# JavaScript Basic CRUD: Products Lab

# Problem

In this exercise we are going to create a simple web page which will contain the following functionalities: **listing products** with **name** and **price**, **creating** a new product, **editing** and **deleting** an existing product (CRUD).

# Overview

## Requirements

* **Express** framework
* **Handlebars** view engine
* **Mongoose** ORM
* **MongoDB**

## Data Model

The Product entity holds 2 properties

* **name – non-empty text**
* **price – number**

## Project Skeleton

You will be given the applications’ skeletons, which holds about **90%** of the logic. You’ll be given some **files** (**controllers**, **models**, **views**, etc.). The files will have **partially implemented logic**, so you’ll need to write some code for the application to **function properly**.

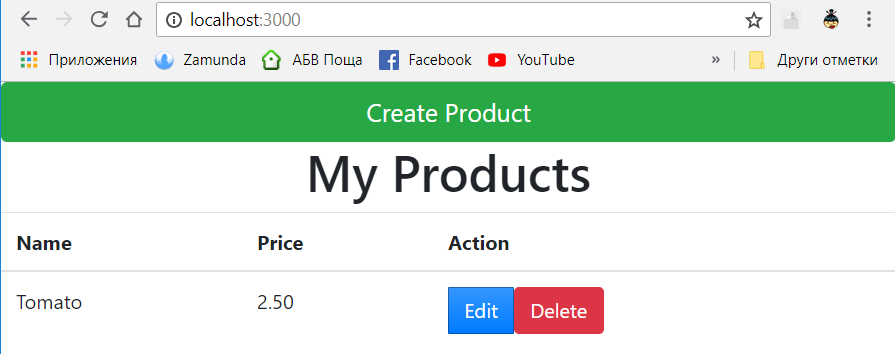
The application’s views will be given to you fully implemented. You only need to include them in your business logic.

Everything that has been given to you inside the skeleton is **correctly implemented** and if you write your code **correctly**, the application should work just fine. You are free to change anything in the Skeleton on your account.

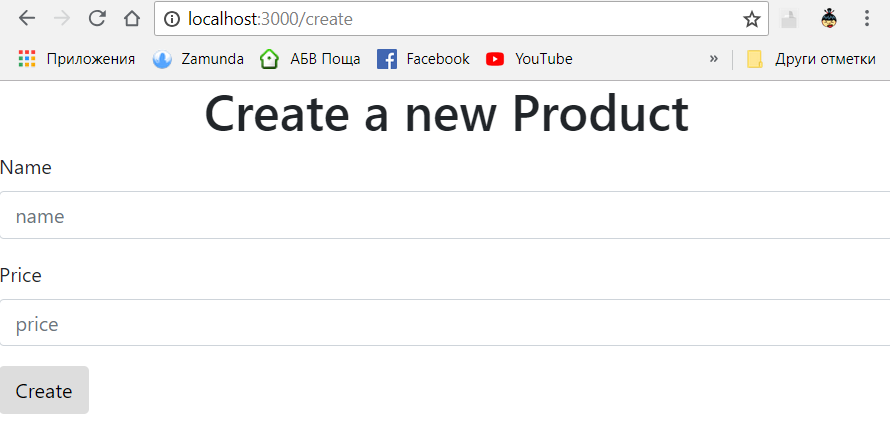
## User Interface

This is the user interface or how the application’s pages should look in their final form (fully implemented). You have several pages, described below:

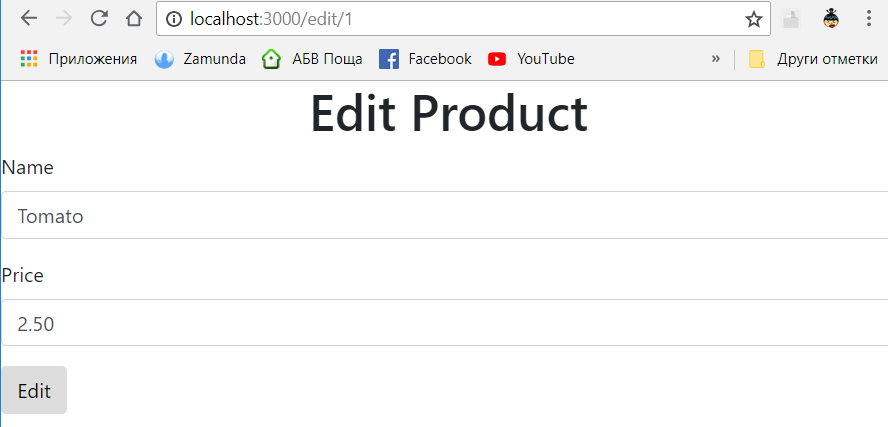
### Index Page



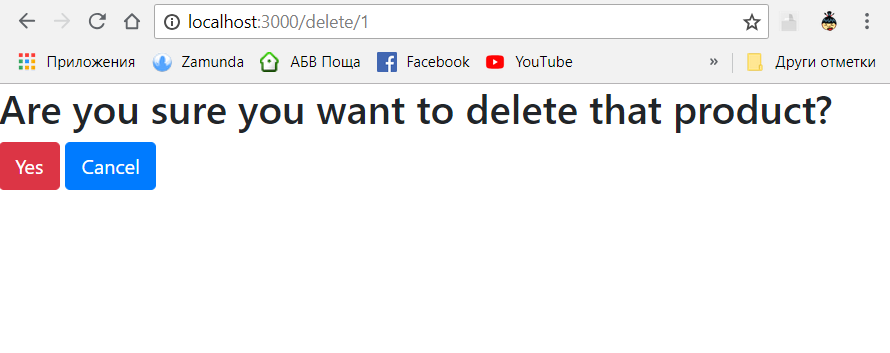
### Create Page



### Edit Page:



### Delete Page



# Implementation

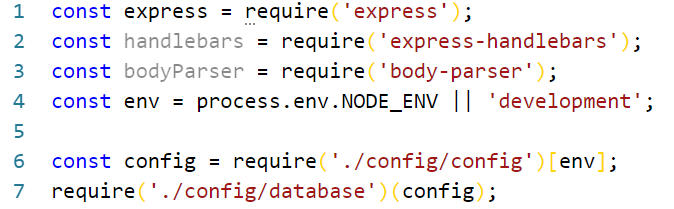
## Install Dependencies

Type **npm install** to install following dependencies inside **package.json**:



## Explore the Startup File

Explore **index.js**. It contains all of the server logic and the routing:



* **express** – we are going to use it for the server
* **handlebars** – this is going to be the view engine (handlebars)
* **bodyParser** – we need it to read info from forms (like our create product form)
* **env** – this is going to be the environment of our project
* **config** – this is the configuration we are going to use (we will implement it later on)

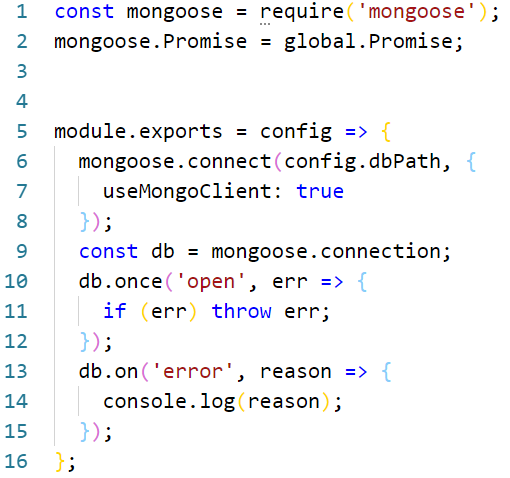
## Database Configurations

Now before creating a new product, we need somewhere to store our products. We have a folder name **"config"** and inside we have a **JavaScript file** named **"config.js":**



* This will **set** the **port** for our app and will **create** a database (make sure you have started MongoDB)

We also have a **database.js** file:



* This code will **connect** mongoose to the database and will **log** a message when the connection is **made**. In case of **error**, it will log an error message.

## Layout

Now let us fill the **main.hbs**.

