**LO1. Introduction to Nodejs and REST**

Content :

[What is it? 1](#_Toc98834757)

[First Intro Web App 4](#_Toc98834758)

[Additional Routes 6](#_Toc98834759)

[Additional routes with parameters 6](#_Toc98834760)

[RESTful Routes 7](#_Toc98834761)

**LO1.1. Introduction**

One of the defacto standards that has emerged in recent years for web application development is Nodejs. You will see reference to various different frameworks such as React, Angular, Vue and others but all of these are Node-based frameworks. As we will see upcoming, these frameworks are entirely dependent on Node.

### What is it?

Nodejs is a javascript run-time environment. Recall that javascript is an interpreted language and requires something to act as the interpreter to execute javascript code. Previously, in CWEB601, we learned how to write javascript that could execute in a web browser. The browser in that case was the runtime environment. Prior to Nodejs, that was the only option and there was no way to execute javascript code on a local machine. With Nodejs installed, we can now execute javascript locally which means we can now build applications which can run locally. In this course, we'll look at building a web application in the form of a locally run web server.

**LO1.2. Installation**

There are various tools we'll need to develop our web application. The following are required:

1. Nodejs
2. An IDE of some kind
3. Postman

Nodejs can be downloaded from <https://nodejs.org/en/download/>. It is available for Windows, MacOS or Linux.

There are several IDEs available that support javascript and this is really all you need. Technically, we can simply use a basic text editor but using some form of IDE will make life easier and it will also make it easier to work with GIT with our code as well since most IDEs provide support for version control. The IDE used here will be Visual Studio Code which can be downloaded at no cost from <https://code.visualstudio.com/download>, also for all three of the above environments.

Postman is a tool that we will need to test our backend before moving on to front end development. Some parts of our web application can be checked easily but much of it cannot be unless we have either a completed front end or a tool that emulates what that front end would do. That is the role that Postman fills. It can be downloaded from <https://www.postman.com/downloads/> .

Installing the above software is straightforward in that the install programs can be run by clicking through with just default settings. Note that once complete, you may want to have environment variables set up to point to the installed directories for node. In windows 10 this can be done using the following steps(NOTE: EXERCISE CAUTION HERE AS A MISTAKE CAN CAUSE SERIOUS PROBLEMS FOR YOUR SYSTEM):

1. Open File Explorer and right click on 'This PC' -> select 'Properties'.  The following window should appear:

Graphical user interface, text, application, email

Description automatically generated

2. Click on the text "Advanced system settings" on the left side. The following dialog will appear. Click the bottom button "Environment Variables..." to open the Environment Variables dialog.

Graphical user interface, text, application, email

Description automatically generated Graphical user interface, text, application

Description automatically generated

3. Here we will set the Path variable. The upper part of the window allows us to set parameters only for the current user, the lower part for system variables. Setting the Path in the lower part will set it for all users. Find the Path variable and either double-click it or select it and click the 'Edit...' button.

In the resulting dialog, click 'New' and enter the path to where the Node.exe executable is installed. You only need to specify the folder location here. The standard on windows should be something like "C:\Program Files\nodejs\".

Graphical user interface, text, application

Description automatically generated

4. Once this is done, click OK on each dialog and then close the original System window.

On MacOS, you may not need to do anything as the node executable files should end up in /usr/local/bin which is already in the $PATH variable. If you need to, this can be done a couple of different ways. You can add a $PATH via your bash profile or .zshrc file if using zsh. Or to add the path more directly, modify the /etc/paths file:

1. Open your Mac terminal

2. Execute the following command:

sudo nano /etc/paths

3. Enter your password when prompted

4. navigate to the bottom of the file and add the path for node.

5. Type control-x to quit

6. Enter Y to save

**LO1.3. Your First Nodejs Application**

Before we create an actual application, let's do the standard Hello World test to make sure Node is set up correctly.

I'll be using Visual studio code. Create a folder somewhere to store your application. In VS Code, select Open Folder from the File menu. Open the folder you just created.

