COURSEWARE

- Professional Skills
- Agile Fundamentals
- Jira
- Git
- Databases Introduction
- Java Beginner
- Maven
- Testing (Foundation)
- Java Intermediate
- HTML
- CSS
- Javascript
- Spring Boot
- Selenium
- Sonarqube
- Advanced Testing (Theory)
- Cucumber
- MongoDB
- Express
  - Intro
  - Handling Requests
  - Middleware
  - Error Handling
  - Router
  - Mongoose
- NodeJS
- React
- Express-Testing
- Networking
- Security

Router

Contents

- Overview
- Importing Router
- Using Router
- Project Structure
- Exporting Routes
- Base path
- Nesting middleware
- Tutorial
- Exercises

Overview

So far we have created routes *directly* on **app**.
Whilst this is okay Express provides us with the capability to define collections of routes using Express Router.

Importing Router

Router is provided with the Express package and can be imported into a module using **require()**.

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

This will create a new Router object from the Express object.
Router object objects function like a 'mini-application' - they have their own middleware and routes but these are *isolated* to the router rather than being used throughout the whole **app**.

Using Router

Express Router uses *exactly* the same functions that we saw in *Request Handling*.

```
router.get(path, callback);

router.put(path, callback);

router.post(path, callback);

router.patch(path, callback);

router.delete(path, callback);
```

Project Structure

Typically our routes will be separated into their own modules based on their *domain* and kept in a */routes* folder.

For example, an application that dealt with *customers* and *items* might look like this:

Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
.
├── index.js
├── package.json
├── package-lock.json
└── routes
    ├── items.js
    └── customers.js
```

Exporting Routes

Our routes aren't much good to us if they're just sat in a file in */routes* so in order to add them to our **app** we will first need to export them.

Lets use **items.js** from above as an example.

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

router.get('/get', (req, res) => {
    //
}

router.post('/create', (req, res) => {
    //
}

router.put('/update', (req, res) => {
    //
}

router.delete('/delete', (req, res) => {
    //
}

module.exports = router;
```

And then importing into **index.js**.

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

const itemRoutes = require('./routes/items.js');

app.use(itemRoutes);
```

By creating our routes in this way we can decouple the routes from the main Express application and increase code maintainability.

Base path

When adding our routes to the **app** we can specify a *base path*, this path will be prepended to the paths specified in the router.

For example if we specified a base path for **itemRoutes** like this:

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

const itemRoutes = require('./routes/items.js');

app.use('items', itemRoutes);
```

Then the final routes would become:

```
/items/get
/items/create
/items/update
/items/delete
```

Nesting middleware

Much like individual routes we can nest middleware so it only applies to a particular *collection* of routes.

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

const app = express();

const itemRoutes = require('./routes/items');

const logger = (req, res, next) => {
    console.log(new Date());
    next();
}

app.use('items', logger, itemRoutes);
```

In this example the **logger** middleware will run *before* the itemRoutes, log the date and then pass on the request using **next()**.

Tutorial

There is no tutorial for this module

Exercises

1. Go back to your *request_handling* solution, extract the routes out to another file using Express.Router() and import them into **index.js**

▶ Click here for solution