The Pizza Shop

10.12.2016

Overview

In this tutorial we will be building an example of an online pizza shop ordering system. You will be creating a user interface for your hungry customers. They will enter their names and then select their pizza type and any extra toppings. At the end, you will be providing your customers a total amount to pay for their order.

Initial Setup

I. Create a directory for your project

As you have already been through the bash scripting tutorial, we won't spend too much time on the initial settings phase. Just make a directory to contain your pizza shop project and move to that directory.

```
mkdir pizza

cd pizza
```

ip: you could also achieve the same result with only one command:

```
mkdir pizza && cd pizza
```

Now create your html entry point:

```
touch index.html
```

II. Download your IDE

If you have not done this yet, please download your IDE of choice. I am currently using *Sublime3* because it is easy to learn (alternatively *Atom is very popular*). You don't necessarily need a license to use this software. The unregistered version of it lasts a month, but also after this time period you can keep using it.

Follow the instructions to download the for your OS. You can get them from:

Sublime: https://www.sublimetext.com/3

Atom: https://atom.io/

ymbolic links (aka symlinks) - advanced shortcuts.

*aded your editor, is time to add a really useful shortcut
to allow you to easily open your editor from the terminal
e manually by right clicking on the file and choose the
/Atom.*

```
ln -s "/Applications/Sublime
Text.app/Contents/SharedSupport/bin/subl" ~/bin/sublime
```

```
ln -s /Applications/Atom.app/Contents/Resources/app/atom.sh
/usr/local/bin/atom
```

you can then just type:

```
sublime index.htm
```

mlinks for you...I am sorry :(

Create the structure of your HTML page

Simply open your *pizza-shop.html* file and type something on it. Now, from your terminal, within your pizza folder, type:

```
open pizza-shop.html
```

This should open up your browser and show the content of the file. If you see the url bar, you can see from where in your machine that file has been served. This is a great way of testing the early stages of your application with HTML, CSS, and JS (and is a really good way to start playing). Don't you worry, later on we will show you how to spin up your own node server so that your page won't be served anymore from the local filesystem, but by your server instead. Confused? No, problem, it's normal. If you are confused when you arrive at this point, please ask us to explain the difference in some more detail.

I. What is an HTML document

HTML is the standard markup language for creating Web pages where every element is represented by a tag. The acronym stands for *Hyper Text Markup Language*. There are many things to consider when we are putting in place the skeleton of our HTML page.

