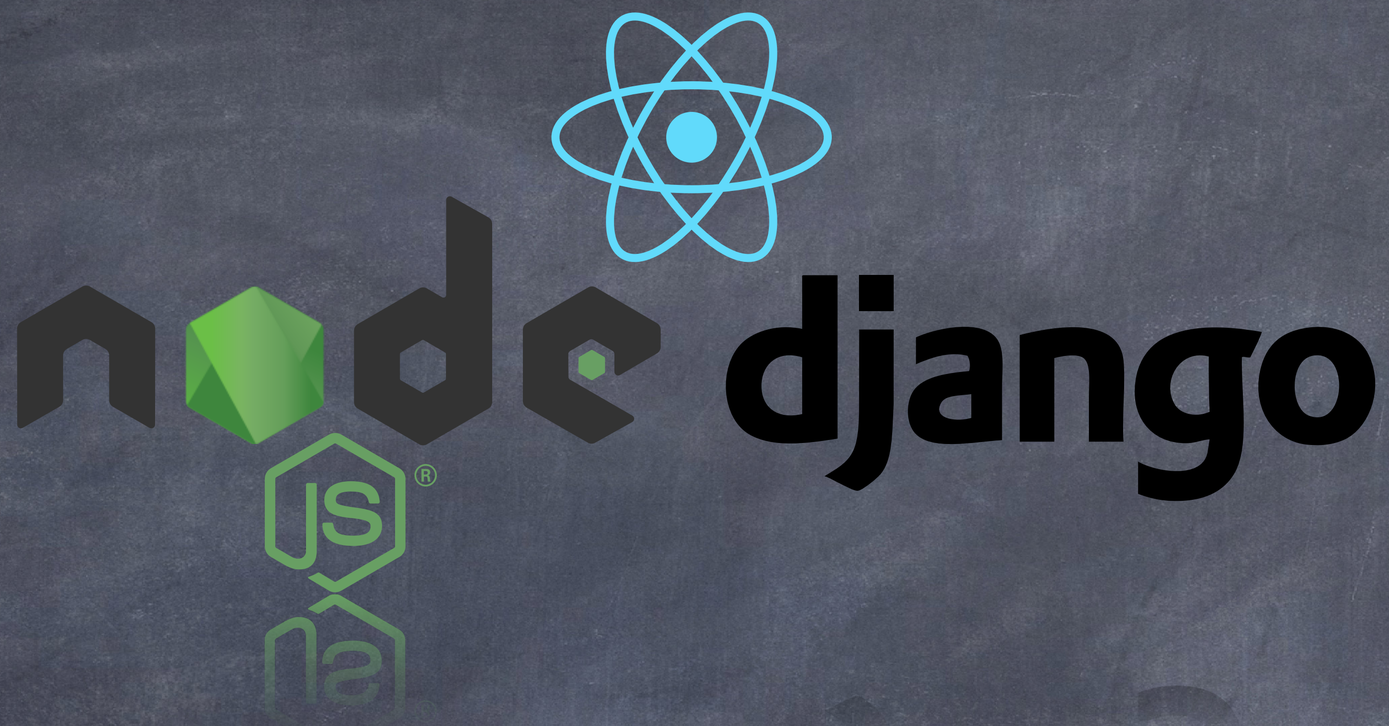


PROGRAMOWANIE APLIKACJI WEBOWYCH
PRACOWNIA APLIKACJI WEBOWYCH

inż. Paweł Palacz

O MNIE

- Wykształcenie:
  - **Inżynier informatyki** (Aplikacje Internetowe i Mobilne) - Collegium Da Vinci w Poznaniu
  - **Technik informatyk** - Zespół Szkół Komunikacji im. Hipolita Cegielskiego w Poznaniu
- Doświadczenie zawodowe:
  - **Fullstack Web Developer (Symfony/ Node/Next.js)** - Fabrity (od 2022)
  - **Self-employed Web Developer** - in2dev (od 2022)
  - **PHP (Symfony) developer** - as4you (2020-2022)

Obraz autorstwa storyset na Freepik

ZASADY OCENIANIA - WYKŁAD

- I semestr
  - 1x sprawdzian
  - minimum 2x kartkówka (nie)zapowiedziana*
  - odpowiedź ustna**
  - praca na lekcji***
  - nieprzygotowanie 1x

Obraz autorstwa storyset na Freepik

ZASADY OCENIANIA - WYKŁAD

- II semestr
  - 1x sprawdzian
  - minimum 2x kartkówka (nie)zapowiedziana*
  - odpowiedź ustna**
  - praca na lekcji***
  - projekt****
  - nieprzygotowanie 1x

Obraz autorstwa storyset na Freepik

ZASADY OCENIANIA - PRACOWNIE

- I semestr
  - 1x sprawdzian
  - zadania z lekcji
  - praca na lekcji*
  - nieprzygotowanie 1x

Obraz autorstwa storyset na Freepik

ZASADY OCENIANIA - PRACOWNIE

- II semestr
  - 1x sprawdzian
  - zadania z lekcji
  - praca na lekcji*
  - projekt**
  - nieprzygotowanie 1x