Inside the folder **"views/layouts"** there is a **"main.hbs"** copy the following code:

<html lang="en">

<head>

<meta charset="UTF-8">

<link rel="stylesheet" href="/css/bootstrap.css">

<title>Document</title>

</head>

<body>

{{{body}}}

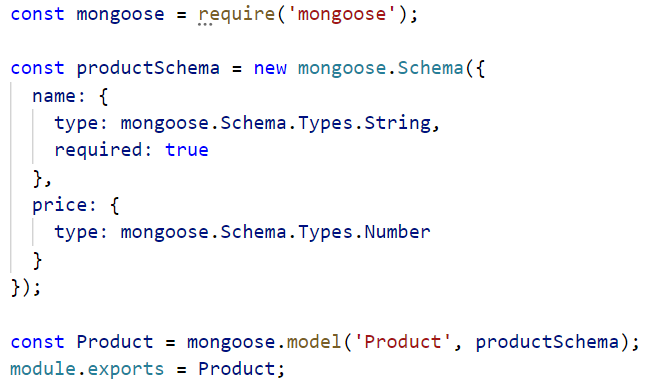
</body>

</html>

* Link the css from bootsrap
* **{{{body}}}** – this is where the chosen from us view is going to render

## Product Model

Inside the **folder called models** and in there **create** a file called **"Product.js"**. In there, add the following code:



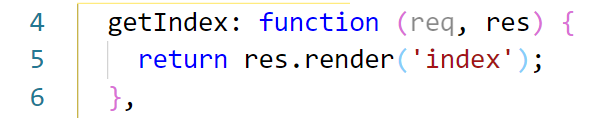
* We create **a schema for the product**. Each product will have a **name (required) and a price**
* Then we create a **model using that schema and we export it**

## Controller

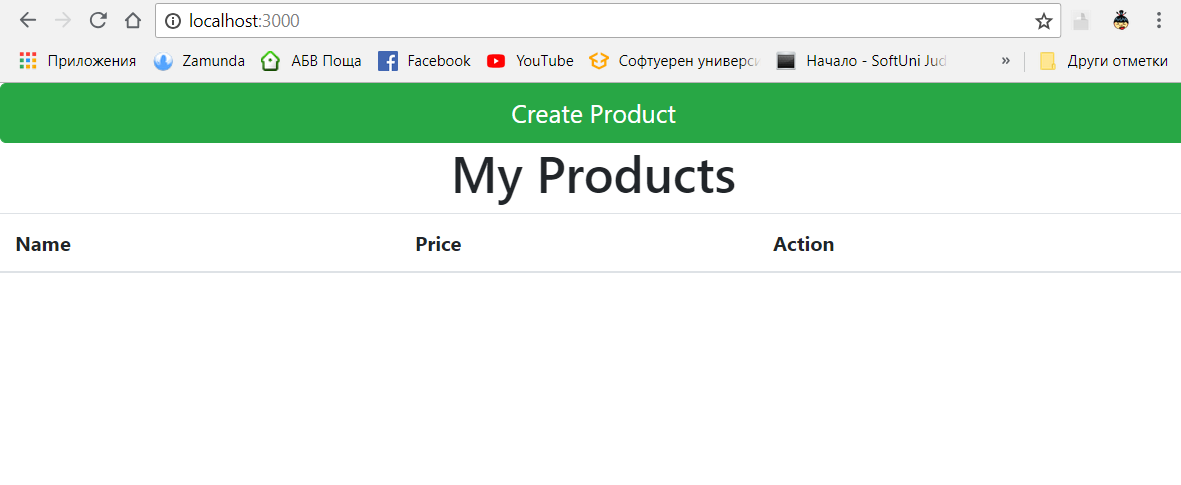
In the controllers folder there is a **home-controller.js** file. There you will find the following code:



* These are the functions we are going to use. We are going to implement them as we need them.
* For now let us just write the **getIndex:**



Restart the server and refresh the browser. You should now see this:



## Create Product View

Inside **"views"** create a **new** file **"create.hbs"** with the following code:

