Express Routing

SENG 4640
Software Engineering for Web Apps
Winter 2023

Sina Keshvadi
Thompson Rivers University

Review

- Node.js and Express represent HTTP requests and responses using JavaScript objects

- We can use these objects' properties and functions to dynamically generate the content that is sent in response to a request

The feature of Express that we'll use for creating different responses for different requests is known as **routing.**

Express Routing

```
var express = require('express');
var app = express();

app.use('/about', (req, res) => {
   res.send('This is the about page.');
});

app.use('/login', (req, res) => {
   res.send('This is the login page.');
});

app.use( /*default*/ (req, res) => {
  res.status(404).send('Not found!');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

index.js file

Express Routing

```
var express = require('express');
var app = express();

app.use('/about', (req, res) => {
   res.send('This is the about page.');
});

app.use('/login', (req, res) => {
   res.send('This is the login page.');
});

app.use( /*default*/ (req, res) => {
  res.status(404).send('Not found!');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

to create a express app (a back-end app)

Express Routing

```
var express = require('express');
var app = express();

app.use('/about', (req, res) => {
   res.send('This is the about page.');
});

app.use('/login', (req, res) => {
   res.send('This is the login page.');
});

app.use( /*default*/ (req, res) => {
  res.status(404).send('Not found!');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

specify a callback function for each individual URL

Express Routing

```
var express = require('express');
var app = express();

app.use('/about', (req, res) => {
   res.send('This is the about page.');
});

app.use('/login', (req, res) => {
   res.send('This is the login page.');
});

app.use( /*default*/ (req, res) => {
  res.status(404).send('Not found!');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

if we get here that means that we could not service the request, so we'll send back the 404 status along with the content not found.

run

```
> node index.js
```

and Open http://localhost:3000/about

Each of those functions is known as **middleware.**

Express Middleware

- A **middleware** is a function that is invoked in the handling of an HTTP request
- This will allow us to serve static files such as HTML, images, etc, that are locally stored.
- It is used in the “middle” between receiving a request and sending a response
- Multiple middlewares can be chained together on the same request

Middleware: Serving Static Files

- The simplest middleware is **`express.static`**, which serves static files that are locally stored

```
varexpress = require('express');
varapp = express();
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).send('Not found!');
});
app.listen(3000, () => {
console.log('Listening on port 3000');
});
```

Middleware: Serving Static Files

- The simplest middleware is **`express.static`**, which serves static files that are locally stored

```
varexpress = require('express');
varapp = express();
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).send('Not found!');
});
app.listen(3000, () => {
console.log('Listening on port 3000');
});
```

specify that any URI that's in the /public folder should serve static content.

Middleware: Serving Static Files

- The simplest middleware is **`express.static`**, which serves static files that are locally stored

```
varexpress = require('express');
varapp = express();
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).send('Not found!');
});
app.listen(3000, () => {
console.log('Listening on port 3000');
});
```

```
URI in /public
```

Middleware: Serving Static Files

- The simplest middleware is **express.static**, which serves static files that are locally stored

```
varexpress = require('express');
varapp = express();
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).send('Not found!');
});
app.listen(3000, () => {
console.log('Listening on port 3000');
});
```

use the express.static middleware function to specify the folder or directory that holds the static content.

Middleware: Serving Static Files

- The simplest middleware is **express.static**, which serves static files that are locally stored

```
varexpress = require('express');
varapp = express();
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).send('Not found!');
});
app.listen(3000, () => {
console.log('Listening on port 3000');
});
```

In this case, there would be a folder or directory called files that would hold, for instance, our html files.

```
<!-- This is files/index.html -->

<html>
<body>
<h1>Hello!</h1>
<img src="images/kitty.jpg">
<!-- File is files/images/kitty.jpg -->
</body>
</html>
```

Here we have an example where we have a static html file.
This would be in the **files** directory called **index.html**.

```
<!-- This is files/index.html -->

<html>
<body>
<h1>Hello!</h1>
<img src="images/kitty.jpg">
<!-- File is files/images/kitty.jpg -->
</body>
</html>
```

```
<!-- This is files/index.html -->

<html>
<body>
<h1>Hello!</h1>
<img src="images/kitty.jpg">
<!-- File is files/images/kitty.jpg -->
</body>
</html>
```