Obraz autorstwa storyset na Freepik

KONTAKT Z PROWADZĄCYM

- Email
  - pawel.palacz@zsk.poznan.pl (zalecany)
- Teams
- E-dziennik

Obraz autorstwa storyset na Freepik

TEMATYKA ZAJĘĆ

Obraz autorstwa storyset na Freepik

TEMATYKA ZAJĘĆ

- Cechy i zastosowania Node.js
- Przygotowanie środowiska pracy dla Node.js - Instalacja, dobór IDE, przydatne narzędzia
- npm - menadżer pakietów dla Node.js
- Rozpoczęcie pracy z Node.js
- Implementacja serwera HTTP w Node.js (moduł http)

TEMATYKA ZAJĘĆ

- Strumienie w Node.js
- Wbudowane moduły Node.js
- express.js - framework dla Node.js
- Połączenie Node.js z relacyjną bazą danych
- Połączenie Node.js z nierelacyjną bazą danych

TEMATYKA ZAJĘĆ

- Własne moduły dla Node.js
- Tworzenie i wykorzystywanie middleware
- Obsługa błędów w express
- Wstęp do React.js
- Podstawy React.js

TEMATYKA ZAJĘĆ

- Hooki
- Hooki - interfejs API
- Testowanie aplikacji React.js
- Stylowanie aplikacji React.js
- Python - podstawy; powtórzenie wiadomości

TEMATYKA ZAJĘĆ

- Django - framework webowy dla języka Python
- Django - Request i Response
- Django - Modele i panel administracyjny
- Django - Widoki i szablony
- Django - formularze i widoki generyczne

TEMATYKA ZAJĘĆ

- Django - testowanie
- Django - dokładna analiza frameworka

To jest ta scena, w której zadajecie pytania

CECHY I ZASTOSOWANIA NODE.JS

HISTORIA NODE.JS

- Pierwsze wydanie: 27 maja 2009 (Linux, OS X)
- Pierwsze wydanie dla Windows: 2011
- Powstanie OpenJS Foundation (połączenie JS Foundation oraz Node.js Foundation): 2019

ALE CO TO TEN NODE?

- Node.js to połączenie silnika V8 (Google), biblioteki libUV oraz kilku innych bibliotek
- Pierwotnym celem powstania Node.js było dodanie do stron internetowych technologii push

ZASTOSOWANIA NODE.JS

- Interfejsy API i serwery
- Bazy danych
- Narzędzia CLI
- Automatyzacja
- Aplikacje desktopowe (Electron)

CECHY NODE.JS

- Asynchroniczność
- Podejście „event-driven”
- Skalowalność
- Brak zakleszczeń (brak blokowania)

PRZEGLĄDARKOWY „HELLO WORLD" W NODE.JS

```
const http = require('http')
const hostname = '127.0.0.1'
const port = 3000

const server = http.createServer((req, res) => {
    res.statusCode = 200
    res.setHeader('Content-Type', 'text/plain')
    res.end('Hello World')
})

server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`)
})
```

PRZYGOTOWANIE ŚRODOWISKA PRACY DLA NODE.JS

INSTALACJA, DOBÓR IDE, PRZYDATNE NARZĘDZIA

INSTALACJA NODE.JS

- Windows
  - https://nodejs.org/en/download
- Linux / macOS
  - https://github.com/nvm-sh/nvm
- Docker
  - https://hub.docker.com/_/node/

PRZYKŁAD DOCKERFILE DLA PROJEKTU W NODE.JS

```
FROM node:18-alpine

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install

EXPOSE 3000

CMD ["npm", "run", "start"]
```

PRZYKŁAD COMPOSE DLA PROJEKTU W NODE.JS

```yaml
services:
  node:
    container_name:  node
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - .:/app
      - /app/node_modules
    restart: always
    ports:
      - "3000:3000"
```

DOBÓR IDE DLA NODE.JS

VS

PRZYDATNE NARZĘDZIA PRZY PRACY Z NODE.JS

- ***ESLint*** - pomaga poprawić niespójne elementy kodu, wyszukuje błędy
- ***TSLint*** - ESLint dla Typescript
- ***Prettier*** - wymusza code-style według opisanych reguł
- ***husky*** - ułatwia tworzenie git hooks

NPM - MENADŻER PAKIETÓW DLA NODE.JS

CO TO JEST NPM?

- Domyślny manager pakietów dla środowiska Node.js
  - Pierwsze wydanie: **12 stycznia 2010**
  - Może być używany do zarządzania warstwą front-end aplikacji WWW
- Repozytorium pakietów (nazywane rejestrem npm)
- Nazwa firmy (npm, Inc.) która nim zarządza

ALE CO TO TEN PAKIET?

- Każda aplikacja w Node.js **może** być pakietem (ang. package)
- Inicjalizacja pakietu jest wymagana do instalacji innych pakietów z repozytorium npm

INICJALIZACJA PAKIETU

$ npm init

INICJALIZACJA PAKIETU

- Po inicjalizacji pakietu zostaje utworzony plik **package.json**
  - pole „name” - nazwa pakietu
  - pole „version” - wersja pakietu (***wersjonowanie semantyczne***)
  - pole „description” - opis pakietu
  - pole „main” - główny skrypt (*entrypoint*) pakietu
  - pole „scripts” - obiekt zawierający skrypty uruchamiane przez CLI npm
  - pole „author” - autor pakietu
  - pole „license” - licencja, na której został wydany pakiet

PRZYDATNE KOMENDY NPM

- **npm i[nstall] <nazwa_pakietu> [—dev]** - instaluje wskazany pakiet [flaga —dev oznacza zależność developerską]
- **npm test** - uruchamia skrypt testów
- **npm uninstall <nazwa_pakietu>** - odinstalowuje wskazany pakiet
- **npx <pakiet> <komenda_pakietu>** - wywołanie komendy z pakietu bez instalowania go jako zależności

