Node JS - Day 2

Macy's Learning Spike

Useful links

- Repository for all labs code + solutions: https://github.com/alcfeoh/di-node-js

- Link to these slides: https://goo.gl/e1Cj86

Today we're going to build...

License Plate Store Home My cart Search Search

Welcome to our store

Browse our collection of License Plates below

2008 Georgia license plate

Ad occaecat ex nisi reprehenderit dolore esse. Excepteur laborum fugiat sint tempor et in magna labore quis exercitation consequat nulla tempor occaecat. Sit cillum deserunt eiusmod proident labore mollit. Cupidatat do ullamco ipsum id nisi mollit pariatur nulla dolor sunt et nostrud qui.

$8 Add to cart »

2015 New Jersey license plate

A beautiful license plate from the Garden State. Year is 2015.

$11 Add to cart »

2013 California My Tahoe license plate

Sunt irure nisi excepteur in ea consequat ad aliqua. Lorem duis in duis nisi sit. Cillum eiusmod ipsum mollit veniam consectetur ex eiusmod nisi laborum amet anim mollit non nulla. Lorem ea est exercitation nostrud consectetur officia laborum fugiat sint. Nostrud consequat magna officia minim et aute nostrud.

$9 Add to cart »

Outline for today

What is Express?

Error handling

Integrating CORS with Express

Advanced routing

Templating with Express

Building Restful APIs with Express

Outline for today

What is Express?

Error handling

Integrating CORS with Express

Advanced Routing

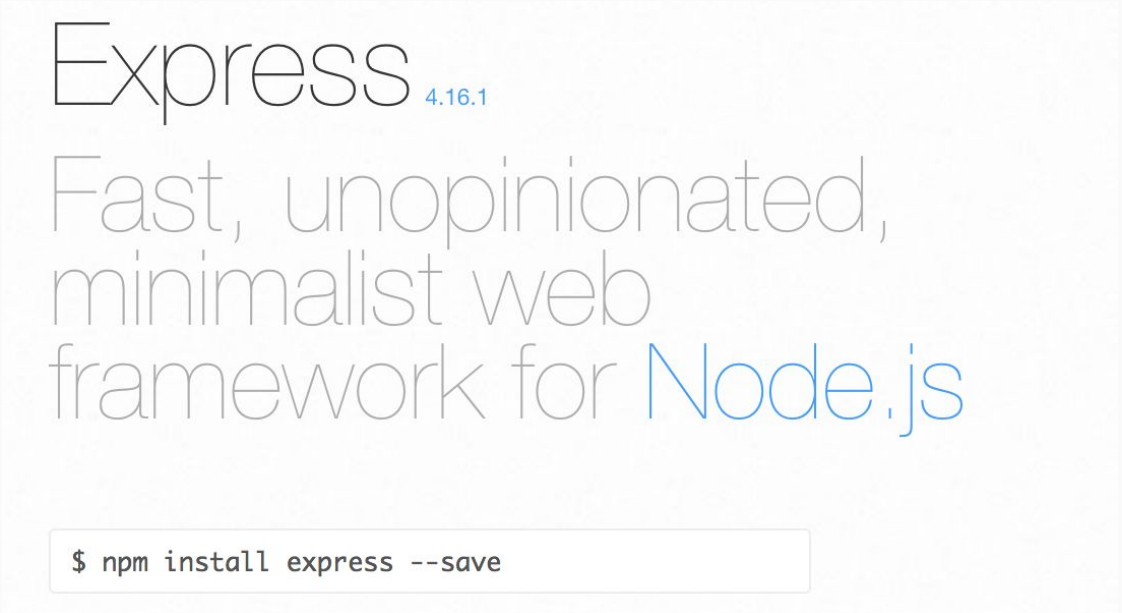
Templating with Express

Building Restful APIs with Express

What is Express?

What is Express?

- Express is a web framework for building web applications, APIs and services



- Official website: http://expressjs.com

How to write a basic HTTP server with Express?

Config is similar to other server modules like express.