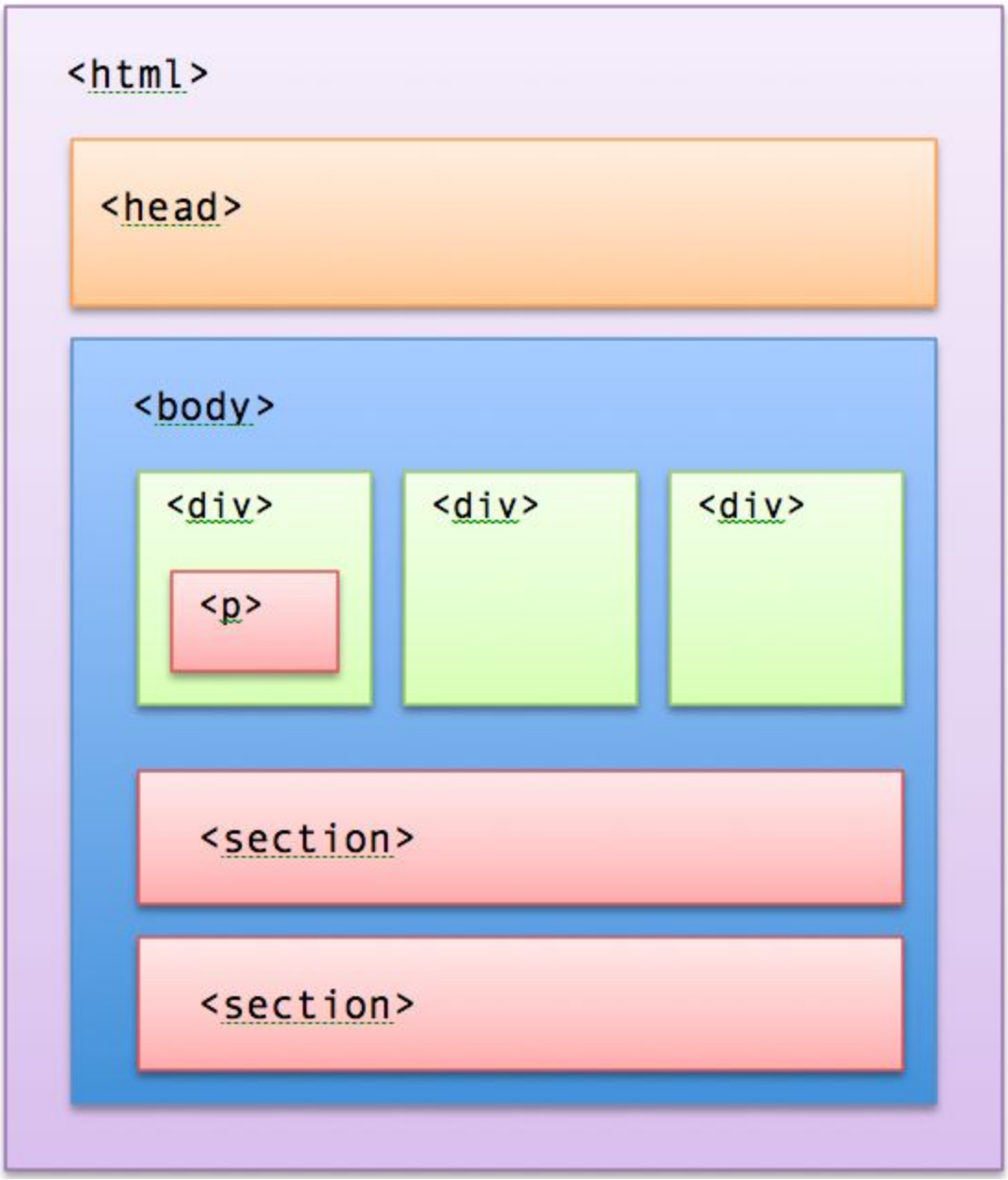
First is `DOCTYPE` but we will leave it on a side for now, but just remember to always include `<!DOCTYPE html>` at the beginning of your document to make sure you are using HTML5 and not previous versions. Think about an HTML page as a tree: you have the root, the trunk, branches, branches branching out from other branches, leaves and sometimes even flowers.

A HTML page is exactly the same: you have a `<html>` element that is the root of your page, a `<head>` that contains informations about your tree (the type for example), a `<title>` to specify the name of your document, a `<body>` the trunk (the VISIBLE CONTENT OF THE PAGE), your semantic tags `<section>`, `<p>` etc and sometimes semantic tags nested with each other `<section><p></p></section>` (that is a "branch" section containing a branch `<p>`). In the last case, our `<p>` element is the child of the <section> element but at the same time, the <section> element is child of the `<body>` (everything, in fact, derives and comes back to the body). Then, inside your `<p>` element, you can have some text `<p>Hello</p>`, these are our leaves. We can as well embellish our `<p>` by giving it some attributes, these are our flowers. This is the basic structure of any HTML page, something that is always present and it looks like:

```
<!DOCTYPE html>

<html>

   <head>

     <title></title>

   </head>

   <body>

   </body>

</html>
```

If you have any doubt about an HTML element, please refer to the W3C standard. The World Wide Web Consortium (W3C) is an international community where Member organizations, a full-time staff, and the public work together to develop Web standards. Basically they are the ones that decide the rules of the game! How something should be used and what browser should support it.

