# C# Web Basics – Exam Preparation

# MUSACA

**MUSACA** (**M**ulti-**U**ser **S**elling and **A**ccounting **C**ashier **A**pplication) is a web platform application for cashiers at selling points. You know how a cashier at a supermarket checks in your products and then tells you how much your cost is, well this one is the same, but kind of like a web app. Anyway, you’ve been employed by the **SB** (**S**lavic **B**abushka) group to implement the **MUSACA** platform for a client of theirs, which is a really big hypermarket company.

However, there are specific requirements that must be followed.

## Technological Requirements

* Use the SIS.WebServer
* Use the SIS.MvcFramework
* Use Entity Framework Core

The Technological Requirements are **ABSOLUTE**. If you do not follow them, you will **NOT** be scored for other Requirements.

Now that you know the Technological Requirements, let us see what the Functional Requirements are.

## Database Requirements

The **Database** of the MUSACA application needs to support **3 entities**:

### User

* Id - a GUID String, Primary Key
* Username - a **string** with **min length** **5** and **max length 20** (**required**)
* Email - a **string** with **min length** **5** and **max length 20** (**required**)
* Password - a string – hashed in the database (**required**)

### Product

* Id - a GUID String**, Primary Key**
* Name - a **string** with **min length** **3** and **max length 10** (**required**)
* Price – a **decimal** with **min** **value** – **0.01**.

### Order

* Id – a GUID String or an **Integer**
* Status – can be one of the following values ("Active", "Completed")
* Issued On – a Date object.
* Products – a collection of Product objects
* **CashierId** – a **GUID foreign key (required)**
* Cashier – a User object

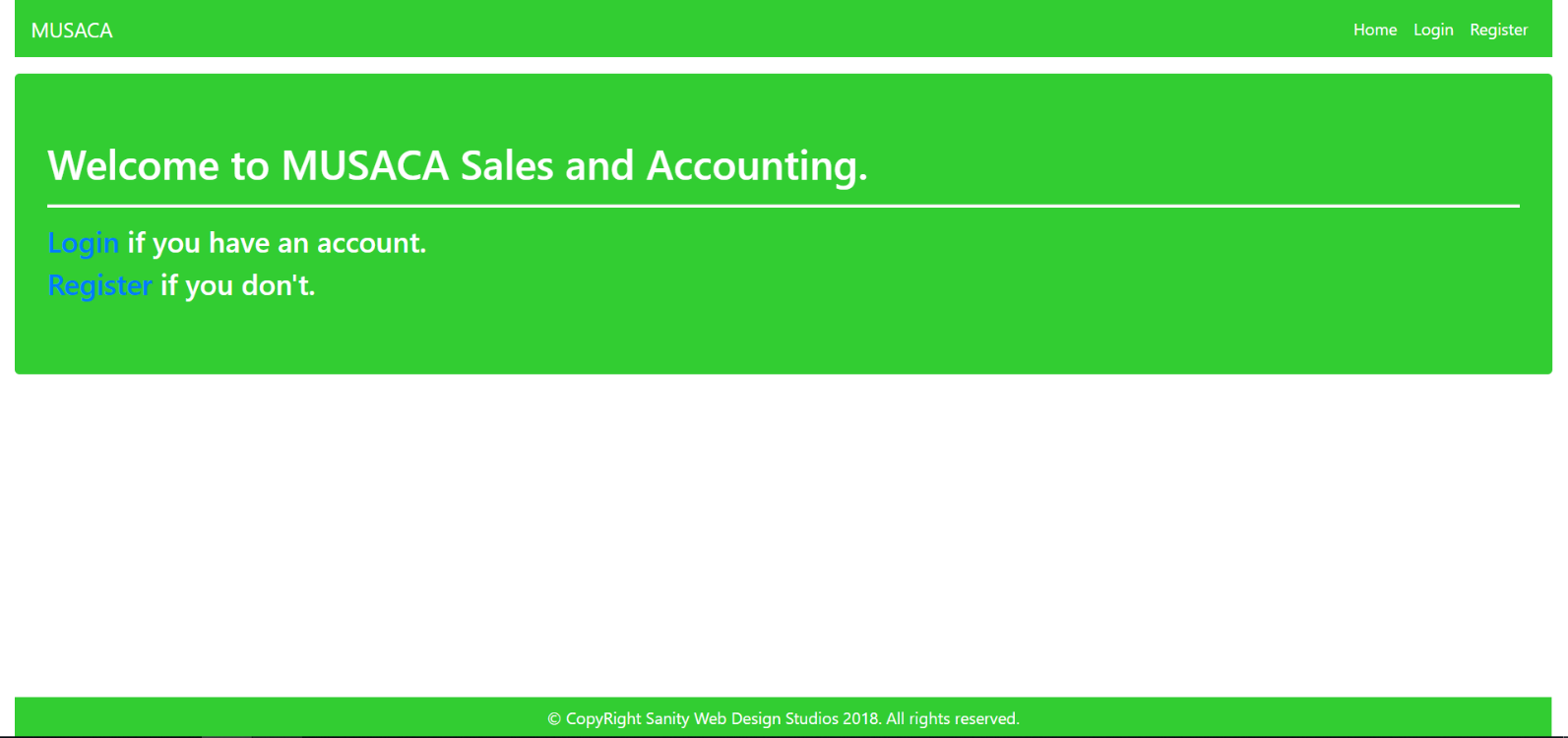
Implement the entities with the **correct datatypes**.