The example on the right returns **'Hello from Express'** when a user navigates to **`localhost:3000`**

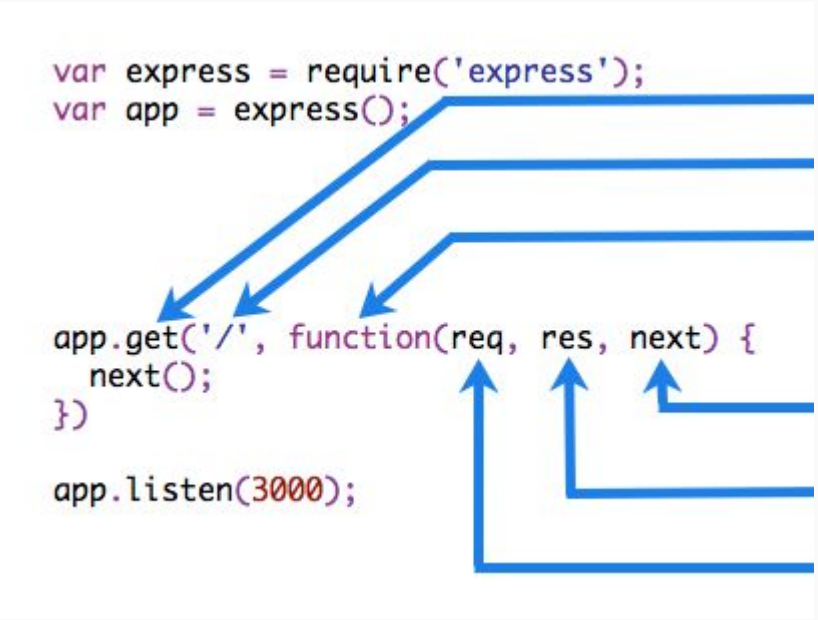
```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (request, resp) => {
    resp.send('<h1>Hello from Express!</h1>')
});

// We can serve static content too!
app.use('/data', express.static('content'));

app.listen(port, (err) => {
    console.log(`server listening on ${port}`)
});
```

Express is based on middleware functions



HTTP method for which the middleware function applies

Path (route) for which the middleware function applies

The middleware function

Callback argument to the next middleware function

HTTP response argument to the function

HTTP request argument to the function

`req`, **`res`**, and **`next`** are parameter names used by convention, though actual parameter names do not matter in javascript

Lab 1 - Basic HTTP server

- **Your mission:** Install **`express`** with npm. Create a new file **express-server.js** and use the **express** module to create a basic server that runs on port 8000. The server should work as follows:
  - When **`/hello`** is accessed, it should return **`Hello Express World!`**
  - When **`/data`** is accessed, it should return the contents of the file **`data/plates.json`**
  - Any other URL should return **`Nothing to see here`**
- **Hint:** You can use the **`fs`** module to read a file with Node.JS

What we just learnt

Express is a web framework for building web applications, APIs and services

Express uses a simple config that is easy to understand

Its official website is http://expressjs.com

Outline for today

What is Express?

Error handling

Integrating CORS with Express

Advanced Routing

Templating with Express

Building Restful APIs with Express

Error handling

How to log errors with Express?

- Express automatically handles errors thrown in your code by logging them and returning a HTTP 500 error along with the error stack
- It is possible to define our own error handler function, which has to be declared with 4 parameters as follows:

```
function logErrors(err ,req, res, next){
  console.error(err);
  res.status(500).send('Server error');
}
```

- Parameters: **`err`** is the error object, **`req`** is the request, **`res`** is the response, **`next`** is a function used to call the next error handler

Tweaking error handling in Express

- Several error handling functions can be defined and chained
- In Express, such functions are called **middleware** and rely on the **`next()`** parameter to let Express pass the error to the next handler (if any):

```
function logErrors(err ,req, res, next){
  console.error(err);
  next(err);
}
```

- The above function has to be registered with `app.use(logErrors)` just before starting the server, after all routes are defined

How to throw an error?

We can simply use the **`throw`** syntax to throw an error

Express error handler functions will then catch that error

Note that Express has a default error handler that returns a HTTP 500 error

```
app.get('error', (request, resp, next) => {
    throw new Error("Something bad happened");
});
```

Lab 2 - Error handling

- **Your mission:** Using the web server we wrote during lab #1, add custom error handling when a user tries to access an unknown URL:

  - First throw an error **“Unknown URL”** along with the path that lead to the error (for instance: **“Unknown URL: /app”**)
  - Then test that code by accessing a wrong URL in your browser
  - Change the server config to add a custom error handler function
  - The handler function should return a **404 HTTP error** with the error message `'Nothing to see here'` to the client

What we just learnt

Express has its own error handling mechanism that we can replace, customize and extend

Error handlers are registered as middleware functions with 4 parameters

Handlers can be chained so that each function can be as specific a needed (one handler for DB errors, 404 errors, etc.)