Here, this image would be in the files/images folder
underneath the root of our node application.

```
<!-- This is files/index.html -->

<html>
<body>
<h1>Hello!</h1>
<img src="images/kitty.jpg">
<!-- File is files/images/kitty.jpg -->
</body>
</html>
```

app_three

we can choose to send back specific files

Middleware: Serving Static Files

- We can use the response object to send back specific HTML files as needed

```
var express = require('express');
var app = express();
app.use('/public', express.static('files'));
app.use( /*default*/ (req, res) => {
res.status(404).send('Not found!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

For instance in this example, when we're handling the 404 not found, rather sending back the string not found, we can choose to send back a file that maybe includes the message that the requested resource couldn't be found

Middleware: Serving Static Files

- We can use the response object to send back specific HTML files as needed

```
var express = require('express');
var app = express();
app.use('/public', express.static('files'));
app.use( /*default*/ (req, res) => {
res.status(404).sendFile(__dirname + '/404.html');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

We'll use the send file function to return a file.
This variable __direname just specifies the location of where we've installed our node app. And we'll say that we've created a 404.html file, that we want to send back as a result of this request.

make a 404.html file
run
> node index.js
open a page that does not exist.

Defining and Using Middleware

- Middleware functions can contain any amount of JavaScript code with any functionality

- They take three parameters: `req`, `res`, and `next`

- `next()` must be called at the end of the function to invoke the next middleware or the final response

lets see how we would create a middleware function that would be in the chain of our requests.

```
var express = require('express');
var app = express();

var logger = (req, res, next) => { var
  url = req.url;
  var time = new Date();
  console.log('Received request for ' + url +
                  ' at ' + time);
  next();
};

app.use(logger);
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).sendFile(__dirname + '/404.html');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

create a function called logger that takes the request, the response, and the next function to invoke.

```
var express = require('express');
var app = express();

var logger = (req, res, next) => { var
  url = req.url;
  var time = new Date();
  console.log('Received request for ' + url +
                  ' at ' + time);
  next();
};

app.use(logger);
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).sendFile(__dirname + '/404.html');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

log to the console that we received a request for this URL at this time.

```
var express = require('express');
var app = express();

var logger = (req, res, next) => { var
  url = req.url;
  var time = new Date();
  console.log('Received request for ' + url +
                  ' at ' + time);
  next();
};

app.use(logger);
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).sendFile(__dirname + '/404.html');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

we must invoke the **next** function so that the next middleware in the chain will be invoked.

```
var express = require('express');
var app = express();

var logger = (req, res, next) => { var
  url = req.url;
  var time = new Date();
  console.log('Received request for ' + url +
                  ' at ' + time);
  next();
};

app.use(logger);
app.use('/public', express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).sendFile(__dirname + '/404.html');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

Meaning that the logger middleware will be invoked on any HTTP request.

```
var express = require('express');
var app = express();

var logger = (req, res, next) => { var
  url = req.url;
  var time = new Date();
  console.log('Received request for ' + url +
                  ' at ' + time);
  next();
};

// app.use(logger);
app.use('/public', logger, express.static('files'));

app.use( /*default*/ (req, res) => {

  res.status(404).sendFile(__dirname + '/404.html');
});

app.listen(3000, () => {
   console.log('Listening on port 3000');
});
```

in this case, we only want to have the logger used when we access the static files in the public folder. But we don't want the logger to be used, for instance, when we have the 404.

to test this example you must have a **files** directory which has a index.html file.

open
   http://localhost:3000/public/
to see the request logs.

to test this example you must have a **files** directory which has a index.html file.

open
   http://localhost:3000/public/
to see the request logs.

you can’t see the output of console.log(“messages”) on your browser. why?

