Webtechnológia és webalkalmazás-fejlesztés - Alapok

Alapok

Magda Donát

Széchenyi István Egyetem, Győr

https://git.mdnd-it.cc/Donat/GKNB_MSTM071

2026. január 28.

SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR

INFORMATIKA TANSZÉK
DEPARTMENT OF COMPUTER SCIENCE

Oktató

Elérhetőségek

Magda Donát, mérnökinformatikus BSc

- **E-mail**: magda.donat@ga.sze.hu
- **Konzultáció**: előre egyeztetve, Google Meet

Tantárgy célja

- Átfogó kép a modern webalkalmazás-fejlesztés folyamatáról
- Kliens-oldali (frontend) és szerver-oldali (backend) fejlesztés
- Adatbázis-kezelés és API-kommunikáció
- Gyakorlati tapasztalat a teljes fejlesztési folyamatban
- Különös hangsúly a JavaScript-alapú technológiákra

Cél

A tervezéstől a működő alkalmazásig: teljes fejlesztési folyamat elsajátítása

Backend technológiák

Szerveroldali fejlesztés

- Node.js – szerveroldali futtatókörnyezet
- Express.js – webes keretrendszer
- Prisma – ORM adatmodellezésre

Eszközök és adatbázis

- SQLite – relációs adatbázis
- Postman – API tesztelés
- Redis – gyorsítótár

REST API-k → Útvonalkezelés → Adatbázis-kapcsolat → Biztonságos adatkezelés

Tematika – 1-3. hét

Alapozás, HTTP, REST API, Express és architektúra

1. hét: Alapok, Node.js, HTTPS, NPM package manager
1. hét: REST API, Architekturális felépítések, CQRS
1. hét: Express.js, Routing, Postman
2. hét: ORM, Database connection, CRUD operations
2. hét: Prisma, Aggregates, Repositories
3. hét: Database CRUD, ORM, CQRS + DTOs, Mappers

Tematika – 3-6. hét

Authentikáció, Authorizáció, Külső szolgáltatások

- 3. hét: AUTH, OAUTH, OAUTH2
- 4. hét: Session, JWT, Cookie
- 4. hét: Hash, Middleware, Services
- 5. hét: Email sending, Templating
- 5. hét: Dependency Injection, Logging
- 6. hét: CORS, Redis, External API
- 6. hét: 1 ZH – Teljes backend írás

Értékelési mód

1. Zárthelyi dolgozat (50 pont)

Backend fejlesztés

- Backend rész: 50 pont

Alapvető komponensek, API-k és adatkezelés

2. Zárthelyi dolgozat (50 pont)

Összetett webalkalmazás

- Teljes frontend–backend integráció
- Önálló, működő alkalmazás
- Egyéni munka

Pontszámítás és jegyek

Összesített pontszám	Érdemjegy
0–50	1 (elégtelen)
51–60	2 (elégséges)
61–70	3 (közepes)
71–80	4 (jó)
81–100	5 (jeles)

Megajánlott jegy feltétele

Legalább 71 pont a két zárthelyi dolgozat összesített eredményéből

Következő lépések

1. Fejlesztői környezet telepítése (Node.js, npm)
2. Git és verziókezelés alapjai
3. Első Express.js alkalmazás elkészítése
4. REST API alapok megismerése

Mi a HTTP?

- HyperText Transfer Protocol - szövegalapú protokoll
- Kliens-szerver kommunikáció alapja a weben
- Állapotmentes (stateless) protokoll
- Request-Response modell
- Port: 80 (HTTP), 443 (HTTPS)

HTTP verziók

- HTTP/1.0 (1996) - Egy kérés per kapcsolat
- HTTP/1.1 (1997) - Perzisztens kapcsolat, pipeline
- HTTP/2 (2015) - Multiplexing, bináris protokoll
- HTTP/3 (2022) - QUIC protokoll, UDP alapú

HTTP vs HTTPS

HTTP

- Titkosítatlan kommunikáció
- Lehallgatható
- Módosítható (Man-in-the-Middle)
- Gyorsabb (nincs titkosítás)
- Port 80

