# Lab: Shop Stop

This lab is part of [“ExpressJSFundamentals” course @ SoftUni](https://softuni.bg/trainings/1642/expressjs-fundamentals-may-2017). The lab itself will be distributed into several parts each containing more concrete information and guide steps on how to develop the functionality specified below.

“*Shop Stop*” is very simple **product** **catalog** website (like [OLX](https://www.olx.bg/), [Amazon](https://www.amazon.com/) – but simpler 😊). The application will consist of **users**, **products** and **categories**. Each **user** can **register**, **login** and **logout**. **Users** also can **create**, **buy**, **edit** or **delete** a **product**. Each **product** has **a** **category** in which it is specified. Site will implement of **searching** for a **product** by the product’s **name** or **category**.

### Table of Contents

* Part I - Laying the Project Fundamentals
* **Part II – Using Third-Party Modules** **(current)**
* Part III – Defining Database Models and Relations
* Part IV – Advanced Functionality
* Part V – User Authentication

# Part II – Using Third-Party Modules

This part will cover two different functionalities: uploading image when product is created instead of specifying url and using .json file as database.

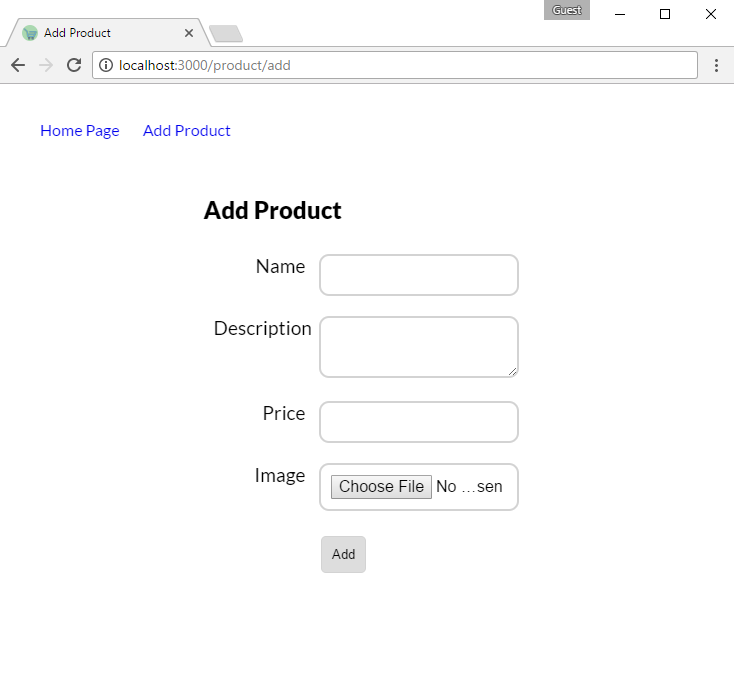
## Upload Image for Product

Our goal for this section is to succeed in uploading images to the web server and saving reference to them in products in order to be retrieved later on.

We should start with the html. Go to "**product/add.html**", for the input field with name "**image**" change it's type from "**text**" to "**file**". Make it only [accept](https://www.w3schools.com/tags/att_input_accept.asp) .png, .jpg and .jpeg files. One more crucial thing, add enctype="multipart/form-data" in the <form> tag:



Let’s start the application to see how the addition of product is looking like:



Now it is time to go back to the **product** **handler** and make sure that the file is uploaded on the server and saved in database.

First, we start with installing "[multiparty](https://www.npmjs.com/package/multiparty)" – for parsing data from the html form and "[shortid](https://www.npmjs.com/package/shortid)" for generating random names for files (this step is left to you). Now require it inside the handler:



Below is given example implementation of how to work with multiparty. It is very crucial to work with right encoding in order to save data in proper format.

For more clarity about how it works check the documentation [here](https://www.npmjs.com/package/multiparty).

**EDIT:**

**51 – Unnecessary return statement.**

**46+ - product??**

**77 - form.parse(req)??**

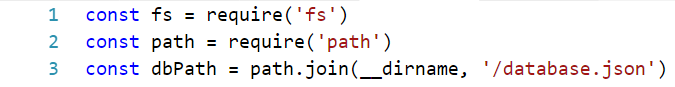
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

There are several key moments…

## JSON File Database

Until now we used in memory array for the most of our operations – we will change that. Let’s save our data in **.json** file so whenever we make changes these changes to be persisted somewhere in a file.