Next create a new file. In the left portion of the window, right click and select New File. A file will be created and you can enter a name for it. Call it app.js.

Since this is just a simple test, type the following and save

console.log("Hello World!!");

You can run this within VS Code or open a command line/terminal window and navigate to the directory. Either way, type the following to run your app:

node app.js

And you should see the text 'Hello World!!' show up as output in the terminal window.

Congratulations! You've run your first Nodejs application.

**LO1.4. Nodejs and Packages**

One of the most fundamental aspects of development with Nodejs is the use of packages. We can program whatever we like from scratch, including web servers, but there's no need to because almost any functionality you could need for an application can be found in Nodejs packages.

Nodejs comes with its own package manager, npm. To search for packages to add to an application, go to [www.npmjs.com](http://www.npmjs.com). The way packages work, you don't download them from the npm website, rather, you use the node package manager npm to import packages via terminal. When you find a package you want on the npm website, it will tell you the two main pieces of information you need: how to install and how to use.

### First Intro Web App

For our first introductory application, we'll build a simple web server. One of the most common packages for this in Node is called ***express***. You can search 'express' on the npm website and you'll find the primary information for it. Express is a simple framework which allows for building web applications. The quick installation listed uses an application generator. We can skip this and just do a simple, usual install using npm.

Before we can do that, we will first need to create an application using npm by issuing the following command on the command line/terminal:

  npm init

This starts the npm application wizard which will prompt you for a basic set of information for your application. For our simple starter, we can accept all defaults here and just press the enter/return key for each until we're done. Once complete, npm will generate a file called package.json in the root of our project directory. This file is very important as it contains project information which will include any packages we add. The reason package.json is important is because it gets used for deployment. We will develop our apps in our own development environment but eventually they will have to be migrated to a production environment which will likely not have all of the packages we installed. That's where package.json comes in. As we install packages, the package information will be added to package.json so it will know what packages are needed when it comes time to deploy our application in a production environment.

Before we write any code, we'll need to add the express package to our project. The usual way to do this at the command line is using the command**npm install \_\_\_\_\_**where \_\_\_\_\_\_ is the package we want to add. Additionally, we'll add the flag **--save** to ensure the package info is added to our package.json file. So type

npm install express --save.

It will take a few seconds for npm to download and install the express package. We can now move on to building our intro app in the next item.

**LO1.5. Creating a Simple Web App**

To create our web app, we will first need a main file for our application. In package.json, this will be listed as "main". In our case, the default is index.js so we'll use that. Create a new file in your project called index.js.

Next we will need to import the express package into our index.js file. This is done in javascript using a require() method and assigning its result to a variable:

const express = require('express');

Then we create an app variable using a call to express:

const app = express();

We now have what we need to build the rest of our application. We first add code at the **end** of our file to start our web server and listen for connections:

app.listen(3010, function(){  
      console.log("The server is running.");  
 });

Note that the callback function can be left out here:

app.listen(3010);

In either case, the server starts and begins listening on port 3010.Note that we can choose pretty much any port as long as its not in the reserved range up to 1024.

Now we'll add a basic root route that will produce some output when we access our server from a web browser using a basic localhost web address. Add the following code after the first two lines but before the app.listen line:

app.get('/',function(req,res){  
    res.send("This is the home page!!");  
 });

There's a lot going on here so let's step through this.

The method app.get will be executed in response to a user request to our server. The request must specify the root of our server which would be, in this case, http://localhost:3010. This target is specified by '/' as the first parameter. If we wanted to respond to the user typing http://localhost:3010/mydirectory/ we would specify '/mydirectory' instead. Also note that you can also use 127.0.0.1 instead of localhost to specify the local computer.

The anonymous function here is a callback function. It is executed in response to the user request. The two parameters to the function req and res represent the request and response. The request is the user request and contains information associated with the user request from their browser. The response is whatever our server sends back to that browser. res.send() is a simple method that simply sends text back to the browser in response to its request.

The get method also represents a wrapper for the HTTP GET method and is the only method that a user triggers manually from their browser when they open a website.