HTTPS = HTTP + SSL/TLS

- Titkosított kommunikáció
- SSL/TLS tanúsítvány szükséges
- Védett adatátvitel
- Autentikáció és integritás
- Port 443

Fontos

Modern webalkalmazásoknál HTTPS kötelező! (SEO, biztonság, böngésző figyelmeztetések)

HTTP Request felépítése

Request struktúra

```
GET /api/users/123 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Accept: application/json
Authorization: Bearer token123

[Request Body - opcionális]
```

- Request Line: Metódus, útvonal, protokoll verzió
- Headers: Metaadatok kulcs-érték párok
- Body: Opcionális adatok (POST, PUT, PATCH)

HTTP Response felépítése

Response struktúra

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 58
Date: Mon, 27 Jan 2026 10:00:00 GMT
Server: nginx/1.20.1

{"id": 123, "name": "Alice", "email": "alice@example.com"}
```

- Status Line: Protokoll, státuszkód, státusz szöveg
- Headers: Metaadatok
- Body: Válasz tartalma (HTML, JSON, XML, stb.)

HTTP Metódusok

GET	Erőforrás lekérése (csak olvasás, idempotens)
POST	Új erőforrás létrehozása (nem idempotens)
PUT	Erőforrás teljes frissítése (idempotens)
PATCH	Erőforrás részleges frissítése (nem idempotens)
DELETE	Erőforrás törlése (idempotens)
HEAD	Mint GET, de csak headerek (nincs body)
OPTIONS	Támogatott metódusok lekérése (CORS)
TRACE	Echo teszt (diagnosztika)
CONNECT	Tunnel létrehozása (proxy)

Idempotens

Többszöri végrehajtás ugyanazt az eredményt adja (GET, PUT, DELETE)

HTTP Státuszkódok - 1xx Informational

100 Continue A kliens folytathatja a kérést

101 Switching Protocols Protokoll váltás (pl. WebSocket)

102 Processing A szerver dolgozik a kérésen (WebDAV)

103 Early Hints Korai válasz header információkkal

Használat

Ritkán használt kategória, főleg speciális protokoll műveleteknél

HTTP Státuszkódok - 2xx Success

200 OK Sikeres kérés (GET, PUT, PATCH)

201 Created Erőforrás sikeresen létrehozva (POST)

202 Accepted Kérés elfogadva, de még feldolgozás alatt

203 Non-Authoritative Proxy/cache módosította a választ

204 No Content Sikeres, de nincs visszaadott tartalom (DELETE)

205 Reset Content Sikeres, űrlap visszaállítása szükséges

206 Partial Content Részleges tartalom (range request)

REST API

200 (GET, PUT, PATCH), **201** (POST), **204** (DELETE)

HTTP Státuszkódok - 3xx Redirection

300 Multiple Choices Több lehetséges választás

301 Moved Permanently Végleges átirányítás (SEO átad)

302 Found Ideiglenes átirányítás (POST → GET)

303 See Other Átirányítás GET-re (POST után)

304 Not Modified Cache friss, nincs módosítás

307 Temporary Redirect Ideiglenes, metódus megmarad

308 Permanent Redirect Végleges, metódus megmarad

Cache és átirányítás

301/308 végleges, 302/307 ideiglenes. 307/308 megőrzi az eredeti metódust!

HTTP Státuszkódok - 4xx Client Error (1/2)

400 Bad Request Hibás kérés szintaxis (validáció)

401 Unauthorized Autentikáció szükséges (login hiányzik)

402 Payment Required Fizetés szükséges (nem használt)

403 Forbidden Hozzáférés megtagadva (nincs jogosultság)

404 Not Found Erőforrás nem található

405 Method Not Allowed Metódus nem engedélyezett

406 Not Acceptable Nem támogatott tartalom típus

407 Proxy Auth Required Proxy autentikáció szükséges

408 Request Timeout Kérés időtúllépés

HTTP Státuszkódok - 4xx Client Error (2/2)

409 Conflict Konfliktus (pl. már létező erőforrás)

410 Gone Erőforrás véglegesen eltávolítva

411 Length Required Content-Length header hiányzik

412 Precondition Failed Feltétel nem teljesült

413 Payload Too Large Request body túl nagy

414 URI Too Long URL túl hosszú

415 Unsupported Media Type Nem támogatott media type

416 Range Not Satisfiable Érvénytelen range

418 I'm a teapot Vicces kód (április 1.)

422 Unprocessable Entity Validációs hiba (WebDAV)

429 Too Many Requests Rate limit túllépés

HTTP Státuszkódok - 5xx Server Error

500 Internal Server Error Általános szerver hiba

501 Not Implemented Metódus nem implementált

502 Bad Gateway Gateway/proxy hiba

503 Service Unavailable Szerver túlterhelt/karbantartás

504 Gateway Timeout Gateway időtúllépés

505 HTTP Version Not Supported HTTP verzió nem támogatott

506 Variant Also Negotiates Konfigurációs hiba

507 Insufficient Storage Nincs elég tárhely

508 Loop Detected Végtelen ciklus (WebDAV)

510 Not Extended További kiterjesztés szükséges

511 Network Auth Required Hálózati autentikáció

REST API-ban gyakori státuszkódok

Metódus	Sikeres	Hiba
GET	200 OK	404 Not Found
POST	201 Created	400 Bad Request, 409 Conflict
PUT	200 OK, 204 No Content	400 Bad Request, 404 Not Found
PATCH	200 OK	400 Bad Request, 404 Not Found
DELETE	204 No Content	404 Not Found
Autentikáció: 401 Unauthorized, 403 Forbidden		
Szerver hiba: 500 Internal Server Error, 503 Service Unavailable		
Validáció: 400 Bad Request, 422 Unprocessable Entity		

Fontos HTTP Headers (Request)

Gyakori Request Headers

