

(Node.js), Express, middleware

backend demo

27.2.2025



Haaga-Helia

Express framework for building REST API backend

- You use Express object and functions => (Express uses the services of the Node.js itself behind the scenes.)
- *"Write less and get more done"*
- Express we use to:
 - Configure the backend server
 - Setup the backend routing, certain request URL pattern to a certain handler function
 - To add middleware to the request handling loop
 - To get URL parameters, GET parameters or POST data to our handler program easily
 - `req.query.age` (The Get parameters of the URL, with `?name="Joe"&age=13`)
 - `req.params.id` (The `:` marked parts of the URL, like `/category/:id`)
 - `req.body.somePropertyOfTheJsonObjectInBody`
 - (Middleware function returned by `express.json` did already parse the request body Json and offers it in `req.body` to you. Either a JSON array object, or straight a single JSON object)
 - Possibly for validation and common error-handling and logging

Middleware? Express.json() as example

- Express allows **middleware** functions to be used in the request handling loop, before the request comes to your own request handling code in the routes folder
- Middleware functions take the request (request-response object pair actually), and do something to it, and then ask next middleware function to continue, thus "pass the control to the next middleware handler".
- That is: they intervene in the request processing pipeline for each and every request. After all the middleware the last function to be used is your own request handler defined with `.get()` `.delete()` `.post()` or so.
- E.g. inside: `index.js` in root folder (the starting point of the backend app)
 - `import express from "express";`
 - `const app = express();`
 - `app.use(express.json());` // adds a function (that `express.json()` returns) to the request handling loop
- ... then in e.g. `/routes/category.js`
 - `if(req.body.name.length > 2)` // `express.json` has parsed the body JSON text as a JS object automatically.

Another example of Middleware - CORS

- CORS = Cross-origin Resource Sharing
- Protection mechanism. Trying to stop illegal connection initiated from another server.
- With the CORS backend app developer can configure what is allowed
- E.g. inside: index.js in root folder (the starting point of the backend app)
 - `import express from "express";`
 - `import cors from "cors";`
 - `const app = express();`
 - `app.use(cors());` // Merely disabling the cross-origin resource sharing safety mechanism! Allowing all. Hazardous!
- ... then in all the rest api endpoints, the CORS allows, in this example, any cross-origin sharing
- In production we should of course use whitelist of allowed connections from client (web or another server), and configure CORS to allow only them.