Outline for today

What is Express?

Error handling

Integrating CORS with Express

Advanced Routing

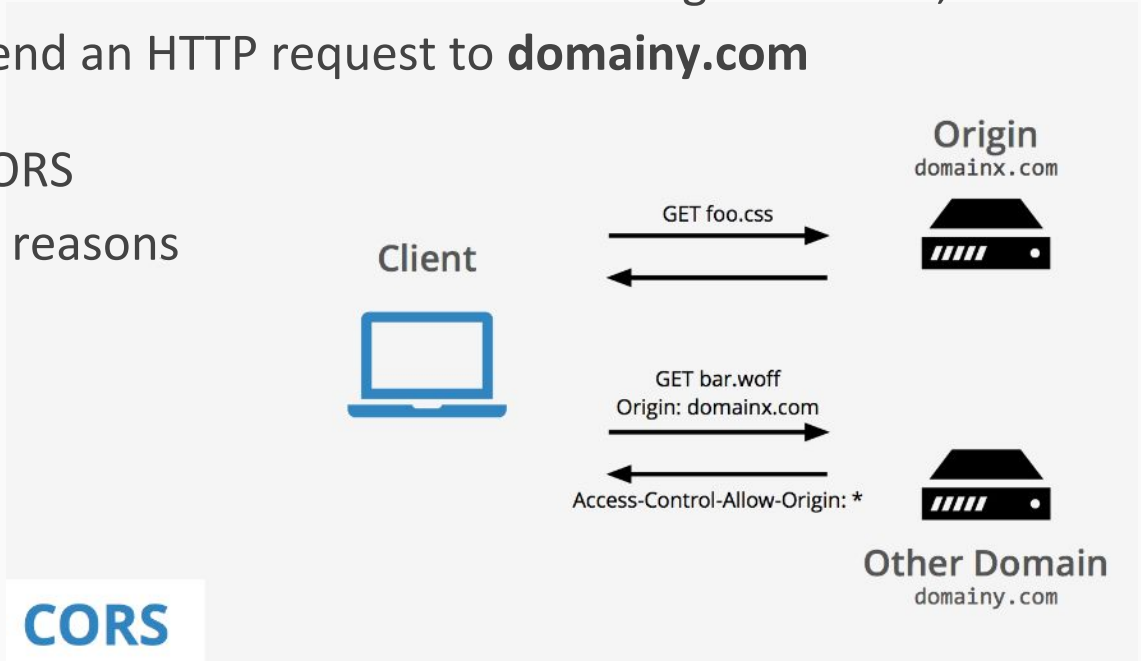
Templating with Express

Building Restful APIs with Express

Integrating CORS with Express

What is CORS?

- CORS stands for **C**ross **O**rigin **R**esource **S**haring
- When enabled, it allows browsers to make cross-origin API calls, i.e. **domainx.com** can send an HTTP request to **domainy.com**
- Browsers prevent CORS by default for safety reasons



How to enable CORS with Express?

CORS requires the use of a specific middleware library called…. **cors**

The middleware can be applied globally or for specific routes only

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

app.use(cors());

app.get('/products/:id', function (req, res, next) {
   res.json({msg: 'This is CORS-enabled for all origins!'})
})

app.listen(80, function () {
   console.log('CORS-enabled web server listening on port 80')
})
```

How to enable CORS for specific routes?

The **cors** function can also take an optional config as a parameter if we want to allow CORS for specific domains only

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

app.get('/products/:id', cors(), function
(req, res, next) {
   res.json({msg: 'This is CORS-enabled for
all origins!'})
})

app.listen(80, function () {
   console.log('CORS-enabled web server
listening on port 80')
})
```

Lab 3 - Enabling CORS

- Start a static **`http-server`** in the **lp-store-ui** directory. This will serve a store user interface on **`http://localhost:8080`**
- Open that page in your browser: The store is empty. That's because CORS isn't enabled in our Express server.
- **Your mission:** Using the web server we wrote during lab #2, change the config to enable CORS. Then you should be able to see data in our store after refreshing **`http://localhost:8080`**
- **Note:** CORS is required because localhost:**8000** and localhost:**8080** are different origins, even though they are two servers running on the same physical machine.