```
Host: api.example.com                # Celszerver
User-Agent: Mozilla/5.0               # Kliens informacio
Accept: application/json              # Elfogadott tartalom tipus
Content-Type: application/json        # Kuldes tartalom tipus
Authorization: Bearer <token>         # Autentikacio (JWT)
Cookie: sessionId=abc123              # Session cookie
Accept-Language: hu-HU,en             # Nyelvi preferencia
Accept-Encoding: gzip, deflate        # Tomoritesi modok
Cache-Control: no-cache               # Cache iranyitas
If-None-Match: "etag123"              # Felteteles keres
Origin: https://example.com           # CORS eredethelye
```

Fontos HTTP Headers (Response)

Gyakori Response Headers

```
Content-Type: application/json      # Valasz tartalom tipus
Content-Length: 1234                # Tartalom merete
Date: Mon, 27 Jan 2026 10:00:00     # Valasz idopontja
Server: nginx/1.20.1                # Szerver informacio
Set-Cookie: sessionId=abc; HttpOnly # Cookie beallitas
Cache-Control: max-age=3600         # Cache idotartam
ETag: "etag123"                     # Verzio azonosito
Expires: Wed, 29 Jan 2026 10:00:00 # Lejarat idopont
Location: /api/users/123            # Atiranyitas cel (3xx, 201)
Access-Control-Allow-Origin: *      # CORS engedelyezes
X-RateLimit-Remaining: 99           # Rate limit info
```

HTTPS - SSL/TLS Handshake

1. Client Hello - Támogatott titkosítási módok
2. Server Hello - Kiválasztott titkosítás + tanúsítvány
3. Certificate Verify - Tanúsítvány ellenőrzése (CA)
4. Key Exchange - Szimmetrikus kulcs létrehozása
5. Finished - Titkosított kommunikáció kezdete

SSL/TLS verziók

- SSL 2.0/3.0 - Elavult, sebezhetőségek
- TLS 1.0/1.1 - Elavult (2020-tól)
- TLS 1.2 - Még használt, de TLS 1.3 ajánlott
- TLS 1.3 - Legújabb, gyorsabb, biztonságosabb

Összefoglalás

HTTP/HTTPS alapok

- HTTP: Request-Response, állapotmentes protokoll
- HTTPS: Titkosított HTTP (SSL/TLS)
- HTTP metódusok: GET, POST, PUT, PATCH, DELETE
- Státuszkódok: 1xx, 2xx, 3xx, 4xx, 5xx
- Headers: Metaadatok kérésben és válaszban
- REST API: 200, 201, 204, 400, 401, 403, 404, 500

Mi a JavaScript?

- Dinamikus, interpretált programozási nyelv
- Eredetileg böngészőkben futó szkriptnyelv (1995, Brendan Eich)
- Ma már szerveroldali fejlesztésre is használható (Node.js)
- Gyengén típusos, prototype-alapú objektum-orientált nyelv
- Az ECMAScript szabvány implementációja

Miért fontos?

A JavaScript az egyetlen nyelv, amely natívan fut a böngészőben, és a Node.js révén a backend fejlesztés alapja is lehet.

Változók és adattípusok

Változó deklaráció

