CWEB280

# Setup For Development

1. Get the latest version of nodejs
   1. ~~I installed~~ [~~https://github.com/coreybutler/nvm-windows/releases/download/1.1.7/nvm-setup.zip~~](https://github.com/coreybutler/nvm-windows/releases/download/1.1.7/nvm-setup.zip)
   2. ~~then ran the follwoing from command prompt:~~

~~nvm install latest~~

**https://nodejs.org/en/download/**

1. Get the latest  version of Webstorm (<https://download.jetbrains.com/webstorm/WebStorm-2021.2.1.exe>)
2. Install the latest version of postman (<https://www.postman.com/downloads/>)
3. Create a folder **cweb2021** somewhere (mine is on the root of my D drive)
4. Open a new command prompt then navigated to the new **cweb2021** folder and run the following commands from the command prompt

npx express-generator --view=hbs lo1serveronly -f

cd lo1serveronly

npm install

npm audit fix --force

npm install bootstrap bootswatch portfinder

## To run the project from command prompt:

npm start

## To run the project in WebStorm

1. Open the cweb2021/lo1serveronly folder in WebStorm as an existing project
2. Once the project is open click “Add Configurations…” near top right
3. In the Run/Debug Configurations window click the plus sign next to the top left
4. Select “npm” from the list of available configurations (left window area)
5. Then on the right window area find the Command (drop down) and select “start”
6. Click the ok button
7. You should see “start” where “Add Configurations…” used to be.
8. Then click the green play button immediately to the right of “start”

## To view the running project :

Open a web browser and navigate to <http://localhost:3000/>

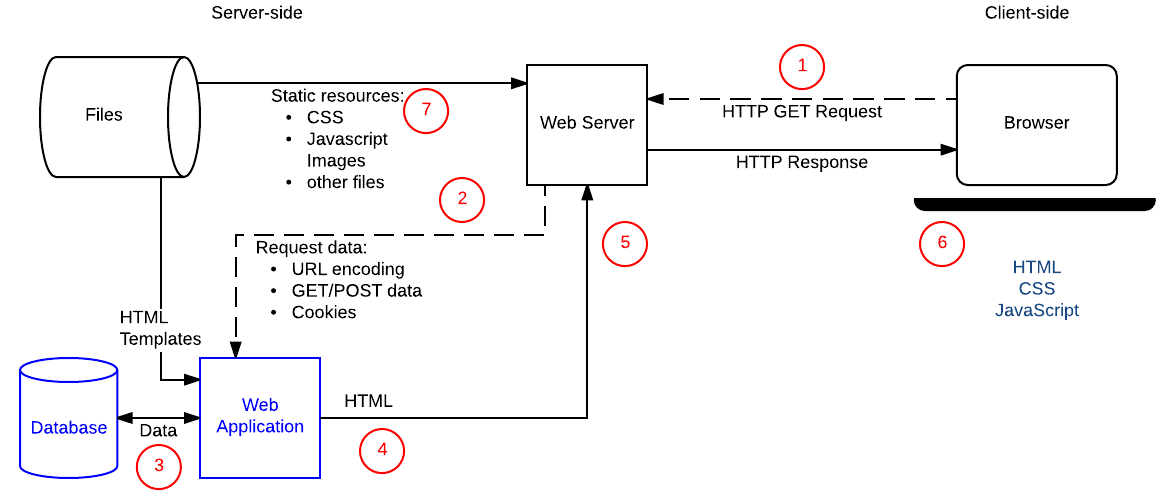
# Client side versus Server Side execution:

Last year you learned about HTML, CSS and JavaScript. It is important to understand the web browser ( Chrome, Fire Fox, Edge) did all work to render the webpage from the CSS and HTML you wrote. The browser also executed the JavaScript on the webpage as well. We will think of the browser running on your local computer as the client. In this class all the HTML, CSS and JavaScript running on the browser is called **Client-side execution**.

When we think about using the internet, we usually think about going to a web site. The web site is hosted on a server, and we must use a particular website address (aka URL) in the browsers address bar to access the website server. The website server sends HTML, CSS and JavaScript to the client browser and then the browser renders the page and executes the JavaScript.

Websites can have a collection of static webpages, but most websites provide a service and allow for users to dynamically interact with the server through the rendered webpage

Let’s take the example of searching for and buying an item off Amazon. There needs to be code that executes on the server that searches a database for the items you are looking for. Then there is additional code on the server that dynamically generates HTML/JavaScript that contains a list of found items’ information. We will call this **Server-side execution**. The dynamically generated HTML/JavaScript is sent the client browser, but the client is completely unaware of all the code that executed on the server. As far as the browser is concerned, it just received static HTML/JavaScript from the server.



Learn More about Server- Side : <https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction>

# Server-side Languages and Frameworks :

Learn More: <https://www.bairesdev.com/blog/top-languages-server-side-scripting/>

There are several programming languages and frameworks used on web servers. The most common are PHP, Node.js, Python, Ruby and Java.

