NODE

EXPRESS ROUTES

Routing

Routing refers to how an application’s endpoints (URIs) respond to client requests

Get method

```
const express = require('express')
const app = express()

// respond with "hello world" when a GET request is made to
//the homepage
app.get('/', (req, res) => {
  res.send('hello world')
})
```

Express supports methods that correspond to all HTTP request methods:

```
app.all('/secret', (req, res, next) => {
  console.log('Accessing the secret section ...')
  next() // pass control to the next handler
})
```

Route Paths

This route path will match requests to the root route, /.

```
app.get('/', (req, res) => {
  res.send('root')
})
```

Route Paths

This route path will match requests to /about.

```
app.get('/about', (req, res) => {
  res.send('about')
})
```

Route Paths

This route path will match requests to /random.text.

```
app.get('/random.text', (req, res) => {
  res.send('random.text')
})
```

Parameterized Routes

- Route path: /users/:userId/books/:bookId
- Request URL: http://localhost:3000/users/34/books/8989
- req.params: { "userId": "34", "bookId": "8989" }

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

Parameterized Routes

Route path: /flights/:from-:to

Request URL: http://localhost:3000/flights/LAX-SFO

req.params: { "from": "LAX", "to": "SFO"

Parameterized Routes

Route path: /plantae/:genus.:species

Request URL: http://localhost:3000/plantae/Prunus.persica

req.params: { "genus": "Prunus", "species": "persica" }

Query String

https://stackabuse.com/?page=2&limit=3

The query parameters are the actual key-value pairs like page and limit with values of 2 and 3, respectively.

Your query parameters can be retrieved from the query object on the request object

A Typical Use Case
/?page=2&limit=3

```
app.get('/', async function(req, res) {
    // Access the provided 'page' and 'limt' query parameters
    let page = req.query.page;
    let limit = req.query.limit;
    let articles = await Article.findAll().paginate({page: page, limit: limit}).exec();
    // Return the articles to the rendering engine
    res.render('index', {
        articles: articles
    });
});
```

NODE

MIDDLEWARES

Middlewares

Middleware functions are functions that have access to

- Request
- Response
- the next middleware function

Middleware functions can 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 function in the stack.

Middleware

```
// Define middleware function
const myMiddleware = (req, res, next) => {
  // Do something with the request (e.g., logging)
  console.log('Middleware executed');
  // Call next() to pass control to the next middleware in the stack
  next();
};
// Use middleware in your application
app.use(myMiddleware);
```

Simple Logger

```
const logMethodAndUrl = (req, res, next) => {
    // Log the HTTP method and URL
    console.log(`Method: ${req.method}, URL: ${req.url}`);
    next();
};
```

Terminate req, res cycle

```
const maintenanceMiddleware = (req, res, next) => {
    // Send a 503 Service Unavailable status code
    res.status(503).send('Site under maintenance. ');
};

//If added at top it will not pass control to any route
```

Types of middlewares

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

Application-level middleware

Bind application-level middleware to an instance of the app object….

```
const express = require('express')
const app = express()
app.use((req, res, next) => {
  console.log('Time:', Date.now())
  next()
})
```

Mounted on /users/:id for all methods

```
app.use('/user/:id', (req, res, next) => {
  console.log('Request Type:', req.method)
  next()
})
```

Route and its handler

```
app.get('/user/:id', (req, res, next) => {
  res.send('USER')
})
```

Series of middlewares

```
app.use('/user/:id', (req, res, next) => {
  console.log('Request URL:', req.originalUrl)
  next()
}, (req, res, next) => {
  console.log('Request Type:', req.method)
  next()
})
```

Second middleware will never be called. why?

```
app.get('/user/:id', (req, res, next) => {
  console.log('ID:', req.params.id)
  next()
}, (req, res, next) => {
  res.send('User Info')
})
// handler for the /user/:id path, which prints the user ID
app.get('/user/:id', (req, res, next) => {
  res.send(req.params.id)
})
```

Middlewares in an array for reusability.

```
function logOriginalUrl (req, res, next) {
  console.log('Request URL:', req.originalUrl)
  next()
}
function logMethod (req, res, next) {
  console.log('Request Type:', req.method)
  next()
}
const logStuff = [logOriginalUrl, logMethod]
app.get('/user/:id', logStuff, (req, res, next) => {
  res.send('User Info')
})
```

Router Level Middlewares

Middlewares can also be applied to express routers

```
const express = require('express')

const app = express()

const router = express.Router()

// a middleware function with no mount path. This code is executed for every request to the router

router.use((req, res, next) => {

  console.log('Time:', Date.now())

  next()

})
```

Error-handling middleware

```
app.use((err, req, res, next) => {
  console.error(err.stack)
  res.status(500).send('Something broke!')
})
```