Below, there is a graphic representation of what the browser does when you open a webpage. Have a look at that, and then keep reading.



When your browser sees a webpage, starts building the HTML document structure based on your tags and then renders your page. The main container or root element is the `<html>` tag. When the browser sees this tag, knows exactly that needs to build a webpage, Everything is contained in this main tag and this tag has usually information about the language used in your website. Straight after that comes the `<head>` element. In here, you can find all the links to import static files and `<meta>` tags (i.e. author, content of your webpage and so on). After your head tag, usually comes the body tag. The `body` is exactly what the tag name is describing: the main body of the document where you can put all your tags. In the example above, inside the body there are 5 elements all at the same level. Being all at the same level, they can be also called siblings.

Last, inside the first `div`, there is a paragraph tag. The paragraph is then, a child of the `div`. This is how the code behind the page above looks like:

```
<!DOCTYPE html>
<html>
   <head>
   </head>
   <body>
      <div>
            <p></p>
      </div>
      <div></div>
      <div></div>
      <section></section>
      <section></section>
   </body>
</html>
```

II. The `<input>` field

The form element is probably the richest element you can find amongst the HTML elements and is commonly used to send data to a back end system; users input data on the page, and then it is sent somewhere. It is not by chance we have decided to start from this!

This is how it looks like:

```
<form>
  First name:<br>
  <input type="text" name="firstname"><br>
  Last name:<br>
```

```
    <input type="text" name="lastname">
    <button type="submit">Submit</button>
</form>
```

As you have now seen, one of the main elements of a form is an input field. The name is self explanatory but it does not really say how much you can do with it just by using the native platform. Remember that the platform is actually providing most of the things you may need on a web page. A common problem is that developers do not realise that and end up re-inventing the wheel. If you understand what the platform can provide you, you will gain enough tools to suggest a better and more productive implementation

As mentioned, the platform provides most of the things you may need. Let's say for example that when a user visits your page, you want an input to be selected by default. Do you think you need to write code for that? Or is that something the platform provides?

Try googling by yourself how to achieve your "html focus on input by default". Give yourself 5 min to experiment, then continue reading.

As you have probably seen, the solution is:

```
<input type="text" name="lastname" autofocus>
```

That autofocus is an attribute, one of the flowers of your <input> branch. Is something you don't necessarily need, but if you do, is available.