# Node.js

Learn More: <https://nodejs.org/en/docs/guides/>

Node.js is a technology that allows JavaScript to execute on a server. Web servers do NOT have browsers running on them, but the JavaScript engine developed in Chrome was separated and re-purposed to execute JavaScript on web servers.

# Node Package Manager

Find Packages from the most popular repository of npm packages: <https://www.npmjs.com/>

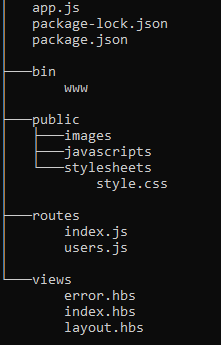
A very important part of Node development is being able use code/packages made by other developers and share code you whish to contribute. Nodejs includes a utility called “npm” that can download and install packages shared by an enormous community of other developers. When we create new development projects, we will often use npm to download and install packages to the project that will give us a huge head start, potentially saving us many hours development.

# Express Framework

Learn More: <https://expressjs.com/> - <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs#guides>

Originally writing JavaScript code to generate webpages was quite tedious. Some developers got together to develop a minimal framework that would make developing webpages easier for other developers. Express is one of many web frameworks that set up a structure (set of guidelines) for Node.js developers.

## New Project Example Directory/Folder Structure

Express includes a command line tool used to generate a very basic working app to build on. We use the “Express application generator” to create a new project with the directory structure shown here.

* The package.json and package-lock.json are used by npm to keep track of which packages are installed in the project. Npm will create a new folder called node\_modules where the package files and folders will be downloaded/stored.
* app.js is the main file that connects all the dependencies from the node\_modules
* bin/www is a script file that contains a script to configure and start the webserver
* The routes directory contain files that determine the valid paths (or web addresses) available on the web site. The path or address used will determine what code will execute or which file will be loaded. Think of then traffic cops for the website.
* The View folder contains templates which are most similar to regular html pages, but use a server-side script code inline with regular html. There may inline script syntax that are supported the most popular are pug and moustache- (Handlebars or “hbs” is more powerful version of moustache)

# Use Bootswatch theme In the project

When you created the project, you have installed bootswatch which is bootstrap theme library to make use of the new themes we need to add a path to the app.js to look in folders for the downloaded CSS files

**\app.js – around line 25 add the lines in yellow**

app.use('/', indexRouter)

app.use('/users', usersRouter)

// add the following line to point to the folder that contains the bootswatch css files

app.use('/bw', express.static(\_\_dirname + '/node\_modules/bootswatch/dist'))

The layout.hbs file is a special template that wraps all other templates – so adding html here adds the same html to all other templates.

**\views\layout.hbs** – change the href for the stylesheet and add a bootstrap container **AROUND** the {{body}} place holder

<!DOCTYPE html>

<html>

<head>

<title>{{title}}</title>

<!--GO to https://bootswatch.com/ to see all the other themes installed in this project -->

<link rel='stylesheet' href='/bw/quartz/bootstrap.css' /><!-— currently using the quartz theme – change it you like -->

</head>

<body>

<div class="container">

{{{body}}}

</div>

</body>

</html>

Restart the webserver for the changes to take effect

# Add a New Route File to the project

To add new webpages to the website we start by adding routes to a new rout file or an existing route file. We will be creating a new route file that will handle serving up all the examples.

In the **\routes** folder, create a new javascript file called **examples.js**

**\routes\examples.js: - change the bootswatch style to the one you picked in layout.hbs**

/\*\*

\* examples.js

\* router for path: http://localhost:3000/examples/

\*/

const express = require('express')

const router = express.Router()

//TODO: code to handle GET request to the /examples/ path

/\* GET content for path: http://localhost:3000/examples/ \*/

router.get('/', function (req, res, next) {

// send content directly to the browser (not using a template) - this can be tedious

res.send('<html><head><title>Examples Index</title> <link rel="stylesheet" href="/bw/quartz/bootstrap.css" /></head>' +

'<body class="container"><h1>Examples</h1><ul>' +

'<li><a href="/examples/simple-code/">Simple Server Code examples</a></li>' +

'</ul></body>')

})

module.exports = router

Now that we have the router file lets add a path for it in the app.js file

**\app.js – add the lines in yellow**

const createError = require('http-errors')

const express = require('express')

const path = require('path')

const cookieParser = require('cookie-parser')

const logger = require('morgan')

// declare router files

const indexRouter = require('./routes/index')

const usersRouter = require('./routes/users')

//add a new const for the examples.js router

const examplesRouter = require('./routes/examples')

const app = express()

// view engine setup

app.set('views', path.join(\_\_dirname, 'views'))

app.set('view engine', 'hbs')

app.use(logger('dev'))

app.use(express.json())

app.use(express.urlencoded({ extended: false }))

app.use(cookieParser())

app.use(express.static(path.join(\_\_dirname, 'public')))

app.use('/', indexRouter)

app.use('/users', usersRouter)

// add the following line to point to the folder that contains the bootswatch css files

app.use('/bw', express.static(\_\_dirname + '/node\_modules/bootswatch/dist'))

