



CPSC-597C A PROJECT III

ASP.NET MVC 5 – TrendStyle Shopping Website with PayPal

Professor : Mr. Ausif Mahmood

NAME: ANANTHI SELVAMANI

ID: 1004738

Table Of Contents

S:No	Title	Page No
1	Introduction	3
2	Project SetUp	4
	-Software Requirements	4
3	Admin Area	5
	-DTO	6
	-Tables	7
	-Methods in Pages Controller	9
	-DropZone	21
4	Front End Part of the WebSite	23
	-Account SetUp	27
	-Enable User Roles	29
	-Add PayPal to Cart	30
5	SnapShots	32

Introduction

In this project a Shopping Website is designed using MVC pattern. There are two parts to this Website. One part in which where the admin can only access and the other part where the user can login and view the products and place the order. Based on the userRoles this functionality has been added to the project. Once the user places an order the user is navigated to the pay pal page where the payment process takes place.

Project Setup

Software Requirements:

Language of Implementation: C#

Operating System: Windows 10

Database: Local DB (SQL Server)

ADMIN AREA

The first step in the project is to Create Area which is equivalent to a mini mvc application inside the main mvc application. It gives better structure to larger applications. Code files inside area can be maintained and upgraded successfully from the rest of the app.

After creating admin area - the SHARED folder structure will be empty. So copy paste the _layout.cshtml, ViewStart.cshtml file from the actually project and place it here. An adminarearegistration page gets created which actually returns the view for the admin access and has the default routes configuration for the admin part. On the ViewStart.cshtml edit the name of the layout that has to be rendered when the admin access the site.

For the database part click on App_Data folder –Create New Item – Select SQL Server Database – create a database (TrendStyle.mdf). Once the DB is created click on the file and select “Add Ignored Files to Source Control”

Now create a Dashboard controller inside Admin area. Create a view specific to that controller. In the BundleConfig.cs file add scripts which includes jquery and bootstrap and in the layout page render this bundle created.

```
bundles.Add(new ScriptBundle("~/bundles/scripts").Include(  
    "~/Scripts/jquery-{version}.js",  
    "~/Scripts/bootstrap.js"));
```

List of DTO created in Project:

- PagesDTO
- SideBarDTO
- CategoryDTO
- ProductDTO
- Db.cs
- OrderDetailsDTO
- OrderDTO
- RolesDTO
- UserDTO
- UserRolesDTO

List of Controllers in Admin Area:

- DashboardController
- PagesController
- ShopController

Add Entity Framework.6.1.3 into the project references through References-Manage Nuget Packages. Entity Framework is a ORM framework(Object Relational Mapping). The main advantage of using entity framework is that it returns data in the database as an object or a collection of objects.

Tables created in TrendStyle.mdf

- tblPages(Fields: Id(int), Title(varchar(50)), Slug (varchar(50)), Body varchar(MAX), Sorting int, HasSideBar(bit)
- tblSideBar(Fields: Id int, Body varchar(MAX))
- tblCategories(Id int, Name varchar(50), Slug varchar(50), Sorting int)
- tblProducts(Id int, Name varchar(50), Slug varchar(50), Description varchar(Max), Price number(18,2), CategoryName varchar(50), CategoryId int, ImageName varchar(100))
- tblUsers(Id int, FirstName varchar(50), LastName varchar(50), EmailAddress varchar(50), UserName varchar(50), Password varchar(50))
- tblRoles(Id int, Name varchar(50))
- tblUserRoles(UserId Int, RoleId int)
- tblOrders(OrderId Int, UserId Int, CreatedAt datetime2(7))
- tblOrderDetails(Id Int, OrderId int, UserId int, ProductId int, Quantity Int))

After creating a table create the DTO(Data Transfer objects) for the table. In that DTO created create property for all the table fields. If a particular field is the primary key specify that as KEY in the definition of the class. If the table created and the DTO class name is different specify the table name above the dto created as below.

```
[Table("tblPages")]  
public class PageDTO  
{
```

```

[Key]
public int Id { get; set; }
public string Title { get; set; }
public string Slug { get; set; }
public string Body { get; set; }
public int Sorting { get; set; }
public bool arHasSideb { get; set; }
}

```

Now create ViewModel as PageVM which has the same property as the pageDTO. Create constructor in PageVM and pass DTO to it so that when value gets filled in the db tables the DTO will take all the values and pass it to the fields.

Add the DTO created in the db class through which you can access the table created. While adding the DTO pass it as an entity to the DbSet class which represents an entity set that is used to create, read, update and delete operations.

```

public DbSet<PageDTO> Pages { get; set; }

```

In the default index method now create a list of PageVM and inside the db statements initialize the list which equals to a lambda expression which returns all the pages created through the pageVM and pass the expression as a DTO which calls the constructor in the pageVM file and initialize the list. Finally return that pageList. Add view to this page with the View name as “INDEX”, template type as “List” and Model class to PageVM and check the box Use a Layout page and ADD the view.

In the view create a for loop which displays all the property value from the model and add a if loop to check if the model is empty display “There are no pages to be displayed” else display all the model items.

Methods in PagesController

Add Page:

Add page – create a view as ADD PAGE, Template as Create, Model as pageVM and check the checkbox Use a layout page. A sample view page is created with some tags which denotes the form input fields. The tags @Editorfor- denotes the text area and @labelfor-denotes the label fields. By default it has a ValidationSummary and if suppose a form validation fields the specific validation message gets displayed. We can add even some custom model errors but for now we are going with the default validation messages which comes with the file.

There is a link provided to navigate back to the main page.

Now create a postback method for the AddPage function which takes the pageVM as the parameter. In the postback method first check if the model is valid. Then in the db context declare slug and initialize the pageDTO. If slug value is not specified then take the title as the slug parameter and convert it to lower case and add it as slug. Check if the slug and the title are unique in that case add custom model error which states “THE TITLE AND THE SLUG ALREADY EXISTS. Now set the values to the dto fields and for sorting set it to 100. The ideology behind it is whenever you add a page it will have 100 as sorting value which will be considered as the last added page as no website will have more than 100 pages. I have added a TempData message which carries the success message and in the

view if that tempData is not null display the success message. Now save the dto and the db changes and redirect to the get Add Page method.

Edit Page:

For the edit page create a get method for that in the PagesController and pass the page id as a parameter. In this method first declare the pageVM model then get the specific page using the id parameter. Check whether the page exists and the initialize the model with pageVM. Finally return the model.

Right click and add view for this method. In Add View Enter EDIT PAGE – template would be EDIT, model class would be PAGEVM and check the use a layout page option. Add a tempData message in the view file and add a if loop which checks if the slug is “home” make that title as readonly and for any other title make it editable.

Now create a postback method for EDIT page which takes the page ID as parameter. In this method first check whether the model exist and if exists get the page id. Check the slug value and if the slug contains null or white space take the title value as the slug value and convert it to lowercase. Now make sure the title and the slug value are unique using a lambda function. Now add the values in the model to the dto and save the changes. Add a custom success message to the tempData field and redirect to the EditPage get method.

PAGE DETAILS

In this method pass the page id where you want to view the details. First declare a pageVM then using the Db context get the particular pageDTO

then check if the page exist. If the page doesn't exist display a custom error message as "PAGE DOES NOT EXIST". If page exist initialize the model with the pageDTO and return the model. Now create a view for the pageDetails by clicking AddView – set the view name as PageDetails, template –Details, Model class- PageVM and check the use layout page checkbox. The asp.net will create a default cshtml page with the details related to that model.

DeletePage:

In the Index.cshtml add the loop to check if the slug is not "home" then you can delete that page. If the slug is "home" then the admin cannot delete that page. Link a class of delete to this line. In the layout file I have added the Render scripts tag so now I have added a script to the Index.cshtml file. In the scripts part add a jquery which shows if the confirm page deletion is not clicked it will return false and the admin will not be able to delete the page created.

```
/* Confirm page deletion */
$("a.delete").click(function () {
    if (!confirm("Confirm Page Deletion")) return false;
});
```

Create a get method for the DeletePage function which will take the page id as the parameter. Get the page dto using the id and remove the dto and save the changes to the database. Finally redirect to the index page.

Reorder pages:

I have used JQUERY minified version and in the Index.cshtml file specify an id for the table and class of sorting and for each item in the table specify the item_id as a id value and check if the slug equals "home" then that

particular page cannot be drag and dropped as the home page is always the first page in the application. Then use a ajax call which takes the url, ids as the parameter. Now in the pages controller I have createa a method for ReorderPages. In this method set a intial count to 1 and declare the pageDTO. Then use a for loop which checks for each pageID set the dto to the pageID of the selected one and set the sorting equals count. Finally save the changes to db and increment the count.

Add some custom css in the site.css file which changes the cursor to pointe when the pages are clicked and when it is reordered then there will be a border added to it.

```
table.sorting tr:not(.home) {  
    cursor: pointer;  
    pointer-events: auto;  
}  
  
.ui-state-highlight {  
    border: 1px dashed #0094ff;  
}
```

EditSidebar:

In this get method first declare the SidebarVM model and using the DbContext get the dto and intialise the model. Once the model is initialized with the dto return the model. Add View to it with the View Name as EditSideBar, template as Edit, Model class as SidebarVM and check the use a layout page checkbox. Add a postback method for the EditSidebar which takes the SidebarVM as the parameter which first gets the dto and assign the body to the dto body and finally save the changes. Add a custom success message to this method which states sidebar edited successfully and redirect to the EditSidebar get method.

In the admin _Layout.cshtml file add list items with a anchor reference to it .

```
<li><a href="/admin/pages">Pages</a></li>  
<li><a href="/admin/pages/EditSidebar">Sidebar</a></li>
```

To add more features to the textArea add ckEditor nugetPackage. Right click on the References part and click on Manage NugetPackages and enter ckEditor and install the latest version in the project structure.

To enable the ckEditor to the textarea write the following script in the EditSidebar.cshtml file

```
@section Scripts{  
<script src="~/Scripts/ckeditor/ckeditor.js"></script>  
  <script>  
    CKEDITOR.replace("Body");  
  </script>  
}
```

To allow even html element to save in the text area use the tag [AllowHtml] which can take the html elements and save in the db tables. Whereever the textarea are used in the view add the above script in the view page.

