Nodeschool Toronto

# Building an HTTP Server with Node

# What is an HTTP Server?

- Servers respond to requests from clients
- clients and servers communicate via HTTP
- a request consists of a method and a URL and optionally more data
- anything that makes requests to a server is called a client
  - browser, mobile app, cURL, another server
- server has software that determines how to respond to a request

# Servers then and now

- 90's servers: respond to a request from a browser for a webpage with an HTML document
  serve static files (CSS, images, scripts) included by that HTML document
- Today's servers: respond to requests from anywhere with static files OR
  - dynamic HTML, JSON, XML, and more

## Example

- Web Browser makes a request to nodeschool.io
- (DNS routing takes place)
- Server receives GET request for “/“ URL
- Static file server automatically sends the index.html file located at its root directory, along with a 200 status code (OK)

## Example

- Web Browser makes a request to github.com/nodeschool
- Server receives GET request for ‘/nodeschool’ URL
- Software running on the server:
  - determines that the requested resource is a user with the name “nodeschool”
  - Looks in the database for the user with the name “nodeschool”
  - Finds the template associated with displaying a user
  - Uses info from the database to fill in the blanks in the template
  - Sends the resulting HTML back to the client with a 200 status code (OK)

## Example

- A script running on nytimes.com makes an AJAX request to nytimes.com/services/weather?location=toronto
- Server receives a GET request for /weather with the parameters {location: ‘toronto’} and a header of `Content-type: application/json`
- Software running on server:
  - looks up weather for location “toronto”
  - Formats response as JSON
  - Sends JSON back to client
- script uses data from JSON response to update a <p> tag on the page with the new information

## Example

- Web browser makes a request to shopify.com/asldksj222
- Server receives GET request for /asldksj222
- Software on server:
  - tries to match /asldksj222 to something it has instructions for
  - finds no match
  - responds with the file 404.html and status code 404 (not found)

# Let’s make a node http server

- We’re building the software that receives an HTTP request, determines what to do about it, and sends back a response to a client
- Server software can be written in any language that the server knows how to run
- We’re going to use javascript, so we’ll need a server that runs node

# Getting set up

- make a new flie server.js
- import the `http` module

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

# Creating a Server

- **use the** `http.createServer` **method to create a server**
- **call** `server.listen` **to start the server listening on a specific port.**

```
var server = http.createServer();

server.listen(8080, function () {
  console.log('server is listening on port 8080');
});
```

- **run** `node server.js` **to start the server**
- **our server doesn't actually *do* anything when it gets a request**

# Listening for requests

— **the** `server` **is an instance of** `EventEmitter`**, meaning it has an** `on` **method for listening for events**

```
server.on('request', function (request, response) {
  console.log(request);
});
```

— **visit localhost:8080 in your browser**

# "Hello World"

- **We can add content to our** `response` **object with** `response.write`
- **when we're ready to send our response back, use** `response.end()`

```
server.on('request', function (request, response) {
  response.write('Hello world');
  response.end();
});
```

# Error handling

```
request.on('error', function(err) {
  console.error(err);
});

response.on('error', function(err) {
  console.error(err);
});
```

# Responding with static files

- create an index.html file with some arbitrary content
- use the `fs` module to read the file and write it to the repsonse object

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

server.on('request', function (request, response) {
  fs.readFile('index.html', function (error, contents) {
    response.write(contents);
    response.end();
  });
});
```

# Some basic routing

— **we can do different things based on the requested URL by comparing the `request.url` string**

```
server.on('request', function (request, response) {
  if (request.url === '/home') {
    fs.readFile('index.html', function (error, contents) {
      response.write(contents);
      response.end();
    });
  } else {
    response.write('not found');
    response.statusCode = 404;
    response.end();
  }
});
```

# Dynamic Routing

- Web servers often have "routers", which determine what behaviours to run depending on what URL was entered.
- We can capture he URL or parts of it and use that value as inputs for our software
- think of a URL like a function call: /users/123 could read as `getUser(123)` where `123` is a user ID.
- we can do dynamic routing more easily using a regular expression

```
var re = /^\/currencies\/(\w+)/;
```

This regex will match URLs with the format /currencies/[string] and will remember the value of [string]

# Dynamic Routing cont'd

- **Say we have an object that stores the currency symbol for countries:**

```
var currencies = {
  'CAN': 'CAD',
  'USA': 'USD',
  'FRA': 'EUR',
}
```

- **We want to return the currency name for the country code in the URL**

— **the captured string from the regex is in the return value from** `string.match`

```
var match = request.url.match(re);

if (match) {
  var currency = currencies[match[1]];
  response.write(currency);
  response.end();
} else {
  response.write('not found');
  response.statusCode = 404;
  response.end();
}
```

# Dynamic HTML

— We can combine dynamic routing + file rendering

```
if (match) {
  fs.readFile('./currency.html', 'utf-8', function (error, contents) {
    var output = contents.replace('$contents', currencies[match[1]]);
    response.write(output, 'utf-8');
    response.end();
  });
} else {
```

# Connecting to an API

— **we can bring in the `request` module to make HTTP requests *from* our server**

```
var request = require('request');

function getLiveValue(symbol, callback) {
  request.get('http://api.fixer.io/latest?base=USD', function (error, response, body) {
    var rates = JSON.parse(body).rates;
    callback(rates[symbol]);
  });
}
```

# Returning HTML

```
getLiveValue(currency, function (rate) {
  fs.readFile('./currency.html', 'utf-8', function (error, contents) {
    var string = `The currency for ${country} is ${currency}. <br> 1 USD = ${rate} ${currency}`;
    var output = contents.replace('$contents', string);
    response.write(output, 'utf-8');
    response.end();
  });
});
```

# Returning JSON

```
getLiveValue(currency, function (rate) {
  var output = JSON.stringify({
    currency: currency,
    rate: rate,
  });
  response.setHeader('Content-Type', 'application/json');
  response.write(output, 'utf-8');
  response.end();
});
```