Built In Middlewares

- express.static serves static assets such as HTML files, images, and so on.
- express.json parses incoming requests with JSON payloads. **NOTE: Available with Express 4.16.0+**
- express.urlencoded parses incoming requests with URL-encoded payloads. **NOTE: Available with Express 4.16.0+**

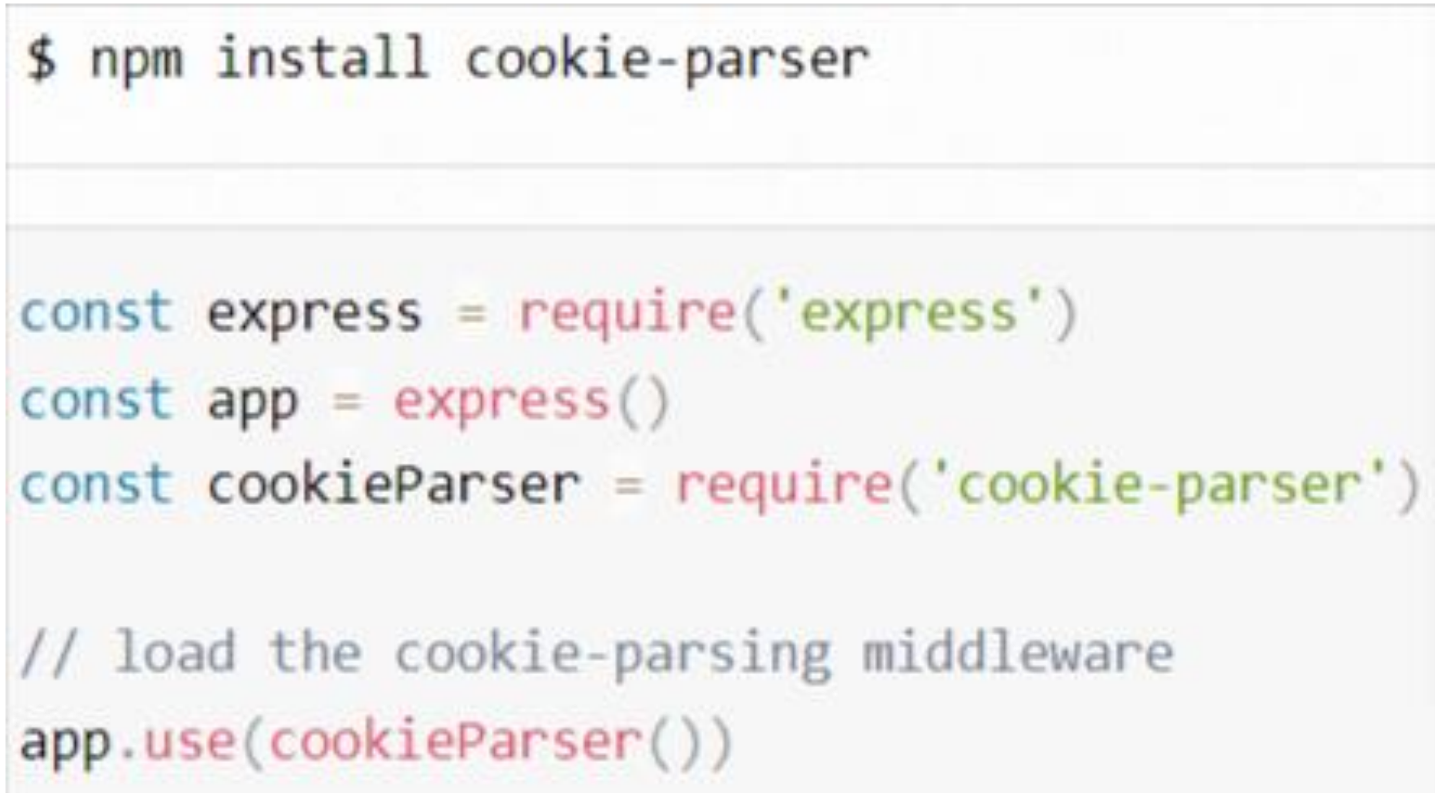
middlewares

express.json() is a built express middleware that convert request body to JSON.

express.urlencoded() just like express.json() but also convert form-data to JSON etc.

- **Extended:true** This option allows to choose between parsing the URL-encoded data with the querystring library (when false) or the qs library (when true). The “extended” syntax allows for rich objects and arrays to be encoded into the URL-encoded format, allowing for a JSON-like experience with URL-encoded. For more information, please see the qs library.

Third-party middleware



```
$ npm install cookie-parser
```

```
const express = require('express')
const app = express()
const cookieParser = require('cookie-parser')

// load the cookie-parsing middleware
app.use(cookieParser())
```

What it Does

```
async function checkSessionAuth(req, res, next) {
  if (!req.session.user) {
    req.flash("danger", "You need to login for this route");
    return res.redirect("/login");
  }
  next();
}
module.exports = checkSessionAuth;
```