Day04 Deployment

part 1 – Folder Setup

1. Copy the folder called **Part12** from **Day03** paste it into a folder called Day04.
2. Copy the html files from **Day02** paste it inside of the folder rename it to just **HTML**. It should contain all the HTML/JS/CSS from the second day of sessions.
3. Your **Day04** folder should look like the image below, before running **npm install**.

|  |
| --- |
|  |

1. Run **npm install** to create the **node**\_**modules** folder and setup the environment.
2. First task is to serve the **HTML** pages we built on **Day02**. Now that we have a **controller** file, everything that goes to the client will pass through there and the root function will handle serving our HTML page when users land on our root which at the moment is <http://localhost/8000>. We would need to change line 3 of controller.js to serv the index.html file instead.

|  |
| --- |
| **const Weight = require('../models/employees);**  **exports.getdefault=function(req, res){**  **res.send(‘../HTML/index.html’);**  **};** |

1. Execute **node http\_server** and go to the address from #5. Of course this will not work because whatever is between the single quotes will simply print out on the browser screen. Node has a different function to handle html files, it is called **sendFile()** and it is attached to the response object.

|  |
| --- |
| **const Weight = require('../models/employees);**  **exports.getdefault=function(req, res){**  **res.sendFile(‘../HTML/index.html’);**  **};** |

Remember to stop and start the service

1. This did not work but it provided some clues, something about a path. In order to serve static pages, we need the **path** package, so in the controller.js file, declare a variable and point it to the **path** package.

|  |
| --- |
| **const Weight = require('../models/employees);**  **const path = require("path");**  **exports.getdefault=function(req, res){** |

1. The path object has a method called **join()** which we can use to obtain the current path of the application. If we then concatenate the root path with the path where our **HTML** files live, we can finally obtain a true absolute path to our files

|  |
| --- |
| **const path = require("path");**  **exports.getdefault=function(req, res){**  **res.sendFile(path.join(\_\_dirname + '/../HTML/index.html'));**  **};** |

1. Although the html file is served, it appears to not know that CSS and JS exists, we need to let Express know that these files exist and that it should use them. Open routes.js and include the following lines:

|  |
| --- |
| **module.exports = function(app){**  **const express= require('express');**  **app.use(express.static(\_\_dirname + '/../HTML'));**  **let controller = require('../controllers/controller');** |

Although we had to require **Express** in the http\_server.js file, we still have to do it again in this file. Also we just need the static method to know where the directory is that contains our html/css/js files.

Note: if the above code does not work in U18, use this instead:  
**app.use(express.static(\_\_dirname + './../HTML'));**

1. At this point, if you navigate to team weights in the navigation menu, you should see some records there already. If you do not see anything then you would need to complete the teamweights.html file.

part 2 – Configuring Team Weights

1. We did not have to do much work with teamweights.html, but we can improve the display a bit. Open that html file and go to where the **h2** element is, there should be a pair of **div** tags with an id of **records**, if not add it now.

|  |
| --- |
| **<main>**  **<h2>Showing records for team</h2>**  **<div id="records"></div>**  **<button id="getData">Get Records</button>**  **</main>**  **<aside>** |

1. In the JS file, look for the displayData() function and where we had <p> tags before change them to <div> tags

|  |
| --- |
| **function displayData(arr) {**  **let outHTML = "";**  **for(let i=0; i < arr.length; i++){**  **outHTML+="<div>"+arr[i].empName + " weighed " + arr[i].empWeight + " Kgs</div>";**  **}**  **document.getElementById("records").innerHTML = outHTML;**  **}** |

1. Now we can target that id in the css, so create a new style for our display of records:

|  |
| --- |
| **#records {**  **margin-left:34px;**  **width:80%;**  **}** |

Refresh the teamweights.html file and adjust the CSS to your liking.

1. We may also want to change the background color of alternating rows just for easier reading, again you can play with the background colors of this code:

|  |
| --- |
| **#records div:nth-child(odd) {**  **background: lightgray;**  **display: block;**  **margin:5px 0px;**  **}** |

(optional)

part 3 – adding a new record (jQuery)

The HTML file enterweight.html is being used to add a new record to the database. We have to configure that file to work with the **putnewdoc** endpoint. Before continuing make sure that the jQuery CDN files are linked in the head tags of the html doc

|  |
| --- |
| **<head>**  **<title> Skillsoft Weight Tracker </title>**  **<link rel="stylesheet" type="text/css" href="styles/styles.css" />**  **<meta content="text/html;charset=utf-8" http-equiv="Content-Type">**  **<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>**  **<script src="https://cdn.jsdelivr.net/npm/jquery-validation@1.19.0/dist/jquery.validate.js"></script>**  **</head>** |

1. Open the scripts.js file and in the jQuery Validator submitHandler section, add a line of code to hit our api (around line 48)

|  |
| --- |
| **submitHandler:{**  **function(form){**  **$.post('http://localhost:8000/putnewdoc',**  **form.submit();**  **}** |

Notice the endpoint and comma

1. We will *serialize* the form fields prior to posting so change the next line. The serialize method will be the next parameter in the submitHandler function

|  |
| --- |
| **submitHandler:{**  **function(form){**  **$.post('http://localhost:8000/putnewdoc',**  **form.serialize(),**  **}** |

1. Finally, add json in quotes as the final parameter

|  |
| --- |
| **submitHandler:{**  **function(form){**  **$.post('http://localhost:8000/putnewdoc',**  **form.serialize(),**  **‘json’**  **}** |

1. Here is the entire function

|  |
| --- |
| **function putNewDoc() {**  **//**  **$(function(){**  **$("form[name='frmCollectWeights']").validate({**  **rules:{**  **empName:{**  **required:true,**  **minlength:3,**  **lettersOnly:true**  **},**  **empWeight:{**  **required:true,**  **digits:true**  **}**  **},**  **messages:{**  **empName:{**  **required:"Name is required",**  **minlength:"Name too short"**  **},**  **empWeight:{**  **required:"Weight is required",**  **digits:"Weight must be a number"**  **}**  **},**  **submitHandler: function (form) {**  **$.post('http://localhost:8000/putnewdoc',**  **$(form).serialize(),**  **'json');**  **}**  **});**  **});**  **//** |

1. When we tested this post endpoint we got a response, something like “created John”, well we can test that out here, just add a function to handle the response

|  |
| --- |
| **submitHandler: function (form) {**  **$.post('http://localhost:8000/putnewdoc',**  **$(form).serialize(),**  **function ( data) {**  **console.log(data);**  **},**  **'json');**  **}** |

1. However the code does not work at the moment, the problem is that the code is expecting json, but getting plain data, so in the controller.js file, change the res.end() method to return json

|  |
| --- |
| **exports.putnewdoc = function(req,res){**  **let empName = req.body.empName;**  **let empWeight = req.body.empWeight;**  **const weight = new Weight();**  **weight.empName = empName;**  **weight.empWeight = empWeight;**  **weight.save({}, function(err) {**  **if (err)**  **res.end(err);**  **res.end(JSON.stringify(`Created ${empName}`));**  **});**  **};** |

This result could now be used on the HTML to give feedback to the user