Seminar 7

Crearea rutelor în Express.js

Definirea rutării înseamnă utilizarea metodelor obiectului aplicației Express care corespund metodelor HTTP pentru a gestiona modul în care aplicația răspunde la cererile clientului. De exemplu, metoda **app.get()** se utilizează pentru a trata cererile de tip GET, în timp ce metoda **app.post()** se folosește pentru a gestiona cererile de tip POST.

O metodă de rută este derivată din una dintre metodele HTTP și este atașată unei instanțe a clasei Express.

Codul de mai jos este un exemplu de rute definite pentru metodele GET și POST către rădăcina aplicației.

```
// GET method route
app.get("/", (req, res) => {
  res.send("GET request to the homepage");
});

// POST method route
app.post("/", (req, res) => {
  res.send("POST request to the homepage");
});
```

Metoda specială de rutare **app.all()** este utilizată pentru a încărca funcții middleware pe o anumită cale astfel încât acestea să se aplice la toate metodele de cerere HTTP. De exemplu, handler-ul următor este executat pentru cererile către ruta **'/users'**, indiferent dacă se folosește metoda GET, POST, PUT, DELETE sau oricare altă metodă de cerere HTTP acceptată în modulul HTTP.

```
app.all("/users", logger);
```

```
const logger = (req, res, next) => {
  console.log(`${req.method} ${req.path} ${res.statusCode}`);
  next();
};
```

Express.js Router

Express.Router este o clasă care permite crearea, gestionarea și organizarea rutelor într-o aplicație node.js într-un mod modular și structurat. Această clasă face parte din modulul express și este folosită pentru a defini și grupa rutele într-un mod reutilizabil și ușor de întreținut.

Prin intermediul acestui router, putem organiza și gestiona rutele într-un mod modular și să le montezi pe căi specifice în aplicația principală.

Astfel, plecăm de la următoarea secvență de cod:

```
const express = require("express");
const orders = require("./utils/constants");
const app = express();

app.get("/", function (req, res) {
  res.send("Hello World");
});

app.get("/orders", function (req, res) {
  res.status(200).json(orders);
});

app.get("/user/:id", function (req, res) {
  const id = req.params.id;
  res.send(`User with id: ${id}`);
});

app.listen(3000);
```

Acum, o să mutăm toate rutele care țin cont de comenzi și de utilizatori în fișiere speciale, încercând să le separăm și să mai eliberăm **index.js**, locul în care este creat serverul.

La sfârșit, codul o să fie structurat în felul următor:

ordersRoutes.js

```
const express = require("express");
const { orders } = require("../utils/constants");
const router = express.Router();

router.get("/", function (req, res) {
  res.status(200).json(orders);});
module.exports = router;
```

usersRoutes.js

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

const router = express.Router();

router.get("/user/:id", function (req, res) {
  const id = req.params.id;
  res.send(`User with id: ${id}`);
});

module.exports = router;
```

Index.js

```
const ordersRoutes = require("./routes/ordersRoutes");
const usersRoutes = require("./routes/usersRoutes");
const express = require("express");
const app = express();

app.get("/", function (req, res) {
  res.send("Hello World");
});

app.use("/orders", ordersRoutes);
app.use("/users", usersRoutes);

app.listen(3000);
```

După cum putem observa, rutele pentru comenzi și utilizatori au fost mutate în fișiere separate și au fost chemate în cadrul fișierului index.js folosind un middleware prin care toate rutele care conțin **'/orders'** o să se ducă să caute ruta în fișierul **'ordersRoutes'** și toate rutele care conțin '/users' o să caute ruta respectivă în cadrul fișierului **'usersRoutes'**.

Funcții middleware

Funcțiile middleware sunt funcții care au acces la obiectul de cerere (**req**), obiectul de răspuns (**res**) și la **următoarea funcție** middleware din ciclul cerere-răspuns al aplicației. Următoarea funcție middleware este în mod obișnuit indicată prin intermediul unei variabile numită 'next'.

Funcțiile middleware pot efectua următoarele sarcini:

1. Să execute orice cod.
2. Să facă modificări asupra obiectelor de cerere și de răspuns.
3. Să termină ciclul cerere-răspuns.
4. Să apeleze următoarea funcție middleware din stiva de middleware.

Dacă funcția middleware curentă nu încheie ciclul cerere-răspuns, trebuie să apeleze **next()** pentru a transfera controlul către următoarea funcție middleware. În caz contrar, cererea va rămâne în așteptare.

Pentru a demonstra modul în care funcționează un middleware, o să creăm o nouă metodă care va afișa în consolă de fiecare dată când cineva apelează una din rute și o să o adăugăm tuturor rutelor.

```
const ordersRoutes = require("./routes/ordersRoutes");
const usersRoutes = require("./routes/usersRoutes");
const express = require("express");
const app = express();

const logger = (req, res, next) => {
  console.log(`${req.method} ${req.path} ${res.statusCode}`);
  next();
};

app.use(logger);

app.get("/", function (req, res) {
  res.send("Hello World");
});

app.use("/orders", ordersRoutes);
app.use("/users", usersRoutes);

app.listen(3000);
```

Astfel, am creat o funcție care se va asigura de faptul că fiecare cerere o să fie afișată în consolă.

Alte exemple de funcții middleware:

```
app.use(express.urlencoded({ extended: true })); // for having access to req.body
app.use(express.json()); // for parsing data with Content-Type: application/json
```

Livrarea de conținut static

Livrarea conținutului static în Express.js se realizează folosind middleware-ul **'express.static'**.

Pași necesari:

1. **Organizarea conținutului static**: Înainte de a putea livra conținut static, trebuie să organizăm fișierele statice într-un director specific. Acest director, de obicei numit **"public"** sau **"static"**, conține resursele statice pe care dorim să le oferim clienților, cum ar fi imagini, fișiere CSS sau JavaScript.
2. **Configurarea middleware-ului**: În codul serverului, trebuie să configurăm middleware-ul **'express.static'** pentru a indica calea către directorul în care se găsesc fișierele statice. Acest lucru se face prin următoarele linii de cod:

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

// Specifică directorul pentru fișierele statice folosind __dirname
app.use(express.static(path.join(__dirname, 'public')));
```

3. **Adăugarea de conținut**: În cadrul folderului "public," se va stoca conținut static, inclusiv imagini, CSS, JavaScript, favicon-uri, fonturi, fișiere multimedia și resurse suplimentare.

Libăria Dotenv

Dotenv este un modul fără dependențe care încarcă variabilele de mediu dintr-un fișier .env în process.env. Stocarea configurației în mediul de lucru, separat de cod.

Pași necesari pentru configurarea librăriei:

1. Se instalează pachetul prin comanda: **npm install dotenv**;
2. Crearea unui fișier .env și adăugarea variabilelor de mediu și valorile lor;

3. Configurarea librăriei în cadrul aplicației în fișierul în care este creat serverul:

```
const dotenv = require("dotenv");
dotenv.config();
```

4. Utilizarea variabilelor de mediu în locul datelor sensibile prin metoda **‘process.env’**. Exemplu:

```
const PORT = process.env.PORT || 3000;
app.use(cors({ origin: process.env.CLIENT_URL || "http://localhost:5500" }));
app.listen(PORT, function () {
  console.log(`Server is running on port http://localhost:${PORT}`);});
```

5. Adăugarea fișierului de .env în fișierul de .gitignore pentru a nu fi urmărit și urcat pe GitHub.