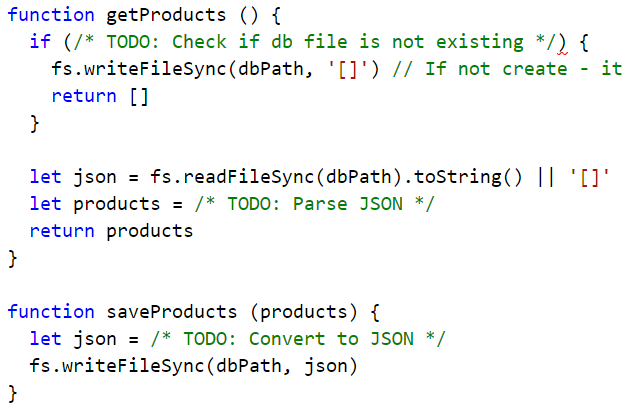
In order to keep data in custom file will need two specific modules: "**fs**" – for read/write **file operations** and "**path**" – for more clarity about **file pathing** and where read/write operations will happen. Last but not least path reference on where that database file is located:



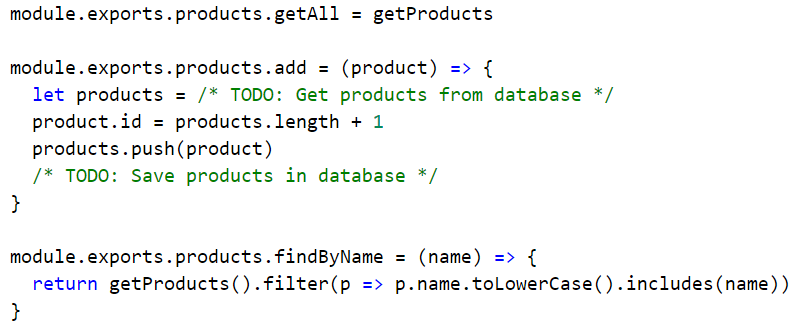
We will have to define two major functions which can be used in our database: getProducts() and saveProducts(products).

The first one will read the data from the file, parse it from json and return it as a result.

The next one however will accept collection of products which will overwrite currently saved (in the file) collection and save it in json format:



Now that we have implemented the helper functions we can change the implementation of exported methods:



Now it is time to test our application and see if every product is saved in the file, no matter how many times the application has been started.

## Minify CSS

Here we will use [Gulp](http://gulpjs.com/) to minify our CSS which is linked in the index page.

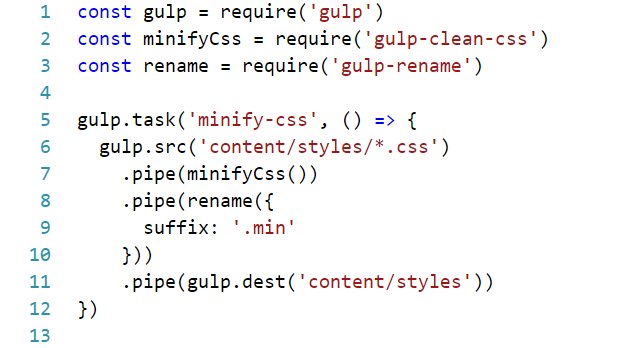
First, let's install gulp if we haven't already:

|  |
| --- |
| npm install gulp-cli -g  npm install gulp --save --save-exact |

Now we can install gulp plugins like [gulp-clean-css](https://github.com/scniro/gulp-clean-css) and [gulp-rename](https://www.npmjs.com/package/gulp-rename):

|  |
| --- |
| npm install gulp-clean-css --save --save-exact  npm install gulp-rename --save –save-exact |

Next, we should create gulp file containing **CSS** task:



Then run the following command and after it's execution new **site.min.css** file.

|  |
| --- |
| gulp minify-css |

Go back to "**home/index.html**" and **reference** the **minified** **version** of the site.css file. Run the application to see if anything is working properly.