## Page Requirements

### Guest Pages

These are the **templates** and **functionalities**, accessible by Guests (**logged out** users).

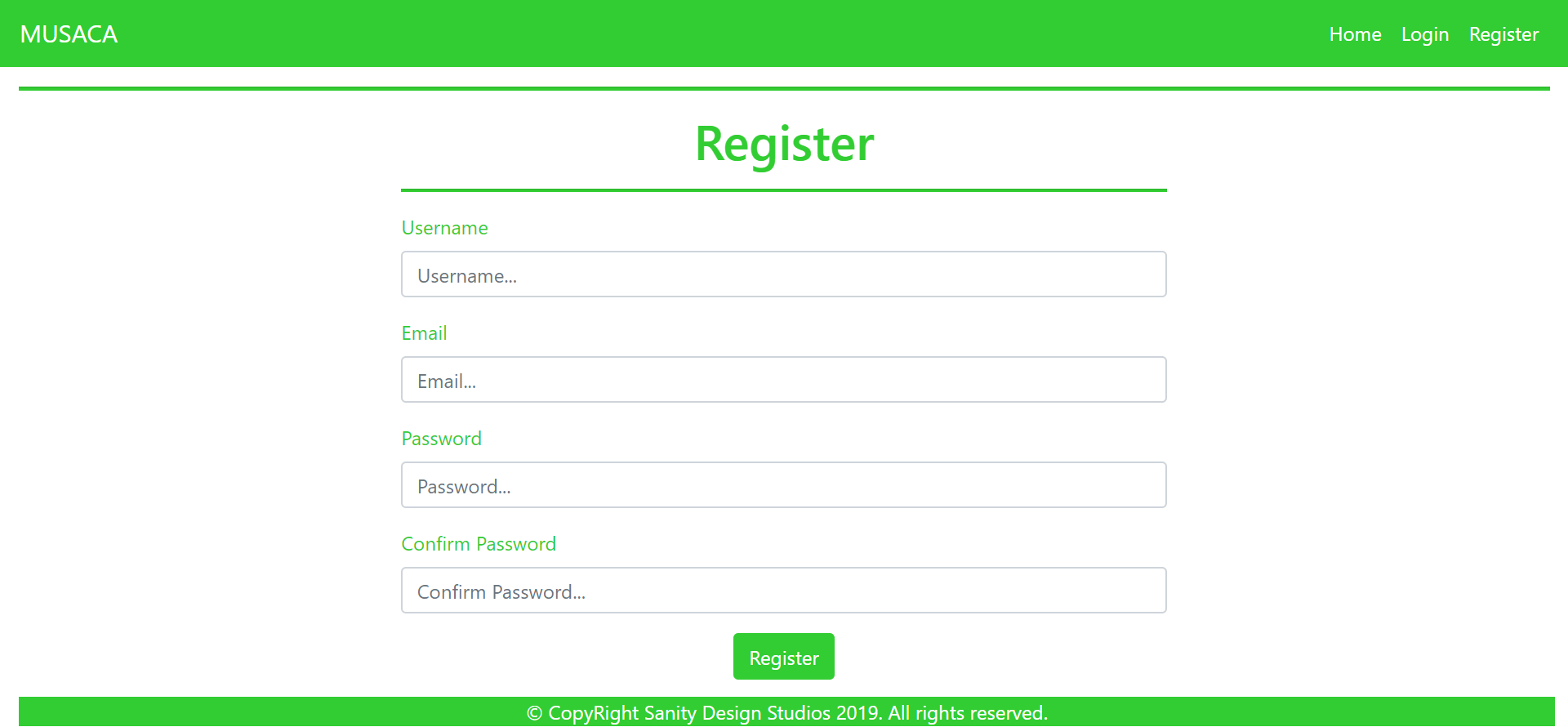
#### Index Page (route = “/Home/Index”) (logged out user)



