Node.js, Express, middleware

idea-case-backend demo

10.2.2023

Haaga-Helia

Node.js (or Node)

- https://nodejs.org
- Since year 2009
- JavaScript runtime environment. Running JS apps outside browser.
- Cross-platform (*Windows, Linux, Mac*, ...)
- Often used for running backend server code
- But used for other things too, like running the development-time frontend development environment of *create-react-app*

Node characteristics

- Fast and Scalable
  - if and when you code all services truly asynchronous (=non-blocking) and keep processing intensive logic outside.
- E.g. use database server for all it can do processing-wise. RDBMSs are really powerful tools for all processing. They have been developed since 1970s by best brains in the World.
- Need to write just little code to create backend
  - especially when using the Express framework (simplifies routine stuff)
- Easy to build incrementally prototyping way

- Google Chrome's V8 JavaScript engine
  - single-thread running,
  - better and better modern ECMAScript support. https://node.green/ and https://nodejs.org/en/docs/es6/

Node – Programming model

- Modules used
  - old times CommonJS modules with module.exports & require
  - nowadays Node supports newer ECMAScript (default or named) export & import
- Each code file is wrapped to be its own visibility block = a module.
  - And you will export the public parts you want to import and use in other modules = files.
- The starting point is e.g. the index.js in the root folder. Anyway the file you start with **npm start** or **nodemon** or **node some.js**
- There is no browser runtime's 'window' object. There would be a 'global' object instead, but don't use it.
- While the Node app is starting we use the Express 'app' object to configure the starting app. E.g. by attaching more and more middleware to the request handling processing pipe / loop
- There are
  - your modules = your local .js or .ts code files
  - ready-made modules, which you import from repositories. E.g. Node&Express modules or third-party modules.

Node – Programming model

- While looking at the code try to identify parts that:
  - Are run at the server startup – Configure, set up handlers and create things. Done with function calls.
  - Are run later when something further happens – Those are function definitions.

LET'S LOOK AT EXAMPLES

Node modules and npm/npx

- Other modules, look e.g. into https://www.npmjs.com/
- 475 000 modules available. Some made by our students last semester as seminar work.
- Be careful, especially these times there might be bad actors trying to sneak malicious code into public npm etc. repositories
  - Sometimes even known programmer was mentioned to have sold his project to outside actor

Most common Commands:

- npm install -g nodemon / npm i -g nodemon Tool installed globally to whole computer
- npm i express Module installed to this project
- npx create-react-app myapp Tool only temporarily downloaded and run immediately
- npm install, npm audit, npm outdated, npm update

Node app creation and added modules

1. Install Node from nodejs.org (LTS version). Installs also npm, npx, …
2. Possibly in e.g. GitHub create a new empty repo with a Node .gitignore template
3. Clone that almost empty repo to a local folder, then go inside that repo folder
4. **npm init** => creates the package.json file where the node modules = project dependencies are listed. Plus some other configuration of the Node app
5. **npm install** would install all dependencies already in the package.json
6. **npm install -save express** would install express module and also add it to package.json
7. **npm install -g nodemon** would install the node monitor tool globally
8. other tools we could install with **npm i / npm install** are e.g.
    a. **cors** – for configuring the CORS security mechanism. "middleware that can be used to enable CORS with various options."https://www.npmjs.com/package/cors
    b. **express** – for easier Node app object configuration e.g. for routing. "A minimal and flexible Node.js web application framework that provides a robust set of features for building web servers" https://www.npmjs.com/package/express
        i. **express.json** built-in middleware function in Express. It parses incoming requests with JSON payloads https://expressjs.com/en/api.html
    c. **express-validator**, express middlewared version of js validator called, well, '**validator**' https://www.npmjs.com/package/express-validator = how to use that 'validator' in express
    d. **knex** – for writing JavaScript to create database actions. and get data back as JSON. https://www.npmjs.com/package/knex
    e. **mysql** or **mariadb** – driver/connector/client for connecting to MariaDB/MySQL database https://www.npmjs.com/package/mysql https://www.npmjs.com/package/mariadb
    f. **winston** – for logging. https://www.npmjs.com/package/winston
    g. **dotenv** – handling environment settings and properties https://www.npmjs.com/package/dotenv

REST API endpoints

- Use the minimalistic URL patterns:
  - GET /product => no id or name given, thus: get <u>all</u> Products
  - GET /product/:id => now id given, thus get one product by id
  - DELETE /product/:id => notice how URL pattern is 100% same as for GET. Thus the http method is important part of the routing.
  - GET /product/cheaperThan/:price => you can build endless special URLs too.
- HTTP Methods we use:
  - POST: create new resource(s) to the backend (in case of auto-increment id, id is not provided in request)
  - PUT: update an existing resource by replacing it with the new version of it (even with auto-increment ids, id is needed and provided in the request, to know which resource to update)
  - GET: fetch the resource(s) from backend
  - DELETE: delete the resource(s) from backend
- Use some tool to test your endpoints constantly. Also after you are ’done with backend’ and doing frontend development. Postman or VS Code REST client