ALTERNATYWY DLA NPM

- **Yarn** - powstał jako alternatywa w czasach słabego rozwoju NPM
- **PNPM** - używa wspólnego cache i instaluje zależności tylko raz
- **Bower** - utworzony dla rozwiązań front-endowych

ROZPOCZĘCIE PRACY Z NODE.JS

INTERAKTYWNE CLI NODE.JS (REPL)

```
$ node
Welcome to Node.js v20.10.0.
Type ".help" for more information.
>
```

UTWORZENIE PROJEKTU NPM (COMMONJS)

$ npm init

UTWORZENIE PROJEKTU NPM (ES MODULES)

$ npm init
$ vi package.json

```
...
"type": "module"
...
```

UTWORZENIE PROJEKTU Z POZIOMU IDE

ROZSZERZENIA PLIKÓW - JAK SIĘ NIE POGUBIĆ

- ***.js*** - składnia zgodna z zapisem w *package.json* (domyślnie ***commonjs***), a w przypadku braku inicjalizacji pakietu ***commonjs***
- ***.cjs*** - skrypt napisany w składni *commonjs*, **niezależnie** od typu wskazanego w *package.json*
- ***.mjs*** - skrypt napisany w składni *ES Modules*, **, niezależnie od typu wskazanego w *package.json***

IMPLEMENTACJA SERWERA HTTP W NODE.JS (MODUŁ HTTP)

IMPLEMENTACJA SERWERA HTTP - COMMONJS

```
// index.[c]js

const http = require('http')
const hostname = '127.0.0.1'
const port = 3000
const server = http.createServer((_req, res) => {
    res.statusCode = 200
    res.setHeader('Content-Type', 'text/plain')
    res.end('Hello World')
})
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`)
})
```

IMPLEMENTACJA SERWERA HTTP - ES MODULES

```
// index.[m]js

import http from "http"
const hostname = '127.0.0.1'
const port = 3000
const server = http.createServer((_req, res) => {
    res.statusCode = 200
    res.setHeader('Content-Type', 'text/plain')
    res.end('Hello World')
})
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`)
})
```

IMPLEMENTACJA SERWERA HTTP - TYPESCRIPT

```
// index.ts

import * as http from „http"

const hostname: string = '127.0.0.1'
const port: number = 3000

const server: http.Server = http.createServer((_req: http.IncomingMessage, res:
http.ServerResponse) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/plain')
  res.end('Hello World')
})

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`)
})
```

PROSTY ROUTING W NODE

```
// index.js

const srv = http.createServer((req, res) => {
    const url = req.url

    if (url === '/') {
        //…
    } else {
        //…
    }
})
```

PROSTY ROUTING W NODE (TS)

```
// index.ts

const srv: Server = createServer((req, res) => {
  const url: string | undefined = req.url

  if (url === '/') {
    //…
  } else {
    //…
  }
})
```

ZWRACANIE PLIKU HTML

```
// index.js

import http from "http"
import { readFile } from 'fs/promises'

const srv = http.createServer(async (req, res) => {
    const url = req.url
    if (url === '/') {
        res.statusCode = 200
        const html = await readFile('./home.html')
        res.setHeader('content-type', 'text/html')
        res.write(html)
        res.end()
    } else {
        //…
    }
})
```

ZWRACANIE PLIKU JSON

```
// index.js

import http from "http"

const srv = http.createServer((req, res) => {
    res.statusCode = 200
    const json = {
        hello: 'world',
    }
    res.setHeader('content-type', 'application/json')
    res.write(JSON.stringify(json))
    res.end()
})
```

IMPLEMENTACJA FORMULARZA

```html
<!doctype html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>Formularz</title>
</head>
<body>
    <form action="/message" method="POST">
        <input type="text" name="message">
        <button type="submit">Send</button>
    </form>
</body>
</html>
```

