1 Alapok elsajátítása: Node.js és JavaScript

1.1 JavaScript alapok.

1.2 Node.js alapok

2 Express telepítése és első alkalmazás

2.1 Express telepítése

Kezdd egy új Node.js projekt létrehozásával, majd telepítsd az Express-t
npm install express
paranccsal.

2.2 Első szerver létrehozása

Készíts egy alapvető Express szervert, ami egy egyszerű "Hello World" üzenetet ad vissza.

3 URL paraméterek és lekérdezési paraméterek kezelése

Cél: Megérteni, hogyan lehet URL paraméterekkel és lekérdezési paraméterekkel dolgozni.

Az Express-ben az adatok fogadásának főbb módjai a következők:

1. **Lekérdezési paraméterek** (req.query)

```
// GET /search?name=John&age=30

app.get('/search', (req, res) => {
    const name = req.query.name; // "John"
    const age = req.query.age;   // "30"
    res.send(`Name: ${name}, Age: ${age}`);
});
```

2. **Útvonal paraméterek** (req.params)

```
// GET /users/123

app.get('/users/:id', (req, res) => {
    const userId = req.params.id; // "123"
    res.send(`User ID: ${userId}`);
});
```

3. **Törzs (body) paraméterek** (req.body) – JSON és URL-kódolt adatok.

HTML

```
<form method="POST" action="/users">
  <input type="text" name="name">
  <input type="number" name="age">
  <button type="submit">Submit</button>
</form>
```

JS:

```
app.use(express.urlencoded({ extended: true })); // Middleware az URL-kódolt adatok kezeléséhez

app.post('/users', (req, res) => {
    const { name, age } = req.body;
    res.send(`Received user: ${name}, Age: ${age}`);
});
```

4. **Fájlok feltöltése** (multer middleware-rel)

HTML:

```
<form action="/upload" method="POST" enctype="multipart/form-data">
  <input type="file" name="myfile">
  <button type="submit">Upload</button>
</form>
```

JS:

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' }); // Mappába menti a fájlokat

app.post('/upload', upload.single('myfile'), (req, res) => {
    res.send(`File uploaded: ${req.file.originalname}`);
});
```

5. **Fejléc adatok** (req.headers)

```
app.get('/headers', (req, res) => {
    const userAgent = req.headers['user-agent'];
    res.send(`User Agent: ${userAgent}`);
});
```

6. **Cookie-k fogadása** (cookie-parser middleware-rel)

```
const cookieParser = require('cookie-parser');
app.use(cookieParser());

app.get('/cookies', (req, res) => {
   const myCookie = req.cookies.myCookieName;
   res.send(`Cookie value: ${myCookie}`);
});
```

Gyakorlati feladat:

- Hozz létre egy útvonalat, amely egy URL paraméter alapján fogad be egy adatot, például: /users/:id, és visszaadja az adott felhasználót.
- Készíts egy olyan útvonalat, amely lekérdezési paramétereket (*query parameters*) kezel, például: */search?name=John*.

4 Statikus fájlok kiszolgálása

Hogyan szolgálhatók ki statikus fájlok (HTML, CSS, képek stb.) Express alkalmazásból.

```
const express = require('express'); //-- synchronus csatlakozás
const app = express();
const port = 3000;
const cors = require('cors');
app.use(cors({origin: 'http://localhost:3000'}));
const fs = require('fs'); //-- lehetővé teszi a képek betöltése
const path = require('path'); //- lehetővé teszi a könyvtár betöltése
app.use(express.static(path.join(__dirname, 'public'))); //-- statikus fájlok betöltése

app.get('/', (req, res) => {
   res.header('Content-Type', 'text/html; charset=utf-8');
   res.status(201).sendFile(__dirname + '/public/index.html');
});