#### Login Page (route = “/Users/Login”) (logged out user)

#### 

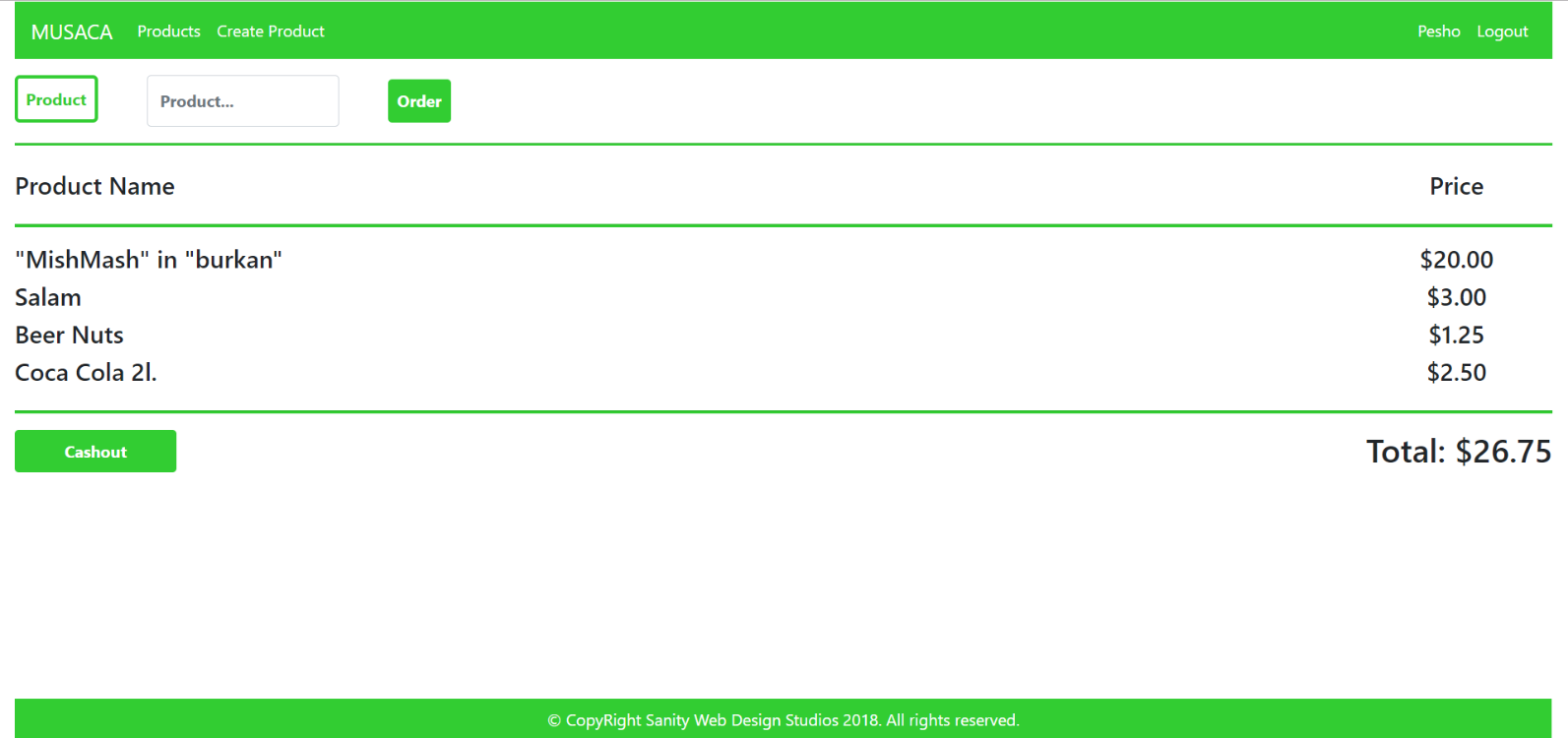
#### Register Page (route = “/Users/Register”) (logged out user)



