Introduction to Computer Science I
COMP 2406A – Winter 2020

Express (Part 1)

Dave McKenney
david.mckenney@carleton.ca

Learning Outcomes

by the End of this Lecture, Students that have Completed the Reading Assignment and Review Questions should be Able to:

Use basic routing functionality of Express

Handle HTTP requests/responses using Express

Connect vs. Express

The goal of Connect is to provide a middleware framework

That is, a way to create/include middleware within an HTTP server app

Connect vs. Express

Connect, however, is lacking in some regards

For example, our middleware was used for all types of requests (POST, GET, etc.)

This makes sense – it wasn’t designed for this

Connect vs. Express

Express, on the other hand, is a web framework

Extending from Connect, Express provides the same middleware functionality

But also provides significantly more web-based utility

Getting Started with Express

```
npm install express

const express = require('express’);

let app = express();
```

Consider 11-ex1-barebones-express.js

Adding Middleware to Express

Within express, we can use the same middleware as we used in the Connect examples

Express even provides some shorthand aliases for several of these middlewares

Adding Middleware to Express

app.use(express.static(“dirname”)); creates a static server for the given directory

Uses the serve-static middleware to easily serve up the static resources your application needs (GET .html, .js, images, etc.)

Can use multiple static servers, see 11-ex2-static-server.js

Adding Middleware to Express

You can also add a route prefix:

```
app.use('/static', express.static('public’));
```

This will serve requests to
http://localhost:3000/static/someResource.html
from the local directory ‘public’

Adding Middleware to Express

Similarly, Express has shorthand for incorporating body parsers:

```
app.use(express.json());
app.use(express.urlencoded({extended: true}));
```

Express Request/Response Objects

The Express request and response objects have the original Node.js objects as a base

That is, it provides the same functionality at minimum (e.g., req.method, req.url)

Express automatically adds various useful properties/methods to the objects we have access to

Express Request Objects

The request object in Express includes:

req.body – the body data from the body parser, if any

Express Request Objects

The request object in Express includes:

req.query – an object containing the query parameter key/values, if any

Express req.query

```
GET /search?q=tobi+ferret
console.log(req.query.q) // => 'tobi ferret'

GET/shoes?ord=desc&shoe[color]=red&shoe[type]=nike
console.log(req.query.ord) // => 'desc'
console.log(req.query.shoe.color) // => 'red'
console.log(req.query.shoe.type) // => 'nike'

GET /shoes?color[]=blue&color[]=black&color[]=red
console.log(req.query.color) // => ['blue', 'black', 'red']
```

Express Request Objects

The request object in Express includes:

req.accepts(types) – returns either the best-matched content type specified in the types array, or false if none match

Allows for content negotiation - returning a desired type of data for the client

Express req.accepts(str)

```
// Accept: text/html
        req.accepts('html’) // => "html"

// Accept: text/*, application/json
        req.accepts('html’) // => "html"

        req.accepts('text/html’) // => "text/html"

        req.accepts(['json', 'text’]) // => "json"

        req.accepts('application/json’) // =>
                                        "application/json"
```

Express req.accepts(str)

```
// Accept: text/*, application/json
        req.accepts('image/png’) // => undefined
        req.accepts('png’) // => undefined

// Accept: text/*;q=.5, application/json
        req.accepts(['html', 'json’]) // => "json"
```

Express Request Objects

The request object in Express includes:

req.get(field) – returns the specified request header field if it exists, false otherwise

Express Request Objects

The request object in Express includes:

req.is(type) – returns false if the content type of the request body does not match the specified type, null if no body, the matching content if there is a match

Can be used to determine how to process the body

Express req.is(str)

```
// With Content-Type: text/html; charset=utf-8
req.is('html’) // => 'html'
req.is('text/html’) // => 'text/html'
req.is('text/*’) // => 'text/*'

// When Content-Type is application/json
req.is('json’) // => 'json'
req.is('application/json’) // => 'application/json'
req.is(text’) // => false
```

Express Response Objects

The response object in Express includes:

res.status(code) – sets the status code of response

Can be chained: res.status(404).send(“Unknown resource.”)

