Go to **home handler** and update it to look like this:

|  |  |
| --- | --- |
|  | **Remove** any **route** and **http** **checks** and **extract** every single if-case to a **function**. |
|  |  |
|  |  |

## Home Page Update

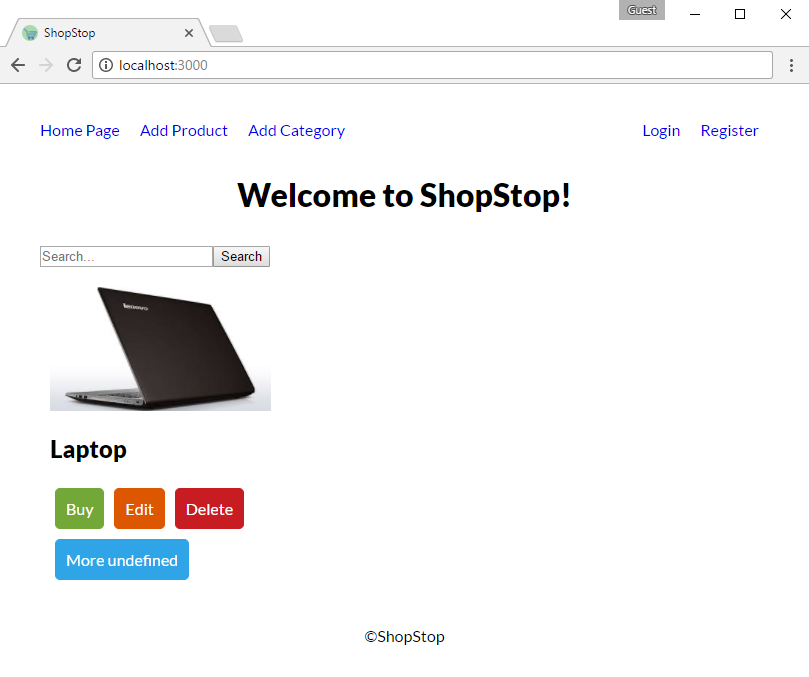
Before we add new functionality go to **"home/index.pug"** and add the following:

|  |
| --- |
| a.btn.btn-small.btn-success(href=`/product/buy/${product.\_id}`)= 'Buy'  a.btn.btn-small.btn-warning(href=`/product/edit/${product.\_id}`)= 'Edit'  a.btn.btn-small.btn-danger(href=`/product/delete/${product.\_id}`)= 'Delete'  a.btn.btn-small.btn-primary(href=`/category/${product.category.name}/products`)  =`More ${product.category.name}` |

These are links for the functionality we are about to add. Keep in mind that these links should be below each product like this:



If everything went well on the **home page** below every product (if any) there should be four new fancy buttons:

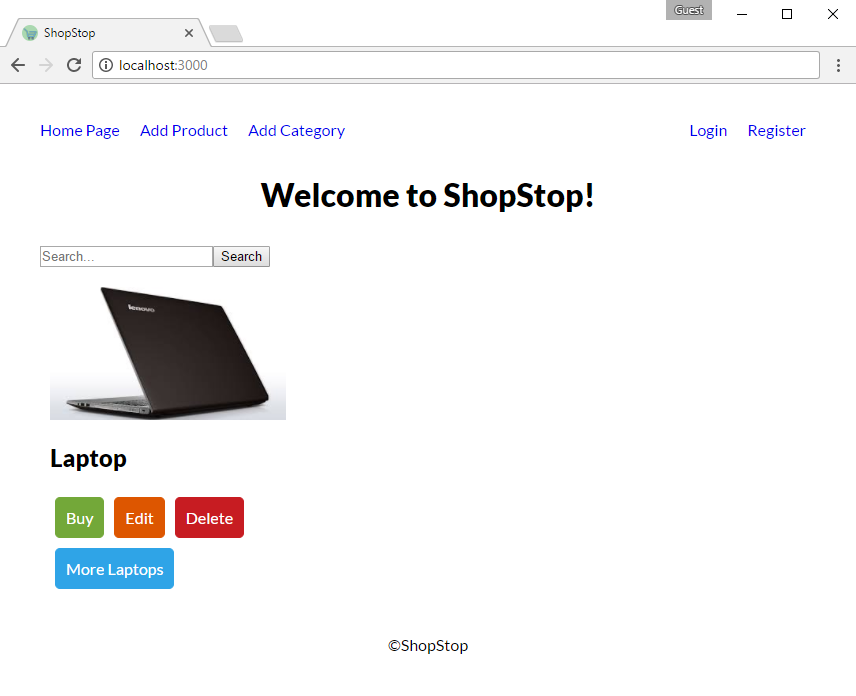


Note that we have "**undefined**" in one of our buttons – so let’s fix it.

The problem is that we want to use product’s category name there, but in order to do that we have to take the name from the database. We are going to use mongoose "[populate](http://mongoosejs.com/docs/populate.html)". Go to **home handler** and modify **index** function to be more like this:



Now, start again and see if it is working properly:

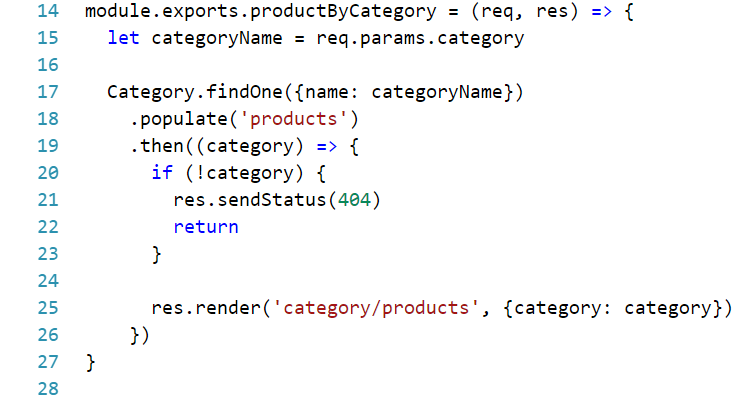


## Show Products by Category

Implementation of showing products by category is next step in developing our application or basically said what will happen when we **click** on "**More Laptops**" button.

At start we should clarify how the above functionality is going to work: first, the **route** will be like this: "/category/:category/products" which means we could get the **category** name **from** the **route**, make **query** to **database** about that specific category and **generate** **view** with all products in given category as response.

So, we should create a new action **category handler**:



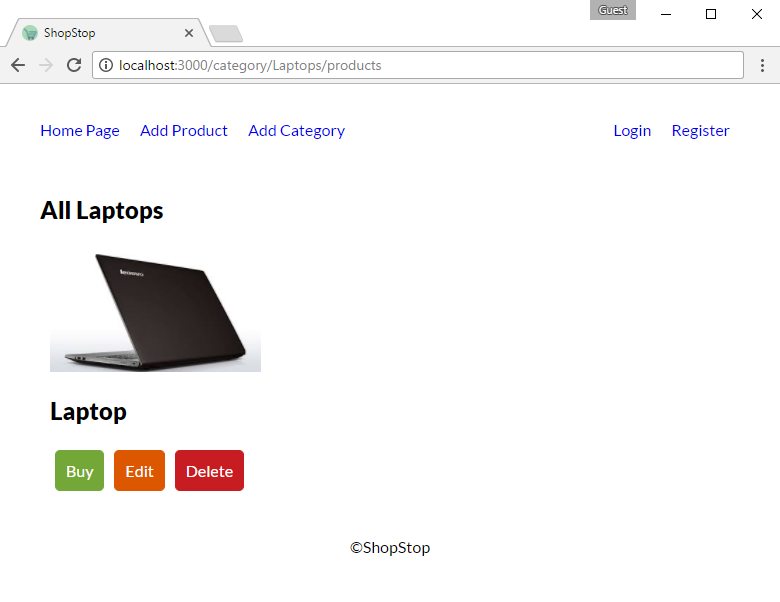
It should **render** the following:

|  |
| --- |
| extends ../layout.pug  block content  h2=`All ${category.name}`  .cards  each product in category.products  .product-card  img.product-img(src=`${product.image}`)  h2=product.name  p=product.description    a.btn.btn-small.btn-success(href=`/product/buy/${product.\_id}`)= 'Buy'  a.btn.btn-small.btn-warning(href=`/product/edit/${product.\_id}`)= 'Edit'  a.btn.btn-small.btn-danger(href=`/product/delete/${product.\_id}`)= 'Delete' |