Middleware Chaining

- Middleware functions are called in the order in which they are specified
- Each uses (or share) the same Request and Response objects
- A middleware function can modify the Request so that it can then be used by subsequent middleware functions “downstream” in the route

Let’s see an example, where different routes use different middleware functions, and one middleware function will modify the request so that it can be used by another middleware function later on.

```
var nameFinder = (req, res, next) => {
  var name = req.query.name;
  if (name) req.username = name.toUpperCase();
  else req.username = 'Guest';
  next();
}

var greeter = (req, res, next) => {
  res.status(200).type('html');
  res.write('Hello, ' + req.username); next();
}

var adminName = (req, res, next) => {
  req.username   = 'Admin';
  next();
}

app.use('/welcome', nameFinder, greeter,
                          (req, res) => { res.end(); } );
app.use('/admin', adminName, greeter,
                          (req, res) => { res.end(); } );
```

we're using app.use to define a root for the /welcome URI.

```
var nameFinder = (req, res, next) => {
  var name = req.query.name;
  if (name) req.username = name.toUpperCase();
  else req.username = 'Guest';
  next();
}

var greeter = (req, res, next) => {
  res.status(200).type('html');
  res.write('Hello, ' + req.username); next();
}

var adminName = (req, res, next) => {
  req.username   = 'Admin';
  next();
}

app.use('/welcome', nameFinder, greeter,
                          (req, res) => { res.end(); } );
app.use('/admin', adminName, greeter,
                          (req, res) => { res.end(); } );
```

The first middleware function that's invoked is the nameFinder

```
var nameFinder = (req, res, next) => {
  var name = req.query.name;
  if (name) req.username = name.toUpperCase();
  else req.username = 'Guest';
  next();
}

var greeter = (req, res, next) => {
  res.status(200).type('html');
  res.write('Hello, ' + req.username); next();
}

var adminName = (req, res, next) => {
  req.username   = 'Admin';
  next();
}

app.use('/welcome', nameFinder, greeter,
                          (req, res) => { res.end(); } );
app.use('/admin', adminName, greeter,
                          (req, res) => { res.end(); } );
```

the query is just part of the URL that follows the URI and provides some information that's been passed by the browser.
If the query contains the name, the name will be defined.

```
var nameFinder = (req, res, next) => {
  var name = req.query.name;
  if (name) req.username = name.toUpperCase();
  else req.username = 'Guest';
  next();
}

var greeter = (req, res, next) => {
  res.status(200).type('html');
  res.write('Hello, ' + req.username); next();
}

var adminName = (req, res, next) => {
  req.username   = 'Admin';
  next();
}

app.use('/welcome', nameFinder, greeter,
                          (req, res) => { res.end(); } );
app.use('/admin', adminName, greeter,
                          (req, res) => { res.end(); } );
```

The *next* function in the root is the function called greeter,

```
var nameFinder = (req, res, next) => {
  var name = req.query.name;
  if (name) req.username = name.toUpperCase();
  else req.username = 'Guest';
  next();
}

var greeter = (req, res, next) => {
  res.status(200).type('html');
  res.write('Hello, ' + req.username);
  next();
}

var adminName = (req, res, next) => {
  req.username   = 'Admin';
  next();
}

app.use('/welcome', nameFinder, greeter,
                          (req, res) => { res.end(); } );
app.use('/admin', adminName, greeter,
                          (req, res) => { res.end(); } );
```

```
var nameFinder = (req, res, next) => {
  var name = req.query.name;
  if (name) req.username = name.toUpperCase();
  else req.username = 'Guest';
  next();
}

var greeter = (req, res, next) => {
  res.status(200).type('html');
  res.write('Hello, ' + req.username);
  next();
}

var adminName = (req, res, next) => {
  req.username   = 'Admin';
  next();
}

app.use('/welcome', nameFinder, greeter,
                          (req, res) => { res.end(); } );
app.use('/admin', adminName, greeter,
                          (req, res) => { res.end(); } );
```