This chaining is common in Express

Express Response Objects

The response object in Express includes:

res.set('Header-Name', 'value') – sets the specified response header to the given value

Can also take an object with many headers

Express Response Objects

The response object in Express includes:

res.type(string) – sets the Content-Type header of the response based on the string

Can take types (e.g., “application/json”) or file extensions (e.g., “.html”)

Express Response Objects

The response object in Express includes:

res.format(object) – takes an object that specifies content-type/function key/values, uses req.accepts to select the appropriate handler and sends response

Useful for providing multiple data formats

Express Response Objects

```
res.format({
        'text/plain': function () {
                res.send('hey’)
        },
        'text/html': function (){
                res.send('<p>hey</p>’)
        },
        'application/json': function () {
                res.send({ message: 'hey’ })
        },
        'default': function () {
          // log the request and respond with 406
          res.status(406).send('Not Acceptable’)
        }
});
```

res.format Example: Note that you could just give function names

Express Response Objects

The response object in Express includes:

res.json(obj) – sends a JSON response containing the given object, uses JSON.stringify()

Automatically sets content-type header and ends response

Express Response Objects

The response object in Express includes:

res.send(body) – sends the given body in the response

Can also res.send(statusCode, body). Sets content type automatically based on input (e.g., string=HTML, object=JSON) and ends response

Express Response Objects

The response object in Express includes:

res.sendFile(path) – sends the file specified by path within the body of the response

Automatically sets Content-Type based on extension

Can also specify options and a callback to be called when sending is complete

Routes and Methods in Express

Our original server designs (using HTTP module) had a lot of code along the lines of:

```
If the request is a GET request:
    If the request if for "/someURL":
        Handle it with this code
    ...
...
```

Routes and Methods in Express

In Connect, we assigned middleware to particular routes

This cleaned our code up a bit

But Connect itself did not account for different request methods (e.g., GET, POST, etc.)

Routes and Methods in Express

Express allows us to set up handlers for requests for specific resources using specific methods

This will allow us to define handling in a way similar to the way we originally did it

e.g., if GET request for /someURL, use someFunc

However, it will provide a cleaner way of doing so

Routes and Methods in Express

Instead of only providing the app.use directive, Express also provides:

app.all(path, function)
app.get(path, function)
app.post(path, function)
app.put(path, function)
app.delete(path, function)

These allow you to define handling for particular HTTP methods (or all methods)

Routes and Methods in Express

If you support multiple methods on a specific resource (we will soon do a lot of this), Express provides a nice way of organizing this

Routes and Methods in Express

```
app.route('/book’) //the resource
 .get(function (req, res) { //handle GET requests
  res.send('Get a random book’);
 })
 .post(function (req, res) { //handle POST requests
  res.send('Add a book’);
 })
 .put(function (req, res) { //handle PUT requests
  res.send('Update the book’);
 })
```

This is less prone to typos and other careless errors

Routes and Methods in Express

```
app.route('/book’) //the resource
 .get(function (req, res) { //handle GET requests
  res.send('Get a random book’);
 })
 .post(function (req, res) { //handle POST requests
  res.send('Add a book’);
 })
 .put(function (req, res) { //handle PUT requests
  res.send('Update the book’);
 })
```

It keeps everything related nearby

Routes and Methods in Express

```
app.route('/book’) //the resource
 .get(function (req, res) { //handle GET requests
  res.send('Get a random book’);
 })
 .post(function (req, res) { //handle POST requests
  res.send('Add a book’);
 })
 .put(function (req, res) { //handle PUT requests
  res.send('Update the book’);
 })
```

Later, we’ll see you can divide handling into modules

Routes and Methods in Express

Consider the re-designed bank code in 11-ex3-express-bank.js and 11-ex3-express-bank-cleaned-up.js

More Advanced Routing

Express provides more advanced routing as well

Next, we will look at this functionality and its relationship with the RESTful design principles we discussed previously

Questions?

Questions?