



ALTERNatywy dla EXPRESS JS

SPIS TREŚCI

- 01** CECHY EXPRESS JS
- 02** NODE JS
- 03** SAILS JS
- 04** NEST JS
- 05** SŁOWA KOŃCOWE

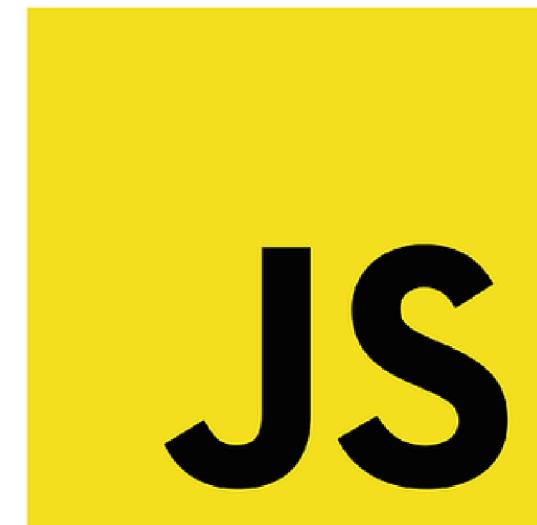
CECHY EXPRESS JS

- Duże community
- Składnia
- Prosty do nauki
- Wsparcie szablonów HTML
- Dużo bibliotek

- Bazowo posiada mało funkcjonalności
- Elastyczność

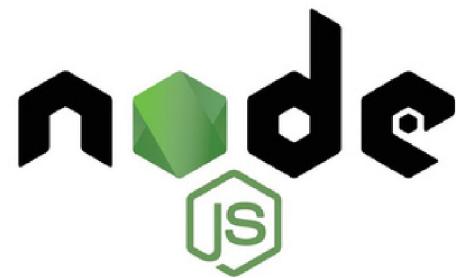
- Bezpieczeństwo
- Obsługa błędów

Express



```
1  const express = require("express");
2  const cors = require("cors");
3  const app = express();
4  const port = 3000;
5
6  // Middleware to enable CORS
7  app.use(cors());
8
9  // Middleware to parse JSON bodies
10 app.use(express.json());
11
12 // Route with URL parameters
13 app.get("/user/:id", (req, res) => {
14   const { id } = req.params;
15   res.send(`User ID: ${id}`);
16 });
17
18 // Route with query parameters
19 app.get("/search", (req, res) => {
20   const { q } = req.query;
21
22   if (q === undefined) {
23     return res.status(400).json({ error: "Missing query parameter 'q'" });
24   }
25
26   res.send(`Search query: ${q}`);
27 });
28
29 // Route handling POST requests
30 app.post("/data", (req, res) => {
31   const data = req.body;
32
33   res.json({ receivedData: data });
34 });
35
36 // Catch-all route for undefined routes
37 app.use((req, res) => {
38   res.status(404).send("404: Page Not Found");
39 });
40
41 // Start the server
42 app.listen(port, () =>
43   console.log(`Server running at http://localhost:\${port}`));
44 );
```