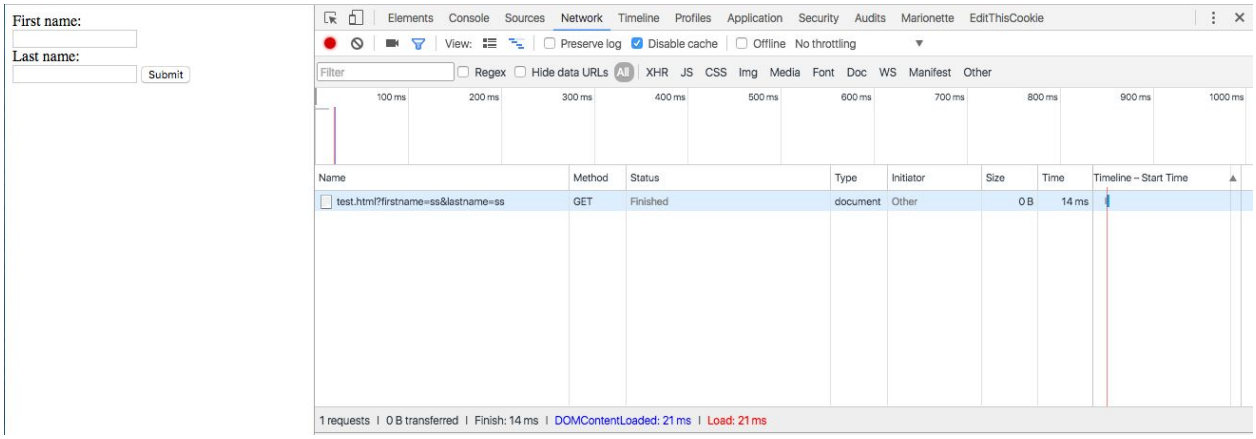
III. Submitting the form

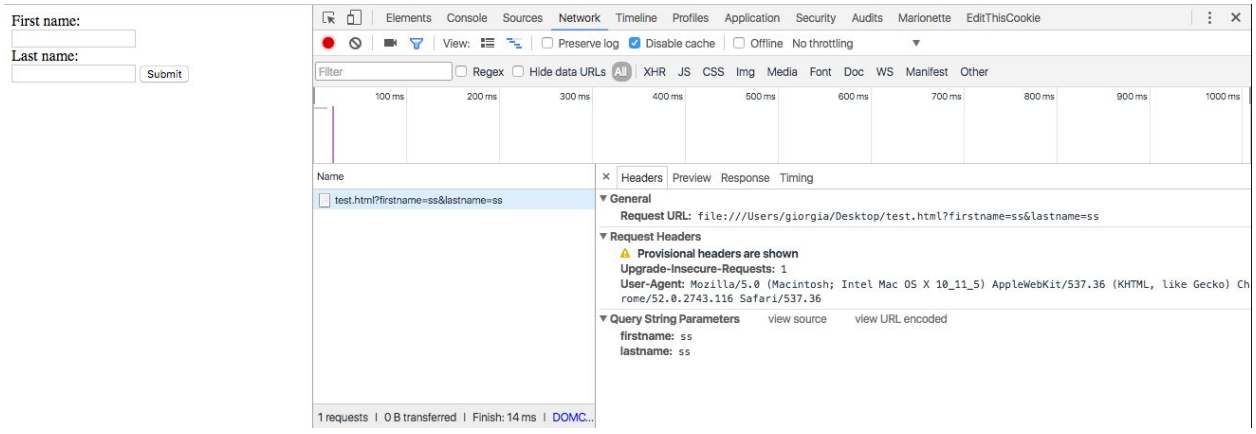
Now, fill the inputs with name and surname and click submit. Nothing seems to have happened, right? Well that is not entirely true. Let's have a look on what is happening behind the scenes and let's open the developer tools: right click anywhere on the page and click "Inspect". This will open the developer tool panel.

Let's have a little tour of the developer tool (for this tutorial I am referring to the Chrome Dev Tool):

- Elements tab: you can see the html elements on your page and their structure
- Console tab: that is where you can use and debug Javascript. Try doing something like 1+1, see?
- Sources tab: here you can see the files requested by your application and your files when in debugger mode.
- Network tab: that is where all can see all your web requests (images, files, analytics and so on).

The Network tab is what we want to be looking at right now. So keeping your developer tool open on the Network tab, try to click again on the submit button and see what happens. Now click on the result reported under the "Name" column and you should see something like this:





Let's now add some radio buttons:

```
<form>
   <legend> Customer Details </legend>
```

```
    First name:<br>
    <input type="text" name="firstname"><br>
    Last name:<br>
    <input type="text" name="lastname">
    <fieldset>
       <div><label> <input type=”radio” name=”size”> Small </label></div>
       <div><label> <input type=”radio” name=”size”> Medium </label></div>
       <div><label> <input type=”radio” name=”size”> Large </label></div>
     </fieldset>
     <legend> Pizza Style </legend>
         <div> <label><input type=”radio” name=”size”> Thin </label></div>
         <div><label> <input type=”radio” name=”size”> Thick
</label></div>
         <button type="submit">Submit</button>
</form>
```