Adding images to page

To add images in the textArea I have used Roxy Fileman. To add the Fileman to ckEditor navigate to the ckEditor config.js and add these following lines

```
var roxyFileman = '/fileman/index.html?integration=ckeditor';  
config.filebrowserBrowseUrl = roxyFileman;  
config.filebrowserImageBrowseUrl = roxyFileman + '&type=image';  
config.removeDialogTabs = 'link:upload;image:upload';
```

Now in the textarea there will be a image icon and on clicking that the roxy fileman fileupload gets opened in which we can navigate to the directory where the picture exist and by clicking upload function the image gets added to the page.

Shop Controller

Create a table called as tblCategories and create a CategoryDTO with all the properties in the tblCategories table. This CategoryDTO is placed in the model data outside folder of Admin because there will be a shopController which is going to exist in the Admin Area as well as the general mvc structure. Create CategoryVM with the similar fields in the CategoryDTO and add default constructor with no parameters and a constructor with the CategoryDTO parameter.

Create the Categories method in the ShopController which declares a list, and using the DB context initialize the list and finally return the view with the list. Now Right click and add a view with the View Name as Categories, template as List and Model Class as CategoryVM and check the Use a Layout page checkbox. Through ajax I have added Category name to the project and edit the category name already existing in the project. So create a div tag in the Categories.cshtml and name it as a class= new-cat. Inside that create an anchor tag which when clicked will trigger

a ajax call. Use a span class of ajax-text which will display an image loader when the anchor tag is clicked.

```
<div class="new-cat">

  <input type="text" id="newcatname" />
  <p>
    <a href="#" id="newcata">Add a new Category</a>
    <span class="ajax-text">

      
    </span>
  </p>

</div>
```

Display the Category name in the table and add only a link to Delete the category. Add a class of delete to this link and make the value of the Category as readonly. So when the category name is double clicked it can be edited. I have added a if look to check if the model is empty then “There are no Categories Available” message gets displayed.

Add the confirm deletion and the reorder functionality from the Pages(Index.cshtml) file and use it in the Shop(Categories.cshtml) file. Add the css for the ajax loader and the div tag for adding the categories in the site.css file as below.

```
div.new-cat {
  margin-top: 30px;
```

```
position: relative;  
}
```

```
span.ajax-text {  
display: none;  
}
```

A jquery is used to add the categories immediately to the category table and shown in the categories page without any refresh. This should happen when the keypress is done. A post back method AddNewCategory is added in the ShopController which takes the categoryname as the parameter. In this method a id is declared and in the db the validation is done which checks whether the category name is unique. If it is not unique then “titletaken” parameter is returned or else the DTO gets initialized and the categoryname entered gets saved in the db. On adding the category name successfully in the db a id gets created which is converted to string and returned to the calling method.

Reorder functionality if added to the Categories page and in the ShopController the post back method for reorder categories gets added. Similary post back method for DeleteCategories is also added to the ShopController. The script to rename the category is also added to the Category.cshtml file.

A post back method to Rename the Category is added which takes the categoryName and the id as the parameter. First the db context is checked and if the category name edited already exist in the database then “title taken” string gets returned . If not a DTO gets created and whatever name has been changed is saved to the dto and the changes are saved in the db.

ProductDTO

Created a class called ProductDTO with all the properties of the tblProducts. In this property get and set , Id field is the primary key and the CategoryId is the Foreign key. So the Foreign key field is specifically set as below.

```
[ForeignKey("CategoryId")]
```

```
Public virtual CategoryDTO Category {get;set;}
```

Create ProductVM with all the properties of the ProductDTO. Create default constructor and parametrized constructor in View Model. We can add additional functionalities to the view model . To add a product to a category a SelectListItem is used and to add images to each product a IEnumerable <string> is used as below.

```
public IEnumerable<SelectListItem> Categories { get; set; }  
public IEnumerable<string> Images { get; set; }
```

Add a get AddProduct method in the shop controller where the model gets initialized first. Then using Db the model.Categories is assigned to the Categories id and name. Finally the model is returned with the view. Right click and create a view as Add Product , template- Create and Model class as ProductVM and check the Use a layout page checkbox. To add images to a product change the Begin form syntax with the below line

```
@using (Html.BeginForm("AddProduct","Shop",FormMethod.Post, new  
{ enctype = "multipart/form-data" })))
```

Modify the Category html tag to a DropDownListFor which displays the list of categories existing and add a new div tag which is used to upload a image for the product. A jquery is written which takes the url of the selected image first then a reader object is created. On this reader onload create a function which will take the imgpreview as the id and set the attribute to the src value of target and set the image width and height. Finally the image will read all input files and upload that selected file. The following is the code to preview a selected image

```
$(function () {  
    /* preview selected image */
```

```
    function readURL(input) {  
        if (input.files && input.files[0]) {  
            var reader = new FileReader();  
            reader.onload = function (e) {  
                $("#img#imgpreview")  
                    .attr("src", e.target.result)  
                    .width(300)  
                    .height(300);  
            }  
            reader.readAsDataURL(input.files[0]);  
        }  
    }  
}
```

```
$("#ImageUpload").change(function()  
{  
    readURL(this);  
});
```

```
});
```

Create a postback method for the AddProduct which takes the ProductVM model as a parameter and a HttpPostedFileBase filename as a parameter. In this method the model state is first checked and the whether the product name is unique or not is checked. A product id field is declared and saved to the product dto. Once all the details filled in the AddProduct page is genuine the details are saved to the dto and the changes are saved in the db.

A region is created and the code to upload image is placed in it. First the necessary directories are created. Then the file is uploaded. The extension of the uploaded file is got and verified. If the extension is allowed then the image name is initialized and saved to the dto. The original and the thumb image paths are set and saved to the db.

Import the NugetPackaged PagedList.mvc to the project. This PagedList provides the feature of Pagination and filtering the products based on the category added. So a get method to display all the products is written which first declares the list of ProductVM, and set the page number field, initialize the list, populate the categories based on the selected list, set pagination and finally return the view with the list.

Create a view with name as Products, Template-List, Model class-ProductVM and check the use a layout page checkbox. Import the following two lines in the Product.cshtml file

```
@using PagedList.Mvc;
```

```
@using PagedList;
```

Add a dropdown to select the list of Categories and when a category gets picked products specific to that category only gets displayed. To add pagination to the field add the following line to the Products.cshtml file

```
@Html.PagedListPager((IPagedList) ViewBag.OnePageOfProducts, page  
=> Url.Action("Products", new { page, catId = ViewBag.SelectedCat })))
```

Create a get method for the EditProduct which takes the product id as the parameter. In this method declare the productVM, get the specific product. Make sure the product exists. If the model exist initialize the model, make a select list. Get all the images related to the product and finally return the view with the model. Create a View as EditProduct, template as Edit, model class as ProductVM and check the use a layout page checkbox. This View will contain all the details specific to the selected product.

Create a post back method for the EditProduct method which takes the ProductVM model and HttpPostedFileBase as the parameters. To edit the product first get the product id, then populate all the product specific details. Check the model state and update any values needed. Make sure all the details edited are unique and genuine then update the products and set a custome success message. Add the image upload code from the Add product in this file. Finally redirect to the page EditProduct.

In the DeleteProduct method pass the id of the product that needs to be deleted. First get the product specific details from the db and if the item exists remove the products from the db and save the changes. From the images folder first identify the image using the id and if the directory exists delete the pathstring. Finally redirect to the Products page.

DropZone:

Add a set of pictures for the products added in the database. Right click the References and clicked on Manage Nuget Packages and install dropzone. First check if the model contain any images to it or not. If more images are to be added to the product the add the following js lines.

```
@if (!Model.Images.Any())
{
    <h2>There are no images for this product</h2>
}
```

```
<form action="/admin/shop/SaveImages" method="post"
enctype="multipart/form-data" class="dropzone" id="dropzoneForm">
    <div class="fallback">
        <input type="file" name="file" multiple/>
        <input type="submit" value="Upload"/>
    </div>
</form>
```

```
<br/>
```

```
@foreach (var image in Model.Images)
{
    <div style="display:inline-block; margin-right: 15px;">
        
        @Html.ActionLink("delete", "DeleteImage", "Shop", new { @class
= "deleteimage", data_name = image })
    </div>
}
```

Include the below link in the scripts folder:

```
<link href="~/Scripts/dropzone/basic.css" rel="stylesheet" />
<link href="~/Scripts/dropzone/dropzone.css" rel="stylesheet" />
@section Scripts {
<script src="~/Scripts/dropzone/dropzone.js"></script>
```

```
/* Dropzone js */
```

```
Dropzone.options.dropzoneForm = {
  acceptedFiles: "image/jpeg,image/png",
  init: function () {
    this.on("complete",function(file){
      if(this.getUploadingFiles().length === 0 &&
this.getQueuedFiles().length === 0){
        location.reload();
      }
    });
  }
};
```

```
    this.on("sending",function(file,xhr,formData){
      formData.append("id",@Model.Id);
    });
  }
};
```

Now create a saveGalleryImages post back method in the ShopController which takes the image id as the parameter. In this method loop through the files and check if the file is not null. If so set the appropriate directory paths and image paths as in image upload and save the original and thumb image in the project. I have added a separate js to delete the individual images added to a product. Finally create a post back method for the

DeleteImage which takes the id of the product and the imageName as the parameter. Use System.IO.File to check for a file image existence and if the image exist delete the path and the image from the folder.

FRONT END PART OF THE WEBSITE

Create a PagesController which takes a empty string as a page parameter. In this method first the page value is set in which if the page is null the page is set to home page. Next declare model and dto. Now if the page exists. Get the page DTO and set a title for the page. Check if a model exist for a page and return the view with the model. Create a view for the page with the View name as Index , templates as Details, Model class as PageVM. Remove all the default tags in the page and set the ViewBag.Title and set the raw html part of the page with the Model.Body

In the RouteConfig.cs file remove the default routes and add the custom routes to it as follows:

```
routes.MapRoute("Pages", "{page}", new { controller = "Pages", action = "Index" }, new[] { "TrendStyle.Controllers" });
        routes.MapRoute("Default", "", new { controller = "Pages", action = "Index" }, new[] { "TrendStyle.Controllers" });
```

Create a pagesMenuPartial in the pagesController which declares a list of PageVM and get all the pages except the home and return the partial view

with the list. Now add this `pagesMenuPartial` in the `layout.cshtml` file and in the `routes.config` file create an entry for the partial view.

