

EXPRESS JS



Agenda

2

- Use ExpressJS to build
 - ▣ REST API
 - ▣ Server side rendering

What is it?

3

- ❑ Minimalist web framework
- ❑ Provides more features on top of Node.js http module
- ❑ Great for building REST API
- ❑ Is considered fast
- ❑ Alternatives: Koa, Strapi, Hapi, Sails.js

Hello World

4

□ yarn add express

```
const express = require('express');  
  
const app = express();  
  
app.get('/', (req, res) => res.send('Hello World!'));  
  
app.listen(3000, () => console.log('Server is running'));
```

Basic Routing

5

- `app.METHOD(path, handler)`
 - ▣ `app` is an instance of `express`
 - ▣ `METHOD` is an HTTP request method, in lowercase
 - ▣ `PATH` is a path on the server
 - ▣ `HANDLER` is the function executed when the route is matched

```
app.get('/', function (req, res) {  
  res.send('Hello World!')  
})  
  
app.post('/', function (req, res) {  
  res.send('Got a POST request')  
})
```

Serving Static Files

6

- Use **express.static**

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

- The name of the static folder is not part of the URL
- `http://localhost:3000/index.html` is mapped to `public/index.html`
- Can define multiple static entries (order is important)
- `public` is relative to current directory. Probably it is better to use

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

express.static options

7

- **index** – Enable/disable the default index.html
- **maxAge** – Modify the Cache-Control header
- **setHeaders** – Set custom headers
- **fallthrough** – If file does not exist, continue to next middleware
- <https://github.com/expressjs/serve-static>

```
app.use(express.static(path.resolve(__dirname, './public'), {  
  index: ["index.html", "about.html"],  
  maxAge: 1000 * 60 * 5,  
}));
```

Route Matching

8

- ❑ Order is important
- ❑ Query string are not part of the route path
- ❑ Route path can be
 - ❑ string
 - ❑ string pattern `?+*()`
 - ❑ Regular expression

```
app.get("/abc*", function(req, res) {  
  res.send("Yo yo");  
});
```


Route Parameters

9

- The captured values are stored inside **req.params**

```
app.get("/api/contact/:id", function(req, res) {  
  res.send("id is " + req.params.id);  
});  
  
app.get("/api/contact/:id?", function(req, res) {  
  res.send("id might be undefiend");  
});
```

- Can append regular expression

```
app.get("/api/contact/:id(\\d+)", function(req, res) {  
  res.send("id is " + req.params.id);  
});
```

- Be aware that **req.params.XXX** is always a string

Unnamed Route Parameter

10

- Parentheses implies a captured value that can be retrieved using `req.params[INDEX]`

```
app.get("/api/contact/ab(c|d)/*", function(req, res) {  
  res.json(req.params);  
});
```

- For the URL `/api/contact/abc/ori`
 - ▣ `req.params[0]` → `c`
 - ▣ `req.params[1]` → `ori`

res.write

11

- Part of Node.js http module (Not ExpressJS)
- Sends the data to the client
- Must invoke **res.end**
- Automatically writes some default HTTP headers ☹

```
app.get("/api/contact/:id", function (req, res) {  
  res.writeHead(404, {  
    "Content-Type": "text/html",  
  });  
  res.write("a");  
  res.write("b");  
  res.write("c");  
  res.end();  
});
```

res.send

12

- ExpressJS API 😊
- Will set the **Content-Type** header according to the type of data you specify
- string → text/html
- Buffer → application/octet-stream
- object/array – application/json

```
app.get("/api/contact/:id", function (req, res) {  
  res.send({  
    ok: true  
  });  
});
```

Other Response Methods

13

- ☐ download
- ☐ json
- ☐ jsonp
- ☐ redirect
- ☐ render
- ☐ sendFile

Express Router

14

- Define a standalone “mini-app”
- Later, associate it with a route path

```
const {Router} = require('express');

const router = Router();

router.get("/details", function(req, res) {
  res.json({
    id:123,
    name: "Ori"
  });
});

router.post("/update", function(req, res) {
  res.json({
    ok: true,
  });
});

module.exports = router;
```

Attach a Router

15

```
const express = require('express');  
const path = require('path');  
const profile = require('./profile');  
  
const app = express();  
  
app.use("/profile", profile);  
  
app.listen(3000, () => console.log('Server is running'));
```

Supported URL is
/profile/details



Middleware

16

- A function that has access to the request & response object
- Can perform
 - ▣ Make changes to request/response
 - ▣ End the request-response cycle
 - ▣ Call the next middleware
- Must call **next** if does not end the current cycle
- Almost every thing is a middleware ...

Recap

17

□ Simple GET handler

```
app.get("/api/contact", function(req, res) {  
  res.send([]);  
});
```

This is a
middleware

□ Can be written as

```
app.get("/api/contact/:id?", function(req, res, next) {  
  if(!req.params.id) {  
    next();  
    return;  
  }  
  
  res.send({id: req.params.id});  
});  
  
app.get("/api/contact", function(req, res) {  
  res.send([]);  
});
```

Move to next
handler

Interception

18

```
app.use(function(req, res, next) {  
  res.locals.requestId = ++nextRequestId;  
  next();  
});  
  
app.use(function(req, res, next) {  
  const before = performance.now();  
  console.log(`BEGIN(${res.locals.requestId})`);  
  next();  
  
  res.on("finish", function() {  
    const after = performance.now();  
    console.log(`END(${res.locals.requestId})`);  
  });  
});
```

Attach a
unique id for
every request

Collect some
performance
counters

Default Error Handler

19

- ❑ Express handles any synchronous error encountered by a middleware
- ❑ The error is written to the client
- ❑ Including stack trace
- ❑ But not under production environment
 - ▣ `NODE_ENV=production`
- ❑ Asynchronous errors will kill the process ☹️


Custom Error Handler

20

- Define a middleware with 4 parameters

```
app.get("/api/contact", function(req, res) {  
  throw new ApplicationError(123, "Oooops");  
  res.send([]);  
});  
  
app.use(function(err, req, res, next) {  
  if(err.errorCode) {  
    res.json({  
      ok: false,  
      errorCode: err.errorCode,  
      errorMessage: err.message,  
    });  
  }  
  else {  
    next(err);  
  }  
});
```

Delegate to
the default
error handler



Debugging

21

- ❑ Express use the popular **debug** module
- ❑ By default no debug output
- ❑ Must use environment variable to see the debug output

```
DEBUG=express:* node main.js
```

View Engine

22

- Use any compliant engine: Pug, Mustache, EJS
- Specify
 - ▣ views directory
 - ▣ View engine name

```
app.set("views", "./views");  
app.set("view engine", "ejs");
```

- Use **res.render**

```
app.get("*", function(req, res) {  
  res.render("index.ejs", {  
    id: 1,  
    name: "Ori",  
  });  
});
```

EJS Syntax

23

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <h1>Hello EJS</h1>

  <div>ID: <%= id %></div>
  <div>Name: <%= name %></div>

  <% if(admin) { %>
    <h2>Admin section</h2>
  <% } %>
</body>
</html>
```

Summary

24

- ❑ ExpressJS is fun
- ❑ Abstraction layer on top of http module
- ❑ Middleware is powerful