IMPLEMENTACJA FORMULARZA

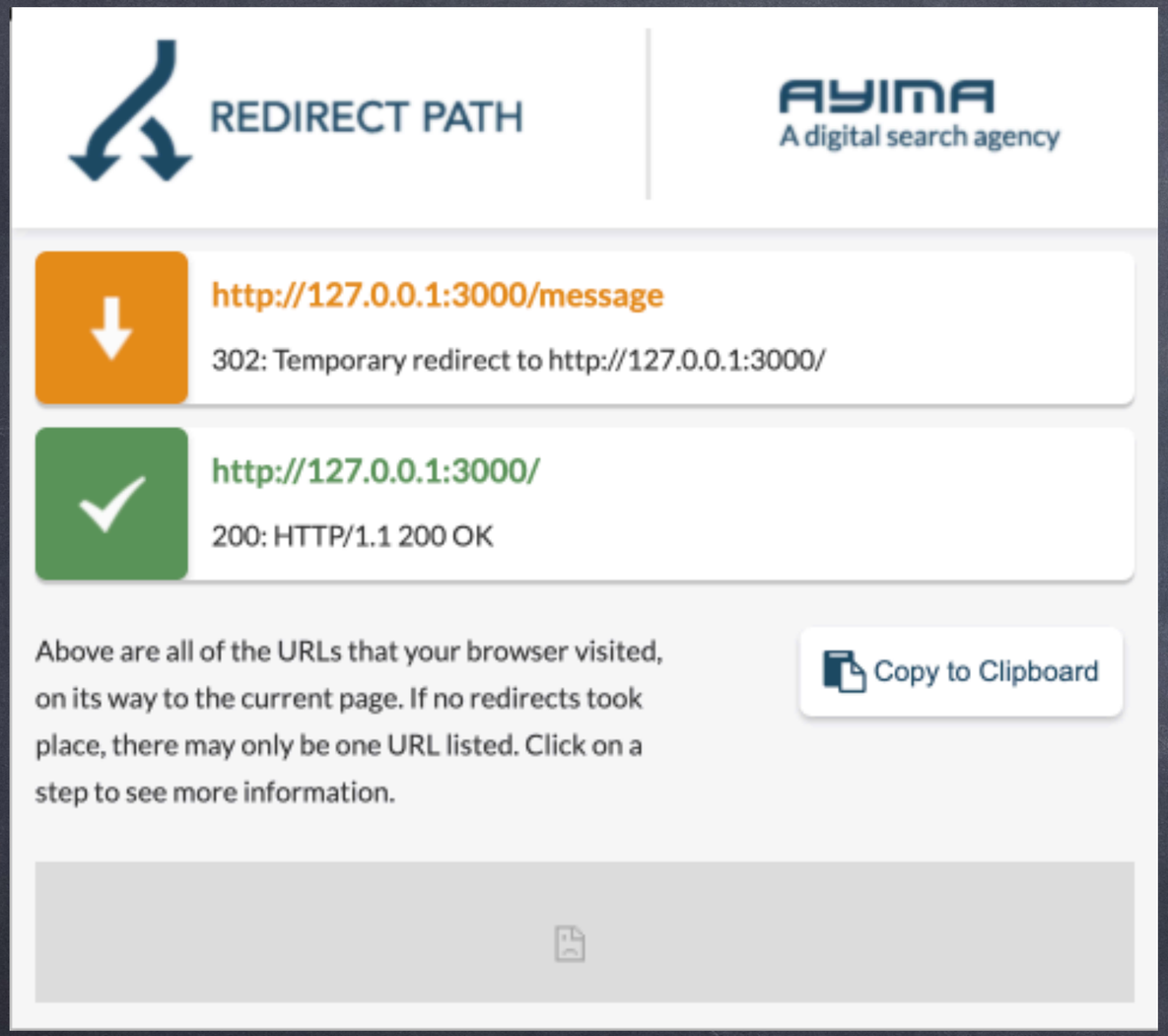
```
// index.[c]js

const http = require('http')
const { writeFile, readFile } = require('fs/promises')

const srv = http.createServer((req, res) => {
    const url = req.url
    const method = req.method

    if (url === '/') {
        const html = readFile('form.html')
        return res.end(html)
    }
```

IMPLEMENTACJA FORMULARZA

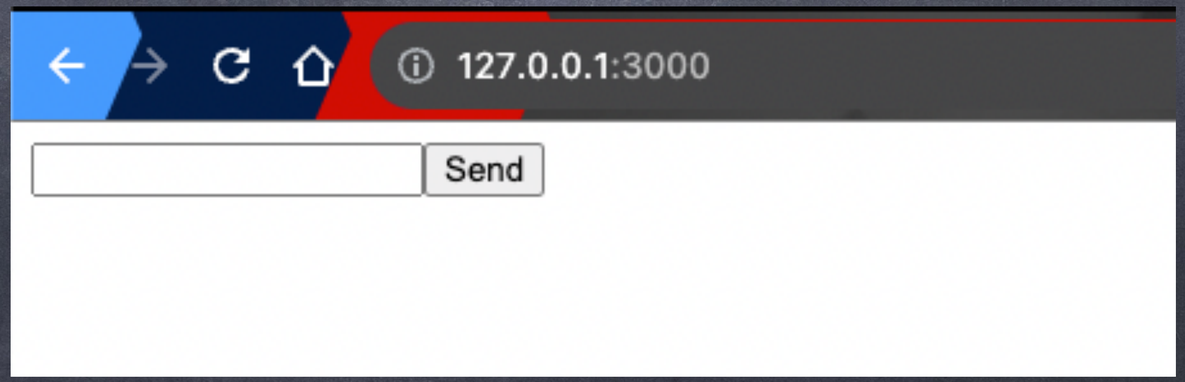
```
if (url === '/message' && method === 'POST') {
        const body = []
        req.on('data', (chunk) => {
            console.log(chunk.toString())
            body.push(chunk)
        })

        req.on('end', async () => {
            const parsedBody = Buffer.concat(body).toString()
            const message = parsedBody.split('=')[1]
            await writeFile(`message-${Date.now().toString()}.txt`, message)
            res.statusCode = 302
            res.setHeader('Location', '/')
            return res.end()
        })
    }
})
```

IMPLEMENTACJA FORMULARZA

```
srv.listen(3000, '127.0.0.1', () => {
    console.log(`Listening on http://127.0.0.1:3000`)
})
```




```
✘ untitled ▶ node form.js
Listening on http://127.0.0.1:3000
message=test
```



MODUŁ URL

UŻYCIE MODUŁU URL (CSJ)

```
// index.[c]js

const url = require('url')

const addr = 'http://localhost:8080/default.html?year=2023&month=september'

const query = url.parse(addr, true)
console.log(query.host) // logs 'localhost:8080'
console.log(query.pathname) // logs '/default.html'
console.log(query.search) // logs '?year=2023&month=september'

const qdata = query.query // creates an object: { year: 2023, month:
'september' }
console.log(qdata.month) // logs 'september'
```