```
// Konstans (nem valtoztathato)
const PI = 3.14159;

// Blokk scope valtozo
let count = 0;
count = 1; // OK

// Fuggvenyi scope (kerulendo!)
var oldStyle = "legacy";
```

Primitív típusok

```
let num = 42;            // Number
let str = "Hello";       // String
let bool = true;         // Boolean
let empty = null;        // Null
let undef = undefined;   // Undefined
let big = 123n;          // BigInt
let sym = Symbol("id"); // Symbol
```

Tömbök és objektumok

Tömbök (Arrays)

```
const fruits = ['apple', 'banana'];

// Elem hozzaadasa
fruits.push('orange');

// Bejaras
fruits.forEach(fruit => {
  console.log(fruit);
});

// Szures
const long = fruits.filter(
  f => f.length > 5
);
```

Objektumok (Objects)

```
const user = {
  name: 'Alice',
  age: 25,
  greet() {
    return `Hi, I'm ${this.name}`;
  }
};

// Eleres
console.log(user.name);
console.log(user['age']);

// Uj tulajdonsag
user.email = 'alice@example.com';
```

Függvények

Hagyományos függvénydeklaráció

```
function add(a, b) {
  return a + b;
}
```

Arrow function (ES6+)

```
const add = (a, b) => a + b;

const square = x => x * x;  // Egy parameter, kapcsos zarojel nelkul

const greet = name => {
  const message = `Hello, ${name}!`;
  return message;
};
```

Destructuring és Spread operátor

Destructuring

```
// Tomb destructuring
const [first, second] = [1, 2, 3];

// Objektum destructuring
const {name, age} = user;

// Fuggveny parameterenel
function print({name, age}) {
  console.log(`${name}: ${age}`);
}
```

Spread operator (...)

```
// Tomb osszefuzes
const arr = [...arr1, ...arr2];

// Objektum masolasa
const copy = {...original};

// Rest parameter
function sum(...numbers) {
  return numbers.reduce(
    (a, b) => a + b, 0
  );
}
```

Aszinkron programozás - Callback

Callback függvények

```
function fetchData(callback) {
  setTimeout(() => {
    const data = {id: 1, name: 'User'};
    callback(data);
  }, 1000);
}

fetchData((data) => {
  console.log(data);
});
```

Callback Hell problémája

Többszintű beágyazott callback-ek olvashatatlan kódot eredményeznek.

Aszinkron programozás - Promise

Promise használata

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const data = {id: 1, name: 'User'};
      resolve(data);
      // reject(new Error('Failed'));
    }, 1000);
  });
}

fetchData()
  .then(data => console.log(data))
  .catch(error => console.error(error))
  .finally(() => console.log('Done'));
```

Aszinkron programozás - Async/Await

Modern async/await szintaxis

```
async function loadUser() {
  try {
    const user = await fetchData();
    console.log(user);

    const posts = await fetchUserPosts(user.id);
    console.log(posts);

    return {user, posts};
  } catch (error) {
    console.error('Error:', error);
  }
}

// Hasznalat
loadUser();
```

ES6+ Modern jellemzők

- Template literals: `‘Hello ${name}‘`
- Default parameters: `function greet(name = ’Guest’)`
- Class syntax:

Class példa

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    return ‘Hi, I’m ${this.name}‘;
  }
}

const alice = new Person(’Alice’, 25);
```

Összefoglalás

JavaScript alapok

- Változók: `const`, `let` (kerüljük a `var`-t)
- Adattípusok: primitívek és objektumok
- Tömbök és objektumok manipulációja
- Függvények: hagyományos és arrow function
- Destructuring és spread operátor
- Aszinkron programozás: callback, Promise, async/await
- Modern ES6+ jellemzők

Mi a Node.js?

- JavaScript futtatókörnyezet a V8 motoron alapulva
- Szerveroldali JavaScript fejlesztés lehetősége
- Eseményvezérelt, nem-blokkoló I/O modell
- Kiváló nagy számú egyidejű kapcsolat kezelésére
- Létrehozta: Ryan Dahl (2009)

Miért Node.js?

- Ugyanaz a nyelv frontend és backend oldalon
- Hatalmas ökoszisztéma (npm)
- Gyors fejlesztés és prototípuskészítés
- Skálázható, nagy teljesítményű alkalmazások

Node.js architektúra

V8 JavaScript Engine

- Google Chrome motorja
- JIT (Just-In-Time) fordítás
- C++ nyelven íródott
- Rendkívül gyors végrehajtás

libuv

- Event loop implementáció
- Aszinkron I/O műveletek
- Cross-platform támogatás

Event Loop

1. Timers (setTimeout, setInterval)
2. Pending callbacks
3. Poll (I/O műveletek)
4. Check (setImmediate)
5. Close callbacks

Single-threaded, de aszinkron!

Első Node.js program

hello.js

```
console.log('Hello, Node.js!');
console.log('Node verzio:', process.version);
console.log('Platform:', process.platform);
console.log('Aktualis konyvtar:', __dirname);
console.log('Fajl neve:', __filename);
```

Futtatás

```
node hello.js
```

Fontos

A Node.js nem böngésző! Nincs `window`, nincs `document`, de van `global` és `process`.

Modulrendszer - CommonJS

math.js - Modul exportálás

```
function add(a, b) {
  return a + b;
}

function multiply(a, b) {
  return a * b;
}

module.exports = {
  add,
  multiply
};
```

app.js - Modul importálás

```
const math = require('./math');

console.log(math.add(2, 3));
// 5

console.log(math.multiply(4, 5));
// 20
```

Beépített modulok

```
const fs = require('fs');      // File system
const path = require('path');  // Path mukodtatas
const http = require('http');  // HTTP szerver
```

ES Modules (ESM)

math.mjs - Export

```
export function add(a, b) {
  return a + b;
}

export function multiply(a, b) {
  return a * b;
}

// Default export
export default {
  add,
  multiply
};
```

app.mjs - Import

```
// Named import
import { add, multiply } from './math.mjs';

console.log(add(2, 3));

// Default import
import math from './math.mjs';

console.log(math.multiply(4, 5));
```

package.json

ESM használatához: {"type": "module"} a package.json-ben

File System műveletek

Fájl olvasás - Szinkron

```
const fs = require('fs');

const data = fs.readFileSync('input.txt', 'utf8');
console.log(data);
// Blokkolja a vegrehajtast!
```

Fájl olvasás - Aszinkron (Callback)

```
fs.readFile('input.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Hiba:', err);
    return;
  }
  console.log(data);
});
// Nem blokkolja a vegrehajtast!
```

File System - Promises API

Fájl olvasás - Promise

```
const fs = require('fs').promises;

async function readData() {
  try {
    const data = await fs.readFile('input.txt', 'utf8');
    console.log(data);
  } catch (error) {
    console.error('Hiba:', error);
  }
}

readData();
```

Fájl írás

```
await fs.writeFile('output.txt', 'Hello Node.js!', 'utf8');
```

HTTP szerver létrehozása

Egyszerű HTTP szerver

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

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain; charset=utf-8');
  res.end('Hello, Node.js szerver!\n');
});

const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Szerver fut: http://localhost:${PORT}`);
});
```