If you now try to submit your form again, nothing is happening, right? Well, it is because at the moment they all have the same name attribute. The name attribute is used to reference form data after a form is submitted and only form elements with a name attribute will have their values passed when submitting a form. Equally each input field must have a different value for the name attribute or only one of them will be submitted. Time to change that:

```
<form>
  <legend> Pizza Size </legend>
<fieldset>
       <div><label> <input type=”radio” name=”small”> Small </label></div>
       <div><label> <input type=”radio” name=”medium”> Medium </label></div>
       <div><label> <input type=”radio” name=”large”> Large </label></div>
  </fieldset>
  <legend> Pizza Style </legend>
  <div>
       <label><input type=”radio” name=”thin”> Thin </label>
  </div>
  <div>
       <label> <input type=”radio” name=”thick”> Thick </label>
```

```
  </div>
  <button type="submit">Submit</button>
</form>
```

So, if you now check the developer tool (under the **Query String Parameters**) after checking your radio buttons you will something like:

medium:
on
thick:
on

And if you look at the URL, you can see exactly the same going on:

file:///Users/giorgia/Desktop/test.html?medium=on&thick=on

Let's change our name for the pizza style to style instead of size and add a different value for each of the radio buttons:

```
<fieldset>
      <legend> Pizza Size </legend>
      <div><label> <input type=”radio” name=”size” value=”small”> Small
</label></div>
      <div><label> <input type=”radio” name=”size” value=”medium”> Medium
</label></div>
      <div><label> <input type=”radio” name=”size” value=”large”> Large
</label></div>
</fieldset>
<fieldset>
      <legend> Pizza Style </legend>
      <div><label> <input type=”radio” name=”style” value=”thin”> Thin
</label></div>
      <div><label> <input type=”radio” name=”style” value=”thick”> Thick
</label></div>
      <button type="submit">Submit</button>
</fieldset>
```

Now let's try to submit again and check the network tab.

So, something is actually happening, right? But right now we have nowhere to send our data to. Equally, if you check the network tab, you can see that the type of request you are making is a GET. Well, that is the default that the browser offers, although is not recommended to use a GET request to send data.

A GET request, as you can see, puts all the values the user types in the URL bar, this isn't a great way of sending data, so instead we can use something called "POST", this sends the data behind the scenes instead and stops it all being put in the URL. You can do this with:

```
<form id="pizza-order" action="/order" method="post">
```

This is just telling when to send our information and the method to use. Now let's refresh and try to submit again: As you can see, the url changed and if we inspect the network tab we can actually see the data we have submitted.

IV. Installing node and npm on your machine

If you already have node and pm installed on your machine, please skip this part and go directly to the next section: Spinning up your own server.

Let's say that node is a Javascript environment for now and npm is what you use to require specific sets of tools for your application. The concept is way more complex than this, but you shouldn't be too worried about that! If you are interested though, you can have a look at the official node page: https://nodejs.org/en/ and have a general feeling of what node can do!

Instructions for Mac OS:

You first need homebrew: http://brew.sh/

Click that link and copy and paste in your terminal the first snippet. Wait until the installation is done, the you can finally install node and npm:

Just type: `brew install node` and after the installation is complete, to check if node is installed correctly, type: `node -v` (this is the command to check which version of node is your machine running. To see if npm is installed type in the terminal `npm -v`. You should have something back like: *v6.2.2* for the first command and something like *v3.9.*5 for the second command. If you are having troubles with the installation, please ask one of the tutors to help you. As one last thing, from your project folder, type `npm init`. This will prompt you some questions, you can either reply or press enter until all the questions are all gone! You should be now good to go.

Instructions for Mac Windows:

Follow the instructions on https://nodejs.org/en/download/

V. Spinning up your own server!

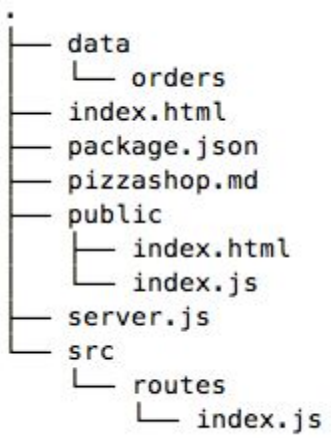
Disclaimer: This part is highly technical but fully documented below. If you are running out of time and you just want to have your app up and running, feel free to skip this part and come back to it later. If you decide NOT to skip this part, skip the disclaimer info and go straight to the next paragraph. If you decide to skip this part, please clone this repository inside your pizza repository. After you have cloned the repository, move your `index.html` file inside the public folder.