UŻYCIE MODUŁU URL (MJS)

```
// index.[m]js

import { parse } from 'url'

const addr = 'http://localhost:8080/default.html?year=2023&month=september'

const query = parse(addr, true)
console.log(query.host) // logs 'localhost:8080'
console.log(query.pathname) // logs '/default.html'
console.log(query.search) // logs '?year=2023&month=september'

const qdata = query.query // creates an object: { year: 2023, month:
'september' }
console.log(qdata.month) // logs 'september'
```

UŻYCIE MODUŁU URL (TS)

```
// index.ts

import { parse, type UrlWithParsedQuery } from 'url'
import { type ParsedUrlQuery } from "querystring"

const addr = 'http://localhost:8080/default.html?year=2023&month=september'

const query: UrlWithParsedQuery = parse(addr, true)
console.log(query.host) // logs 'localhost:8080'
console.log(query.pathname) // logs '/default.html'
console.log(query.search) // logs '?year=2023&month=september'

const qdata: ParsedUrlQuery = query.query // creates an object: { year: 2023,
month: 'september' }
console.log(qdata.month) // logs 'september'
```

WYSYŁKA MAILI

npm install nodemailer
[npm install @types/nodemailer --dev]

WYSYŁKA MAILI (CJS)

```
// index.[c]js

const nodemailer = require('nodemailer')

const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
        user: 'user@gmail.com',
        pass: 'secret-password'
    }
})
```

WYSYŁKA MAILI (CJS)

```
const mailOptions = {
    from: 'user@gmail.com',
    to: 'user@example.com',
    subject: 'Sending E-mail using Node.js',
    text: 'Easy way to send emails'
}

transporter.sendMail(mailOptions, (err, info) => {
    if (err) {
        console.error(err)
    } else {
        console.log(`Email sent: ${info.response}`)
    }
})
```

WYSYŁKA MAILI (MJS)

```
// index.[m]js

import { createTransport } from "nodemailer";

const transporter = createTransport({
    service: 'gmail',
    auth: {
        user: 'user@gmail.com',
        pass: 'secret-password'
    }
})
```

WYSYŁKA MAILI (MJS)

```
const mailOptions = {
    from: 'user@gmail.com',
    to: 'user@example.com',
    subject: 'Sending E-mail using Node.js',
    text: 'Easy way to send emails'
}

transporter.sendMail(mailOptions, (err, info) => {
    if (err) {
        console.error(err)
    } else {
        console.log(`Email sent: ${info.response}`)
    }
})
```

WYSYŁKA MAILI (TS)

```
// index.ts

import {createTransport} from 'nodemailer'
import {Options} from "nodemailer/lib/mailer";

const transporter = createTransport({
  service: 'gmail',
  auth: {
    user: 'user@gmail.com',
    pass: 'secret-password'
  }
})
```

WYSYŁKA MAILI (TS)

```
const mailOptions: Options = {
  from: 'user@gmail.com',
  to: 'user@example.com',
  subject: 'Sending E-mail using Node.js',
  text: 'Easy way to send emails'
}

transporter.sendMail(mailOptions, (err, info) => {
  if (err) {
    console.error(err)
  } else {
    console.log(`Email sent: ${info.response}`)
  }
})
```

WYSYŁKA MAILI - WIELU ODBIORCÓW

```
// index.ts

const mailOptions: Options = {
  from: 'user@gmail.com',
  to: 'user@example.com, user2@example.com',
  subject: 'Sending E-mail using Node.js',
  text: 'Easy way to send emails'
}
```

WYSYŁKA MAILI - FORMAT HTML

```
// index.ts

const mailOptions: Options = {
  from: 'user@gmail.com',
  to: 'user@example.com',
  subject: 'Sending E-mail using Node.js',
  html: '<h1>Welcome</h1><p>That was easy!</p>'
}
```

PRZYJMOWANIE PLIKÓW

npm install formidable
[npm install @types/formidable --dev]

PRZYJMOWANIE PLIKÓW

```
<!--upload-form.html-->

<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Formularz</title>
</head>
<body>
    <form action="/file-upload" method="post" enctype="multipart/form-data">
        <input type="file" name=„filetoupload"><br />
        <input type="submit">
    </form>
</body>
</html>
```

PRZYJMOWANIE PLIKÓW

```
// index.[c]js

const formidable = require('formidable')
const http = require('http')
const fs = require('fs/promises')
const {readFile} = require("fs/promises");

const srv = http.createServer(async (req, res) => {

    if (req.url === '/file-upload') {
        const form = new formidable.IncomingForm()
```

PRZYJMOWANIE PLIKÓW

```
        form.parse(req, async (err, fields, files) => {
            const oldPath = files.filetoupload.filepath
            const newPath = `./upload/${files.filetoupload.originalFilename}`
            await fs.rename(oldPath, newPath)
            res.statusSoce = 302
            res.setHeader('Location', '/')
            res.end()
        })
    }
    else {
        res.writeHead(200, {'Content-Type': 'text/html'});
        const html = await readFile('upload-form.html')
        return res.end(html);
    }
})

srv.listen(3000)
```

STRUMIENIE W NODE.JS

CO TO SĄ STRUMIENIE?