Böngészőben: `http://localhost:3000`

Process objektum

Process információk és műveletek

```
// Kornyezeti valtozok
console.log(process.env.NODE_ENV);

// Parancssori argumentumok
console.log(process.argv);

// Kilepes
process.exit(0);  // Sikeres kilepes
process.exit(1);  // Hiba

// Esemenyek
process.on('exit', (code) => {
  console.log(`Kilepes kod: ${code}`);
});
```

Path modul

Útvonal műveletek

```
const path = require('path');

// Utvonalak osszefuzese
const filePath = path.join(__dirname, 'data', 'users.json');

// Kiterjesztes lekerese
const ext = path.extname('file.txt');  // .txt

// Fajlnev bazisresz
const base = path.basename('/user/local/file.txt');  // file.txt

// Konyvtar resz
const dir = path.dirname('/user/local/file.txt');  // /user/local

// Abszolut utvonal
const abs = path.resolve('data', 'users.json');
```

Events - Eseménykezelés

EventEmitter használata

```
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();

// Esemeny figyelese
myEmitter.on('event', (data) => {
  console.log('Esemeny tortent:', data);
});

// Esemeny kivaltas
myEmitter.emit('event', {message: 'Hello Events!'});
```

Node.js eseményvezérelt architektúrája alapja!

Node.js vs. Böngésző

Jellemző	Node.js	Böngésző
JavaScript engine	V8, SpiderMonkey	V8, SpiderMonkey, stb.
DOM API	Nincs	Van
window objektum	Nincs	Van
global objektum	global	window
File system	Van (fs)	Nincs (limitált)
Modulrendszer	CommonJS, ESM	ESM
HTTP szerver	Készíthető	Nem
Használat	Backend, CLI	Frontend

Összefoglalás

Node.js alapok

- V8 engine alapú JavaScript futtatókörnyezet
- Event-driven, non-blocking I/O modell
- Modulrendszer: CommonJS és ES Modules
- Beépített modulok: fs, path, http, events
- Process objektum: környezeti változók, argumentumok
- File system műveletek: szinkron, aszinkron, Promise
- HTTP szerver készítése
- EventEmitter: saját események létrehozása

Mi az npm?

- Node Package Manager - Node.js csomagkezelő
- A világ legnagyobb szoftver registry-je (2+ millió csomag)
- Node.js-szel együtt települ
- Csomagok telepítése, kezelése, megosztása
- Projekt függőségek kezelése

Verziószám ellenőrzése

```
node -version       npm -version
```

Alternatívák

yarn, pnpm - gyorsabb, hatékonyabb csomagkezelők

npm init - Projekt inicializálás

Interaktív projekt létrehozás

```
npm init
```

Gyors inicializálás alapértelmezett értékekkel

```
npm init -y
# vagy
npm init --yes
```

Eredmény

Létrejön a package.json fájl, amely a projekt metaadatait tartalmazza

package.json struktúra

Alapvető package.json

```
{
  "name": "my-backend-app",
  "version": "1.0.0",
  "description": "Backend alkalmazas Node.js-sel",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "test": "jest"
  },
  "keywords": ["nodejs", "backend", "api"],
  "author": "Magda Donat",
  "license": "MIT",
  "dependencies": {
    "express": "^4.18.2"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

Csomagok telepítése

Telepítés production függőségként

```
npm install express
# vagy rovidebben
npm i express

# Tobbe csomag egyszerre
npm install express prisma dotenv
```

Telepítés development függőségként

```
npm install --save-dev nodemon
# vagy
npm i -D nodemon jest eslint
```

Mi a különbség?

dependencies: éles környezetben is kell | **devDependencies**: csak fejlesztéshez

Csomagok kezelése

Telepítés specifikus verzióval

```
npm i express@4.18.0

# Legfrissebb verzio
npm i express@latest
```

Globális telepítés

```
npm install -g nodemon
npm i -g typescript
```

Eltávolítás

```
npm uninstall express
npm un express

# Globalis eltavolitas
npm uninstall -g nodemon
```

Frissítés

```
npm update
npm update express
```

Semantic Versioning (SemVer)

Verzió formátum:

`1.2.3` → 1 (major) . 2 (minor) . 3 (patch)

- MAJOR: Nem kompatibilis API változások
- MINOR: Új funkciók, visszafelé kompatibilis
- PATCH: Hibajavítások, visszafelé kompatibilis

Jelölés	Jelentés
`1.2.3`	Pontosan 1.2.3 verzió
`^1.2.3`	$\geq$ 1.2.3 és $<$ 2.0.0 (MINOR frissítések)
`~1.2.3`	$\geq$ 1.2.3 és $<$ 1.3.0 (PATCH frissítések)
`*` vagy `latest`	Legfrissebb verzió

npm scripts

Scripts definiálása package.json-ben