NODE JS



- Jest wydajniejszy
- Pełna kontrola nad architekturą
- Większa kontrola nad zapytaniami HTTP
- Biblioteki dla Express JS nie są kompatybilne z Node JS
- Starsze biblioteki
- Wyższy próg wejścia
- Bez framework'a pisze w nim coraz mniej osób

```
1 const http = require('http');
2 const url = require('url');
3
4 const port = 3000;
5
6 // Create a server
7 const server = http.createServer((req, res) => {
8   const parsedUrl = url.parse(req.url, true);
9   const { pathname, query } = parsedUrl;
10
11 // Middleware to enable CORS
12 res.setHeader('Access-Control-Allow-Origin', '*');
13
14 // Middleware to parse JSON bodies
15 if (req.method === 'GET' || req.headers['content-type'] === 'application/json') {
16   let body = '';
17
18   req.on('data', (chunk) => {
19     body += chunk.toString();
20   });
21
22   req.on('end', () => {
23     if (req.method !== 'GET') {
24       try {
25         req.body = JSON.parse(body);
26       } catch (e) {
27         res.writeHead(400, { 'Content-Type': 'application/json' });
28         return res.end(JSON.stringify({ error: 'Invalid JSON body' }));
29     }
30   });
31 }
```

```
31
32 // Route handling
33 if (req.method === 'GET') {
34   if (pathname.startsWith('/user/')) {
35     const id = pathname.split('/').pop();
36     res.end(`User ID: ${id}`);
37   } else if (pathname === '/search') {
38     const { q } = query;
39     if (!q) {
40       res.writeHead(400, { 'Content-Type': 'application/json' });
41       res.end(JSON.stringify({ error: "Missing query parameter 'q'" }));
42     } else {
43       res.end(`Search query: ${q}`);
44     }
45   } else {
46     res.writeHead(404, { 'Content-Type': 'text/plain' });
47     res.end('404: Page Not Found');
48   }
49 } else if (req.method === 'POST') {
50   if (pathname === '/data') {
51     res.writeHead(200, { 'Content-Type': 'application/json' });
52     res.end(JSON.stringify({ receivedData: req.body }));
53   } else {
54     res.writeHead(404, { 'Content-Type': 'text/plain' });
55     res.end('404: Page Not Found');
56   }
57 } else {
58   res.writeHead(405, { 'Content-Type': 'text/plain' });
59   res.end('405: Method Not Allowed');
60 }
61 });
62 } else {
63   res.writeHead(415, { 'Content-Type': 'text/plain' });
64   res.end('415: Unsupported Media Type');
65 }
66 });
67
68 // Start the server
69 server.listen(port, () => {
70   console.log(`Server running at http://localhost:\${port}`);
71 });
--
```

SAILS JS

- Model View Controller
- Blueprint API
- Wiele rzeczy jest wbudowane w framework
- Waterline ORM
- Zbudowane na Express JS
- Konwencja ponad konfigurację
- Znacznie mniejsze community
- Wysoki próg wejścia



- Aplikacja stworzona zgodnie z docsami (wybrałem pustą templatkę)

- W config/security.js dodajemy:

```
16 module.exports.security = {  
17   ...  
18   >   /*****  
19   >   |  
20   >   |  
21   >   |  
22   >   |  
23   >   |  
24   >   |  
25   >   |  
26   >   |  
27   >   |  
28   >   |  
29   >   |  
30   >   |  
31   >   cors: {  
32   >     allRoutes: true,  
33   >     allowOrigins: '*',  
34   >     allowCredentials: false,  
35   >   },
```

- Sails używa body-parsera, ale można zmienić jego ustawienia

- W api/controllers tworzymy UserController.js zawierający:

```
2 module.exports = {
3   ...
4   getUser: function (req, res) {
5     const { id } = req.params;
6     return res.send(`User ID: ${id}`);
7   },
8 }
```

- W api/controllers tworzymy SearchController.js zawierający:

```
2 module.exports = {
3   ...
4   search: function (req, res) {
5     const { q } = req.query;
6
7     if (q === undefined) {
8       return res.status(400).json({ error: 'Missing query parameter \'q\' ' });
9     }
10
11     return res.send(`Search query: ${q}`);
12   },
13 }
```

- W api/controllers tworzymy DataController.js zawierający:

```
2 module.exports = {
3   ...
4   postData: function (req, res) {
5     const data = req.body;
6     return res.json({ receivedData: data });
7   },
8 }
```

- W config/routes.js dodajemy:

- Page Not Found jest obsługiwany bez dodatkowych ustawień i zwracany jest html z views/404.ejs

```
11 module.exports.routes = {
12   > /**
13    * Route with URL parameters
14    'GET /user/:id': 'UserController.getUser',
15
16    * Route with query parameters
17    'GET /search': 'SearchController.search',
18
19    * Route handling POST requests
20    'POST /data': 'DataController.postData',
21
22    * Catch-all route for undefined routes
23    '/*': {
24      fn: function (req, res) {
25        return res.status(404).send('404: Page Not Found');
26      },
27      skipAssets: true,
28      skipRegex: /^\/api\//,
29    },
30  };
31
32
33
34
35
36
37
38
39
40 }
```

NEST JS

- Konwencją jest TS ale jest kompatybilny z JS
- Wstrzykiwanie zależności
- Wspiera testowanie bez dodatkowych bibliotek
- Community porównywalne z Expressem
- Test Driven Development
- Zbudowany na Express JS bądź Fastify JS
- Konwencja ponad konfigurację
- Angularowa struktura
- Duże powiązanie z Angularem