`pwd` (make sure it returns `pizza` as current location)

```
git clone git@gitlab.com:fless-tutorial/fless-tutorial-server-only.git
```

```
cp index.html public
```

Time to structure our project now! Create a file in your root folder (the root folder is the folder you called `pizza`) named: `server.js`. Create as well a folder in the root called `data`, another folder called `src` (short for 'source') and inside this folder another folder called `routes`. Now again in the root, add a folder called `public`. Inside the `routes` folder, add a file called `index.js` and move your `index.html` into your `public` folder and still inside your public folder add a file called `index.js`. Now your project should look like:



You should have as well the package.json file that you have generated when you run the command *npm init* in the previous section.

```
'use strict';
```

```
const express = require('express');
const bodyParser = require("body-parser");

// standard set up to receive data
const PORT = 8081;
const app = express();
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(express.static( __dirname + '/public'));

// load our logic and run it
const routes = require('./src/routes');
routes(app);

// all set up, run the server
app.listen(PORT);
console.log('Running on http://localhost:' + PORT);
```

So, from top to bottom, what this file does is:

- requiring a set of “tools”, express web server, and body parser
- telling to your machine what PORT to use to connect (there are many many many ports that your machine could use, but try to stick with 8081 for now)
- Telling the server how to process incoming data
- Setting where the static folder is (we are telling the application where your code is)
- And bootstrapping the endpoints for your app `routes(app)`

This is a lot of information, I know, but the general idea you need to remember is that server.js is the entry point of your application: it does some setup, tells the application where to find your code, where to store data and anything else generic about your application. The one you are writing, as you may have imagined from the file extension, is Javascript (aka JS)! To be more precise it is a server side Javascript language, called Node.js the only (so far) JS server side language. Now, add this to your `package.json` (after author):

```
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
```

```
  }
```

Now, copy and paste inside your `routes/index.js` file (the ones in bold are some comments to explain you what is going on):

```
const fs = require('fs');
const filename = 'data/orders'; // this is the file we save the orders to

module.exports = function(app) {
  app.get('/list', function (req, res) {// when we make a request to /list
    fs.readFile( filename, 'utf8', function (err,data) {//read the order file
      if (err) {
        console.log('something went wrong ' + err);
      } else {
        res.send(data); // send the file contents back to the browser
      }
    });
  });

  app.post('/order', function (req, res) {// when we get a request to /order
    console.log(req.body); // output to see it on the server

  // build the data: "Joe,joe@hotmail.com, large, pepperoni, italian, cheese"
    var data = req.body.firstname + ',' +
    req.body.email + ',' +
    req.body.size + ',' +
    req.body.style + ',' +
    req.body.type + ',' +
    req.body.toppings + '\n'; // finish the line with a "\n" newline character

    fs.appendFile(filename, data, function (err) {// save the order to file
      if (err) {
        console.log('something went wrong ' + err);
      }
```

```
    });
    res.send('written'); // return to the browser that it was successful
  });
}
```

VI. Time to see some data coming through

Now we should be all set to see some data coming through from our form to our terminal, but before we need to add this to our form:

```
<form id="pizza-order" action="/order" method="post">
```

As you can see we change the method property from GET to POST so that our application knows the data needs to be submitted and processed to a specific resource. From your terminal, now run `node server.js`, and go to http://localhost:8081 and try to submit again your form. You should see in your terminal something like:

```
{ firstname: 'giorgia',
  lastname: 'amici',
  size: 'small',
  style: 'thin' }
```

This is output by the `console.log(req.body);` in your routes/index.js file. You'll find statements like this very useful for inspecting variables and outputting things while you're trying to figure out what's happening and where. Often developers can litter their code with snippets like, console.log("got here"), so they can see exactly which bits of code are being run. There's many more elaborate and complex ways of following the code flow, but often just logging something is the quick solution!

