## Serve Static Files

In this step logic about distributing public files will be implemented – in other words how load different files (.css or simply image) from our server.

Let's begin with adding the back-end logic. In "**handler**" folder add new "***static-files.js***". It will behave like a normal handler but instead of returning html it will return file(s). Our public folder will be the "**content**":

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

The getContentType(url) function will check what the content type of the resource should be (e.g – if the url ends with *"***.css**" the content type should be "**text/css**"). For the full list of content types click [here](https://www.sitepoint.com/web-foundations/mime-types-complete-list/).

Go back to "***handlers/index.js***" and add the static file handler:

|  |  |
| --- | --- |
|  |  |

If you have not [referenced](https://www.w3schools.com/tags/tag_link.asp) the "**site.css**" file and the "**favicon.ico**" in "***home/index.html***" go back and do it.

\*Note make sure that the **href** starts with "**/content/…**"

After that start the web application, now the home page should be slightly different:

|  |  |
| --- | --- |
|  |  |

## Implement Database

In order to **add product** and list all products we should implement some sort of database. In the "**config**" folder add new file called: "***database.js***".

In it there will be in-memory array used as a database. There will be three methods that the "database" should give: get all products, add product and find product by name:

|  |  |
| --- | --- |
|  |  |
|  |  |

Now that we have some sort of database let's continue with adding the functionality of uploading a product to our website.

## Product Create Page

New handler should be created in order to handle all the needed logic for addition of products.

Create "***product.js***" inside the "**handlers**" folder. The handler will be responsible for displaying the (html) form **or** parsing the data from it and put new a product in the database:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Now that the logic about sending a html form is completed the html itself should be created. Go to "**views**" and add new folder "***products***" and add "***add.html***" inside it:

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <meta http-equiv="X-UA-Compatible" content="ie=edge">  <title>ShopStop</title>  <!--  TODO: Link favicon.  TODO: Link css. -->  </head>  <body>  <header>  <nav class="nav">  <ul>  <li><a href="/">Home Page</a></li>  <li><a href="/product/add">Add Product</a></li>  </ul>  </nav>  </header>  <main>  <form class="form center-form" method="post">  <div class="form-group">  <label for="name">Name</label>  <input id="name" name="name" type="text" class="input-field" />  </div>  <div class="form-group">  <label for="description">Description</label>  <textarea id="description" name="description" type="text"   class="input-field"></textarea>  </div>  <div class="form-group">  <label for="price">Price</label>  <input id="price" name="price" type="number" step="0.01"   class="input-field" />  </div>  <div class="form-group">  <label for="image">Image</label>  <input id="image" name="image" type="text" class="input-field" />  </div>  <div class="form-group">  <input class="btn form-btn" type="submit" class="btn" value="Add">  </div>  </form>  </main>  <footer>  <p>&copy; ShopStop</p>  </footer>  </body>  </html> |

One more thing before we test our application – go to "**handlers/index.js**" and add the new handler. The following view should be displayed whenever we try to add new product:

|  |
| --- |
|  |

Now let's go back to the **product** **handler** and implement the POST request – when the data is sent to the server.

If everything went as expected whenever we click the "**Add**" button we should be redirected to home page.

|  |  |
| --- | --- |
|  |  |
|  |  |

## Home Page

In order to validate previous section it would be nice to have a place where all products are displayed. This is why we have to go back to home handler and change things a little.

But before that go the "**home/index.html**" and put a placeholder inside the **<main>** tag:

|  |  |
| --- | --- |
|  |  |

Now let's go the home handler. We should get a reference to our database and from it get all **\*available** products:

|  |
| --- |
|  |

\*Note that we do not filter the products which are already bought – it will be implemented in the next parts.

If you are not fan of replacing strings you can try any lightweight view engine (like [EJS](https://www.npmjs.com/package/ejs)).

Start the application to see if it works as expected:

|  |
| --- |
|  |
|  |

## Search Products by Name

Having products is fun but sometimes you want to filter them by some criteria. This is why we can create a simple form in our home page which will have only one text field (product's name or part of it). After submitting the form all the products containing such text or having same name should be displayed (case-insensitive):

|  |
| --- |
|  |
|  |

#### Hints:

In "**home/index.html**" below the heading add simple form with one input field and one submit button.

Since the form is displayed on the home page the "**home**" handler should process the request. Go to the "**home**" handler and check if there is any filter argument and if there is filter the collection on the output:

|  |  |
| --- | --- |
|  |  |
|  | \*qs comes from querystring module |