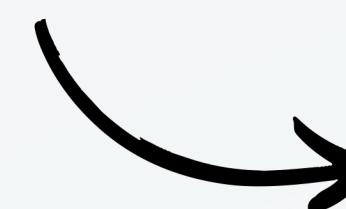


nest

Aplikacja stworzona
zgodnie z [docsami](#)



Nest JS automatycznie
obsługuje parsowanie
JSONów



Aby użyć CORS
wystarczy dodać linijkę
7 w src/main.ts

```
5  async function bootstrap() {  
6    const app = await NestFactory.create(AppModule);  
7    app.enableCors();  
8    await app.listen(3000);  
9  }
```

W src tworzymy
user.controller.ts
zawierający:

```
2 import { Controller, Get, Param, Res, HttpStatus } from '@nestjs/common';
3 import { Response } from 'express';
4
5 @Controller('user')
6 export class UserController {
7   @Get(':id')
8   getUser(@Param('id') id: string, @Res() res: Response) {
9     res.status(HttpStatus.OK).send(`User ID: ${id}`);
10  }
11 }
```

W src tworzymy
search.controller.ts
zawierający:

```
2 import { Controller, Get, Query, Res, HttpStatus } from '@nestjs/common';
3 import { Response } from 'express';
4
5 @Controller('search')
6 export class SearchController {
7   @Get()
8   search(@Query('q') q: string, @Res() res: Response) {
9     if (!q) {
10       res
11         .status(HttpStatus.BAD_REQUEST)
12         .json({ error: "Missing query parameter 'q'" });
13     } else {
14       res.status(HttpStatus.OK).send(`Search query: ${q}`);
15     }
16   }
17 }
```

W src tworzymy
data.controller.ts
zawierający:

```
2 import { Controller, Post, Body, Res, HttpStatus } from '@nestjs/common';
3 import { Response } from 'express';
4
5 @Controller()
6 export class DataController {
7   @Post('data')
8   postData(@Body() data: any, @Res() res: Response) {
9     res.status(HttpStatus.OK).json({ receivedData: data });
10  }
11 }
```

Aby dodać endpointy
wystarczy je zaimportować
oraz dodać do linii 10

```
3 import { UserController } from './user.controller';
4 import { SearchController } from './search.controller';
5 import { DataController } from './data.controller';
6 import { NotFoundMiddleware } from './not-found.middleware';
7
8 √ @Module({
9   imports: [],
10  controllers: [UserController, SearchController, DataController],
11  providers: [],
12 })
```

W src tworzymy
not-found.middleware.ts
zawierający:

```
2 import { Injectable, NestMiddleware, HttpStatus } from '@nestjs/common';
3 import { Request, Response, NextFunction } from 'express';
4
5 @Injectable()
6 export class NotFoundMiddleware implements NestMiddleware {
7   use(req: Request, res: Response, next: NextFunction) {
8     res.status(HttpStatus.NOT_FOUND).send('404: Page Not Found');
9   }
10 }
```

Aby zastosować ten
middleware dodajemy
importy

```
2 import { Module, MiddlewareConsumer, RequestMethod } from '@nestjs/common';

```

Konfigurujemy
aplikacje

```
13 export class AppModule {
14   configure(consumer: MiddlewareConsumer) {
15     consumer
16       .apply(NotFoundMiddleware)
17       .exclude(
18         { path: 'user/:id', method: RequestMethod.GET },
19         { path: 'search', method: RequestMethod.GET },
20         { path: 'data', method: RequestMethod.POST },
21       )
22       .forRoutes({ path: '*', method: RequestMethod.ALL });
23   }
24 }
```

Normalnie w Nest JS Page
Not Found jest obsługiwane
za pomocą funkcji filtrującej.
Przykład.

SŁOWA KOŃCOWE

- Frameworki to tylko narzędzia
- Na Backendzie JS to nie jedyna opcja
- Zachęcam do samodzielnego głębszego poznania Sails JS i Nest JS
- Istnieją inne frameworki do Node.js. Takie jak Koa, Meteor, Fastify i wiele innych

Prezentacja oraz kod dostępne są na moim GitHubie:
<https://github.com/karmatys8/JS/tree/main/prez>

DZIĘKUJĘ ZA UWAGĘ

*Jeżeli ktoś ma jakieś pytania
to teraz jest czas je zadać!*