Create a `SideBarPartial View` in the `PagesController` which first declares the model and initialize the model and finally return the partial view with the model. Create a view as `SibeBarPartial` , template-Details, Model-`SideBarVM` and check the checkbox create as a partial view. In the main body clear all the lines and add the code `@Html.Raw(model.Body)`. In the main `_Layout.Html` file create an html action for this sidebarpartial page created. To make the pages title highlighted when clicked on the li item include the class of active value to it.

ShopController.

Create a `shopController` where the default `Index` method is redirected to the `Index` method in the `pagesController`. Create a `CategoryMenuPartial` in which declare a list of `CategoryVM` and initialize the list and finally return the partial with the list. Create a View `CategoryMenuPartial`, Template-List and Model class as `CategoryVM` and check the Create as a partial view checkbox. Replace all the content of the view with the ul list which list all the existing categories in the project.

Create a `Category` method in the `ShopController` which takes the `categoryname` as a parameter in which first declare a list of `ProductVM`. Get the `categoryId` and initialize the list. Get the category name and return the view with the list. Create a view with the name as `Category`, template-list, and model class-productVM and check the use a layout page checkbox.

Add a `productDetails` method which takes a name as a parameter. In this method first declare the VM and the dto and initialize the product id. First

check if the product exists. If the product exists initialize the productDTO and get the inserted id, Initialize the model. For each product get the gallery images related to the product and finally return the view with the model. Create a productDetails view with the model class as ProductVM. In this view page display the image related to the product and display all the product specific details. Create a anchor link to Add to Cart.

Now install Fancybox in the project via manage NugetPackages. Add the jquery for the fancybox part

CartController:

Create a cartVM to handle the cart functionalities. The cartVM looks like this.

```
public class CartVM
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public int Quantity { get; set; }
    public decimal Price { get; set; }
    public decimal Total { get { return Quantity * Price; } }
    public string Image { get; set; }
}
```

In the cart controller create a CartPartial where the CartVM gets initialized first. The quantity is set to zero and the price is initially set to 0m. Next check for the cart session and get the total quantity and the price. Finally return the partial view with the model. Create a partial view as CartPartial, with template as details and Model class as CartVM. Replace all the tags created with the following lines:

```
<div class="ajaxcart" style="color:white;">
    Your cart: @Model.Quantity item(s) - $@Model.Price
```

```
<br/>
```

```
@Html.ActionLink("Checkout","Index")  
</div>
```

In the Routes file add the Cart controller and the cart partial view. Now add the CartIndex method with the logic as initializing the cart list and then check if the cart is empty. If the cart is not empty calculate the total and save the total in the view Bag. Finally return the view with the model. Create a Index View with the template as list and model class as cartVM. Add anchor tags to increment/decrement/remove product from the cart in the index page.

Create js to add the items in the cart. In the cartController add the AddToCartPartial method which first initializes the cartVM list, Get the product specific details, check if the product is already in the cart and if not add the product to the cart and if increment is clicked increment the product count and finally calculate the total qty and the price of the product and add that to the model

Add the Code for Incrementing the product in the cart, decrementing the product and removing the product from the cart in the Cart Index View. In the cart controller create a method to increment the product which takes the productID as a parameter. First is to initialize the cart list then to get the cartVM from the list. Increment the quantity and store the result and finally return the result as a json with data.

The same step is repeated for the decrementProduct and in this additionally when the product value gets decremented and the value goes below 0 then the window.reload() is called which refreshes the browser and no action

takes place. When the RemoveProduct is clicked first the cart gets initialized and then the model is get from the list and the item is removed from the particular model in the list.

Account setup:

Create UserDTO and UserVM with all the properties of the User table. Specify the required fields as Required. Create table tblRoles with the fields Id and Name and add the values admin, user to it. Create RoleDTO with these two fields in the project. Create UserRoleDTO with these two fields and set both the fields as primary key and specify an order for the table columns. Two foreign keys are used in the table specify that as below.

```
namespace TrendStyle.Models.Data
{
    [Table("tblUserRoles")]
    public class UserRolesDTO
    {
        [Key,Column(Order = 0)]
        public int UserId { get; set; }
        [Key,Column(Order = 1)]
        public int RoleId { get; set; }

        [ForeignKey("UserId")]
        public virtual UserDTO User { get; set; }
        [ForeignKey("RoleId")]
        public virtual RolesDTO Role { get; set; }
    }
}
```

Create AccountController and create a method called as CreateAccount which will return the CreateAccount View. Create a new view as CreateAccount, template as Create and model class as UserVM and check the use a layout page checkbox. For the password field replace the tags with @Html.PasswordFor value. In the routes file add an entry to this create account page.

Specify the authentication mode in the Web.Config as follows:

```
<authentication mode="Forms">  
  <forms defaultUrl ="~/Account/Login" loginUrl ="~/Account/Login"></forms>  
</authentication>
```

Create a loginUserVM with the fields UserName, password, rememberMe of type bool. Add a method login in the Account controller which first check if the user is not logged in and if the username field is not null or empty and has some value it will redirect to the user-profile page and finally it will return the view. Create a new view as Login, template-Create, Model class as – LoginUserVm and check the use a layout page option.

Create a postBack method for the create account which first checks the model state, check if the password match and the username is unique. If the details are genuine the userDTO is created and saved. For all the user create a roleID of 2 is given which specifies the userrole. A custom TempData message is added which shows the success message and the user is redirected to the login method.

A postback for the login method is added in the account controller where a validation is placed to check the user name and password and if the details exist in the database the user is navigated to the home page. Finally a logout method is added in the account controller which will redirect to the account/login method.

A userProfileVM is created which will show all the user specific details. A userNav partial is added which shows all the user specific details in the application.

Enable userRoles:

An application_Authenticate Request method is added in the global.asax.cs as follows.

```
protected void Application_AuthenticateRequest()
{
    // Check if user is logged in
    if (User == null) { return; }

    // Get username
    string username = Context.User.Identity.Name;

    // Declare array of roles
    string[] roles = null;

    using (Db db = new Db())
    {
        // Populate roles
        UserDTO dto = db.Users.FirstOrDefault(x => x.UserName ==
username);

        roles = db.UserRoles.Where(x => x.UserId == dto.Id).Select(x
=> x.Role.Name).ToArray();
    }

    // Build IPPrincipal object
```

```

        IIdentity userIdentity = new GenericIdentity(username);
        IPrincipal newUserObj = new GenericPrincipal(userIdentity,
roles);

```

```

        // Update Context.User
        Context.User = newUserObj;
    }
}
}

```

Adding Paypal to Cart.

Login to Paypal Sandbox (<https://developer.paypal.com>) and create a sample paypal account and add some credit to it. To add paypal to a third party add the following code in the Index part of the cart. Create a paypal partial view with the same code given below:

```

<div class="paypaldiv">
    <form class="paypalform"
action="https://www.sandbox.paypal.com/cgi-bin/webscr"
method="post">
        <input type="hidden" name="cmd" value="_cart">
        <input type="hidden" name="upload" value="1">
        <input type="hidden" name="business" value="jultranet-
facilitator@gmail.com">

```

```

        @foreach (var item in Model)
        {
            <input type="hidden" name="item_name_@count"
value="@item.ProductName">
            <input type="hidden" name="amount_@count"
value="@item.Price">
            <input type="hidden" name="quantity_@count"
value="@item.Quantity">
            count++;

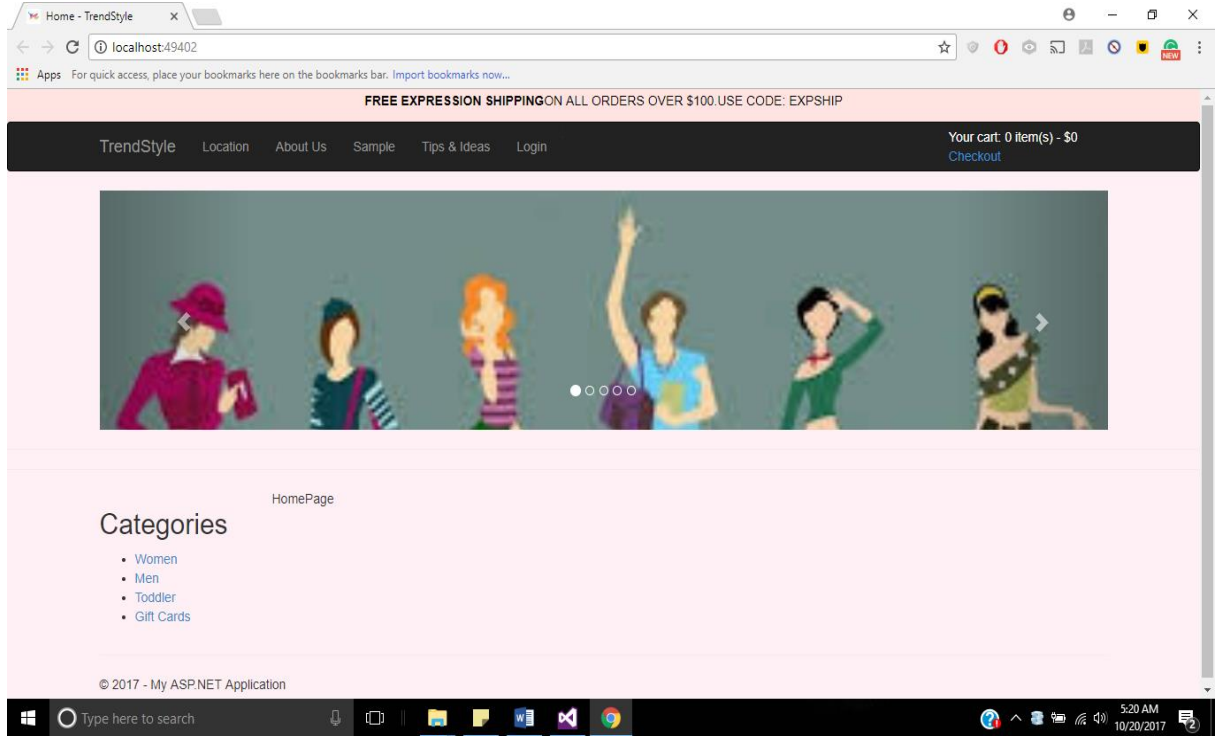
```

```
}
```