To test that this is working, first run the app using **node index.js.** You should see the text from the app.listen callback showing on the command line but you won't have access to the command line. This is because the server is running and waiting for connection requests. We can now open a web browser and type localhost:3010 into the address window to access our app. You should see the text from the app.get callback showing in the window.

Graphical user interface, text, application

Description automatically generated

Use control-c in the terminal to stop the server.

**LO1.6. Finishing the Introductory Web App**

We've seen how to create a basic web app with a root or index route. Eventually we'll need other types of routes as well.

### Additional Routes

To specify additional routes, we simply add '/' and then the name of the route as the first parameter for app.get (or post, put, delete). Lets add another route called 'next'.

app.get('/next',function(req,res){  
 res.send("This is the NEXT page");  
 });

Here we've done mostly the same thing we did to create the root route but now we've specified the route as /next. To access this, save and restart your web server and enter localhost:3010/next in the browser address bar and press return/enter.

### Additional routes with parameters

We can also allow a mechanism to pass a parameter as part of the route. By adding text prefixed with ':' after a route name, whatever the user enters in the spot when calling the route will get passed in the request parameters to the web server. Let's try this out:

app.get('/next/:myParam',function(req,res){  
 var myparam = req.params.myParam;  
 res.send("This is the " + myparam + " page");  
 });

So here we have a few new things. First, we've added a parameter to the route called :myParam. We then retrieve the value passed from the client's browser by referring to req.params.<param\_name>.  We can then use that value however we need to. Try the route out by saving and restarting the server, then entering localhost:3010/next/ in the address bar and adding any random text you want on the end. You should see the text you entered in the page that gets displayed.

That's it for our first, introductory express web app!

**LO1.7. REST**

REST stands for REpresentational State Transfer and is currently the most popular architecture for web application development. Some things that make REST appealing for web development include:

1. It can return data without exposing methods
2. Can be used over almost any protocol but HTTP for web
3. Supports any content type - mainly JSON
4. Works in HTTP with just regular HTTP methods

One notable thing that is not supported by REST is the ability to maintain state as REST is considered stateless.

There are some important constraints for REST:

1. Client-Server Separation
2. Stateless
3. Cacheable
4. Uniform Interface
5. Layered System

**Client Server Separation**

The client and server are separate from one another and one is not dependent on the other. Ie. You should be able to change the front end of your application without affecting the data structure in the database.

**Stateless**

As we mentioned above, REST APIs are stateless. This means that any call to an API can be made independently of any other. It also implies that any API call contains all of the information it needs to execute and complete. It should never have to rely on any contextual information or information from the server for this.

**Cacheable**

APIs should be designed to encourage data caching. When a server returns data, it can include information similar to version information so a client doesn't need to re-request the same data.

**Uniform Interface**

By designing a uniform interface to the API, all requests to the API look the same regardless of whether they come from Firefox or any other browser. This principle helps with #1.

**Layered System**

There may be numerous layers between the Client and the Server to perform various functions such as security or load-balancing or other functions. These layers should be invisible to the client and have no affect on how the client communicates with the server.

### RESTful Routes

A RESTful API will be implemented to provide a set of RESTful routes which encapsulate so-called CRUD functionality. With web applications, these are set up using HTTP methods GET, POST, PUT and DELETE. The routes look like the following:

7 REST routes:

new       /dataitem/new  GET         show a new dataitem creation form  
create    /dataitem      POST       create a new dataitem and then redirect

index     /dataitem      GET         list all dataitems  
show      /dataitem/:id  GET         show a single dataitem  
  
edit      /dataitem/:id/edit GET  show an edit form for a single dataitem  
update    /dataitem/:id      PUT  update a particular dataitem and then redirect  
  
destroy   /dataitem/:id  DELETE    delete a particular dataitem and then redirect

Note that while there are seven routes, some can be implemented in the front-end while some must be implemented as the back-end. The individual routes using PUT, POST and DELETE are implemented in the backend while the front-end provides an initial form or view(GET) which in turn triggers these routes.