And here is the **routing** map:



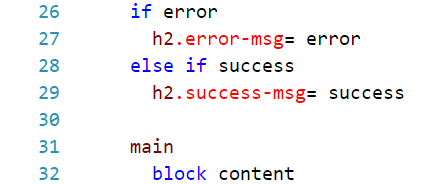
Now start the application and [test](http://imgur.com/gallery/GpWZtCn):



## Message System

In this step we will see how we can send messages to user with simplified **message** **system**.

First go to "**views/layout.pug**" and add the following:



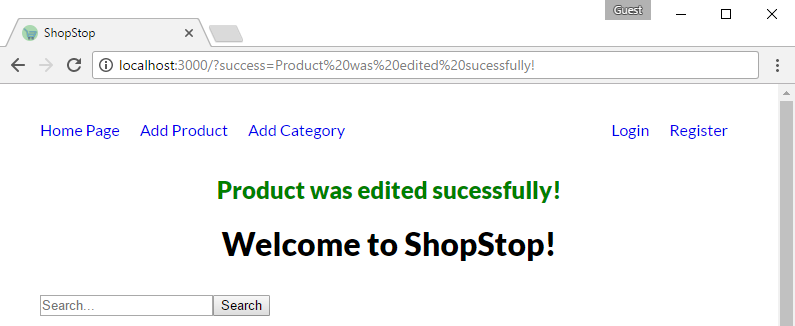
The code above will enable sending "**error**" or "**success**" strings when rendering the view.

When we add the functionality about **editing/deleting/buying** a **product** we will have to make **POST** request which will **redirect** to **home page** with some message. But how the message is going to be passed? The answer is pretty simple – through **query string**: "/?error=This%20is%20error" or "/?success=This%20is%20success"

Go to **home handler** and add the following code:

|  |  |
| --- | --- |
|  | This code will read from the query string for any key-value pair with key: "**error**" or "**success**" and it will take it's value and put it as a message to the layout. |

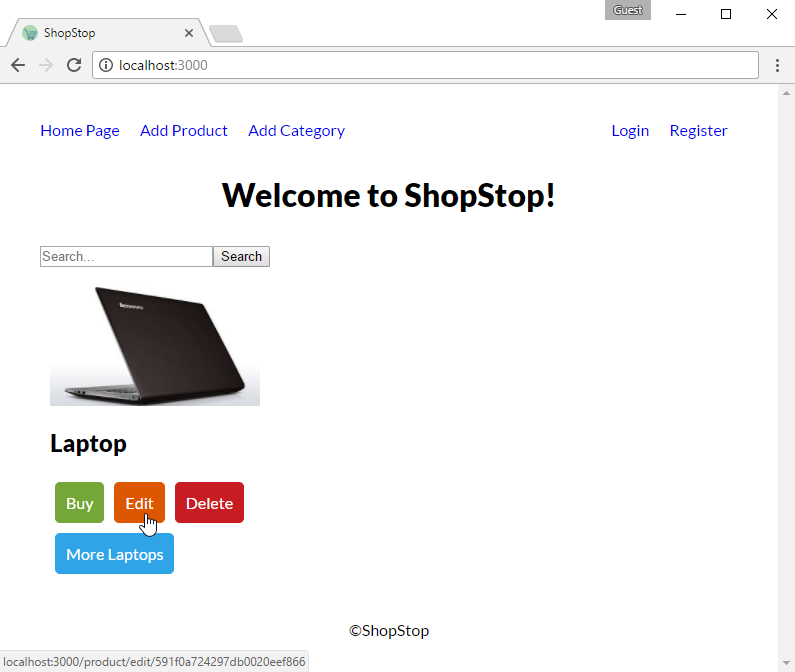
Before moving forward to next part – test if the following message system is working – start the application and use the following url: "/?success=Product%20was%20edited%20sucessfully!". This will lead to:

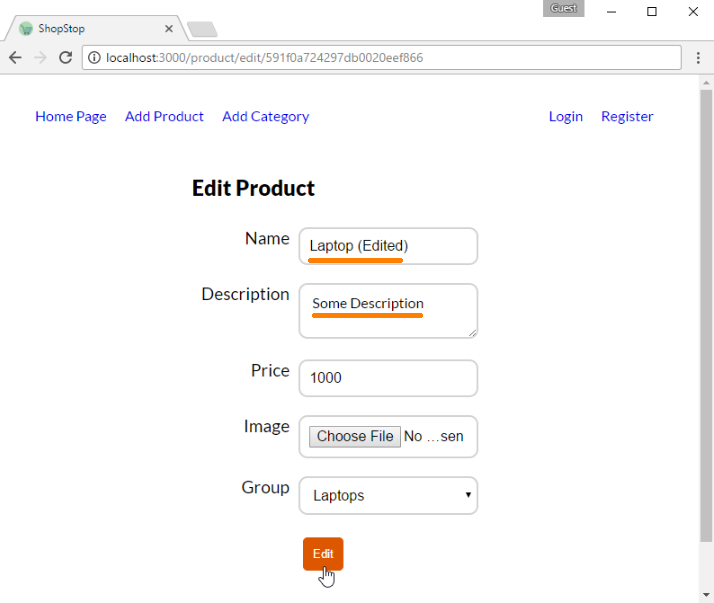


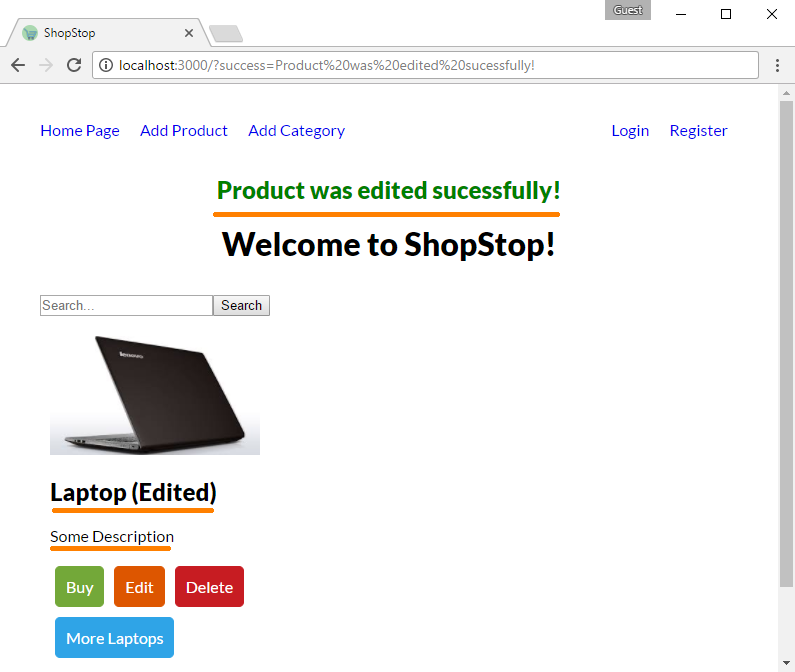
**\*WARNING:** Since the message which is displayed could be edited in the query string – it is not a good practice doing it this way. In the next part of lab (**Part V – User Authentication**) we will see **sessions** and **cookies**. You should **use** **one** of those in order to **improve** current **messaging** **system**.

## Edit Product

Now let’s add the functionality of editing product. Here is what it should look like:







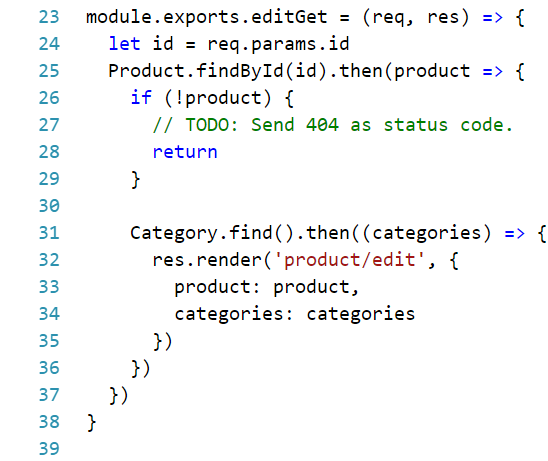
Before diving into writing the logic behind editing product we should consider two key things:

1. Product's picture should be **edited** if **only** a new one is **uploaded** (e.g. if we edit only title there is no need of uploading same picture)
2. With changing product's category we should **remove** **the product's reference** in **old category** and **add** it to the **new** one.