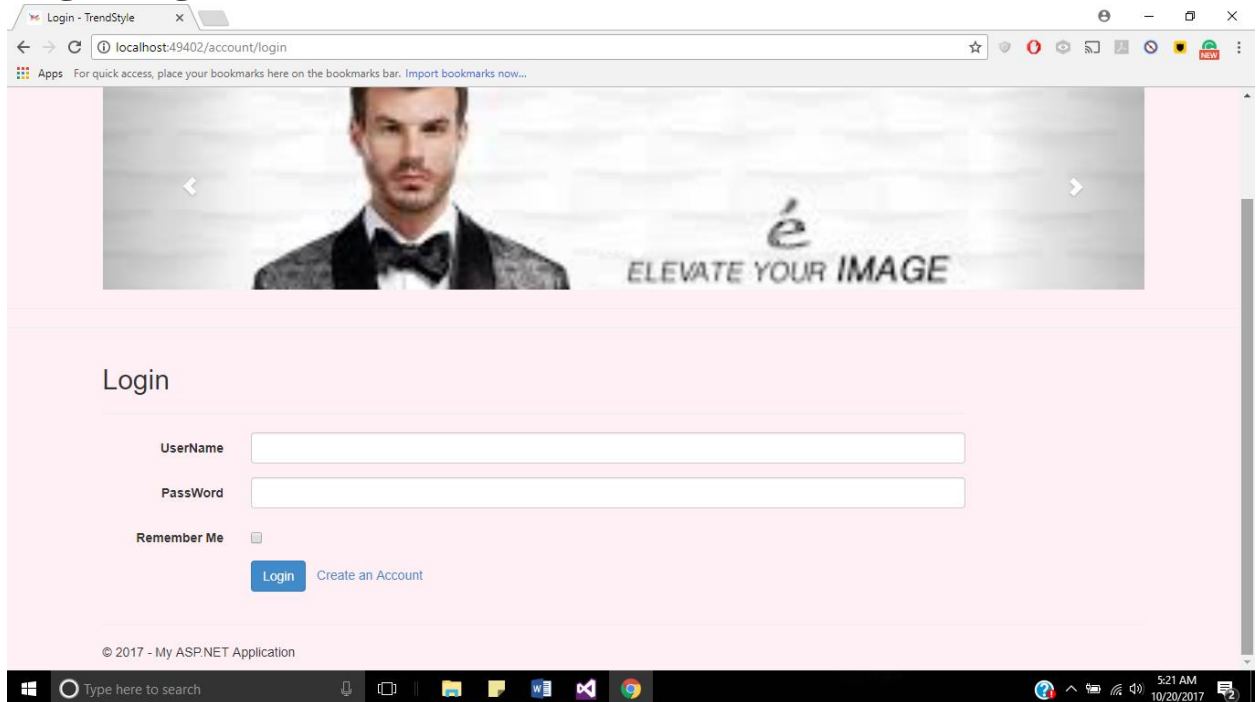
```
    <input type="hidden" name="currency_code" value="USD">
    <input type="image"
src="http://www.paypal.com/en_US/i/btn/x-click-but01.gif"
name="submit" alt="Make payments with PayPal - it's fast, free and
secure!">
    </form>
</div>
```

SNAPSHOTS

• Main Page:



• Login Page:



- **Create Account:**

Create Account

FirstName

LastName

Email

UserName

Password

Confirm Password

Phone Number

Create

© 2017 - My ASP.NET Application

- **Validation message in Create Account Page:**

FirstName
The FirstName field is required.

LastName
The LastName field is required.

Email
The EmailAddress field is required.

UserName
The UserName field is required.

Password
The Password field is required.

Confirm Password
The ConfirmPassword field is required.

Phone Number
The PhoneNumber field is required.

Create

© 2017 - My ASP.NET Application

Create Account - TrendStyle

localhost:49402/account/create-account

Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Create Account

• Username Ananthi05 is taken.

FirstName Ananthi

LastName Selvamani

Email ananthiselvamani0512@gmail.com

UserName Ananthi005

Password

Confirm Password


Phone Number 2039194142

Create

Login - TrendStyle

localhost:49402/account/login

Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...



Login

You are now registered and can login.

UserName

PassWord

Remember Me ☐

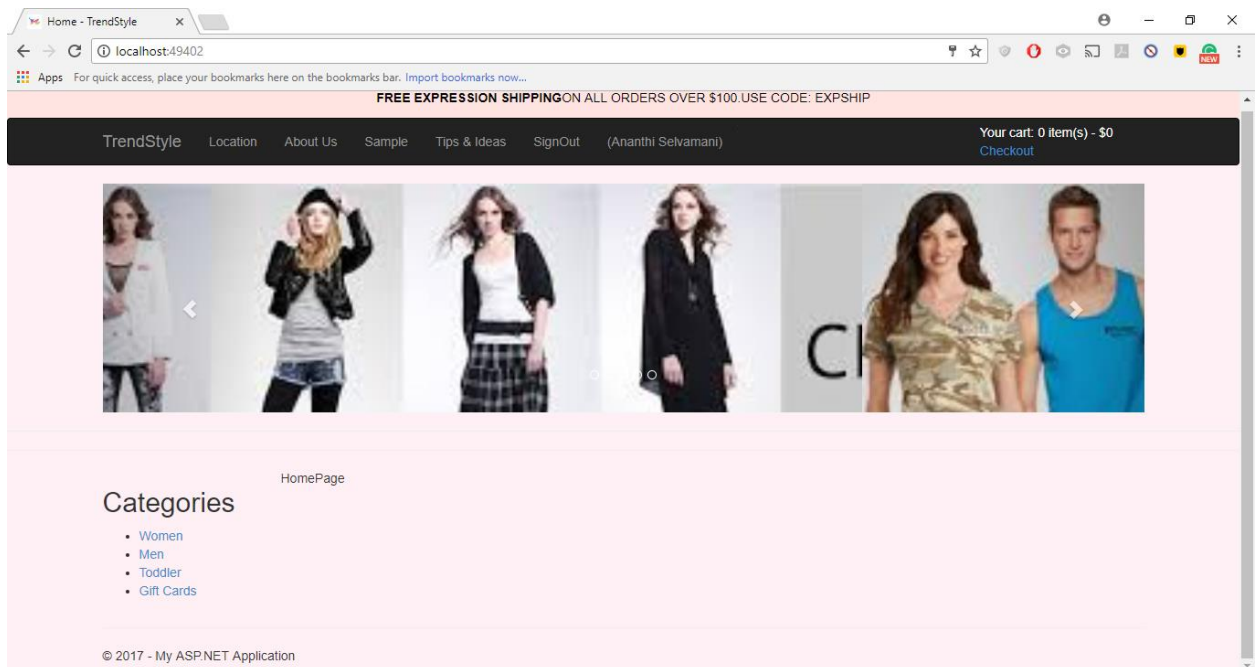
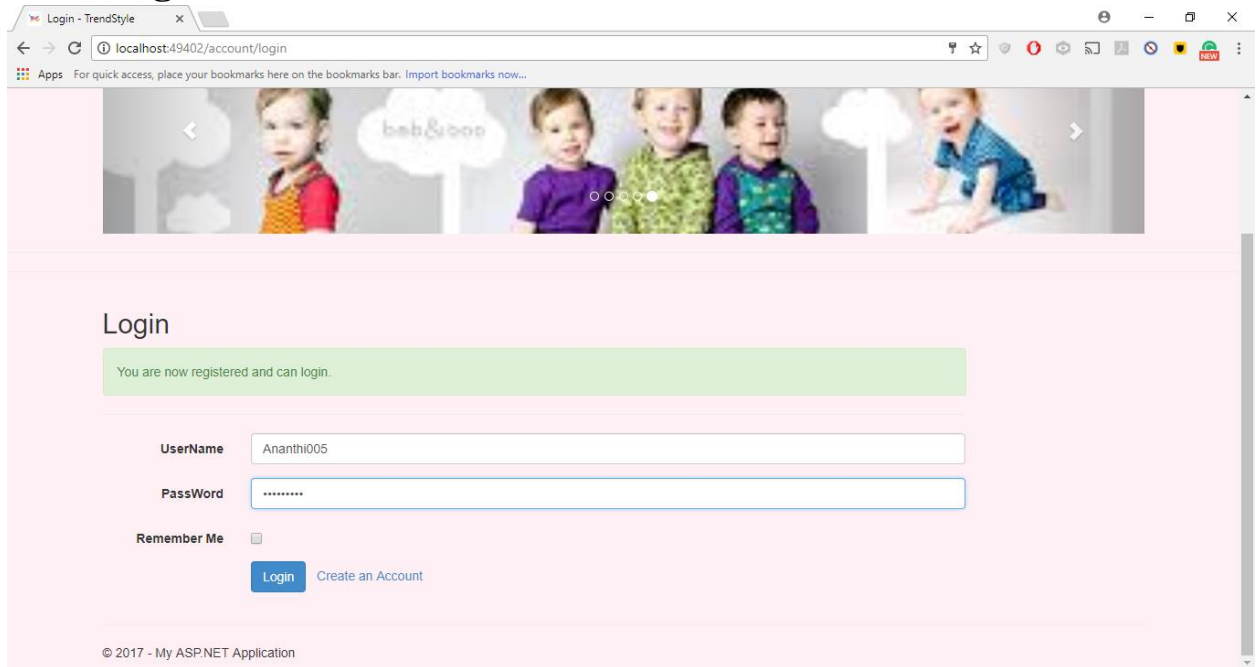
Login Create an Account

