

Happy 30th Birthday World Wide Web!

Express.js

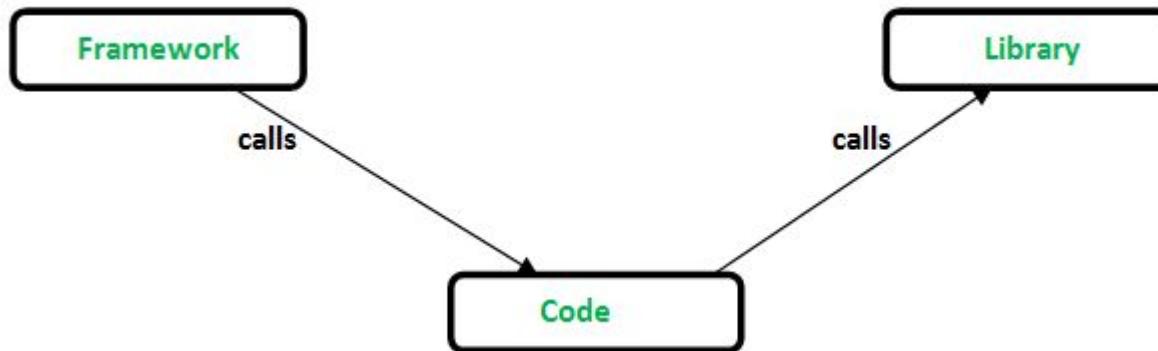
COEN 161

Libraries vs Frameworks

- Libraries
 - Provide helper functions/objects/modules to make coding easier and more reusable
 - Have a limited scope (e.g. Strings, DOM, etc.) making them less complex
 - Your code calls the library
- Framework
 - Provides open ended function definitions where you can write your own custom code
 - Can be made up of many libraries to extend functionality but increasing complexity
 - The framework later calls your code when it decides it is appropriate, this logic is usually built into the framework
 - This is called **inversion of control**

Libraries vs Frameworks

- With a library, we *control* the library functions from our code
- With a framework, the *framework* controls when your code is called



Technology Stacks - Then

L → Linux



A → Apache



M → MySQL

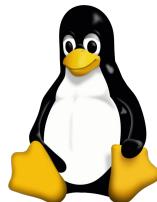


P → PHP



Technology Stacks - Then → Now

L → Linux



A → Apache



M → MySQL



P → PHP



M → MongoDB



E → Express.js



A → AngularJS



N → Node.js



Technology Stacks - Variations

M → MongoDB 

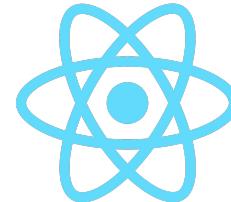
E → Express.js 

A → AngularJS 

N → Node.js 

M → MongoDB 

E → Express.js 

R → React.js 

N → Node.js 

Technology Stacks - More Variations

M → MongoDB



M → MySQL



E → Express.js



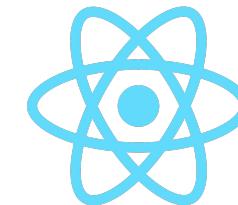
E → Express.js



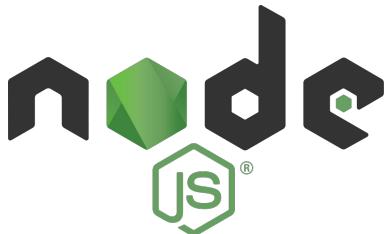
A → AngularJS



R → React.js



N → Node.js



N → Node.js



NPM Init

- The `init` command can be used to set up a new or existing npm package
- After answering a few questions, a new `package.json` file will be created

```
-bash-4.2$ npm init
package name: (upload_demo)
version: (1.0.0)
description: a demo of an upload form with Node.js
entry point: (step1.js)
test command:
git repository:
keywords:
author: <your email>
license: (ISC)
About to write to <path to your directory>/package.json:
```

NPM Init

```
{  
  "name": "upload_demo",  
  "version": "1.0.0",  
  "description": "a demo of an upload form with Node.js",  
  "main": "step1.js",  
  "dependencies": {  
    "formidable": "^1.2.1"  
  },  
  "devDependencies": {},  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "alberto.diaz.tostado@gmail.com",  
  "license": "ISC"  
}
```

- This file can be used to configure many things about your project including the name, version, and **dependencies!**

Express.js

- "Fast, unopinionated, minimalist web framework for Node.js"
- If you're using Node.js, you can use Express.js
- Start by creating a directory to hold your application

```
mkdir myapp
```

```
cd myapp
```