app.get('/login', (req, res) => {
   res.header('Content-Type', 'text/html; charset=utf-8');
   res.status(201).sendFile(__dirname + '/public/login.html');
});
```

Gyakorlat:

- Készíts egy alap weboldalt, amelynek a HTML fájljait, CSS stílusait és képeit az Express a /public mappából szolgáltatja ki.
- Használd az express.static() middleware-t a statikus fájlok kiszolgálására.

-

5 Routing (útvonalkezelés)

Routing: Tanuld meg, hogyan kezelheted a különböző útvonalakat (app.get(), app.post(), stb.) az Express-ben.

- **Dinamikus útvonalak**: Ismerd meg a dinamikus útvonalak használatát, például hogyan lehet paramétereket átadni az URL-ben (/users/:id).

Gyakorlati feladat:

- Készíts egy szervert, amely különböző útvonalakon különböző adatokat szolgáltat.
- Készíts egy egyszerű "felhasználó" (user) API-t, amelyben a felhasználók adatait egy tömbben tárolod (kezdetben az adatok lehetnek statikusak).
    - GET /users – adjon vissza egy listát az összes felhasználóról.
    - POST /users – fogadjon el egy új felhasználót JSON formátumban és adja hozzá a tömbhöz.
    - GET /users/:id – adja vissza az adott felhasználót a megadott id alapján.
    - PUT /users/:id – frissítse az adott felhasználót az id alapján.
    - DELETE /users/:id – törölje a felhasználót az id alapján.
- Implementálj egyszerű validációkat, például azt, hogy
    - a felhasználó neve kötelező legyen POST esetén.
    - születési dátuma két dátum közötti legyen (18 és 140 év közötti)
    - fizetése a havi minimálbérnél nagyobb legyen (teljes munkaidőben foglalkoztatott munkavállalónak 266 800 Ft, szakképzettséget igénylő munkakörben 326 000 Ft.)
    -

6 REST API készítése

Express remekül alkalmas RESTful API-k készítésére. Tanuld meg, hogyan kezelheted a különböző HTTP metódusokat (GET, POST, PUT, DELETE), és hogyan strukturálhatod API-jaidat.

Gyakorlat:

- **Adatkezelés és JSON válaszok**: Készíts egy REST API-t, amely JSON adatokat szolgáltat.
- Hozz létre egy egyszerű CRUD (*Create, Read, Update, Delete*) alkalmazást, amely pl. felhasználók adatait kezeli.

7 Middleware használata

Az Express middleware-ek az Express alkalmazás szíve-lelke. Ezek olyan funkciók (függvények), amelyek hozzáférést biztosítanak a bejövő kérésekhez, és módosíthatják azokat, illetve válaszokat generálhatnak vagy átadhatják a vezérlést a következő middleware-nek a láncban. A middleware-ek alapvető szerepet játszanak az alkalmazás logikájának kezelésében és szervezésében.

Amikor egy HTTP kérést kap a szerver, az Express végigfut a **middleware lánc**on, és minden egyes middleware megkapja a következő három dolgot:

- **Kérés objektumot** (req): A kliens által küldött kérésről szóló információk.
- **Válasz objektumot** (res): Ezen keresztül küldheted vissza a válaszokat a kliensnek.
- **Következő middleware hívása** (next()): Ez a függvény hívja meg a következő middleware-t a láncban.

A middleware-lánc a programban való fizikai elhelyezkedés alapján épül fel.

```
app.use((req, res, next) => {
  console.log('Első middleware');
  next();
});

app.use((req, res, next) => {
  console.log('Második middleware');
  next();
});

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

A next() hívás tudja, hogy melyik a következő middleware a láncban, mert az Express a middleware-eket belsőleg sorban regisztrálja, amikor az app.use() vagy app.get(), stb. metódusokat meghívod. Amikor egy middleware meghívja a next()-et, az Express automatikusan a következő regisztrált middleware-t futtatja.

A middleware-ek rugalmas módon szervezik az alkalmazás működését, lehetőséget adva a kód újra felhasználására, tisztább logika kialakítására és a moduláris felépítésre.

7.1 Middleware típusok

7.1.1 Alkalmazás szintű middleware

Az egész alkalmazásra érvényesek, minden útvonalra és HTTP metódusra lefutnak. Ezt általában az app.use() metódussal definiálják.

```
app.use((req, res, next) => {
  console.log('Alkalmazás szintű middleware.');
  next();
});
```

7.1.2 Route (útvonal) szintű middleware

Csak egy adott útvonalon, vagy útvonalcsoporton futnak le. A middleware függvényt itt paraméterként adhatod át egy adott útvonalnak.

```
app.get('/user/:id', (req, res, next) => {
  console.log('Csak a /user/:id útvonalra fut le');
  next();
}, (req, res) => {
  res.send('Felhasználói információk');
});
```

7.1.3 Harmadik fél által készített middleware-ek

Express middleware-eket harmadik felek is készítenek, amelyek megkönnyítik például a hitelesítést, a fájlfeltöltést, a naplózást stb. Ezeket NPM csomagokon keresztül telepítheted és használhatod.

```
const morgan = require('morgan');
app.use(morgan('combined')); // Naplózza a kéréseket
```

7.1.4 Hiba middleware

Ez egy speciális típusú middleware, amelyet a hibák kezelésére használnak. Egy hiba middleware-t négy paraméterrel definiálnak: *err, req, res* és *next*. Ezek csak akkor hívódnak meg, ha valami hiba történik az alkalmazásban.