So now, your application is not served from your local file system but from your own mini server!!WOOP WOOP! Even more than that, if you now visit http://localhost:8081/list we will see a list containing all our orders!

Now back to the html form, time to try adding some required fields. This will stop the form from being submitted if these fields are not completed. This is a native part of the browser, as we did for the autofocus try to figure out how to make some of your fields required, give it few minutes and then continue reading.

Again, you have probably already figured out, this is how you make a field required:

```
<input type="text" name="firstname" required>
```

Now try to submit again an order trying to leave blank one of the required field and have a look at what happens in the network tab. Interesting, right? And all that functionality is already supported by all the browsers and you do not need to code it! Now, let's swap the surname input fields with an email input field like so:

```
<input type="email" name="email" required>
```

Adding the `type="email"`, gives us a lot of functionality from the platform that we don't need to code. Try typing in the email input, a simple letter or a number and try to submit the form. The browser will itself show an error to the user: “Please include an @”. So, this is still really basic form validation. In fact, if you just type a “random@.com” in your input form, also if we can see it is not a valid email, the browser will still submit it because it meets the criteria that it needs to recognise something as an email address.

Time to add apizza type dropdown:

```
<fieldset>
    <legend>Pizza Type</legend>
    <select name="type">
        <option value="margherita">Margerita</option>
        <option value="marinara">Marinara</option>
        <option value="hawaiian">Hawaiian</option>
        <option value="meatball">Meatball</option>
    </select>
</fieldset>
```

It is really important to add to the `<select></select>` the *name* attribute, otherwise the browser won't know how to send the data as and it won't send it. Have a play now with the dropdown and check your network tab, your terminal and http://localhost:8081/list.

Is now time to add some ingredients the customers can choose from when making their own pizzas. We need to make sure we submit all of them to make our customers happy! This is not a mandatory field, so let's make sure we inform them about this.

Let's add some checkboxes now:

```
<fieldset>
    <legend>Extra Toppings: If you want, choose some extra toppings. We will charge you 0.25 each</legend>
    <input type="checkbox" name="toppings" value="mozzarella"> Mozzarella <br>
    <input type="checkbox" name="toppings" value="salami" checked> Salami<br>
</fieldset>
```

Now that we have added some extra checkboxes, let's submit some extra toppings to our pizza. Let's check again the network tab, the terminal and the http://localhost:8081/list page. All good? Now, we can add our pizza, customise them but still something is missing... We don't know how much our order is gonna cost! Well, is not time to introduce frontend JS to our pizza shop! First, we need a really famous library called jQuery:

```
<head>
<title>PIZZA SHOP</title>
<script
type="text/javascript"src="https://cdnjs.cloudflare.com/ajax/libs/jquery/
3.1.1/jquery.min.js"</script>
</head>
```

jQuery

Now we need to tell our application where to find this new file. Add this into your file (just before the </body> tag):

```
<script src="index.js"></script>
```

Then write this is your index.js file:

```
alert( "Hello World." );
```

Time to refresh. See? your JS file is loaded and ready to be used. Now, first things first. To make sure that our index.js file is run only once the entire structure of the HTML is ready, write this:

```
$( document ).ready(function() {
    console.log( "ready!" );
});
```

Now, what we need to do is to use jQuery to iterate through our list of extra toppings and see which one is "checked" and then multiplying the value of checked fields by £0.25.