- Jedna z podstawowych koncepcji aplikacji Node
- Metoda obsługi danych oraz sekwencyjnych operacji IO
- Zamiast wczytywać do pamięci cały plik odczytują fragmenty danych „kawałek po kawałku”
- Umożliwiają pracę z plikami większymi niż ilość dostępnej pamięci operacyjnej

DO CZEGO WYKORZYSTUJEMY STRUMIENIE?

- Usługi „streamingowe”
- Praca z mediami
- Praca z dużymi zbiorami danych
- „Komponowalność” w naszym kodzie - przesyłanie strumieniowe między fragmentami kodu

DLACZEGO STRUMIENIE?

- Efektywność pamięciowa - nie musimy ładować do pamięci dużych zbiorów danych, aby je przetwarzać
- Efektywność czasowa - rozpoczęcie przetwarzania danych natychmiast po ich uzyskaniu zajmuje znacznie mniej czasu niż czekanie z przetwarzaniem do momentu przesłania całego ładunku

TYPY STRUMIENI W NODE.JS

- **Writable** - strumienie, do których możemy zapisywać dane. ***fs.createWriteStream()*** pozwala nam zapisywać dane do pliku za pomocą strumieni.
- ***Readable*** - strumienie, z których można odczytywać dane. ***fs.createReadStream()*** pozwala nam odczytać zawartość pliku.
- ***Duplex*** - strumienie, które można odczytywać i zapisywać. - przykład: net.Socket
- ***Transform*** - strumienie, które mogą modyfikować lub przekształcać dane podczas ich zapisu i odczytu. Na przykład w przypadku kompresji plików można zapisywać skompresowane dane i odczytywać zdekompresowane dane do i z pliku.

A CZY TE STRUMIENIE JUŻ GDZIEŚ MIELIŚMY?

```
import {createServer, Server} from "http";

const srv: Server = createServer((req, res) => {})
```

- *req* - strumień Request
- *res* - strumień Response

PRAKTYCZNE PRZYKŁADY

TWORZENIE STRUMIENIA READABLE

TWORZENIE STRUMIENIA READABLE

```
// index.[c]js

const Stream = require('stream')
const readableStream = new Stream.Readable()

// stream is initialized, we can send data to it
readableStream.push('ping!')
readableStream.push('pong!')
```

ASYNCHRONICZNY ITERATOR

```
[file: text.txt]

This is a test!
```

ASYNCHRONICZNY ITERATOR

```
// index.[m]js

import * as fs from 'fs'

async function logChunks(readable) {
    for await (const chunk of readable) {
        console.log(chunk)
    }
}

const readable = fs.createReadStream('./text.txt',  {encoding: 'utf8'})

logChunks(readable)

// Output:
// 'This is a test!\n'
```

TWORZENIE STRINGA ZE STRUMIENIA READABLE

```
// index.[m]js

import { Readable } from 'stream'
import assert from "assert"

async function readableToString(readable) {
    let result = ''
    for await (const chunk of readable) {
        result += chunk
    }
    return result
}

const readable = Readable.from('Good morning!', {encoding: 'utf8'})

assert.equal(await readableToString(readable), 'Good morning!')
```

TWORZENIE READABLE Z ITERATORÓW

```
// index.[m]js

import { Readable } from 'stream'

async function * generate() {  // <-- async generator function
    yield 'hello'
    yield 'streams'
}

const readable = Readable.from(generate())

readable.on('data', (chunk) => {
    console.log(chunk)
})
```

TRYBY ODCZYTU

TRYB PRZEPŁYWU (FLOWING MODE)

- W trybie przepływu, aby odczytać dane ze strumienia, można nasłuchiwać zdarzeń danych i dołączyć wywołanie zwrotne.
- Gdy fragment danych jest dostępny, odczytywany strumień emituje zdarzenie danych, a wywołanie zwrotne (callback) jest wykonywane.

TRYB PRZEPŁYWU (FLOWING MODE)

```
// index.[m]js

import * as fs from 'fs'

let data = ''

const readerStream = fs.createReadStream('./file.txt') //Create a readable
stream

readerStream.setEncoding('utf-8') // Set the encoding to be utf8.

readerStream.on('data', (chunk) => {
    data += chunk
})
```

TRYB PRZEPŁYWU (FLOWING MODE)

```
readerStream.on('end', () => {
    console.log(data)
})

readerStream.on('error', (err) => {
    console.error(err.stack)
})

console.log("Program Ended")
```

TRYB WSTRZYMANIA (PAUSED MODE)

- W trybie wstrzymania wystarczy wielokrotnie wywoływać metodę *read()* na instancji strumienia, aż każdy fragment danych zostanie odczytany.