### User Pages

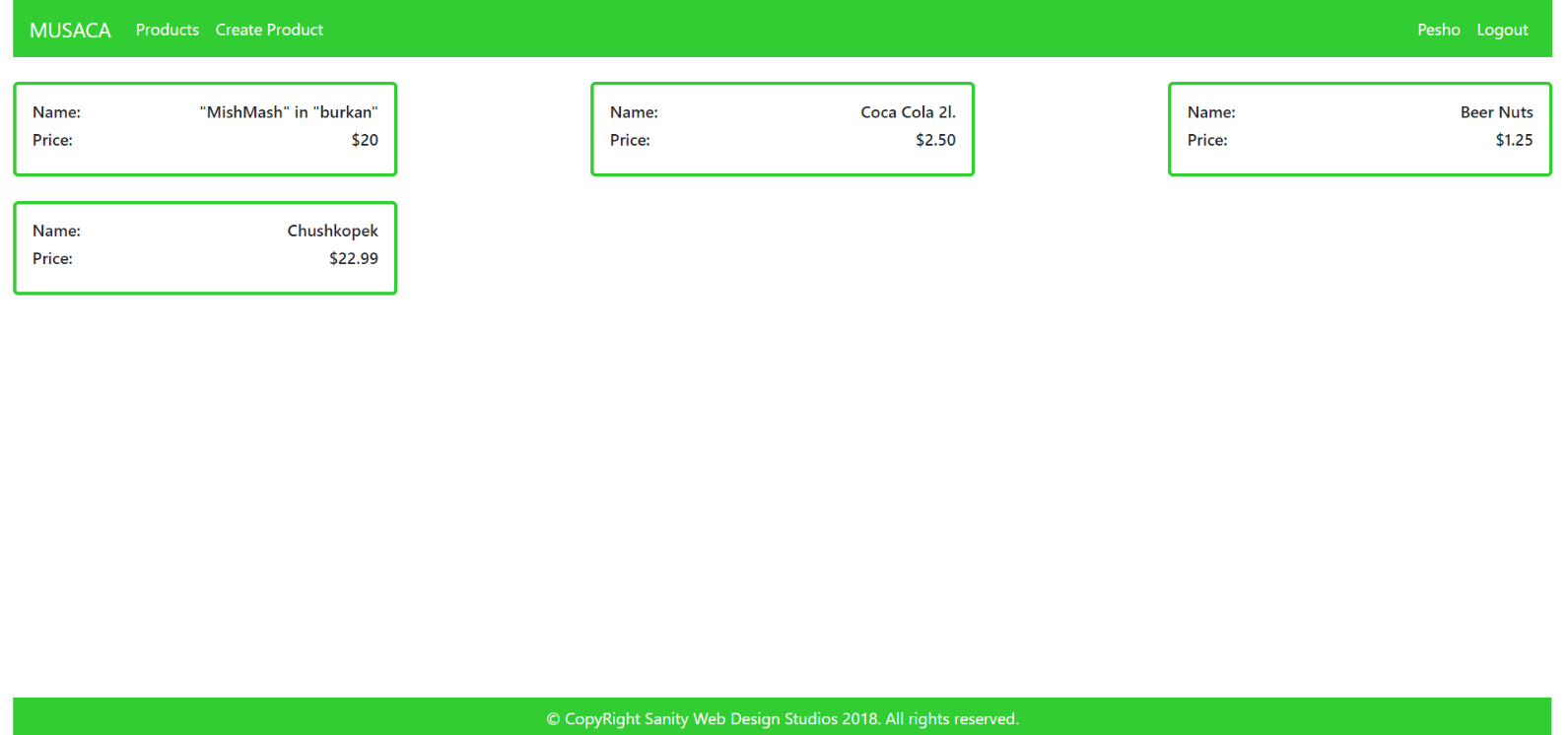
These are the **templates** and **functionalities**, accessible by Users (**logged in** users).