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Valami elromlott!');
});
```

7.1.5 Saját middleware

Használd az Express beépített middleware-jeit, mint a express.json() vagy a express.static(), és ha nem találsz megfelelőt, akkor készítsd el a sajátodat.

Leggyakrabban az alábbi területeken lesz rá szükséged:

- Kérések feldolgozása: *Lehetővé teszik az adatok, pl. JSON vagy form adat feldolgozását.*
- Naplózás: *Segíthetnek naplózni a kérések érkezését.*
- Hitelesítés: *Ellenőrizhetik a felhasználók jogosultságait.*

- Statikus fájlok kiszolgálása: *Statikus fájlokat szolgálhatnak ki, például HTML, CSS, JavaScript*.
- Hiba kezelés: *Kezelhetik az alkalmazásban felmerülő hibákat*.

Gyakorlatok:

- Implementálj egy middleware-t, ami minden kérés előtt naplózza a kérés időpontját.
- Hozz létre egy egyszerű logger middleware-t, amely minden kérésnél kiírja a konzolra az időbélyeget, az útvonalat és a HTTP metódust.
- Készíts egy hibakezelő middleware-t, amely kezeli a nem létező útvonalakat (404-es hibák).

```
app.use((req, res, next) => {
   res.status(404).send('Az oldal nem található!');
});
```

  A 404-es hibakezelő middleware-t mindig a route-ok után kell elhelyezni, hogy a többi route ellenőrzése után fusson le.

- Alkalmazz egy body-parser middleware-t a JSON adat kezelésére a POST és PUT metódusok esetén.

```
function validateRequest(req, res, next) {
  if (!req.body.name || req.body.name.length < 5) {
    return res.status(400).send('Name must be at least 5 characters long');
  }
  next();
}
```

- Készíts a kérések számának a korlátozására egy middleware-t

```
let requestCounts = {};

function rateLimitMiddleware(req, res, next) {
  const userIp = req.ip;
  requestCounts[userIp] = (requestCounts[userIp] || 0) + 1;

  if (requestCounts[userIp] > 100) {
    return res.status(429).send('Too many requests');
  }

  next();
}
```

- Készíts a CORS kezelésre

```
function corsMiddleware(req, res, next) {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Methods', 'GET,POST,PUT,DELETE');
  next();
}
```

8 Adatbázis integráció

8.1 MySQL adatbázis elérésére middleware segítségével

```
const express = require('express');
const mysql = require('mysql');
const app = express();
const port = 3000;

// MySQL kapcsolat létrehozása
const db = mysql.createConnection({
    host: 'localhost',   // Az adatbázis szerver címe
    user: 'root',        // MySQL felhasználó
    password: '',        // MySQL jelszó
    database: 'pizza'    // Az adatbázis neve
});

// Kapcsolat indítása
db.connect((err) => {
    if (err) {
        console.error('Hiba történt a MySQL kapcsolódás során: ', err);
        return;
    }
    console.log('Csatlakozás a MySQL adatbázishoz sikeres.');
});

// GET kérés, amely JSON formátumban küldi vissza a vevő adatokat
app.get('/vevok', (req, res) => {
    const sql = 'SELECT * FROM vevo'; // SQL lekérdezés
    db.query(sql, (err, results) => {
        if (err) {
            console.error('Hiba a lekérdezés során: ', err);
            res.status(500).json({ error: 'Adatbázis hiba' });
            return;
        }
        res.json(results); // Az eredményt JSON formátumban küldjük vissza
    });
});