TRYB WSTRZYMANIA (PAUSED MODE)

```
// index.[m]js

import * as fs from 'fs'

const readableStream = fs.createReadStream('./file.txt')

let data = ''

let chunk
```

TRYB WSTRZYMANIA (PAUSED MODE)

```
readableStream.on('readable', () => {
    while ((chunk=readableStream.read()) != null) {
        data += chunk
    }
})

readableStream.on('end', () => {
    console.log(data)
})
```

TWORZENIE STRUMIENIA WRITABLE

TWORZENIE STRUMIENIA WRITABLE

```
// index.[m]js

import * as fs from 'fs'

const readableStream = fs.createReadStream('file1.txt')
const writableStream = fs.createWriteStream('file2.txt')

readableStream.setEncoding('utf8')

readableStream.on('data', (chunk) => {
    writableStream.write(chunk)
})
```

UŻYCIE WRITABLE.END()

```
// index.[m]js

// Write 'hello, ' and then end with 'world!'.
import * as fs from 'fs'

const file = fs.createWriteStream('example.txt')

file.write('hello, ')

file.end('world!')
// Writing more now is not allowed!
```

PRZEKAZYWANIE DANYCH ZE STRUMIENIA READABLE DO WRITABLE

```
// index.[m]js

import * as Stream from 'stream'

const readableStream = new Stream.Readable()
const writableStream = new Stream.Writable()

writableStream._write = (chunk, encoding, next) => {
    console.log(chunk.toString())
    next()
}

readableStream.pipe(writableStream)

readableStream.push('ping!')
readableStream.push('pong!')
writableStream.end()
```

UŻYCIE ASYNCHRONICZNEGO ITERATORA (I)

```
// index.[m]js

import * as util from 'util';
import * as stream from 'stream';
import * as fs from 'fs';
import {once} from 'events';
import assert from "assert";
const finished = util.promisify(stream.finished) // (A)
```

UŻYCIE ASYNCHRONICZNEGO ITERATORA (2)

```
async function writeIterableToFile(iterable, filePath) {
    const writable = fs.createWriteStream(filePath, {encoding: 'utf8'})
    for await (const chunk of iterable) {
        if (!writable.write(chunk)) { // (B)
            // Handle backpressure
            await once(writable, 'drain')
        }
    }
    writable.end() // (C)
    // Wait until done. Throws if there are errors.
    await finished(writable)
}
```

UŻYCIE ASYNCHRONICZNEGO ITERATORA (3)

```
await writeIterableToFile(['One', ' line of text.\n'], 'tmp/log.txt')
assert.equal(
    fs.readFileSync('tmp/log.txt', {encoding: 'utf8'}),
    'One line of text.\n'
)
```

UŻYCIE ASYNCHRONICZNEGO ITERATORA - WYJAŚNIENIE

- Domyślna wersja funkcji ***stream.finished()*** jest oparta na wywołaniu zwrotnym (***callback***), ale można ją przekształcić w wersję opartą na obietnicy za pomocą funkcji ***util.promisify()*** (linia A)
- Zapis do strumienia writable podczas obsługi ciśnienia wstecznego [***backpressure***] (linia B)
- Zamknięcie strumienia writable i oczekiwanie na zakończenie zapisu (linia C)

PIPELINE()

- ***Piping*** to mechanizm, w którym dostarczamy dane wyjściowe jednego strumienia jako dane wejściowe do innego strumienia.
- Jest on zwykle używany do pobierania danych z jednego strumienia i przekazywania danych wyjściowych tego strumienia do innego strumienia.
- Operacje pipingu ***nie mają ograniczeń***. Innymi słowy, piping jest używany do przetwarzania danych strumieniowych w wielu krokach.

PIPELINE()

- W ***Node 10.x*** wprowadzono ***stream.pipeline()***.
- Jest to metoda modułu do potokowania między strumieniami, przekazująca błędy i prawidłowo czyszcząca oraz zapewniająca wywołanie zwrotne po zakończeniu potokowania.

PIPELINE() - PRZYKŁAD (1)

```
import { pipeline } from 'stream'
import * as fs from 'fs'
import * as zlib from 'zlib'

// Użyj interfejsu API potoku, aby łatwo połączyć ze sobą serię strumieni
// i otrzymać powiadomienie, gdy potok zostanie w pełni ukończony.
// Potok do wydajnego gzipowania potencjalnie dużego pliku wideo:
```

PIPELINE() - PRZYKŁAD (2)

```
pipeline(
    fs.createReadStream('The.Matrix.1080p.mkv'),
    zlib.createGzip(),
    fs.createWriteStream('The.Matrix.1080p.mkv.gz'),
    (err) => {
        if (err) {
            console.error('Pipeline failed', err)
        } else {
            console.log('Pipeline succeeded')
        }
    }
)
```

INTERFEJSY API NODE WYKORZYSTUJĄCE STRUMIENIE (1)

- ***net.Socket*** to główny interfejs API Node, na którym oparte są strumienie i który stanowi podstawę większości następujących interfejsów API
- ***process.stdin*** zwraca strumień podłączony do stdin
- ***process.stdout*** zwraca strumień podłączony do stdout

INTERFEJSY API NODE WYKORZYSTUJĄCE STRUMIENIE (2)

- ***process.stderr*** zwraca strumień podłączony do stderr
- ***fs.createReadStream()*** tworzy strumień do odczytu pliku
- ***fs.createWriteStream()*** tworzy strumień zapisujący do pliku

INTERFEJSY API NODE WYKORZYSTUJĄCE STRUMIENIE (3)

- ***net.connect()*** inicjuje połączenie strumieniowe
- ***http.request()*** zwraca instancję klasy http.ClientRequest, która jest strumieniem zapisującym
- ***zlib.createGzip()*** kompresuje dane przy użyciu gzip (algorytmu kompresji) do strumienia

INTERFEJSY API NODE WYKORZYSTUJĄCE STRUMIENIE (3)

- ***zlib.createGunzip()*** dekompresuje strumień gzip
- ***zlib.createDeflate()*** kompresuje dane przy użyciu deflate (algorytmu kompresji) do strumienia
- ***zlib.createInflate()*** dekompresuje strumień deflate

WBUDOWANE MODUŁY NODE.JS

ASSERT

- Moduł assert zapewnia sposób testowania wyrażeń. Jeśli wyrażenie ma wartość 0 lub false, następuje błąd asercji i program zostaje zakończony.
- Moduł ten został stworzony do użytku wewnętrznego przez Node.js.