```
{
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "test": "jest",
    "build": "tsc",
    "lint": "eslint .",
    "format": "prettier --write ."
  }
}
```

Scripts futtatása

```
npm run dev
npm run test
npm start     # 'run' nelkul is mukodik (start, test, stop)
```

node_modules és package-lock.json

node_modules

- Telepített csomagok mappája
- Nagyon nagy lehet (100+ MB)
- Ne commitoljuk git-be!
- .gitignore-ba: `node_modules/`
- Újragenerálás: `npm install`

package-lock.json

- Pontos verziókat rögzít
- Függőségek fája
- Determinisztikus telepítés
- Commitoljuk git-be!
- Reprodukálható build-ek

Fontos

`npm install` a package-lock.json alapján telepít (ha létezik)

Hasznos npm parancsok

Információk lekérése

```
# Telepitett csomagok listaja
npm list

# Elavult csomagok
npm outdated

# Csomag informacio
npm info express

# Csomag keresese
npm search prisma
```

Karbantartás

```
# Cache torlese
npm cache clean --force

# Fuggosegek auditja
npm audit

# Sebezhetosegek javitasa
npm audit fix

# Projekt tisztitas
rm -rf node_modules
npm install
```

npx - Csomag futtatás telepítés nélkül

npx használata

```
# Egyszeri futtatas telepites nelkul
npx create-react-app my-app
npx prisma init
npx tsx index.ts

# Lokalis csomag futtatas
npx nodemon index.js
```

Előnyök

- Nem szennyezi a globális telepítéseket
- Mindig a legfrissebb verzió
- Gyors prototípuskészítés
- CLI eszközök futtatása

Gyakori csomagok Backend fejlesztéshez

Production dependencies

```
npm i express              # Web framework
npm i @prisma/client       # ORM kliens
npm i dotenv               # Kornyezeti valtozok
npm i cors                 # CORS middleware
npm i bcrypt               # Jelszo hash
npm i jsonwebtoken         # JWT token
npm i express-validator    # Validacio
```

Development dependencies

```
npm i -D nodemon           # Auto-restart
npm i -D prisma            # ORM CLI
npm i -D typescript        # TypeScript
npm i -D @types/node       # Node.js tipusok
```

.npmrc - npm konfiguráció

Projekt .npmrc fájl

```
# Pontos verziot ment (^ nelkul)
save-exact=true

# Engine strict mod
engine-strict=true

# Automaikus audit
audit=true
```

package.json - Node verzió megkötés

```
{
  "engines": {
    "node": ">=18.0.0",
    "npm": ">=9.0.0"
  }
}
```

npm Registry és publikálás

- npmjs.com - Publikus npm registry
- Ingyenes, bárki publikálhat csomagot
- Saját csomag publikálás: `npm publish`
- Privát csomagok: fizetős npm Pro/Teams
- Alternatív registry-k: GitHub Packages, Verdaccio

Saját csomag publikálás lépései

1. `npm login` - Bejelentkezés
2. package.json konfiguráció
3. `npm publish` - Publikálás
4. `npm version patch/minor/major` - Verziókezelés

Összefoglalás

npm alapok

- npm: Node Package Manager, 2+ millió csomag
- `npm init` - projekt inicializálás
- `npm install/uninstall/update` - csomagkezelés
- package.json: projekt metaadatok, függőségek, scripts
- Semantic Versioning: MAJOR.MINOR.PATCH
- dependencies vs devDependencies
- node_modules (ne commit), package-lock.json (commit)
- npm scripts: egyéni parancsok definiálása
- npx: csomagok futtatása telepítés nélkül