- Next, create a package.json file in that directory

```
npm init
```

Express.js

- Next, install express

```
npm install express --save
```

- Note: the --save option tells npm to add express to our dependencies list
- If you want to temporarily install express, use the --no-save option

```
npm install express --no-save
```

- At this point, our directory is ready to use express

Express.js Hello World

- Create a file called `app.js` (or whatever you decided to name your entrypoint) and add the following to it

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

app.get('/', (req, res) => res.send('Hello World!'));

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

Express.js Hello World

- Express runs on Node, so to start our Express server we use the same command we've been using to run our Node servers!

```
node app.js
```

- Our server runs and listens to port 3000 (<http://localhost:3000/>)
- It will only respond to **GET** requests for the path /
- All other requests will return a 404 Not Found

Express Routing

- *Routing* refers to determining how an application responds to a client request to a particular endpoint (URI path)
- In Express, each route can have one or more handler functions, which are executed when the route is matched

`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.

Express Routing

- GET

```
app.get('/', function (req, res) {  
  res.send('Hello World!');  
});
```

- POST

```
app.post('/', function (req, res) {  
  res.send('Got a POST request');  
});
```

Express Routing

- PUT

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

- DELETE

```
app.delete('/user', function (req, res) {  
  res.send('Got a DELETE request at /user');  
});
```

Static Files in Express

- In Node, we had to leverage the `fs` module to serve static files such as images, CSS files, and JavaScript files
- In Express, we can use the `express.static` built-in middleware function
`express.static(root, [options]);`
- The `root` argument specifies the root directory from which to serve static assets
- For example, the following serves static files from a directory called `public`
`app.use(express.static('public'));`

Static Files in Express

- Express looks up the files relative to the static directory, so the name of the static directory is not part of the URL.

`http://localhost:3000/images/kitten.jpg`

`http://localhost:3000/css/style.css`

`http://localhost:3000/js/app.js`

`http://localhost:3000/images/bg.png`

`http://localhost:3000/hello.html`

Static Files in Express

- To use multiple static assets directories, call the `express.static` middleware function multiple times
- Express looks up the files in the order in which you set the static directories with the `express.static` middleware function

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

Express Middleware

- "Express is a routing and middleware web framework that has minimal functionality of its own"
- *Middleware* functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle
- The next middleware function is commonly denoted by a variable named next

Express Middleware

- 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.
- 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

Express Middleware

- An Express application can use the following types of middleware:
 - Application-level middleware*
 - Router-level middleware
 - Error-handling middleware
 - Built-in middleware
 - Third-party middleware*

Application-level middleware

- You can "bind" application-level middleware to an instance of the app object by using the app.use() and app.METHOD() functions

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

```
app.use(function (req, res, next) {  
  console.log('Time:', Date.now()); // log the current time  
  next(); // pass control to the next middleware  
});
```

Application-level middleware

- Middleware can also be mounted for a specific path, so that it is called for any requests for that path

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

- In fact, if you want to specify a particular method for a path, you can use the `app.METHOD()`, which takes a handler that is also considered middleware

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

Application-level middleware

- This example shows multiple routes being defined, but the second route will never get called because the first does not call next()

```
app.get('/user/:id', function (req, res, next) {  
  res.send('User Info');  
});
```

```
// handler for the /user/:id path, which prints the user ID  
app.get('/user/:id', function (req, res, next) {  
  res.end(req.params.id);  
});
```

Third-party middleware

- Use third-party middleware to add functionality to Express apps
- Install the Node.js module for the required functionality

```
npm install cookie-parser
```

- Then load it in your app at the application level

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

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

Third-party middleware

- The [cookie-session](#) middleware provides a lightweight client-side cookie session implementation

```
var cookieSession = require('cookie-session');
var express = require('express');
var app = express();

app.use(cookieSession({
  name: 'session',
  keys: ['key1', 'key2']
}));
```

Third-party middleware

- The `req.session` object contains the session cookie
- Set any property on this object to send a `Set-Cookie` header back in the response
- This example creates a simple, session based, view counter

```
app.get('/', function (req, res, next) {
  // Update views
  req.session.views = (req.session.views || 0) + 1

  // Write response
  res.end(req.session.views + ' views')
})

app.listen(3000)
```

Resources

<https://expressjs.com/>

<https://expressjs.com/en/guide/using-middleware.html>

<http://mean.io/>