What we just learnt

CORS allows to enable Cross Domain requests

Express can enable CORS at the server config level as well as at the route level

Outline for today

What is Express?

Error handling

Integrating CORS with Express

Advanced Routing

Templating with Express

Building Restful APIs with Express

Advanced Routing

Express supports all HTTP requests

So far we've been using HTTP GET, but we can send data to the server using HTTP PUT and POST

```
// GET method route
app.get('/', function (req, res) {
   res.send('GET request')
})

// POST method route
app.post('/', function (req, res) {
   res.send('POST request')
})
```

How to extract request parameters?

Parameters have to be specified in the **path**, then extracted using **req.params**

The name defined in the **path** corresponds to the name of the parameter in **req.params**

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

Optional parameter

A question mark can be used to mark a parameter as optional

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

Route config with string patterns

Full routing guide: https://expressjs.com/en/guide/routing.html

This route path will match `acd` and `abcd`.

```
app.get('/ab?cd', function (req, res) {
  res.send('ab?cd')
})
```

This route path will match `abcd`, `abbcd`, `abbbcd`, and so on.

```
app.get('/ab+cd', function (req, res) {
  res.send('ab+cd')
})
```

This route path will match `abcd`, `abxcd`, `abRANDOMcd`, `ab123cd`, and so on.

```
app.get('/ab*cd', function (req, res) {
  res.send('ab*cd')
})
```

This route path will match `/abe` and `/abcde`.

```
app.get('/ab(cd)?e', function (req, res) {
  res.send('ab(cd)?e')
})
```

Route config with regular expressions

Full routing guide: https://expressjs.com/en/guide/routing.html

This route path will match anything with an “a” in the route name.

```
app.get(/a/, function (req, res) {
  res.send('/a/')
})
```

This route path will match `butterfly` and `dragonfly`, but not `butterflyman`, `dragonflyman`, and so on.

```
app.get(/.*fly$/, function (req, res) {
  res.send('/.*fly$/')
})
```

Lab 4 - Add an item to the cart

- In this lab, we're going to create a server endpoint that adds a new item to the cart of our store.
- **Your mission:** Using the web server we wrote during lab #3, create a new HTTP PUT endpoint **`/cart`** that takes an **`id`** as a parameter
  - The **`id`** has to be validated to make sure it comes from an actual item
  - If the **`id`** is valid, add the plate object to an array called **`cart`** and return a string **"{id} was added to the cart"** as a response to the client
- **Hint:** Create a **`plates`** object that contains the list of all license plates so that you can easily validate existence of the **`id`**

Lab 5 - See and edit cart contents

- In this lab, we're going to create a server endpoint to render cart items.
- **Your mission:** Using the web server we wrote during lab #4, create a new HTTP GET endpoint **`/cart`** that returns all cart items
  - Test your cart at http://localhost:8080/cart.html
  - Create a new HTTP DELETE endpoint **`/cart`** that takes an **`id`** as a parameter and removes an item from the **`cart`** array
- Now you can add, see and remove items from your cart!

What we just learnt

Express allows for further customization of route config

Express can handle parameters, HTTP headers, request payload, etc. before handling a request

Express supports all kinds of HTTP requests (GET, POST, PUT, DELETE)

Outline for today

What is Express?

Error handling

Integrating CORS with Express

Advanced Routing

Templating with Express

Building Restful APIs with Express

Templating with Express

Serving static files

The highlighted code allows to serve the contents of the **content** folder at the **/data** URL.