<h1 style="text-align:center">Create a new Product</h1>

<form action="/create" method="POST">

<div class="form-group">

<label for="exampleInputName1">Name</label>

<input type="text" class="form-control" placeholder="name" name="name">

</div>

<div class="form-group">

<label for="exampleInputName1">Price</label>

<input type="text" class="form-control" name="price" placeholder="price">

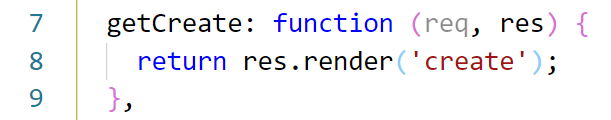
</div>

<button type="submit" class="btn btn-default">Create</button>

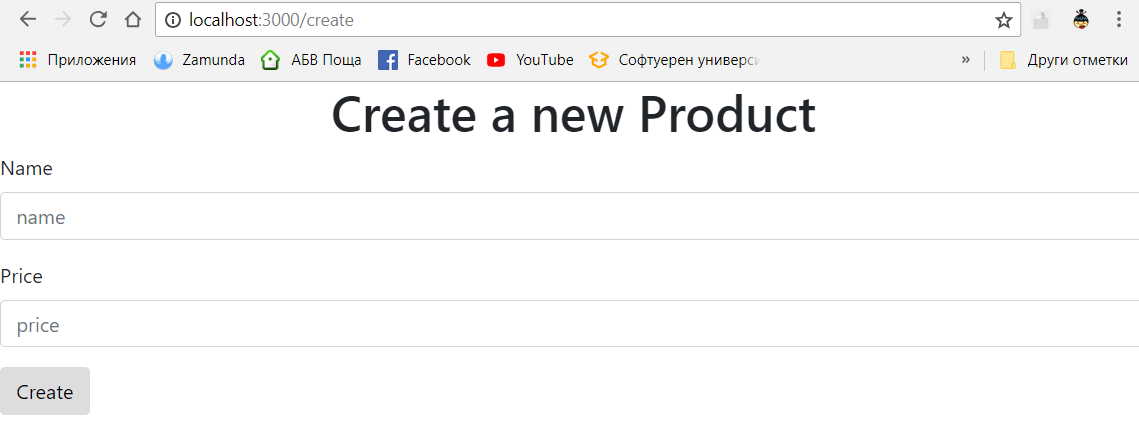
</form>

* **<form action="/create" method="POST">** - this means that when the form is submitted we will make POST on route "/create")

Now go in the **home-controller.js** file and implement get for the **"/create"** route that will render the **"create.hbs"** file:



Restart server, reload browser, click on **"Create Product"** and you should see this:



## Rendering the Products

The next step is to render all the products we have in the database.

First, let us change the **index.hbs** a little bit. Inside the **<tbody>** add the following HTML:

{{#each products}}

<tr>

<td>{{this.name}}</td>

<td>{{this.price}}</td>

<td>

<a href="/edit/{{this.id}}"><button type="button" class="btn btn-primary">Edit</button></a>

<a href="/delete/{{this.id}}"><button type="button" class="btn btn-danger">Delete</button></a></td>