© 2017 - My ASP.NET Application

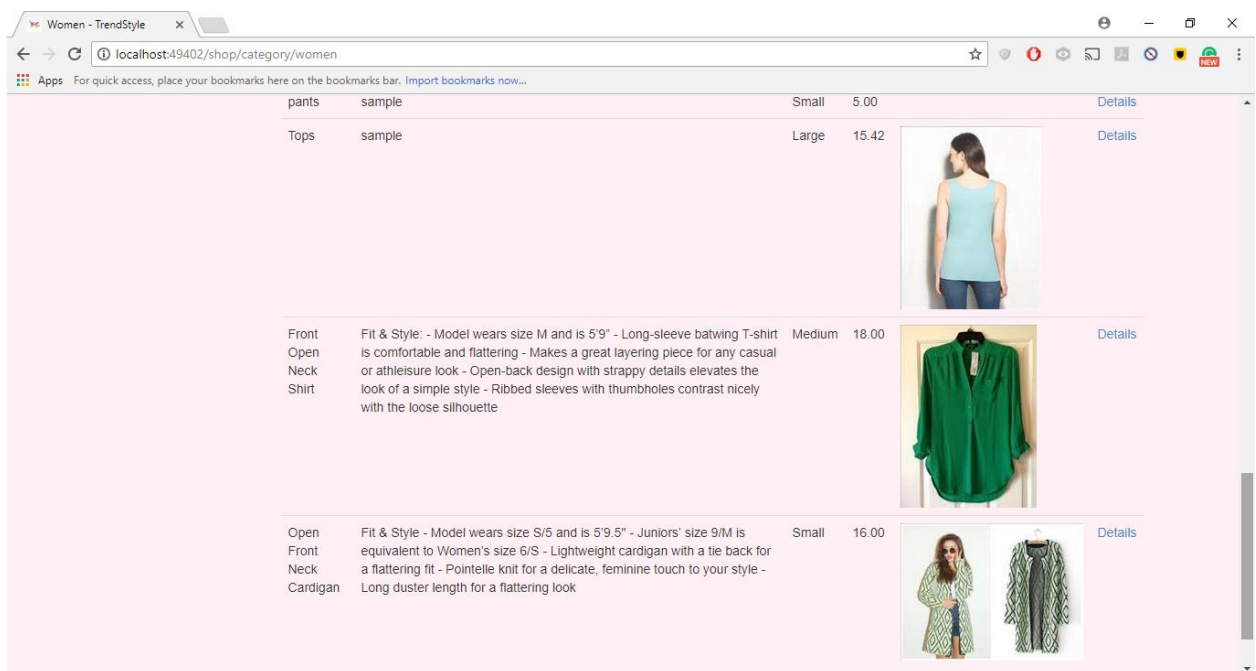
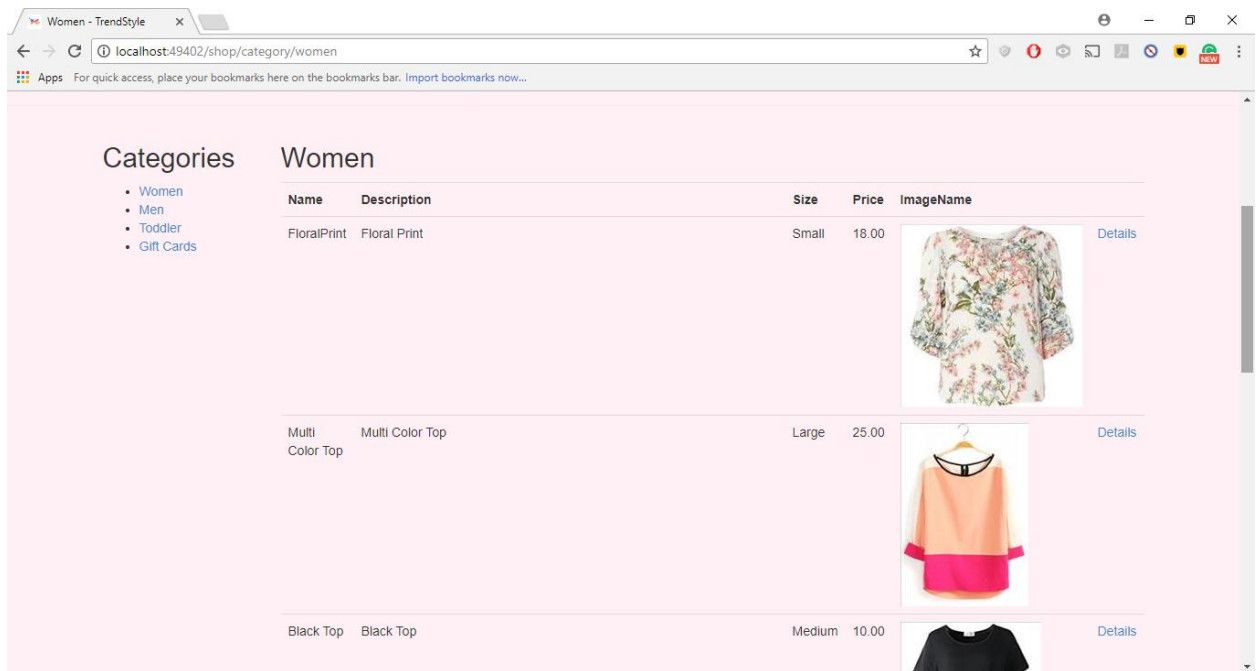
Type here to search

5:25 AM 10/20/2017

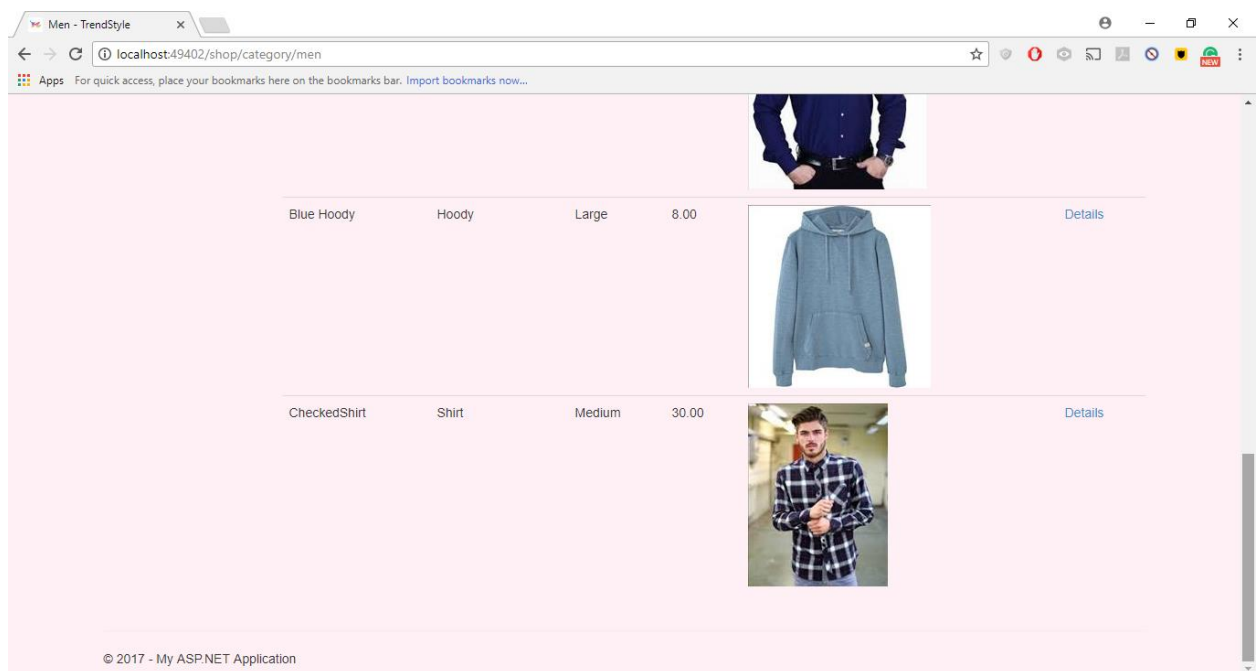
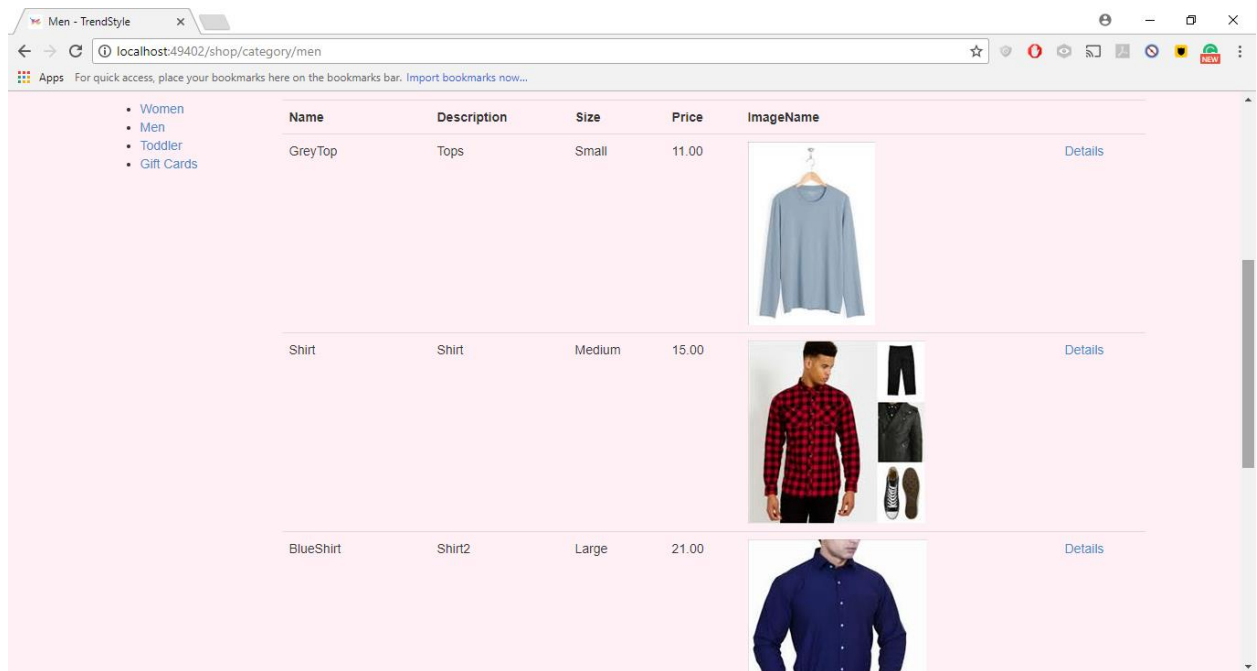
• User Login:



• Select Category Women:



- **Select Category Men:**



• Select Toddler

Toddler - TrendStyle




localhost:49402/shop/category/toddler

Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Categories

- Women
- Men
- Toddler
- Gift Cards

Toddler

Name	Description	Size	Price	ImageName	
Costume1	KID	12-20MONTHS	25.00		Details
Costume2	kid	0-12months	18.00		Details
KID clothes	kids	12-24 months	32.00		Details