As a result, a user trying to access **/data/index.html** would get the **index.html** file located in the content folder

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (request, resp) => {
    resp.send('<h1>Hello from Express!</h1>')
});

// We can serve static content too!
app.use('/data', express.static('content'));

app.listen(port, (err) => {
    console.log(`server listening on ${port}`)
});
```

Custom templating engines

Express allows us to create our own template rendering engine

In our example, our template syntax uses expressions like this: **#message#**

Template stored in index.ntl:

```
<h1>#title#</h1>
<div>#message#</div>
```

Code to call our template rendering mechanism:

```
app.get('/', function (req, res) {
   res.render('index', {
        title: 'Hey',
        message: 'Hello there!'
        });
});
```

Custom templating engines

Our engine is registered for the **ntl** extension

All it does here is simple substitutions

Views are defined in their own folder. They are files with a **`.ntl`** extension, such as **`index.ntl`**

Template rendering engine:

```
app.engine('ntl', function (filePath, options, cb) {
  // define the template engine
    fs.readFile(filePath, function (err, content) {
        var rendered = content.toString()
                .replace('#title#', '<title>' +
          options.title + '</title>')
                    .replace('#message#', '<h1>' +
          options.message + '</h1>')
        return cb(null, rendered)
    })
})

// specify the views directory
app.set('views', './views')

// register the template engine
app.set('view engine', 'ntl')
```

Handlebars as a templating engine

Popular template engines such as Handlebars have their own Node modules that we can use out of the box.

All we have to do is register our template engine.

```
var express = require('express');
var exphbs  =
require('express-handlebars');

var app = express();

app.engine('handlebars',
exphbs({defaultLayout: 'main'}));

app.set('view engine', 'handlebars');

app.get('/', function (req, res) {
   res.render('home');
});

app.listen(3000);
```

Lab 6 - Serving static content

- In this lab, we're going to get rid of our static **`http-server`** and serve the store's HTML, Javascript and CSS directly from Express.
- **Your mission:** Using the web server we wrote during lab #5, add the required configuration to serve content from the **lp-store-ui** directory.
- Now we only have one server running on http://localhost:8000
- As a result, let's disable CORS since it's not needed anymore
- **Hint:** Use the sample config provided a few slides earlier

What we just learnt

Express allows to serve static files and can use template engines

Static content can be served using **`express.static`**

We can write our own rendering engines

Popular templating engines like **`handlebars`** can be used with Express

Outline for today

What is Express?

Error handling

Integrating CORS with Express

Advanced Routing

Templating with Express

Building Restful APIs with Express

Building Restful APIs with Express

What does Restful mean?

- **Rest** stand for **Re**presentational **S**tate **T**ransfer
- A Restful web service relies on HTTP methods to achieve different operations:
  - **`HTTP GET /plates`** returns the list of all plates => **GET = READ**
  - **`HTTP GET /plates/{id}`** returns a given plate
  - **`HTTP PUT /plates/{id}`** updates a given plate => **PUT = UPDATE**
  - **`HTTP POST /plates`** creates a new plate => **POST = CREATE**
  - **`HTTP DELETE /plates/{id}`** deletes a given plate => **DELETE = DELETE**
- Rest is a convention, not an obligation
- Data used by Restful services uses the JSON syntax (**J**ava**S**cript **O**bject **N**otation)

Our server is already Restful

But things can be improved: All of our code is in one single file right now, which is not scalable

We can move some routes to their own config file as shown here

/routes/index.js:

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

router.get('/', function(req, res, next) {
 res.render('index', { title: 'Express' });
});
module.exports = router;
```

/app.js:

```
var index = require('./routes/index');
var users = require('./routes/users');

var app = express();
//...
app.use('/', index);
app.use('/users', users);
```

Express Generator

Express-generator is a package that generates the default structure of any Express application

More information: https://expressjs.com/en/starter/generator.html

```
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

The generated app has the following directory structure

Lab 7 - Improving scalability of our Restful web service

- In this lab, we're going to refactor our cart routes into a specific **`cart-routing.js`** file. Our data models **`cart`** and **`plates`** will also get moved to a **`data-models.js`** file.
- **Your mission:** Using the web server we wrote during lab #6, create the files mentioned above and move the appropriate pieces of code to each file.
- Then include those files in your main server config file

Lab 8 - Using Express Generator to create a sample project

- In this lab, we're going to install **`express-generator`** globally with **`npm`**.
- **Your mission:** Install Express Generator and create a sample project:

```
express myapp
```

- **Hint:** Use the official documentation for assistance: https://expressjs.com/en/starter/generator.html

What we just learnt

Express allows us to structure our code in modules for more scalable Restful servers

Node modules can be used to cut our server into different pieces

`express-generator` can make that task even easier by generating a default project structure for us

Thanks for your attention

I need your feedback before you leave

In case you have any question:

al@interstate21.com