VII. Some maths with JS

First things first. Open your developer console in your browser and type (and then press Enter):

```
$("input[name='toppings']");
```

This is how you get all the input fields with the name 'toppings', and that is exactly what we want. Now, if you try to add *.length* (do not forget the dot!) to the previous snippet you will see how many are they. Now, that is still not good enough because we only need the checked ones. Go on and check some of the toppings, then in your developer console, run:

```
$("input[name='toppings']:checked");
```

Now, try doing the same, still from your developer console, but this time let's save the result in a variable.

```
var total = $("input[name='toppings']:checked");
```

So, in Javascript, the *var* is the keyword for "variable assignment". What that means is that *var* is a special word in JS that can only be used to assign some value to some variable. In our case, the variable name is *total* (the variable name always goes on the left of the equal sign) and the value (that always goes at the right of the equals sign) is the value.

Now, if you type:

```
console.log(total);
```

you should see the same result!

We cannot cover all the basics of JS in this tutorial, we will need one entire tutorial only for that. We will try to cover as much as possible here, but if you have any doubt do not hesitate to ask any of the tutors. Is time to come back to our JS file, *index.js* and try to put everything we have done so far in that document rather than just in the developer console. Open up your index.js and write:

```
$( document ).ready(function() {
    console.log( "ready!" );
    var total = 0;
    total = $("input[name='toppings']:checked").length * 0.25
```

```
        console.log(total)
});
```

Nothing new so far, right? The only thing we did not go through is `var total = 0;` This is not essential for this tutorial, but if you'd like to know more, just ask one of the tutors. If you know reload the page, in your console you should now see two things: *"ready!"* and the value of your variable total (that should be equal to to 0 at the beginning). So, right now, if we check any of the toppings nothing happens. That is because our snippet of code has already been executed and it gets executed only once. So, we need to make our page a bit more dynamic. The idea is to run that code snippet once the customer has submitted the form, so that it will give a total for the topping. JQuery is going to help us with that because it provides a *submit()* function that will wait until a form is submitted before running a specific part of your code. The first thing you need, is the *id* selector of your form. If you check your html file again, you will see `<form id="pizza-order">` and that is exactly what we need. We need to use the id because the id is a unique identifier in an HTML document -there cannot be 2 same ids in the same document- so by selecting our for with the id with are 100% sure we are targeting only that form. Just try typing this and pressing enter in your console:

```
$( "#pizza-order" ).submit();
```

You see? Your form has now been submitted. Add this to your JS file:

```
$("#pizza-order").submit(function() {
      console.log(total)
      return total;
});
```

So, your JS now should be looking something like:

```
$( document ).ready(function() {
   console.log( "ready!" );
   $("#pizza-order").submit(function() {
        var total = $("input[name='toppings']:checked").length * 0.25
        console.log(total)
        return total;
     });
});
```

Be sure that you have ticked the *Preserve Log* option in your network tab or you won't be able to see anything once you change page!

So now, that is actually working and is outputting in the console to total in price of our toppings. As the last thing to do in this tutorial, we will notify the user of the topping price. Instead of console.log, type:

```
$('#toppings').append( "<p>" + total + "</p>" );
```

This is appending a new paragraph inside your #toppings fieldset and pass the total. Easy peasy!

You arrived at the end of this tutorial but if you want you can continue reading for an extra task that won't be guided.

VIII. Extra task: total and CSS styling

A. As your extra non guided task, try to use what we did above for the total for the toppings and using it to compute an overall total for the order. The process will be really similar to the one we did together!

B. As a second extra task, try to style your pizza shop. Try by adding some css. What I can tell you is that you will need a main.css file in your public folder and include it in your html file. To achieve that, have a look at this link: http://www.w3schools.com/tags/tag_link.asp.

If you get stuck, you can contact me and I will help or send a code snippet! You can contact us at exercise@fless.co.uk.

Thanks again for attending and if you want to follow the second part of the workshop where we will be covering email templating, databases, seo techniques and UX principles, contact us at courses@fless.co.uk.

Milestones