ASSERT - DOSTĘPNE METODY

Metoda	Opis działania	Składnia
assert()	Sprawdza, czy wartość jest prawdziwa. Tak samo jak assert.ok()	assert(expression, [message])
deepEqual()	Sprawdza, czy dwie wartości są równe	assert.deepEqual(value1, value2, [message])
deepStrictEqual()	Sprawdza, czy dwie wartości są równe, używając ścisłego operatora równości (===).	assert.deepStrictEqual(value1, value2, [message])
doesNotThtow()	assert.doesNotThrow potwierdzi, że funkcja nie jest w stanie rzucić błędu	assert.doesNotThrow(fn, [error], [message])

ASSERT - DOSTĘPNE METODY

Metoda	Opis działania	Składnia
equal()	Sprawdza, czy dwie wartości są równe, używając operatora równości (==)	assert.equal(value1, value2, [message])
deepEqual()	Sprawdza, czy dwie wartości są równe	assert.deepEqual(value1, value2, [message])
fail()	Rzuca błąd AssertionError z podanym komunikatem błędu lub domyślnym komunikatem błędu	assert.fail([message])
ifError()	Rzuca określony błąd, jeśli określony błąd ma wartość true	assert.ifError(value)

ASSERT - DOSTĘPNE METODY

Metoda	Opis działania	Składnia
notDeepEqual()	Sprawdza, czy dwie wartości nie są równe	assert.notDeepEqual(value1, value2, [message])
notDeepStrictEqual()	Sprawdza, czy dwie wartości nie są równe, używając operatora strict not equal (!==)	assert.notDeepStrictEqual(value1, value2, [message])
notEqual()	Sprawdza, czy dwie wartości nie są równe, używając operatora nierówności (!=)	assert.notEqual(value1, value2, [message])
notStrictEqual()	Sprawdza, czy dwie wartości nie są równe, używając operatora strict not equal (!==)	assert.notStrictEqual(value1, value2, [message])

ASSERT - DOSTĘPNE METODY

Metoda	Opis działania	Składnia
ok()	Sprawdza, czy wartość jest prawdziwa	assert.ok(value, [message])
strictEqual()	Sprawdza, czy dwie wartości są równe, używając ścisłego operatora równości (===)	assert.strictEqual(value1, value2, [message])
throws()	Oczekuje, że funkcja fn zgłosi błąd	assert.throws(fn, [error], [message])

BUFFER

- Moduł buffer zapewnia sposób obsługi strumieni danych binarnych.
- Obiekt Buffer jest obiektem globalnym w Node.js i nie jest konieczne importowanie go za pomocą słowa kluczowego require/import.

CRYPTO

- Moduł kryptograficzny zapewnia sposób obsługi zaszyfrowanych danych.

FS

- Moduł File System umożliwia pracę z systemem plików komputera.

HTTP

- Moduł HTTP zapewnia sposób przesyłania danych przez Node.js za pośrednictwem protokołu HTTP (Hyper Text Transfer Protocol).

HTTPS

- Moduł HTTPS zapewnia sposób przesyłania danych przez Node.js za pośrednictwem protokołu HTTP TLS/SSL, który jest bezpiecznym protokołem HTTP.

NET

- Moduł Net umożliwia tworzenie serwerów TCP i klientów TCP.

OS

- Moduł OS dostarcza informacji o systemie operacyjnym komputera.

OS - WYBRANE METODY I WŁAŚCIWOŚCI

Metoda / Właściwość	Opis działania
arch()	Zwraca architekturę procesora systemu operacyjnego
constants	Zwraca obiekt zawierający stałe systemu operacyjnego dla sygnałów procesu, kodów błędów itp.
cpus()	Zwraca tablicę zawierającą informacje o procesorach komputera.
endianness()	Zwraca kolejność procesora (BE, LE) *
EOL	Zwraca znacznik końca linii dla bieżącego systemu operacyjnego.

OS - WYBRANE METODY I WŁAŚCIWOŚCI

Metoda / Właściwość	Opis działania
freemem()	Zwraca ilość wolnej pamięci w systemie
hostname()	Zwraca nazwę hosta systemu operacyjnego
loadavg()	Zwraca tablicę zawierającą średnie wartości obciążenia (1, 5 i 15 minut)
networkInterfaces()	Zwraca interfejsy sieciowe, które mają adres sieciowy
platform()	Zwraca informacje o platformie systemu operacyjnego.

OS - WYBRANE METODY I WŁAŚCIWOŚCI

Metoda / Właściwość	Opis działania
release()	Zwraca informacje o wersji systemu operacyjnego
tmpdir()	Zwraca domyślny katalog systemu operacyjnego dla plików tymczasowych
totalmem()	Zwraca całkowitą liczbę pamięci systemu
type()	Zwraca nazwę systemu operacyjnego
uptime()	Zwraca czas działania systemu operacyjnego w sekundach
userInfo()	Zwraca informacje o bieżącym użytkowniku

PATH

- Moduł Path zapewnia sposób pracy z katalogami i ścieżkami plików.