</tr>

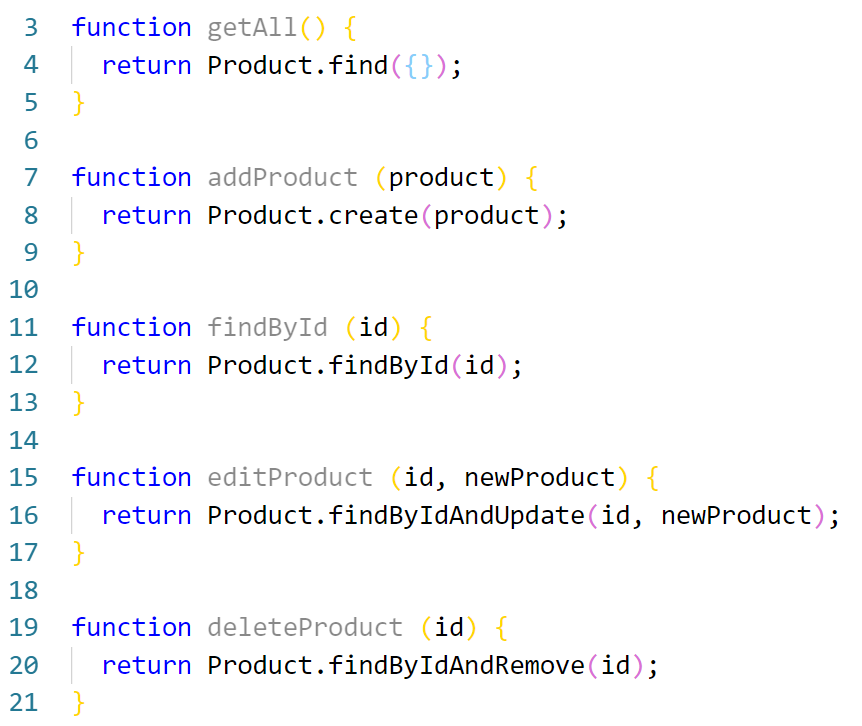
{{/each}}

* It will **loop through each product** and render its **name, price and two buttons(edit and delete)**

In order to loop through the products, we need to **pass them to the view**.

So let us create the **"service"** functions that will do this.

Inside **"home-controller.js"** add the following code:

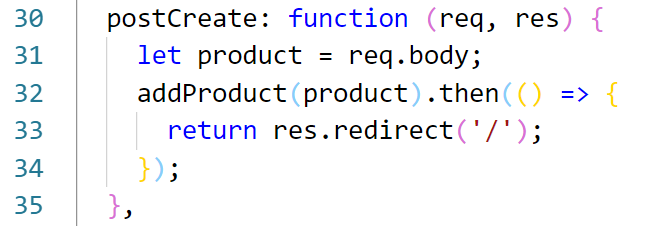


* These are all the methods we are going to use to **create**, **read**, **update** and **delete** a product
* As you see, we use **Product**. You have to require it at the **beginning** of the file:



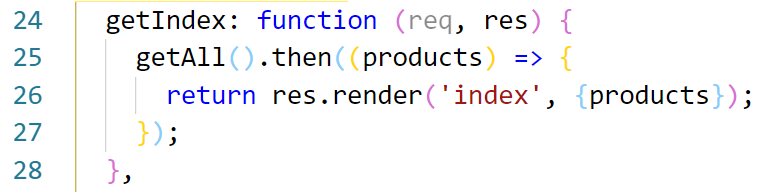
## Creating a Product

Now let us implement the **post on "/create"**. Go back to the **home-controller.js** and add the following:

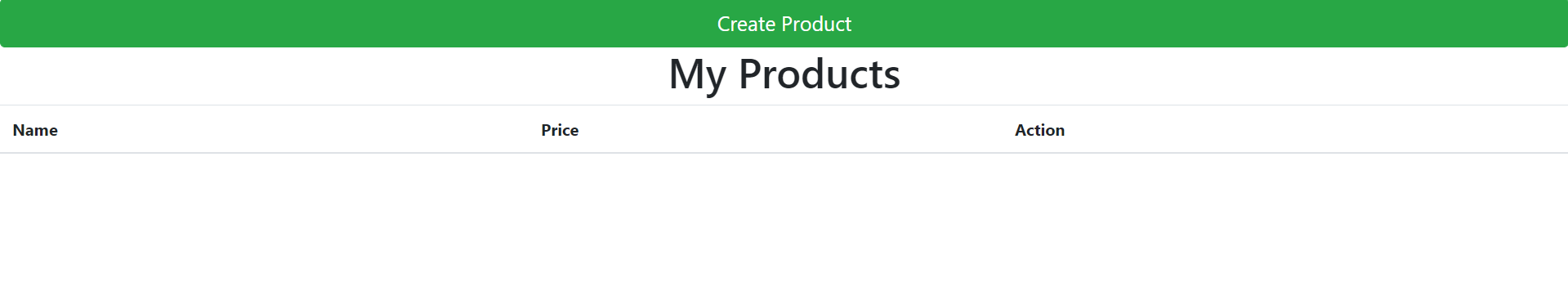


## Listing All Products

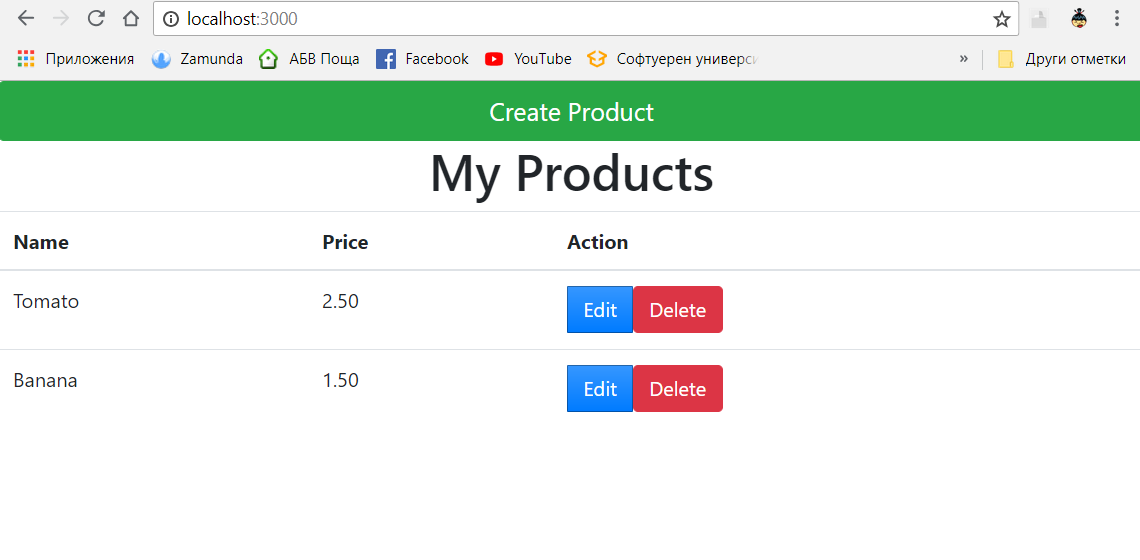
Next, we are going to modify our get on **"/"**, so we pass the products to the view:



**Restart** the server and **reload** the page. You should see something like this:



Try **adding** some products:



## Edit Product

First **create** the **"edit.hbs"** (in the **"views" folder** again).

It will be the same as the **"create.hbs"**, but the fields will be **filled with the existing data**