With keeping this arguments in mind we are ready to create our **product/edit** **view**:

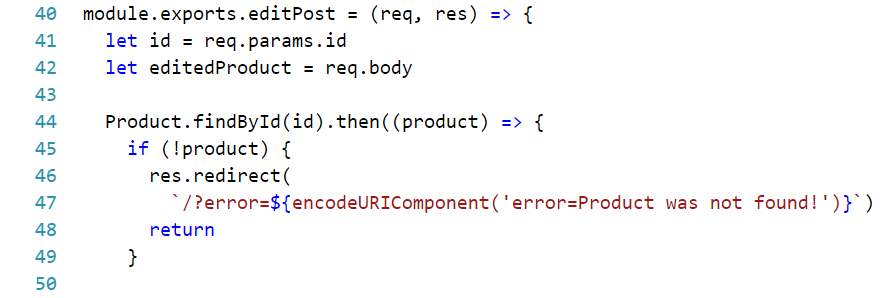
|  |
| --- |
| extends ../layout.pug  block content  form.form.center-form(method='post', enctype='multipart/form-data')  h2 Edit Product  .form-group  label(for='name') Name  input#name.input-field(name='name', type='text', value=product.name )  .form-group  label(for='description') Description  textarea#description.input-field(name='description', type='text')= product.description  .form-group  label(for='price') Price  input#price.input-field(name='price', type='number', step="0.01" min="0" value=product.price)  .form-group  label(for='image') Image  input#image.input-field(name='image', type='file', accept='.jpg,.jpeg,.png')  .form-group  label(for='description') Group  select.input-field(name='category')  each category in categories  if (category.\_id.equals(product.category))  option(value=`${category.\_id}` selected)=category.name  else  option(value=`${category.\_id}`)=category.name  .form-group  input.form-btn.btn.btn-warning(type='submit', value='Edit') |

Now in the **product** **handler** and add action for visualizing the edit form:



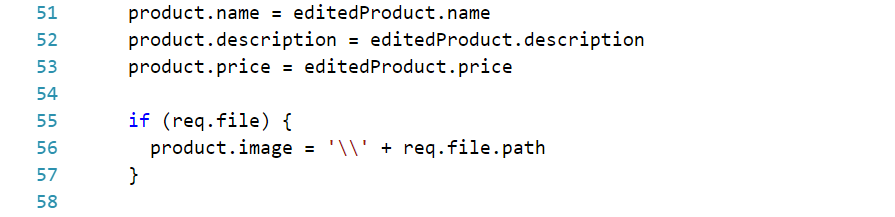
\*Hint: [here](https://stackoverflow.com/a/42370052/5591237) is how you could send status code

This was the easy part, now we have to implement the real logic behind editing a product:



Note the usage of [encodeURIComponent](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent) – it will simply **escape** any **invalid** url **characters** (like "/" or " ")

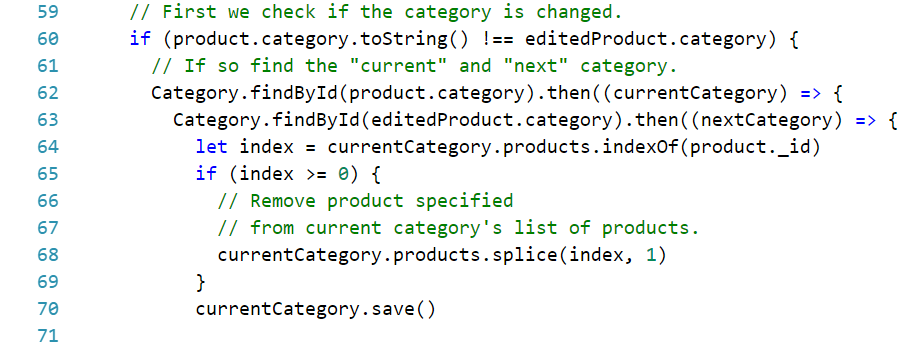
Then we continue with editing the product object:

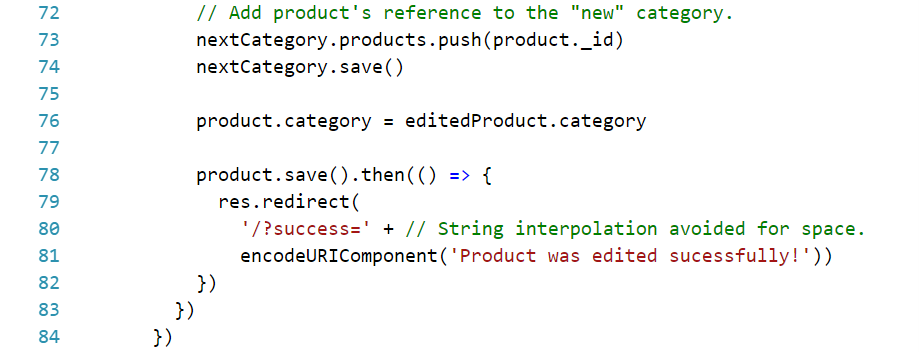


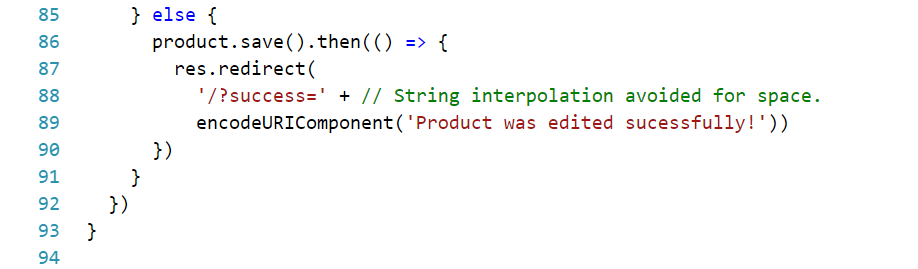
We just **re-assign** **values** passed **from** the **form** in order to **update** the **current** **product** object.

The next part is little messy though. The reason is because we have to deal with the relation between product and category.

Whenever we **change** some **product's** **category** we should go to that **old category** and **remove product's reference** from it and last but least in the **new category** we should **add product's reference**. So here is a suggestion of how it could be done:







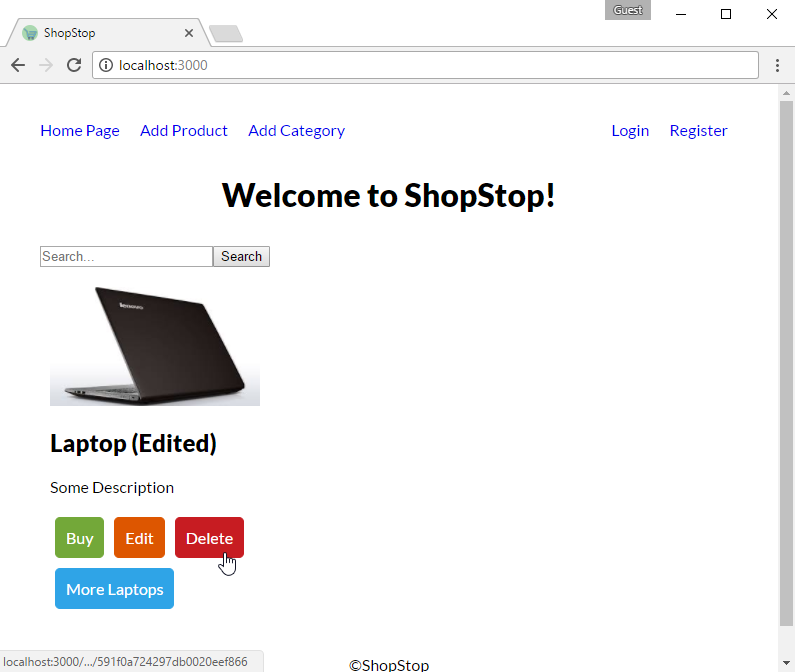
One last thing before we move on – **routing**:

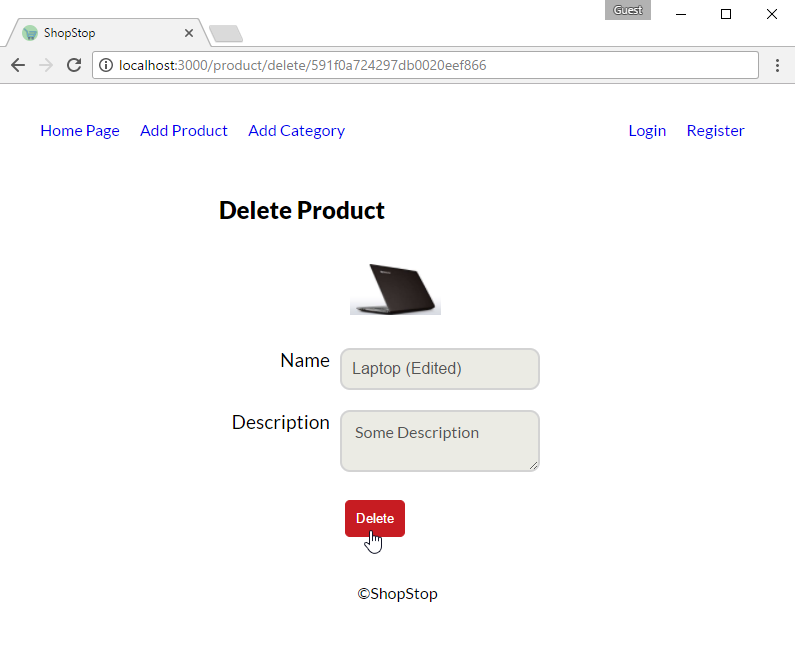


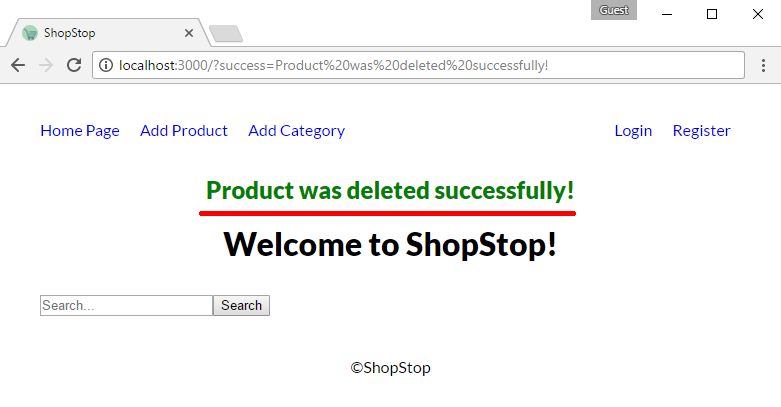
## Delete Product

Moving on next is the deletion of a product. Keep in mind that when you **delete** a **product** you should **delete** it's **reference** **in** the **category** of which is part of.

Here is how it should look like:







Here is the html:

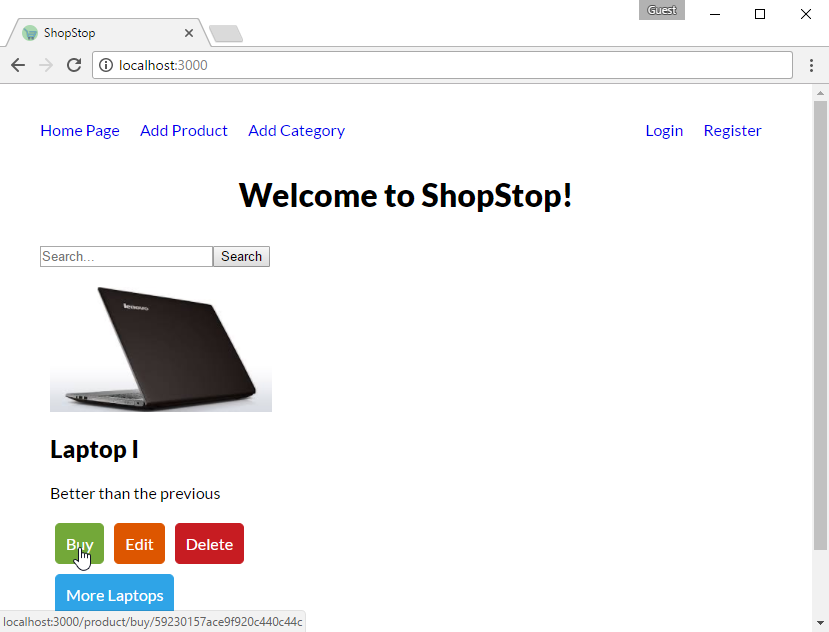
|  |
| --- |
| extends ../layout.pug  block content  form.form.center-form(method='post', enctype='multipart/form-data')  h2 Delete Product  .form-group  label  .product-card  img.product-img(src=`${product.image}`)  .form-group  label(for='name') Name  input#name.input-field(name='name', type='text', value=product.name disabled )  .form-group  label(for='description') Description  textarea#description.input-field(name='description', type='text' disabled )= product.description  .form-group  input.form-btn.btn.btn-danger(type='submit', value='Delete') |

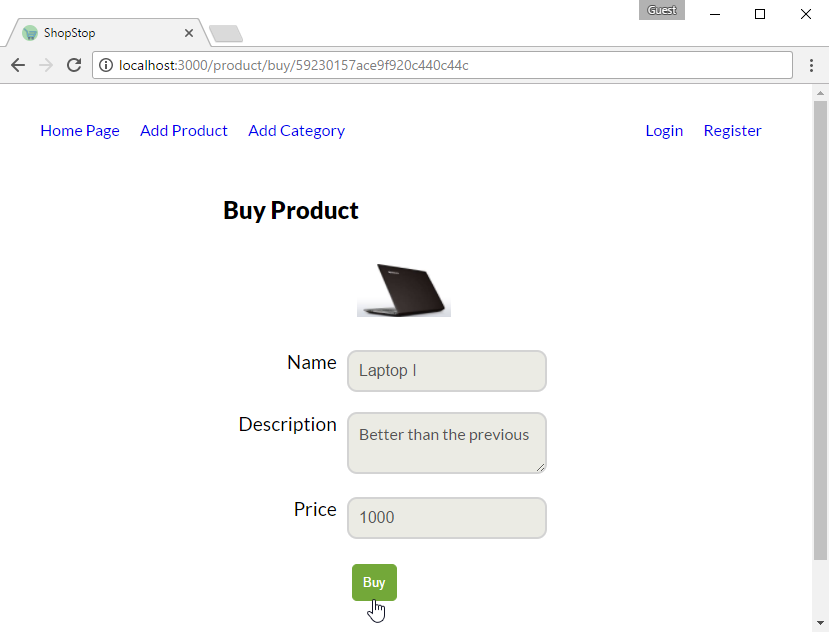
Follow the guidelines from the previous part and you should be able to implement the removal of product on your own.

**\*Hint: when deleting a product, delete product's picture as well.**

## Buy Products

For this part generate only GET request handler displaying the following view:





Use the following html:

|  |
| --- |
| extends ../layout.pug  block content  form.form.center-form(method='post')  h2 Buy Product  .form-group  label  .product-card  img.product-img(src=`${product.image}`)  .form-group  label(for='name') Name  input#name.input-field(name='name', type='text', value=product.name disabled )  .form-group  label(for='description') Description  textarea#description.input-field(name='description', type='text' disabled )= product.description  .form-group  label(for='price') Price  input#price.input-field(name='price', type='number', value=product.price disabled )  .form-group  input.form-btn.btn-success.btn(type='submit', value='Buy') |

**\*Note** that only the **GET** action should be implemented. When we have users we will come back for this one.

**In the next part** we will implement the functionality behind **buying a product** and we will cover up **user authentication**.