#### LoggedIn Index Page (route=”/Home/Index”) (logged in user)r

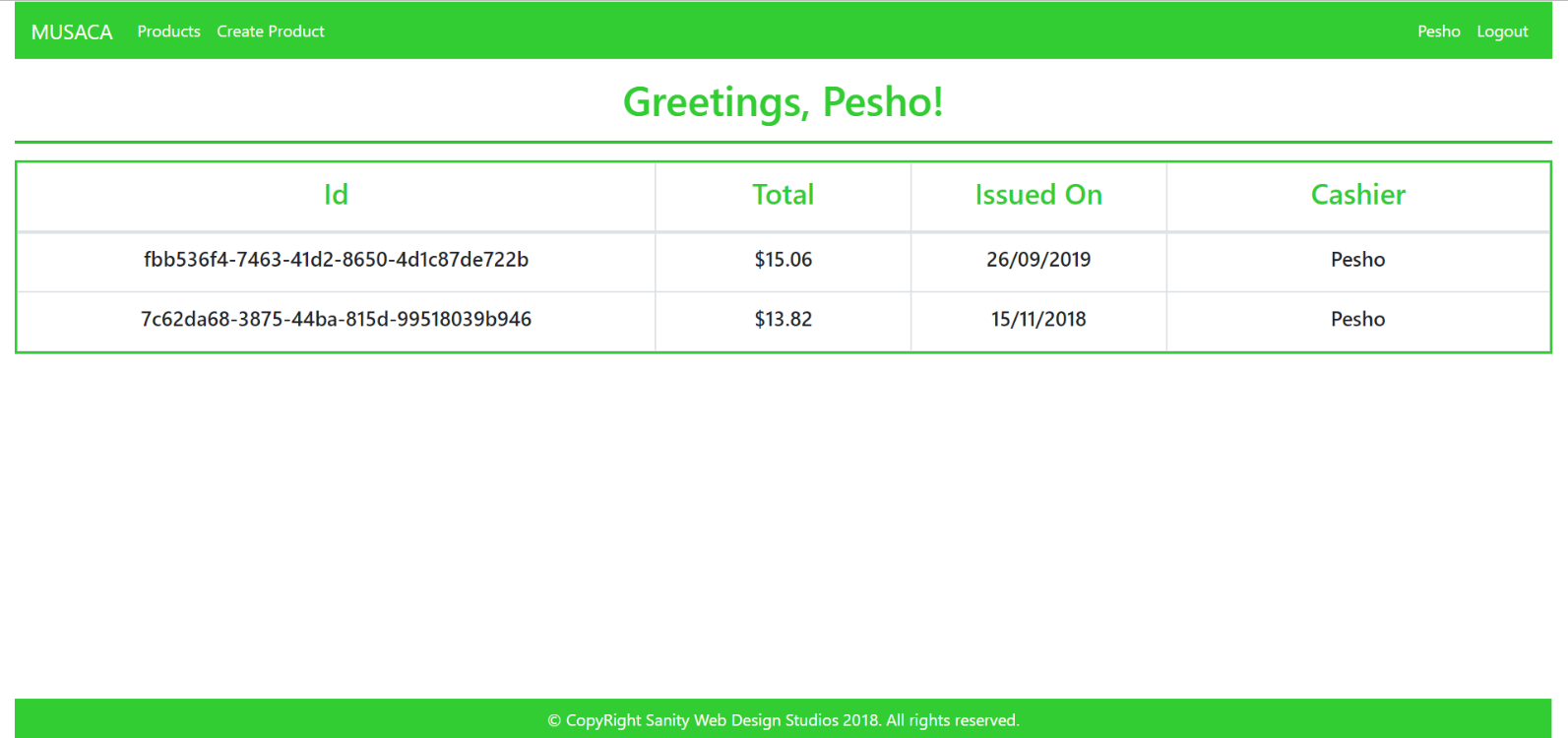


#### Product Create Page (route=”/Products/Create”) (logged in user)

#### Products All Page (route=”/Products/All”) (logged in user)



#### Profile Page (route=”/Users/Profile”) (logged in user)



Some of the templates have been given to you in the application skeleton, but the others will be for you to implement, so make sure you implement them correctly. You can use the given ones as helpers.