<h1 style="text-align:center">Edit Product</h1>

<form action="/edit/{{product.id}}" method="POST">

<div class="form-group">

<label for="exampleInputName1">Name</label>

<input type="text" class="form-control" value="{{product.name}}" name="name">

</div>

<div class="form-group">

<label for="exampleInputName1">Price</label>

<input type="text" class="form-control" value="{{product.price}}" name="price">

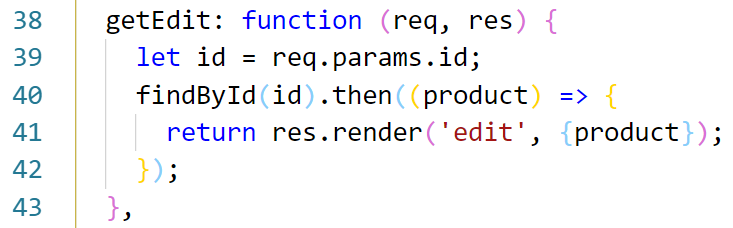
</div>

<button type="submit" class="btn btn-default">Edit</button>

</form>

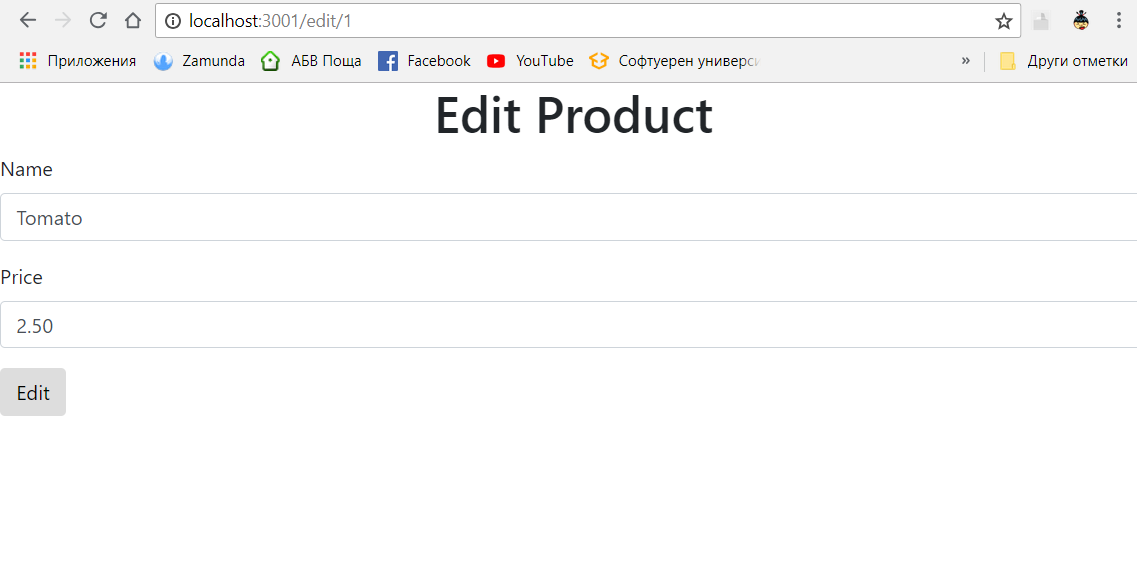
* In order to fill the fields, we need to have the existing product (we will pass it to the view)
* The action of the form will be to **"/edit/{productID}" with method "POST"**, so we know which product to edit

Now let us implement the get on edit:

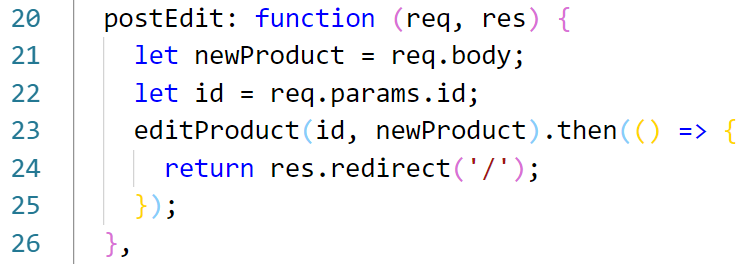


* **id** – we get the id from the **request url**
* We use the **findById** function.
* Then after we **get the product**, we **pass it to the view** and render it

Restart server, reload and try to edit a product:



The post does **not** work yet. So that is the next step:



* We use the function **editProduct**

Restart the server, reload and try to edit a product.

## Delete Product

The last part is to delete a product. Let us first **create** the view **"delete.hbs":**

<h2>Are you sure you want to delete that product?</h2>

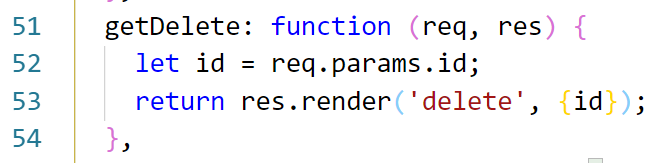
<form action="/delete/{{id}}" method="POST">

<button type="submit" class="btn btn-danger">Yes</button>

<a href="/"><button type="button" class="btn btn- primary">Cancel</button></a>

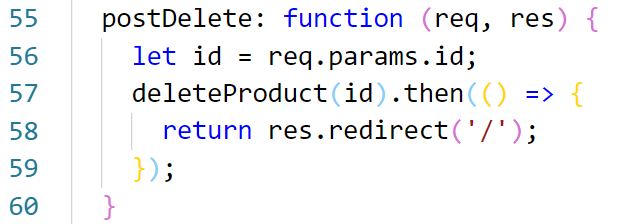
</form>

Go to **home-controller.js** and add the following code:



* We pass the id, because we want to know which **product** we want to **delete**

Now let us create the post on delete:



Restart the server, reload and now you have a **web page with all of the CRUD operation**