Node API backends

Michael Chang
Spring 2023
Plan for today

Recap: Node servers in Express
   Defining a route, returning JSON

Express middleware
   Storing variables about a request
   Reading request body

Aside: CORS

Designing clean REST APIs
Aside: directory structure

**Stuff we created/defined**

api: NodeJS code for defining API routes

public: HTML/CSS/(frontend) JS sent to the browser

lib: Code we provide (both client and server)
  - Only has the auto-refresh code

server.js: Script that starts the Node server
Aside: directory structure

Stuff common to all Node projects

- package.json: Metadata about the project, including dependencies
- package-lock.json: Info about the exact packages you've installed for the project
- node_modules: The actual packages you installed

Notes

- When sending your project, delete node_modules
- When downloading a project, run "npm install" to create node_modules
- If something is wrong with npm install, try deleting package-lock.json
Note: frontend/backend separation

Client and server both in JS, but separate
- Typically running on separate machines
- Client "calls" the server via fetch; can't call a function
- Server responds with JSON; can't return arbitrary values (classes, etc.)

Client modules
- Client imports modules from public dir
- Can include external libraries with <script> tags

Server modules
- import Node builtin libs and npm packages
- npm to install external libraries
Express middleware

**Function that runs before handler for route**

```javascript
app.get("/students/:id", (req, res, next) => {
  res.locals.student = STUDENTS[req.params.id];
  next();
}, (req, res) => { ... });
```

**res.locals**: information about this request

(Can't use global variables, because multiple reqs handled in parallel)

**next()**: call next function in the "chain"

Allows multiple middlewares, then final handler

Don't send response and also call next()
Express middleware

**app.use to add middleware**

```javascript
app.use("/students/:id", (req, res, next) => {
    res.locals.student =
        STUDENTS[req.params.id];
    next();
});
```

**Call the middleware function for all requests starting with /students/:id**

- Sets res.locals.student
- Later endpoints can use it
Reading request body

Need to interpret request body as JSON

Does not happen automatically

body-parser

Maintained by Express devs, but separate npm package
Provides middleware to read request body in various formats

Usage

import bodyParser from "body-parser";
app.use(bodyParser.json());
app.post("/...", (req, res) => {
  let id = req.body.id;
  ...
});
Aside: CORS

Normally can't fetch() from different "origin"

Origin = host and port
E.g. if server running on another machine, or another port on same machine

    fetch("http://localhost:1931/api");

Cause: CORS

Prevents malicious web sites from reading content from your pages/APIs

Solution

    import cors from "cors";
    app.use(cors());
API design tips

Each "thing" in your system has unique URI

E.g. the "mchang" student accessed via /students/mchang

If you need a way to look up students by other fields, use query string

E.g. /students?firstName=Michael

Think of paths like folders

/students

  /students/mchang

  /students/mchang/courses

Not: /student_courses/mchang
API design tips

Use HTTP methods effectively

- GET requests should never update data
- Use PATCH when updating a resource, and DELETE for deleting
- When updating/deleting a resource, use the resource URI
  - E.g. PATCH /students/mchang
  - Not: PATCH /students/mchang/update
- Use POST for creating, and for misc actions
  - These may need a suffix, like POST /users/mchang/enroll
API design tips

**Use request body to send objects**
- E.g. when creating or updating a resource

**Use HTTP errors to report problems**
- E.g. 400 means request missing parameter or can't be completed
- Include error message in the JSON
  - Could be human readable, or program readable, or both
Summary

**Today**
Writing API backends in Node

**Before next time**
Project proposal
assign3.1
Set up MongoDB (see lecture schedule)

**Next time**
Persistent data storage (databases)