Static Files, Middleware

CS 390 – Web Application Development

J. Setpal

October 24, 2022

Table of Contents

Table of Contents

WIWYT – Static Files

- Express is pretty incredible: it allows us to develop full-fledged APIs like they're Hello World projects.

WIWYT – Static Files

- Express is pretty incredible: it allows us to develop full-fledged APIs like they're Hello World projects.
- Serving static files is an essential component to a web API; understanding it is useful to help us build better software.

WIWYT – Middleware

- Express works fundamentally as an abstraction layer over the traditional API implementation. It's a big reason why it's so easy to work with.

WIWYT – Middleware

- Express works fundamentally as an abstraction layer over the traditional API implementation. It's a big reason why it's so easy to work with.
- Middleware allows us fine-grained control over the routing process within the API, enabling us to extend the Express functionality depending on the use-case.

Table of Contents

Static Files – General Idea, Syntax

The syntax for serving a static file is straightforward:

```
// ... some code
app.use('/location', express.static('path/to/dir'))
// ... more code
```

Static Files – General Idea, Syntax

The syntax for serving a static file is straightforward:

```
// ... some code
app.use('/location', express.static('path/to/dir'))
// ... more code
```

The directory is crucial to **containerize** file serving!

Let's Build a File Server!

If you can view this screen, I am making a mistake.

Table of Contents

Middleware – Understanding Hooks

Idea: EVERYTHING IS MIDDLEWARE!

Middleware – Understanding Hooks

Idea: EVERYTHING IS MIDDLEWARE!

Q: What happens when we run an Express function? How does Express interpret it?

Middleware – Understanding Hooks

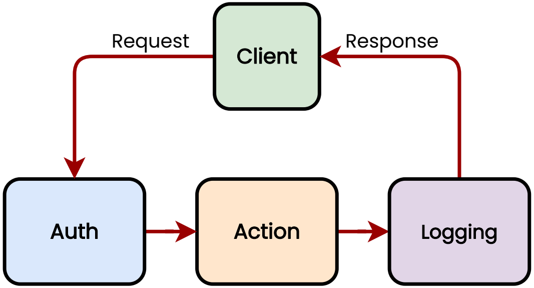
Idea: EVERYTHING IS MIDDLEWARE!

Q: What happens when we run an Express function? How does Express interpret it?
A: It runs a series of functions sequentially - like traversing a linked list.

Middleware – Understanding Hooks

Idea: EVERYTHING IS MIDDLEWARE!

Q: What happens when we run an Express function? How does Express interpret it?
A: It runs a series of functions sequentially - like traversing a linked list.



Middleware – Syntax

Let's break down some sample code:

```
// ... some code
app.get('/', function, (req, res) => {
                    res.send('Hello from the Express
                        API!!');
           })

function f(req, res, next) {
         console.log('f');
         next();
}
// ... some code
```

Middleware – Passing Values Between Functions

Passing data is incredibly important; it's what allows functions to communicate.

Middleware – Passing Values Between Functions

Passing data is incredibly important; it's what allows functions to communicate. Here's how we do it:

```
// ... some code
app.get('/', function, (req, res) => {
                    console.log(`${req.f}`);
                    res.send('Hello from the Express
                        API!!');
           })

function f(req, res, next) {
         console.log('f');
         req.f = true;
         next();
}
// ... some code
```

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.
- We can't update the {`req, res`} variables after next is called. At the end of the chain, the result is sent to the client.

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.
- We can't update the {`req, res`} variables after next is called. At the end of the chain, the result is sent to the client.

Secondly:

- Middleware is called in order of declaration.

Middleware – Some Nuance

There's **two** pitfalls to avoid. Firstly:

- `next()` is **not** a `return`. It remains in the stack, and is called at the end of the chain.
- We can't update the {`req, res`} variables after next is called. At the end of the chain, the result is sent to the client.

Secondly:

- Middleware is called in order of declaration.
- Don't accidentally call authentication after the action!

Let's Implement Server Logging!

If you can view this screen, I am making a mistake (again).

Table of Contents

On Wednesday –

We'll cover working with Forms!

Also, sending, and understanding the difference between from types, like GET and POST.

Homework – Build an API!

- The API should statically serve files.

Homework – Build an API!

- The API should statically serve files.
- There should be a dynamic index to list the files!

Homework – Build an API!

- The API should statically serve files.
- There should be a dynamic index to list the files!
- Password-protected by argument (doesn't need to be secure; naive implementation is OK).

Homework – Build an API!

- The API should statically serve files.
- There should be a dynamic index to list the files!
- Password-protected by argument (doesn't need to be secure; naive implementation is OK).
- Add a fun item please :)

Starter code, and the announcement with full details will be posted shortly!

Thank you!

Have an awesome rest of your day!

Slides: `https://www.cs390.dev/slides/static,middleware.pdf`

If anything's incorrect or unclear, please ping: `jsetpal@purdue.edu`
I'll patch it ASAP.