Then it will call next which will go to the next middleware, which in this case, is just a little anonymous function that's gonna call res.end to send the HTML and close the connection.

```
var nameFinder = (req, res, next) => {
  var name = req.query.name;
  if (name) req.username = name.toUpperCase();
  else req.username = 'Guest';
  next();
}

var greeter = (req, res, next) => {
  res.status(200).type('html');
  res.write('Hello, ' + req.username); next();
}

var adminName = (req, res, next) => {
  req.username   = 'Admin';
  next();
}

app.use('/welcome', nameFinder, greeter,
                          (req, res) => { res.end(); } );
app.use('/admin', adminName, greeter,
                          (req, res) => { res.end(); } );
```

sets the username to admin, and then invokes next like previous one.

```
var express = require("express");
var app = express();

var nameFinder = (req, res, next) => {
 var name = req.query.name;
 if (name) req.username = name.toUpperCase();
 else req.username = "Guest";
 next();
};

var greeter = (req, res, next) => {
 res.status(200).type("html");
 res.write("Hello, " + req.username);
 next();
};

var adminName = (req, res, next) => {
 req.username = "Admin";
 next();
};

app.use("/welcome", nameFinder, greeter, (req, res) => {
 res.end();
});
app.use("/admin", adminName, greeter, (req, res) => {
 res.end();
});

app.listen(3000, () => {
 console.log("Listening on port 3000");
});
```

Full code

Try opening a web page that Welcomes you by name.
What would the URL look like?

open
http://localhost:3000/admin

and

http://localhost:3000/welcome?name=alice

and
http://localhost:3000/welcome

and
http://localhost:3000/

Middleware, Routes, and Routers

- We may find that the same combinations of middleware functions are being used in multiple routes

- We can combine middleware functions into “sub-routes” using **Routers** and then use those in our routes

```
var nameFinder = (req, res, next) => { . . . }
var greeter = (req, res, next) => { . . . }

var adminName = (req, res, next) => { . . . }


var logger = (req, res, next) => { . . . }

var header = (req, res, next) => { . . . }

var footer = (req, res, next) => { . . . }


app.use('/welcome', logger, nameFinder, header,
      greeter, footer, (req, res) => { res.end(); } );


app.use('/admin', logger, adminName, header,
      greeter, footer, (req, res) => { res.end(); } );
```

we have several middleware functions (header and footer like others)

```
var nameFinder = (req, res, next) => { . . . }
var greeter = (req, res, next) => { . . . }

var adminName = (req, res, next) => { . . . }


var logger = (req, res, next) => { . . . }

var header = (req, res, next) => { . . . }

var footer = (req, res, next) => { . . . }


app.use('/welcome', logger, nameFinder, header,
      greeter, footer, (req, res) => { res.end(); } );


app.use('/admin', logger, adminName, header,
      greeter, footer, (req, res) => { res.end(); } );
```

we can see in both of these routes we have this sub route of header, greeter, footer.

```
var nameFinder = (req, res, next) => { . . . }
var greeter = (req, res, next) => { . . . }

var adminName = (req, res, next) => { . . . }

var logger = (req, res, next) => { . . . }

var header = (req, res, next) => { . . . }

var footer = (req, res, next) => { . . . }

var commonRoute   = express.Router();
commonRoute.use(header, greeter, footer);


app.use('/welcome', logger, nameFinder, header,
      greeter, footer, (req, res) => { res.end(); } );

app.use('/admin', logger, adminName, header,
      greeter, footer, (req, res) => { res.end(); } );
```

It would be easier if we could combine these into some common sub route

```
var nameFinder = (req, res, next) => { . . . }
var greeter = (req, res, next) => { . . . }

var adminName = (req, res, next) => { . . . }

var logger = (req, res, next) => { . . . }

var header = (req, res, next) => { . . . }

var footer = (req, res, next) => { . . . }

var commonRoute   = express.Router();
commonRoute.use(header, greeter, footer);


app.use('/welcome', logger, nameFinder, commonRoute,
      (req, res) => { res.end(); } );

app.use('/admin', logger, adminName, commonRoute,
      (req, res) => { res.end(); } );
```

Summary

- **Routing** allows us to specify different functionality for different HTTP requests
- Routing uses **middleware** functions, each of which handles a different part of the functionality
- Middleware functions can be chained together and can pass values to each other by modifying the Request object
- **Routers** allow us to combine middleware functions into common “sub-routes”