web programming

intro to node.js


oli


rec

agenda

1. intro to node.js
2. modules
3. project structure
4. intro to express

1. intro to node.js

what's node.js?

- platform to execute JavaScript
  (outside a browser)

- built on top of JavaScript V8
  (JavaScript engine that powers Google Chrome)

- uses a non-blocking I/O model
  (event-driven, perfect for real-time applications)

what's node.js?

- its code is absolutely open-source
  (you can find and collaborate with it in GitHub)

- written in C++ and JavaScript
  (most of its codebase is written in JavaScript)

node vs. browser

browser

`window` global variable

node.js

`process` global variable

start a node process

```
$ node
```

start a node process

```
$ node <file-name.js>
```

node *hello-world.js*

```
setTimeout(function() {
  console.log('world');
}, 2000);

console.log('Hello');
```

php alternative

```
echo('Hello');
sleep(2);
echo('world');
```

“Node never sleeps.”

–Ryan Dahl

2. modules

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

require

- module system for referencing to other files

- these files can be located:
  - node's standard library
  - relative to our codebase
  - third-party, via package manager

require (std lib)

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

const onDone = function() {
  console.log('Success!');
};

fs.writeFile('./test.txt', '😊', 'utf8', onDone);
```

require (relative)

```
// add.js
module.exports = function(a, b) {
   return a + b;
};

// index.js
const add = require('./add')

add(1, 2)
```

require (3rd-party)

```
// we need to install it first!
// npm install --save express

const express = require('express')

const app = express()
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

Added in: v0.1.22

- `<Object>`

A collection of all the standard HTTP response status codes, and the short description of each. For example, `http.STATUS_CODES[404] === 'Not Found'`.

http.createServer([options][, requestListener]) [src]

▶ History

- `options` `<Object>`
  - `IncomingMessage` `<http.IncomingMessage>` Specifies the `IncomingMessage` class to be used. Useful for extending the original `IncomingMessage`. **Default:** `IncomingMessage`.
  - `ServerResponse` `<http.ServerResponse>` Specifies the `ServerResponse` class to be used. Useful for extending the original `ServerResponse`. **Default:** `ServerResponse`.
- `requestListener` `<Function>`
- Returns: `<http.Server>`

Returns a new instance of `http.Server`.

The `requestListener` is a function which is automatically added to the `'request'` event.

http.get(options[, callback]) [src]

http.get(url[, options][, callback]) [src]

▶ History

- `url` `<string>` | `<URL>`
- `options` `<Object>` Accepts the same `options` as `http.request()`, with the `method` always set to `GET`. Properties that are inherited from the prototype

Go to https://nodejs.org/api for more documentation

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Hello world');
});

server.listen(8000);
```

```
const http = require(‘http');
let serverCounter = 0;

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.end('Visitor number' + (++serverCounter));
});

server.listen(8000);
```

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

const server = http.createServer(function(req, res) {
  res.writeHead(200, {
    'content-type': 'text/plain'
  });

  res.write(‘hello');

  setTimeout(function() {
    res.end(‘world’);
  }, 2000);
});

server.listen(8000);
```

3. project structure

dividing by *envs*

client

sub-project for the web interface

server

our abstraction layer for the data

dividing by *envs*

more specific

web

sub-project for the web interface

api

our abstraction layer for the data

EXERCISE

Go to your project and follow the instruction on the first issue

```
$ npm init
```

This utility will walk you through
creating a **package.json** file.
It only covers the most common items,
and tries to guess sensible defaults.

See `**npm help json**` for definitive
documentation on these fields
and exactly what they do.

Use `**npm install <pkg>**` afterwards to
install a package and
save it as a dependency in the
package.json file.

```
Press ^C at any time to quit.
package name: (<…>)
version: (1.0.0)
description:
git repository:
keywords:
author:
license: (ISC)
About to write to <…>:
```

REMEMBER

The package.json from LU-0

package.json

```
{
  "name": "web-programming-boilerplate",
  "scripts": {
    "lint:js": "eslint .",
    "lint:css": "stylelint **/*.css",
    "lint": "npm run lint:js && npm run lint:css",
    "pretest": "npm run lint",
    "test": "jest --passWithNoTests"
  },
  "devDependencies": {
    "@ucudal/eslint-config": "...",
    "@ucudal/stylelint-config": "...",
    ...
  }
}
```

package.json

```
{
  "name": "web-programming-boilerplate",
  "scripts": {
    "lint:js": "eslint .",
    "lint:css": "stylelint **/*.css",
    "lint": "npm run lint:js && npm run lint:css",
    "pretest": "npm run lint",
    "test": "jest --passWithNoTests"
  },
  "devDependencies": {
    "@ucudal/eslint-config": "...",
    "@ucudal/stylelint-config": "...",
    ...
  }
}
```

the next part is for the API

```
$ cd api
```

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

EXERCISE

Intialize the api directory

api/index.js

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

const app = express();

app.get('/products', function(req, res) {
  return res.json([
    {
      _id: '1',
      name: 'Thing A'
    }, {
      _id: '2',
      name: 'Thing B'
    },
  ]);
});

app.listen(process.env.PORT || 8000);
```

api/package.json

```
{
  …,
  "scripts": {
    "start": “node index.js”,
    …
  },
  …,
}
```

```
$ npm start
```

3. intro to express

what's express?

- it's one out of many node.js **web frameworks**.
- widely used and has been around for years.
- allows to create **APIs** or **server pages** and **static files**.

what's express?

- has a **routing** mechanism that maps path patterns and http methods.
- it allows you to specify how to render **templates**, if you need to do so.

an express server

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

const app = express();

app.listen(8000);
```

an express server

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

const app = express();   our express instance

app.listen(8000);
```

an express server

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

const app = express();

app.listen(8000);
```

port we are listening

an express route

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

const app = express();

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

app.listen(8000);
```

```
$ curl -i localhost:8000
```

an express route

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

const app = express();

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

app.listen(8000);
```

http method

an express route

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

const app = express();

path (route)
app.get('/', function(req, res) {
   res.send(‘Hello!');
});

app.listen(8000);
```

an express route

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

const app = express();
```

handler

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

app.listen(8000);
```

an express response

could be:

- **simple text**
  `res.send('text')`

- **json**
  `res.json({ myObj: true })`

- **template rendering**
  `res.render('pathToTemplate')`

template rendering

- express doesn't have a built-in template engine.
- instead, it allows you to install whatever fits your needs.

template rendering

- by default, the template files are allocated in a `/views` directory, that means:

```
/<root>
 package.json
 index.js
 /views
   my-template.html
```

template rendering

- the idea to use templates is to provide a context to the response.

```
res.render('index', {
   user: {
      name: 'McLovin'
   }
});
```

<EXERCISE>

Make a multi-page web app in express

- Express serving to a port of choice.
- Resolves the index (`/`) to a HTML with a form.
- Submits the form to `/response` showing the content typed by the user.
- Use `POST` for the form method.

index.js (v0)

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

const app = express();

app.get('/', function(req, res) {
  // code for rendering the HTML.
});

app.listen(8000);
```

index.js (v0)

- has express running in port **8000**.
- handles the **get** to the **index**, but is missing to show the html we need.

```
$ npm i --save nunjucks express-nunjucks
```

index.js (v1)

```
const express = require('express');
const expressNunjucks = require('express-nunjucks');

const app = express();

expressNunjucks(app);

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

app.listen(8000);
```

index.js (`v1`)

- has a **template engine** (nunjucks).
- **renders** an `index.html` from `./views/index.html` for responding to the index.

index.html (v1)

```
<html>
  <head>
    <title>WebProgramming - LU-3</title>
  </head>
  <body>
    <h1>WebProgramming - LU3</h1>
    <form action="/response">
      <input type="text" name="text" />
      <input type="submit" />
    </form>
  </body>
</html>
```

index.html (`v1`)

- has a **form** that redirects its action to `/response`.
- as the **method** for the form is not provided, the default is used, which is **GET**.

index.js (v2)

```
const express = require('express');
const expressNunjucks = require('express-nunjucks');

const app = express();

expressNunjucks(app);

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

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

app.listen(8000);
```

index.js (v2)

- has a **handler** for `/response`.
- sends all the parsed **query string** to its template context.

response.html (v2)

```
<html>
  <head>
    <title>WebProgramming - LU-3 Response</title>
  </head>
  <body>
    <h1>Response</h1>
    <p>{{ text }}</p>
  </body>
</html>
```

response.html (v2)

- display the `text` attribute from the **context**.

```
$ npm i --save body-parser
```

index.js (v3)

```
const express = require(‘express');
const bodyParser = require(‘body-parser');
const expressNunjucks = require('express-nunjucks');

const app = express();

app.use(bodyParser.urlencoded());

expressNunjucks(app);

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

app.post('/response', function(req, res) {
  res.render('response', req.body);
});

app.listen(8000);
```

index.js (v3)

- uses `body-parser` to **parse the payload** from the POST request.

- change the **http method** on the handler for `/response` to **POST**.

- send the parsed payload to the response in the **context**.

index.html (v3)

```
<html>
  <head>
    <title>WebProgramming - LU-3</title>
  </head>
  <body>
    <h1>WebProgramming - LU3</h1>
    <form action=“/response" method="POST">
      <input type="text" name="text" />
      <input type="submit" />
    </form>
  </body>
</html>
```

index.html (v3)

- change the **method** for the form to **POST**.

</EXERCISE>

Make a multi-page web app in express

thanks!