**NOTE**: The templates should look **EXACTLY** as shown above.

**NOTE**: The templates do **NOT** **require** **additional** **CSS**. Only **bootstrap** and the given **style.css** are enough.

**NOTE**: In the given **style.css** you’ll see some helpful classes “bg-musaca”, “text-musaca”, “border-musaca”.

## Functional Requirements

The functionality of the **MUSACA** application is quite simple. The Users are the Cashiers, and they are “checking out products” literally.

### Main functionality

When a User logs in, he sees several things on his Index page.

* A form for **ordering** a Product by name.
* A list of the Products for the current Order.
  + With Product Mame, Product Price
* A [Cashout] button which completes the Order.

### Products

The Products are just data entities. They are only created and persisted. The Orders do not affect them in any way. The Products are like templates, so that the User knows what he is ordering.

### Orders

For each User, there is **exactly 1 Active** Order at all time.

By entering a name of an existent Product, in the **form** on the **top** of the Index page, a User adds a Product to his currently **Active** Order. Upon completing this functionality, the application should **redirect** back to the Index page, where the User (Cashier) can see his Order and the Products.

An Order is created with status – "Active" by **default**. The currently Active Order is visualized on the Index page as a list of its Products, with their Product’s Name and their Product’s price. A Total Price is also visualized, below the list, which is a sum of every Product’s Price.

**NOTE**: Only the currently Active Orders (with status – "Active") is visualized on the Index page.

Clicking the [Cashout] button on the bottom of the Index page should **change** the currently Active Order’s status to "Completed".

Another Order should be **created** with the **currently logged-in** User as Cashier and it should become the current Active Order.

**NOTE**: Upon Registering a User, you should create the initial Active Order for it.

### Users

All Users can View All Products, Create Products, Cashout and View All Personal Orders.

Each User can check data about all the Products, by clicking on the [Products] button from the **navbar**. This should **redirect** him to the Products All page.

Each User can check data about all **HIS Completed** Orders, by clicking on the [{Username}] button from the **navbar**. This should redirect him to the Profile page. On that page, the User can check a list of only **HIS** Orders, and only the Completed ones (status = "Completed").

Each User can also create new Products, by clicking on the [Create Product] button from the **navbar**. This should redirect him to the Product Create page. NOTE: Creating a Product, should **redirect** to the Products All.

**NOTE**: All decimal data is visualized up to the **second digit** after the decimal point.

## Security Requirements

The Security Requirements are mainly access requirements. Configurations about which users can access specific functionalities and pages.

* Guest (not logged in) users can access Index page and functionality.
* Guest (not logged in) users can access Login page and functionality.
* Guest (not logged in) users can access Register page and functionality.
* Users (logged in) can access User LoggedIn Index page and functionality.
* Users (logged in) can access the Product Create page and functionality.
* Users (logged in) can access Products page and functionality.
* Users (logged in) can access Order functionality.
* Users (logged in) can access User Profile page and functionality.
* Users (logged in) can access Logout functionality.

## Code Quality

Make sure you provide the best architecture possible. Structure your code into different classes, follow the principles of high-quality code. You will be scored for the Code Quality and Architecture of your project.

**NOTE:** The **Service Layer** is **required**

## Data Validation

* In case of invalid data during the **login**, redirect to "**Users/Login**"
* In case of invalid data during the **registration**, redirect to "**Users/Register**"
* In case of invalid data during the **product creation**, redirect to "**Product/Create**"

**NOTE:** Use the ValidationSisAttributes.

## Scoring

This section describes how the scoring of the Exam will be made.

### Database Requirements – 10 points.

### Data Validation – 20 points.

### Template Requirements – 10 points.

### Functionality – 40 points.

### Security – 10 points.

### Application Structure – 10 points.