.txt', (req, res) => {
    // ...
});

app.delete('/comments.json', (req, res) => {
    // ...
})
```



Routing paths - pattern

```
// matches: user, users
app.get('/users?', (req, res) => {
    // ...
});

// matches: users, userss, usersss, ...
app.post('/users+', (req, res) => {
    // ...
});

// matches: users, usxxxers, usRANDOMers
app.delete('/us*ers', (req, res) => {
    // ...
});
```

Routing paths - regular expression

```
// matches: file.txt, abc/kot.txt
app.get(/.*\.txt/, (req, res) => {
    // ...
});

// matches: ala, alaMaKota ...
app.post(/^ala.*/, (req, res) => {
    // ...
});
```

Route parameters

Route parameters are named segments of URLs, that are used to capture the values specified at their position in the URL.

The captured values are populated in the req.params object, and the route parameter name is specified in the path as the corresponding keys.

Query string parameters

```
http://localhost:4500?users=12&posts=44
```

Route parameters:

```
http://localhost:4500/users/12/posts/44
```

Route parameters

```
// Path: /users/:userId/posts/:postId
// URL: http://localhost:4500/users/12/posts/44
app.get('/users/:userId/posts/:postId', (req, res) => {
    // req.params: { "userId": "12", "postId": "44" }
});
```

Route parameters

```
// Path: /getFile/:filename.:extension
// URL: http://localhost:4500/getFile/somefile.txt
app.get('/getFile/:filename.:extension', (req, res) => {
    // req.params: { "filename": "somefile", "extension": "txt" }
});
```

Route handlers

A route can have multiple callback functions that are executed sequentially until a call is made to to send a reply to the client.

The condition is that intermediate functions use the callback next function.

Routing procedures can take the form of a function, a series of functions or a combination of both.

Single callback

```
app.get('/ala-ma-kota', (req, res) => {
    // ...
});
```

Multiple callbacks

```
app.get(
    '/ala-ma-kota',
    (req, res, next) => { ...; next() },
    (req, res, next) => { ...; next() },
    (req, res) => { ... }
);
```

Combine callbacks

```
const callback1 = (req, res, next) => { ...; next() }
const callback2 = (req, res, next) => { ...; next() }
app.get(
    '/ala-ma-kota',
    [ callback1, callback2 ],
    (req, res) => { ... }
);
```

app.route()

You can create chainable route handlers for a route path by using app.route().

Because the path is specified at a single location, creating modular routes is helpful, as is reducing redundancy and typos.

app.route()

```
app.route('/users')
    .get((req, res) => {
        // ...
    })
    .post((req, res) => {
        // ...
    })
    .delete((req, res) => {
        // ...
    });
```

Express Router

The Express.Router class is used to create modular, sets of path handling routes.

A router instance is a complete software including routing system, also referred to as a mini application.

Express Router - example

```javascript
// ./dashboard.js
const express = require('express');
const router = express.Router();

router.use((req, res, next) => {
    console.log('time: ', Date.now());
    next();
});

router.get('/', (req, res) => {
    res.send('hello world!');
});
module.exports = router;
```

```javascript
// ./app.js
const express = require('express');
const dashboard = require('./dashboard');
const app = express();

app.use('/dashboard', dashboard);
app.listen(4700,
  () => console.log('server started'));
```

Response methods

- res.download() – send file to download
- res.end() – end response
- res.json() – send json response
- res.redirect() – redirect response
- res.render() – render view
- res.send() – send response with one of possible content types
- res.sendFile() – send file as stream
- res.sendStatus() – send and set status code as body