Express Server

Agenda

- Middleware
- Serving Static Files
- Routing
  - Route Methods
  - Route Paths
  - Route Handlers
  - Route Parameters
  - Handling Query Strings
  - Response Methods
  - app.route()
  - express.Router()
- Body Parser

Middleware

- Is a function that has access to
  - HTTP request object (req)
  - HTTP response object (res)
  - The next middleware function (next)
- Used to perform the following tasks
  - Execute any code
  - Make changes to the request and the response objects
  - End the request-response cycle
  - Call the next middleware in the stack
- An Express app is essentially a series of middleware function calls

Middleware

- Example middleware function - requestTime()

```
var requestTime = function (req, res, next) {
  req.requestTime = Date.now();
  next();
}
```

  - This function adds a property called requestTime to the request object
  - Sets the current date to the property

Middleware

- To load the middleware function
  - Call app.use(), and pass middleware function as the parameter

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

var requestTime = function (req, res, next) {
  req.requestTime = Date.now();
  next();
};

app.use(requestTime);
```

  - The Express app object uses the requestTime middleware function

Middleware

- When a request is made to the root of the app, it displays the timestamp of the request

```
app.get('/', function (req, res) {
  var responseText = 'Hello World!<br>'
  responseText += '<small>Requested at: ' +
                  req.requestTime + '</small>'
  res.send(responseText)
})
```

- Note
  - If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging

Serving Static Files

- Use the ‘express.static()’ built-in middleware function to serve static files
  - Static files are files such as image, CSS file, JS file
  - Use the following code to serve images, CSS files, and JS files in a directory named ‘public’

```
app.use(express.static('public'));
```

  - To create a virtual path prefix for files that are served by the express.static function

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

Routing

- Definition of application end points and how they respond to client requests

```
app.METHOD(PATH, HANDLER)
```

  - Where
    - app – is an instance of express
    - METHOD – is an HTTP request method (lowercase)
    - PATH – is a path on the server
    - HANDLER – is the function executed when the route is matched

Routing

- Route Methods
  - Derived from one of the HTTP methods (GET, POST, PUT, DELETE, etc.)
  - Is attached to an instance of the ‘express’ class
    - Use app.get() to handle GET requests
    - Use app.post() to handle POST requests

```
// GET method route example
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// PUT method route example
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user');
})
```

Routing

- Route Paths
  - Define the ‘endpoints’ at which requests can be made
  - Can be strings, string patterns, or regular expressions
- Endpoint
  - A combination of a URI (path) and a HTTP request method (GET, POST, …)
- Route Handlers
  - Also called ‘handler functions’, generally are callback functions
  - Called when the app receives a request to the specified route and HTTP method
  - Can be a function, an array of functions, or combination of both

Routing

- Route Parameters
  - Named URL segments
  - Used to capture values specified at their position in the URL
  - For e.g., to define a route with route parameters - ‘userId’ and ‘bookId’, the path of the route is specified as follows

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params);
})
```

Routing

- Route Parameters (Continued)
  - The parameter values are populated in the req.params object, with parameter names specified in the path as their respective keys

```
// Actual Request URL:
//   http://localhost:3000/users/34/books/8989

// req.params: { "userId": "34", "bookId": "8989" }

// Within the route handler, 'userId' and 'bookId'
// parameters can be accessed using the following code

  const userId = req.params.userId;
  const bookId = req.params.bookId;
```

Routing

- Handling Query Strings
  - Query string
    - Is a part of the URL, generally appended at the end
    - Contains data as one or more key-value pairs
    - Prefixed by a question mark (?)
    - Multiple key-value pairs are separated using &
  - ‘query’ property of Request object is an object containing a property for each query string parameter in the route
  - If there is no query string, it is the empty object, {}

Routing

– Query string example

```
// Query String Example
// http://example.com/path/to/page?name=hari&age=25

// GET /path/to/page?name=hari&age=25

req.query.name
// => “hari“

req.query.age
// => “26"
```

Routing

- Response Methods
  - Send a response to the client
  - Terminate the request - response cycle
  - If not called, the client request will be left hanging

Method	Description
res.send()	Send a response of various types
res.json()	Send a JSON response
res.render()	Render a view template
res.end()	End the response process

Routing

- app.route()
  - Can be used to create chainable route handlers for a particular route path
  - Used to avoid duplicate route names (and thus typo errors)

```
app.route('/book')
  .get(function (req, res) {
    res.send('Get a random book');
  })
  .post(function (req, res) {
    res.send('Add a book');
  })
  .put(function (req, res) {
    res.send('Update the book');
  })
```

Routing

- express.Router
  - Is used to create modular, mountable route handlers
  - Instance is a complete middleware and routing system
  - Often called as a “mini-app”

Routing

– Create a router file named ‘books.js’ in the app directory

```
var express = require('express');
var router = express.Router();

// define the home page route
router.get('/', function (req, res) {
  res.send('Books home page');
})

// define the about route
router.get('/about', function (req, res) {
  res.send('About books');
})

module.exports = router;
```

Routing

– Now, load the router module in the app

```
var books = require('./books')

// ...

app.use('/books', books)
```

– The app will now be able to handle requests to ‘/books’ and ‘/books/about’

Body Parser

- Middleware used to parse the body of incoming request
- Gets called before the route handlers
- Makes the data available to handlers via ‘req.body’ property

- Install body-parser

```
npm install body-parser
```

Body Parser

- API

```
var bodyParser = require('body-parser');
```

  - bodyParser object exposes various factories to create middlewares like:
    - JSON body parser
    - URL-encoded form body parser
    - Raw body parser
    - Text body parser
  - All middlewares will populate the ‘req.body’ property with the parsed body

Body Parser

- Add a generic JSON and URL-encoded parser as a top-level middleware, which will parse the bodies of all incoming requests

```
var express = require('express')
var bodyParser = require('body-parser')

var app = express()

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json
app.use(bodyParser.json())

app.use(function (req, res) {
  res.setHeader('Content-Type', 'text/plain')
  res.write('you posted:\n')
  res.end(JSON.stringify(req.body, null, 2))
})
```

THANK YOU