// add a new path called examples that uses the code in the examples.js file for routing

app.use('/examples', examplesRouter)

// catch 404 and forward to error handler

app.use(function (req, res, next) {

next(createError(404))

})

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/examples/>

## Handle a Get Request

The router.get( ***[path]***, function (req,res,next) {}) function implies we are expecting a GET request ( default request methods for browsers)

The “path” parameter gets added to the router file name – for example a router file called **catalog**.js has path **“/books**” – then the actual path is webserver.com**/catalog/books**

Then are multiple kinds of request methods but the most common are GET, POST, PUT and DELETE. We will use all of these request methods eventually

Learn More about REST methods - <https://www.restapitutorial.com/lessons/httpmethods.html>

## Sending content to Client

The res.send() function is used to send a string directly to the browser. In this case the string is some html.

# Add a New Route that uses a Template

## Create a new template file with Expressions.

In the **\views** folder create a **text** file called **simple-code.hbs**

**\views\simple-code.hbs**

<!-- https://handlebarsjs.com/guide/expressions.html -->

<h1>{{title}}</h1>

<p>Welcome {{myPosition}}: {{myName}} </p>

Template files make it easier to code html by separating the html content out into their files and not having to put it all into a string variable

Templates also allow for developers to include syntax that can dynamically generate html.

There are many template engines in express. (PUG, Moustache, Handlebars etc). We will be using handlebars but PUG is more popular

Learn more about Template Engines: <https://blog.logrocket.com/top-express-js-template-engines-for-dynamic-html-pages/>

## Render a Template file

Now in the same \routes\examples.js file add handler for a new path “/examples/simple-code/”

**\routes\examples.js – add the code in yellow**

//TODO: code to handle GET request to the /examples/ path

/\* GET content for path: http://localhost:3000/examples/ \*/

router.get('/', function (req, res, next) {

// send content directly to the browser (not using a template) - this can be tedious

res.send('<html><head><title>Examples Index</title> <link rel="stylesheet" href="/bw/quartz/bootstrap.css" /></head>' +

'<body class="container"><h1>Examples</h1><ul>' +

'<li><a href="/examples/simple-code/">Simple Server Code examples</a></li>' +

'</ul></body>')

})

/\* GET content for path: http://localhost:3000/examples/simple-code \*/

router.get('/simple-code', function (req, res, next) {

// call the render function to open the simple-code template

// in the options send in as many parameters as you want

// the options parameters can then be used in the template using the moustache syntax i.e {{ variable }}

res.render('simple-code', {

title: 'Simple Server Code examples', // title used by the layout.hbs - appears at the top the page

myName: 'Your Name',

myPosition: 'Student',

})

})

module.exports = router

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/examples/simple-code/>

Notice that the new template has a title of ‘’Simple Server Code examples” – we passed the **title** option from the example.js router

When the template is rendered, all instances of **{{title}}** are replaced it with “**'Simple Server Code examples**”

Learn More about Handlebars syntax: <https://handlebarsjs.com/guide/expressions.html>

## Conditional Expressions – Add IF statements

**\views\simple-code.hbs – add the code in yellow**

<p>Welcome {{myPosition}}: {{myName}} </p>

<div class="card p-3">

<h2>Determine Random number </h2>

{{randomNum}} is

<!-- https://handlebarsjs.com/guide/builtin-helpers.html#if -->

{{#if randomIsEven}}

<i class="text-success">EVEN</i>

{{else}}

<i class="text-danger">ODD</i>

{{/if}}

</div>

**\routes\examples.js – add the code in yellow**

// some simple logic to generate a random number from 0 to 5

let rnd = Math.floor(Math.random()\*5);

// call the render function to open the simple-code template

// in the options send in as many parameters as you want

// the options parameters can then be used in the template using the moustache syntax i.e {{ variable }}

res.render('simple-code', {

title: 'Simple Server Code examples', // title used by the layout.hbs - appears at the top the page

myName: 'Your Name',

myPosition: 'Student',

randomNum: rnd, // use the random number generated above to display on the page

randomIsEven: rnd % 2 === 0, // check if the random number is even and set a boolean variable

})

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/examples/simple-code/>

## Simple Iterators

**\views\simple-code.hbs - add code to bottom of file**

<div class="card p-3">

<h2>Loop through array of names </h2>

<ul>

<!--https://handlebarsjs.com/guide/builtin-helpers.html#each-->

{{#each names}}

<li>{{ this }}</li>

{{/each}}

</ul>

</div>

**\routes\examples.js – add the code in yellow**

res.render('simple-code', {

title: 'Simple Server Code examples', // title used by the layout.hbs - appears at the top the page

myName: 'Your Name',

myPosition: 'Student',

randomNum: rnd, // use the random number generated above to display on the page

randomIsEven: rnd % 2 === 0, // check if the random number is even and set a boolean variable

names: ['Aaron', 'Betty', 'Carl', 'Debby'] // arrays and objects can also be sent to the template

})

Restart the webserver for the changes to take effect - Navigate to <http://localhost:3000/examples/simple-code/>

The #each expression will loop through the array declared in the examples.js