Toddler - TrendStyle

localhost:49402/shop/category/toddler


Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

KID clothes

kids

12-24 months

32.00




[Details](#)

Clothe4

kid

1T-5T

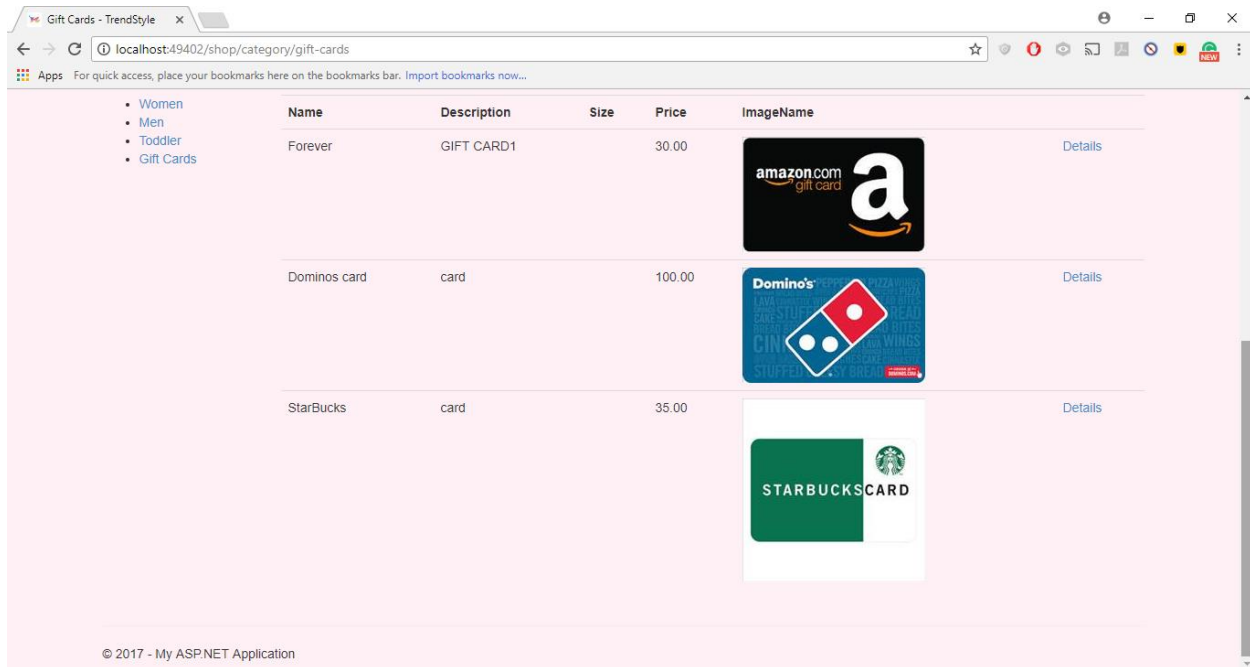
22.00



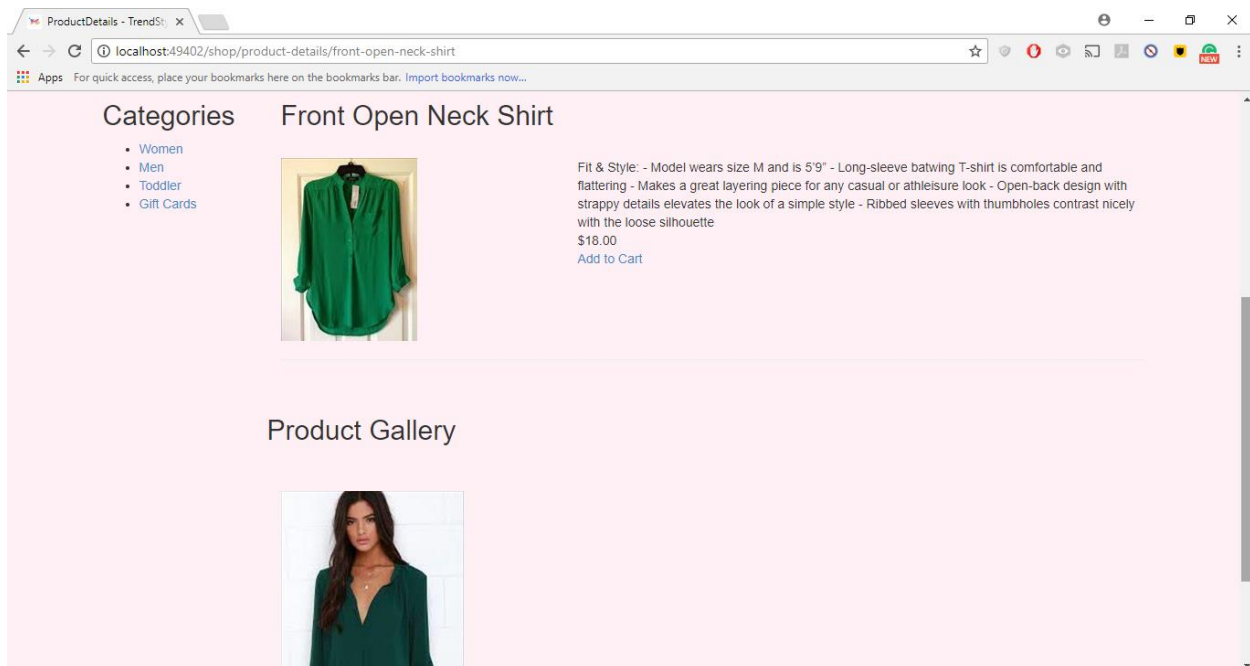
[Details](#)

© 2017 - My ASP.NET Application

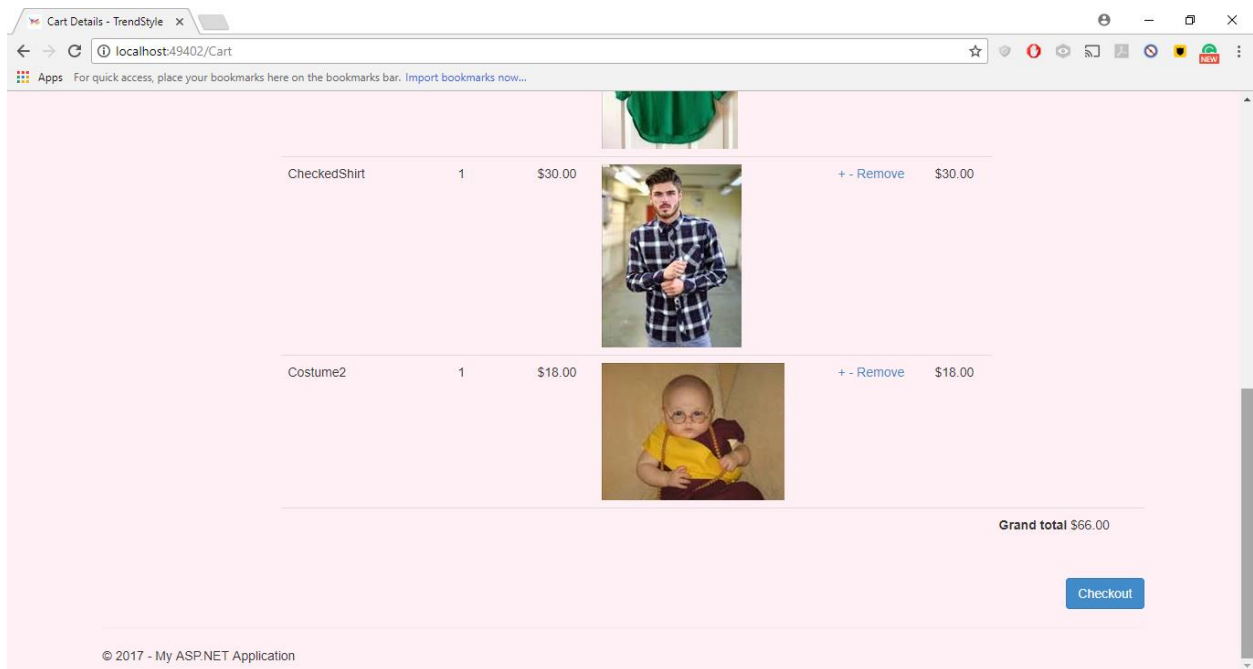
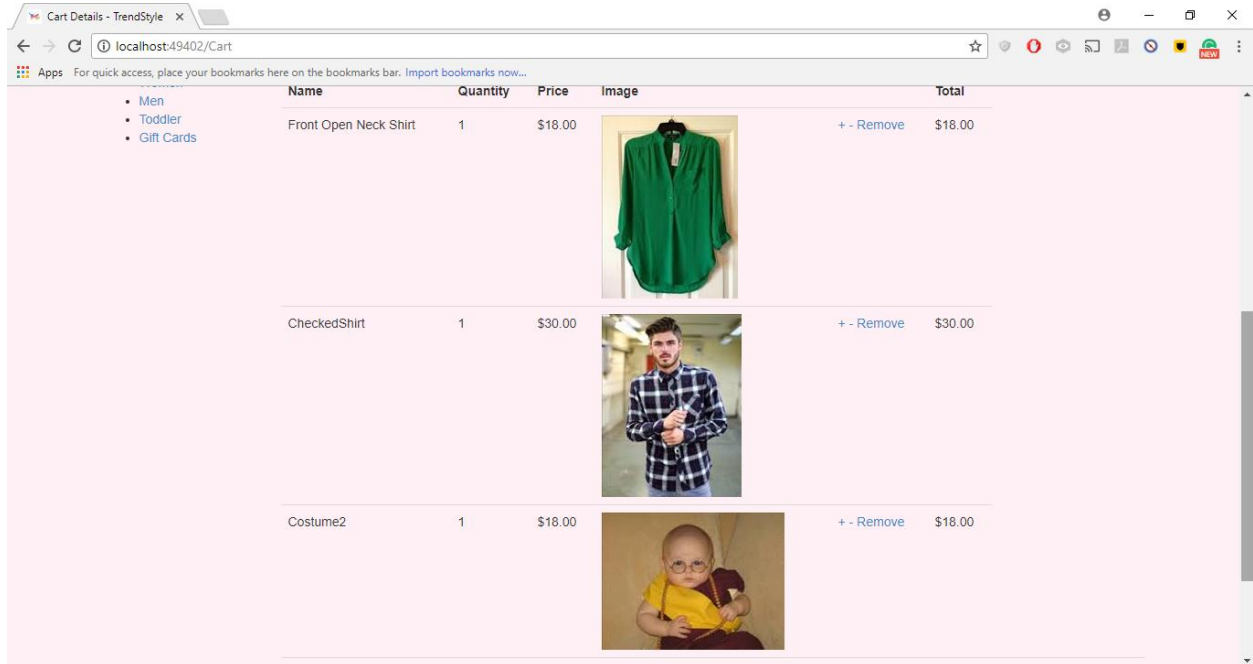
• Select GiftCards:



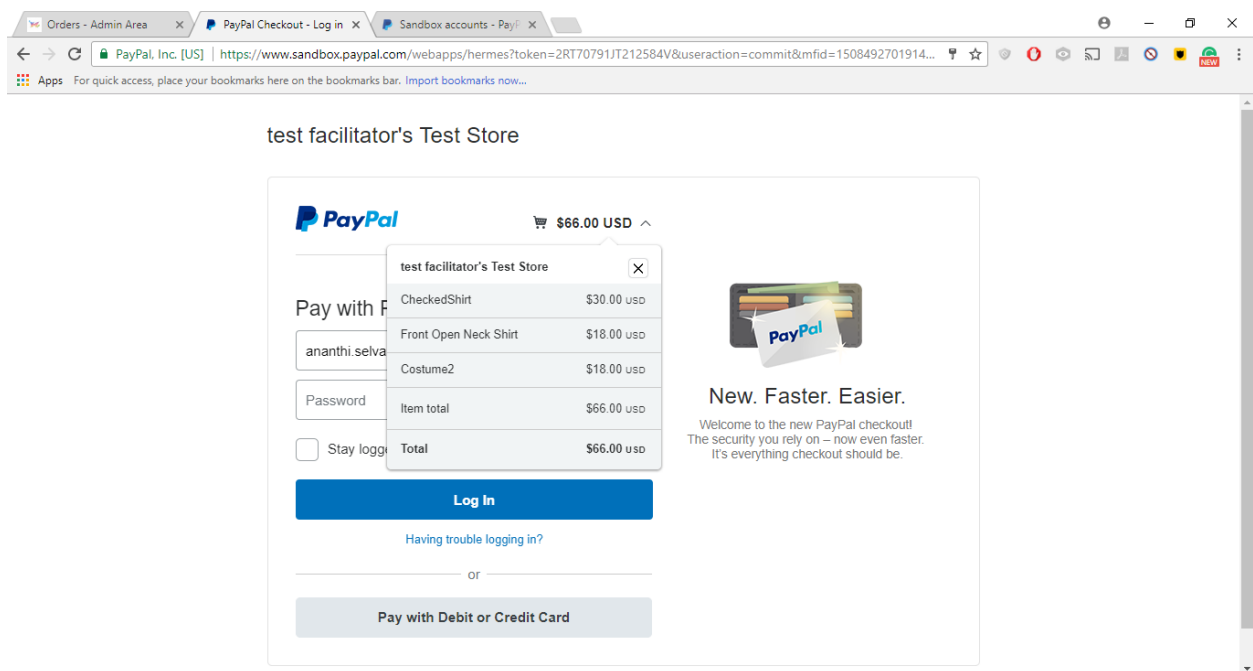
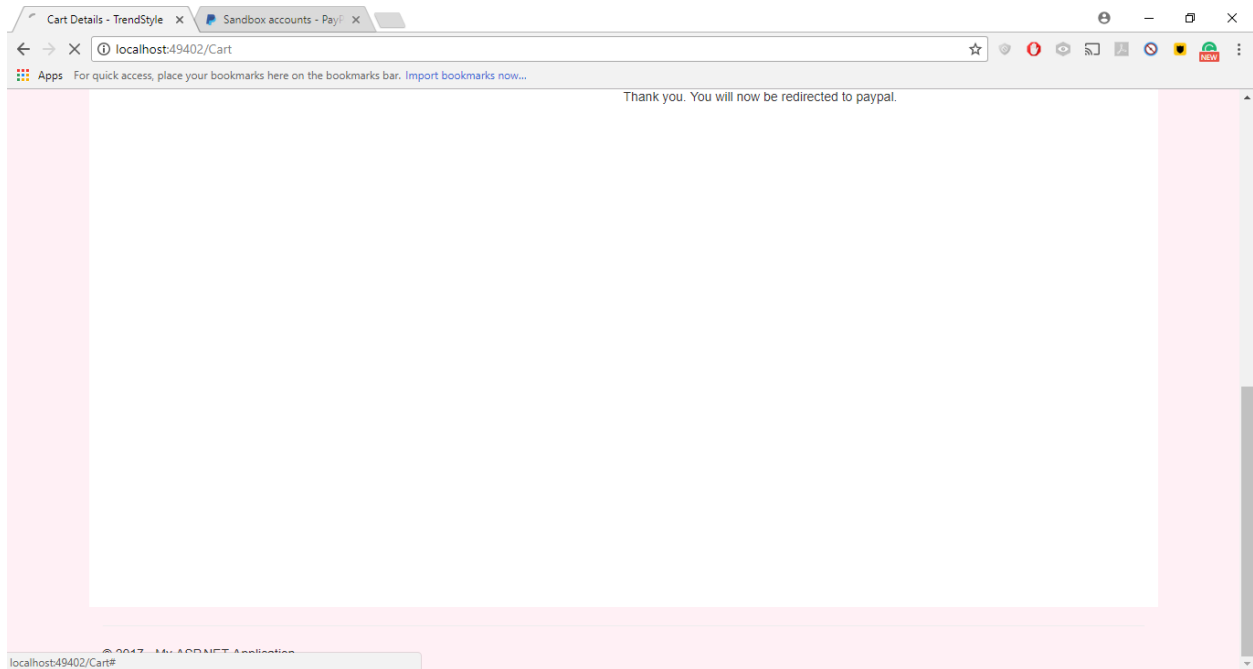
• Click on a Product Image



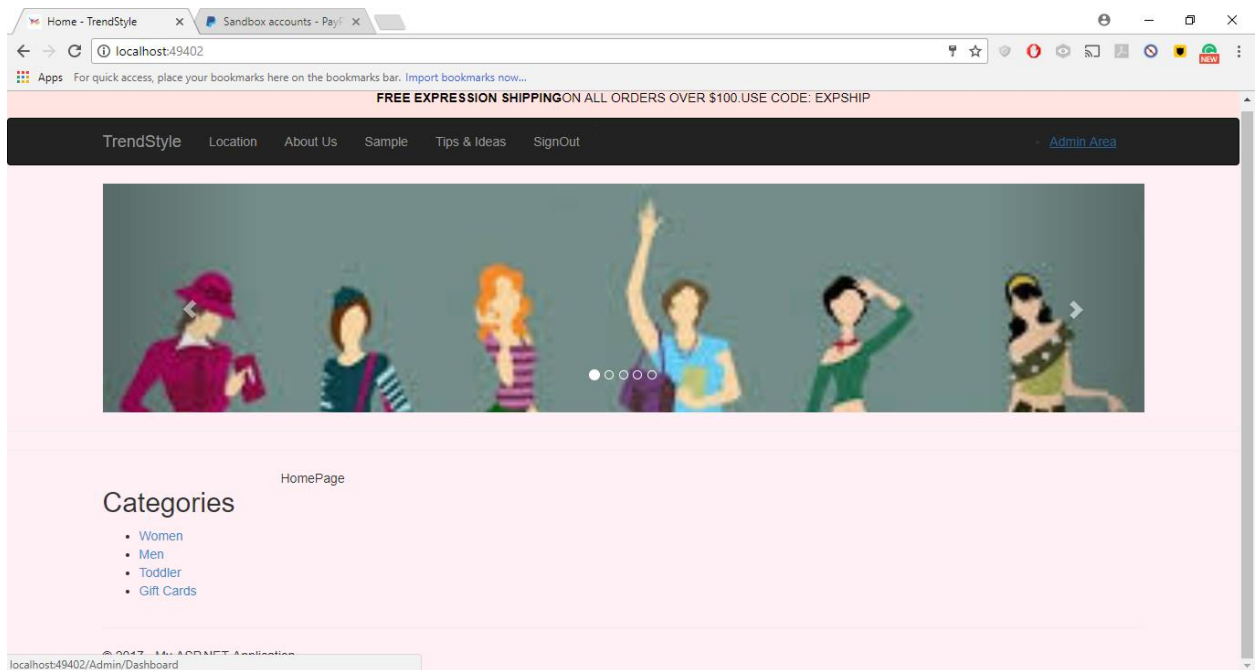
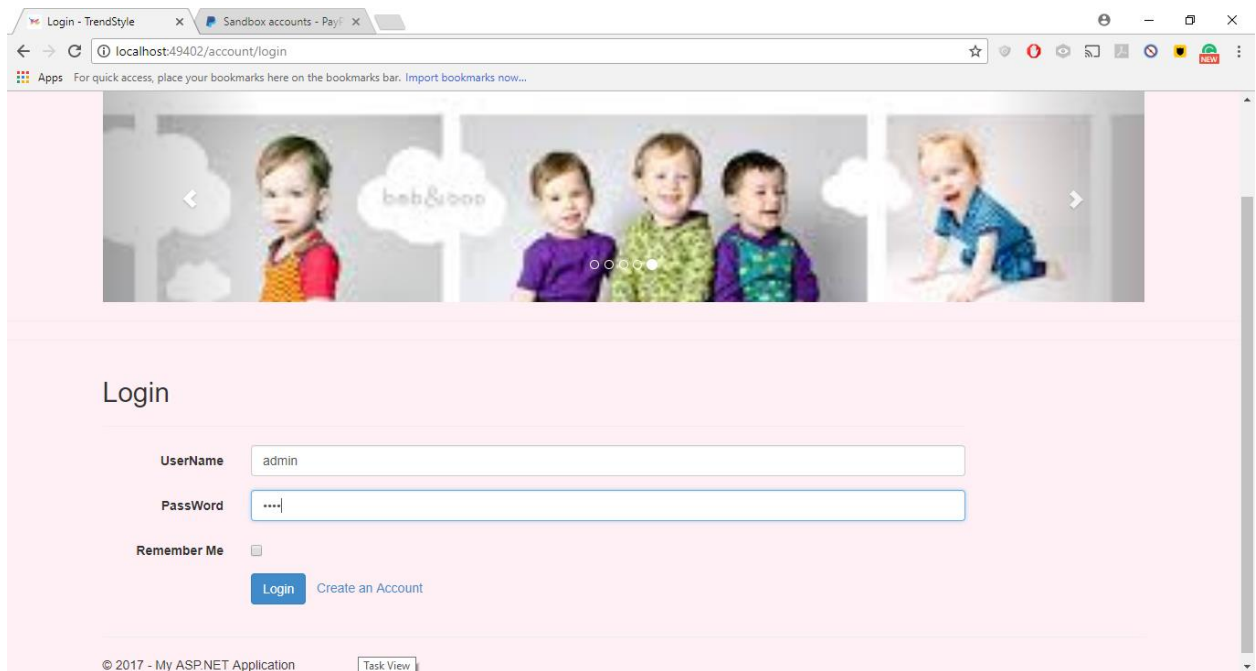
Add more items to the cart:



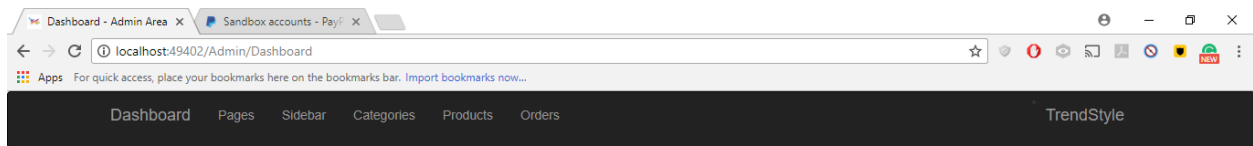
• Click on CheckOut



- **Login as Admin:**



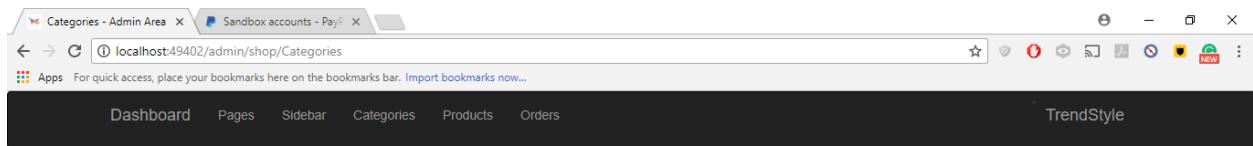
- **Click on Admin Area**



- **Click on Pages:**

Title	Slug	arHasSideb	
Home	home	<input type="checkbox"/>	Edit Details
Location	location-	<input type="checkbox"/>	Edit Details Delete
About Us	trend-style	<input type="checkbox"/>	Edit Details Delete
Sample	sample	<input checked="" type="checkbox"/>	Edit Details Delete
Tips & Ideas	tips-&-Ideas	<input type="checkbox"/>	Edit Details Delete

- **Click on Categories:**



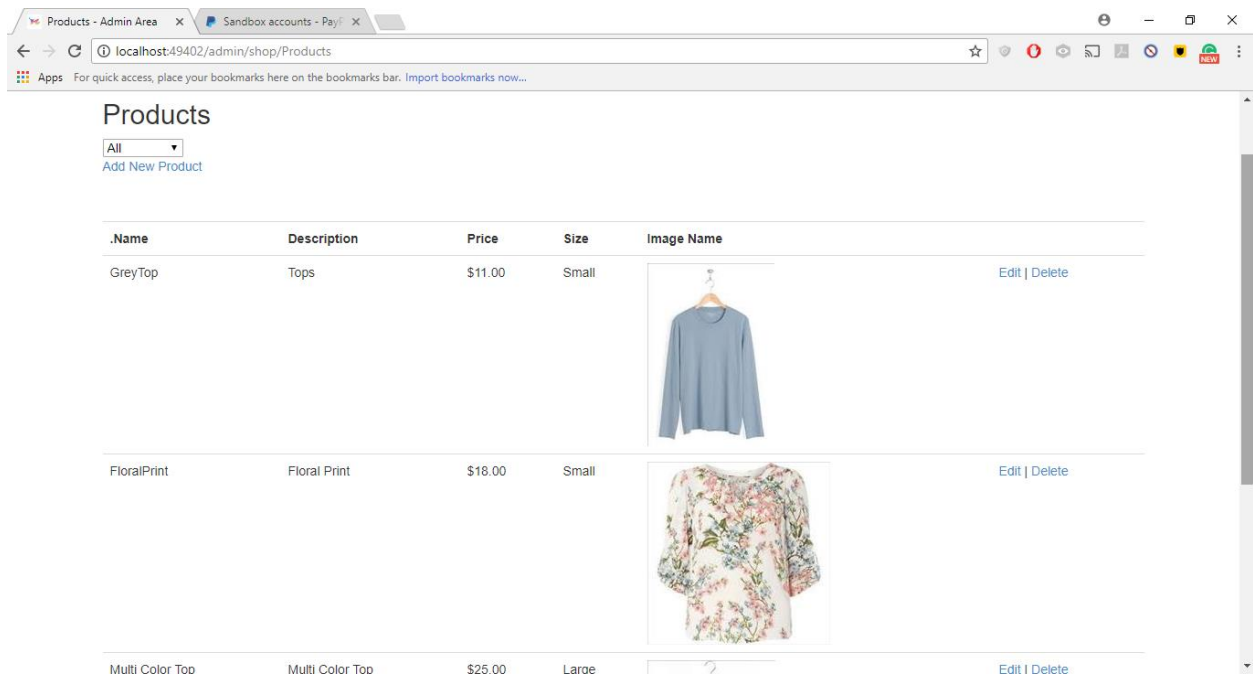
Categories

[Add a new Category](#)

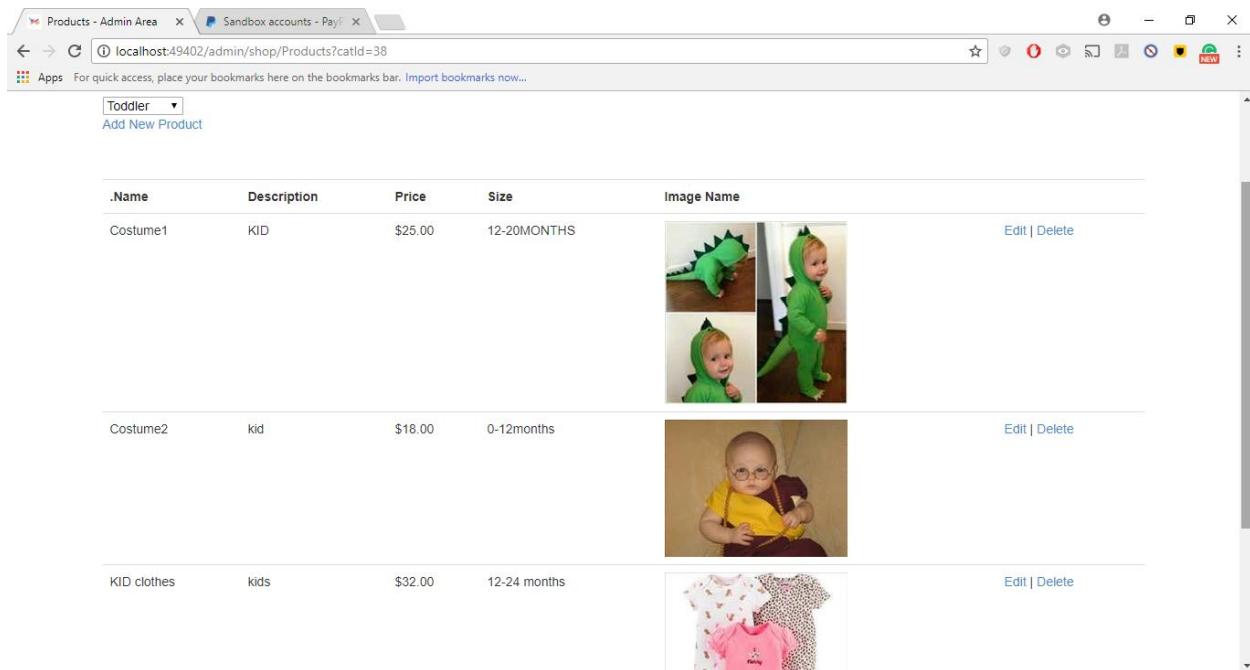
[Add](#)

Name	
Women	Delete
Men	Delete
Toddler	Delete
Gift Cards	Delete

- **Click on Products:**



- **Search by Category:**



• Add New Product

Add Product
ProductVM

Name


Description

Size

Price

Category

Image



• Click Add Product:

Product added successfully

ProductVM

Name

Description

Size

Price



Category

Image No file chosen

[Back to Products](#)

- **Click back to Products and check the Toddler section:**

Toddler

.Name	Description	Price	Size	Image Name	
Clothe4	kid	\$22.00	1T-5T		Edit Delete
sample product 1	sample product1	\$3.00	small		Edit Delete

« 1 2

- **Click Order:**

Orders - Admin Area

Sandbox.accounts - Pay...

localhost:49402/admin/shop/Orders

Apps

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Orders

OrderNumber	UserName	Order Details	Created At	Total
1	Jessie0007	FloralPrint x 1 GreyTop x 2	10/15/2017 12:05:00 AM	\$40.00
2	Jessie0007		10/15/2017 12:07:05 AM	\$0.00
3	Jessie0007	Black Top x 5 FloralPrint x 1	10/15/2017 12:11:24 AM	\$68.00
4	Jessie0007		10/15/2017 12:12:58 AM	\$0.00
5	Jessie0007	FloralPrint x 3	10/15/2017 12:13:46 AM	\$54.00
6	Jessie0007	GreyTop x 8	10/15/2017 12:16:59 AM	\$88.00
7	Jessie0007	GreyTop x 2	10/15/2017 12:18:34 AM	\$22.00
8	Jessie0007	FloralPrint x 1 GreyTop x 1	10/15/2017 2:36:11 AM	\$29.00
9	Jessie0007	BlueShirt x 1 Front Open Neck Shirt x 1 Dominos card x 1 Costume1 x 1	10/16/2017 6:53:12 AM	\$164.00
10	Ananthi005	Front Open Neck Shirt x 1 CheckedShirt x 1 Costume2 x 1	10/20/2017 5:35:58 AM	\$66.00