// Az alkalmazás elindítása
app.listen(port, () => {
    console.log(`Az alkalmazás fut a http://localhost:${port} címen.`);
});
```

8.2 ORM használata

8.2.1 Mongoose

8.2.2 Sequelize

Gyakorlat:

- Hozz létre egy alkalmazást, amely adatokat kér le és tárol egy adatbázisban.
- Integrálj egy MongoDB vagy MySQL adatbázist az alkalmazásba.
- Töltsd le az adatokat az adatbázisból (pl. felhasználói adatokat) és jelenítsd meg őket a REST API-n keresztül.
- Adatok beillesztése és frissítése az adatbázisban a POST és PUT műveletek segítségével.

9 Sablonmotorok használata

Sablonmotor jelentősen megkönnyíti a kód és az adatok elválasztását, a weboldalak dinamikussá tételét, valamint a fejlesztés gyorsítását. Express támogat több sablonmotort.

9.1 EJS (Embedded JavaScript)

<% %> és <%= %> jelölésekkel dolgozik. Egyszerű és közvetlen, támogatja a vezérlési szerkezeteket, mint a ciklusok és feltételek (például for, if).

9.2 Handlebars.js

Használja a „mustache” stílusú kódot {{ }}, amely egyszerű és tiszta megjelenést ad. Támogatja a segédfüggvényeket (helpers) és a részsablonokat, amik segítenek a sablon felépítésében és újrafelhasználhatóságában.

9.3 Pug (korábban Jade)

Különleges és tömör, HTML-típusú kóddal dolgozik, ahol a behúzások számítanak, így kevesebb zárójelet használ. Egyszerű logikai műveletek, ciklusok és feltételek elérhetők benne, valamint részletes CSS és JavaScript támogatás.

9.4 Mustache

„Mustache” stílusú, {{ }} jelölés használatával, ami tiszta, de logikamentes sablonokat eredményez. Minimális logika, mivel a sablonmotor nem támogat ciklusokat vagy feltételeket – ezek JavaScript-ből jönnek. Egyszerű adatmegjelenítéshez használják, mivel a sablonok nem bonyolíthatók túlságosan.

9.5 Nunjucks

Használja a Django-hoz hasonló {% %} és {{ }} szintaxist. Kifejezetten erős sablonkezelés, sok beépített funkcióval (pl. szűrők, feltételek, ciklusok). Frontend és backend sablonoknál egyaránt jól működik, ahol szükségesek a komplexebb sablonstruktúrák.

Moduláris és támogat segédfüggvényeket, komplex feltételeket, ciklusokat, melyek nagyobb rugalmasságot nyújtanak.

Gyakorlat:

- **Sablonmotor beállítása és használata**: Készíts egy egyszerű sablont egy adott motorral, és adj vissza dinamikus tartalmat.
- Integrálj egy templating motort, például **Pug**-ot vagy **EJS**-t az alkalmazásba.
- Készíts egy dinamikus HTML oldalt, amely a szerverről kapott adatokat jeleníti meg *(pl. felhasználók listája*).

10 Hibakezelés és biztonság

- **Hibakezelés**: Tanuld meg, hogyan kezeld a hibákat az Express-ben, mind szinkron, mind aszinkron módon (Promise-ok).
- **Biztonsági intézkedések**: Ismerkedj meg a biztonsági gyakorlattal, pl. CORS kezelése, input validáció, titkosítás.

Gyakorlat:

- Implementálj alapvető hibakezelést és biztonsági middleware-t (pl. Helmet).

11 Autentikáció és jogosultságkezelés

Hitelesítés: Tanuld meg, hogyan implementálhatsz hitelesítést (pl. JSON Web Token vagy session alapú hitelesítés).

- **Jogosultságok kezelése**: Kezeld a felhasználói jogosultságokat a különböző API végpontokon.

Gyakorlat:

- Hozz létre egy bejelentkezési rendszert, ahol felhasználók tokenek segítségével tudnak autentikálni.
- Készíts egy egyszerű bejelentkezési rendszert, ahol a felhasználók regisztrálhatnak és bejelentkezhetnek.
- Használj JWT-t (JSON Web Token) a hitelesítéshez és a védett útvonalakhoz.
- Védj le egyes útvonalakat, hogy csak bejelentkezett felhasználók férhessenek hozzájuk.

- Kezeld a session-öket az **express-session** csomag segítségével.

12 File feltöltés kezelése

Cél: Megismerni, hogyan kezelhetünk fájlokat Express alkalmazásban.

- Implementálj egy fájlfeltöltést, amely lehetővé teszi képek vagy dokumentumok feltöltését a szerverre a **multer middleware** segítségével.
- A feltöltött fájlokat tárold egy dedikált mappában, és jelenítsd meg őket egy oldalon.

13 Paginating és Sorting egy API-ban

Cél: Adatok lapozásának és rendezésének kezelése.

- Készíts egy útvonalat, ahol nagy mennyiségű adatot kell lapozni (pl. felhasználók listája), és adj hozzá lapozást (pagination).
- Add hozzá a rendezési (sorting) lehetőséget egy lekérdezési paraméter alapján (pl. ?sort=name).

14 Tesztelés és telepítés

- **Tesztelés**: Tanuld meg, hogyan tesztelheted az Express alkalmazásod automatikusan (pl. Mocha, Chai vagy Jest segítségével).
- **Telepítés**: Ismerkedj meg a telepítési folyamatokkal, pl. hogyan telepíthetsz alkalmazást Heroku-ra vagy más cloud platformra.

Gyakorlat:

- Írj teszteket az API végpontjaidhoz, majd telepítsd az alkalmazásodat egy felhőszolgáltatóra.

15 További források: