Lecture 6: Fundamentals of Web Development

What is the web, and how does it work?

The Core Process of the Web

At the end of the day, the web is all about the communication of ideas. Everything in the web can be seen as a **request** and a **response**

- When you go to a news website, you're requesting news and receiving news in response.
- When you go to a shopping website, you're requesting product information and receiving relevant information.
- When your server receives input, it determines what to do with that input and outputs the proper response.

Your duty as a web developer is to make that communication possible. Your programs will get a request and give a response, and allow that communication to occur as smoothly as possible.

The request

Every time you navigate to a website, your browser sends a **request** on your behalf to the server.

- There is a lot to an HTTP request! https://developer.mozilla.org/en-US/docs/Web/HTTP
- Each request follows a standardized process! https://www.w3.org/wiki/How_does_the_Internet_work

Every request is formatted in a specific way, with the same data provided on each request.

A request to http://google.com/ would look like:

```
GET /?gws_rd=ssl HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: PREF=ID=1111111111111111:FF=0:TM=1441084937:LM=1441084937:V=1:S=DKD8wPI-NAGQztx5; NID=71=0zCx
Connection: keep-alive
Cache-Control: max-age=0
```

Parts of that request

HTTP Verb signifying what action you are trying to (GET POST PUT DELETE)

The server you want to connect to (the HOST)

The location of the resource you want to access on that server (the location)

Headers

- Headers are metadata about your request
- These include cookies!

What does the server do with that info?

The server reads that request and determines what needs to be done in order to generate a response that makes sense for the data that the server has been given.

The server uses data sent in the request in order to generate a response.

Some common types of data in a request that servers use are:

- Querystring params
- Headers
  - Cookies
- Request body

The response

The server sends back a **response** that is similar to the request. It contains:

- A status code
  - Indicates whether or not the operation succeeded
- A set of headers
  - Cookies, data about the response such as content type
  - This is often “meta-data” about the response.
- Some form of response body:
  - An HTML Document
  - A JSON response
  - A File Stream
  - Plain Text
  - Etc.

```
HTTP/2.0 200 OK
Cache-Control: private, max-age=0
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Tue, 01 Sep 2015 05:31:36 GMT
Expires: -1
Server: gws
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Firefox-Spdy: h2
```

Status Codes

Each response must return a status code indicating whether or not the request was successful, and a description is often included with each of the status code

Status codes in the...

- 200-299 range indicate a successful operation
- 300-399 range indicate some sort of redirection must occur
- 400-499 range indicate an error was made by the client during the request
- 500-599 range indicate some sort of error occurred on the server

You will use different status codes to describe different errors in this course

Some status codes:

- https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

The Browser

There are many different browsers, but they each allow for the same fundamental actions to occur:

- They allow a user to navigate to a url
- They then submit a **request** on the user's behalf to the server at that url
- They receive a **response** back from the server
- They render these responses
- They execute any code that they may have received in these responses, for the lifetime of the user being on the web page.

More than just web pages

There is a fundamental idea you must hold onto from now on:

- Your server is **not** a tool to transmit a *web page* to a user
- Your server is a tool to transmit information in response to a request

Web pages are a very small part of the internet. As a web developer:

- You will transmit text that represents data in JSON format
- You will transmit text that represents an HTML document
- You will transmit text that represents an XML document
- You will transmit binary data
- You will transmit media files

None of these require a browser!

You will often write programs that make requests and rely on responses for data!

JSON?

JSON is JavaScript Object Notation, a very popular way for representing data.

Many technologies in the modern era can easily communicate with each other by using JSON as a common way of representing data between them.

```
{
  'name': 'Philip Barresi',
  'age': 23,
  'favorite_drink': 'coffee',
  'hobbies': [
    'programming',
    'reading',
    'hiking'
  ]
}
```

Running a Node.js server

Express

Express is a very popular node package that is distributed on NPM which allows you to configure and run an entire web server.

- http://expressjs.com/

Express allows you to configure different routes and how they should compose a response.

Essentially, by using the Express module, you will use code to configure a server that will listen to requests and send out responses.

- **This is all a web server is! It's not magic, it's just something that takes in requests and sends back responses!**

What is a route?

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

- http://expressjs.com/en/starter/basic-routing.html

For your web applications, you will configure many routes that will each perform some action

By configuring routes, you can have the same URL perform different tasks based on having different request methods (HTTP Verbs)

Request Methods

There are many different types of request methods.

For this course, you will be using 4:

- GET
- POST
- PUT
- DELETE

Other methods:

- https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods

GET Requests

Let us use a blog as an example.

GET requests are made when the client is requesting the representation of a resource

Location	Client is requesting…	Server is responding with…
http://myblog.com/post/2/	The blog post with an id of *2*	An HTML document with the content of the blog post
http://myblog.com/	The home page of the blog	An HTML document the most recent posts
http://myblog.com/editor/	A page with an editor to write new blog posts	An HTML document with an editor to write posts in
http://myblog.com/post/2.json	The blog post with an id of *2*	A JSON document representing the content of a blog post

POST Requests

POST requests are made when the client is requesting to create some form of resource.

POST data comes with a message body that describes the content

Location	Client is requesting…	Server is responding with…
http://myblog.com/post/	That a blog post be created with	An HTML document with the content of the blog post
http://myblog.com/	The home page of the blog	An HTML document the most recent posts
http://myblog.com/editor/	A page with an editor to write new blog posts	An HTML document with an editor to write posts in
http://myblog.com/post/2.json	The blog post with an id of *2*	A JSON document representing the content of a blog post

Making route modules

If you open *routes/posts.js*, you will see a router being created, with three routes setup; one to get a list of all posts, one to create a new post, and one to get a specific post.

A router is a set of rules that dictate how to respond to requests with particular paths and HTTP verbs.

General Organization

When running a server, running *npm start* should start your server and print a message with its address (including port).

You will have:

- A file that creates, configures, and runs your server (app.js)
- A folder with all your route modules (./routes)
  - An index file in the route folder that returns a function that attaches all your routes to your app (./routes/index.js)
  - Route modules (./routes/posts.js)
- A folder for your data access layer modules (./data)
  - Your connection (./data/mongoConnection.js)
  - Your collection file (./data/mongoCollection.js)
  - Your data modules (./data/posts.js)
  - Potentially, an index file that exports all other data access modules

Requiring a folder

You may require a folder by placing a file called **index.js** inside the folder.

This is useful for organizing things, such as defining all your routes in a *routes* folder and all your data modules in a *data* folder.

Node.js as an API (GET)

The purpose

The purpose of our first servers will be simple: to create an API (Application Program Interface)

- An API is a way to interact with a program.

We will begin with treating our servers as an access point to run our applications.

Our first server will allow us to **read** some blog posts that will be stored in MongoDB.

Seeding our database

“Seeding” a database means adding initial data.

By running *npm run seed* we can run a task we’ve defined in our package.json file, which will run a script that will seed our database with a single user and a single post.

Request data

There is a great deal of data available in a request! For now, we will be focusing on requests and only be making simple responses.

In our blog, we will be using request parameters

- Request parameters are dynamic parameters we define in our route.

You should read up on the API for request data

- http://expressjs.com/en/api.html#req

Sending a response

For now, we will only be using two response methods:

- *res.json(someData)* will send a JSON object as the response with a status code of 200 (unless otherwise specified)
- *res.sendStatus(statusCode)* will issue a response with the provided status code