


When building an MVC e-commerce site, you may want to display product catalog and [product detail pages](#). A product listing serves as a product catalog, in which you can display all [products](#), filtered products or products from specific sections.

This page describes how to display products from the **Products** application in an MVC application connected to Kentico. Regardless of products' position in the product tree, you can display them on your MVC site based on your needs. When creating products, use a [product page type](#) labeled as [content only](#).

 The [Kentico.Ecommerce integration package](#) currently supports only products consisting of a [page](#) and an SKU object (the default product setting). The [stand-alone SKU product mode](#) is not supported in MVC.

To build product detail pages, you need to take care of the following steps:

- [Setting URLs of the product listing](#)
- [Displaying products in the listing](#)

## Setting URLs of the product listing

To provide SEO-friendly identifiers in the URL, you can locate your product listing to an address like `<your domain>/Store /<product page type>`.


Add a new route to your **RouteConfig** class in the **App\_Start** folder:

```
routes.MapRoute(
    name: "Store",
    url: "Store/{controller}",
    defaults: new { action = "Index" },
    constraints: new { controller = "LearningProductType" }
);
```

The example assumes that the system displays only products based on one product page type (named *LearningProductType*) with a controller of the same name and uses its *Index* action to actually display the products (as described below in [Displaying products in the listing](#)).

## Displaying products in the listing

Display all relevant product information to visitors to enable them get to a [detail page](#) of their desired product. Use the **Kentico.Ecommerce** integration package to help you with displaying of product details.

 **Tip:** To view the full code of a functional example directly in Visual Studio, download the [Kentico MVC solution](#) from GitHub and inspect the **LearningKit** project. You can also run the Learning Kit website after connecting the project to a Kentico database.

1. [Generate code files of the product page types](#) you intend to display.
  - a. In the **Page types** application, edit the specific product page type.
  - b. Switch to the **Code** tab.
  - c. Click **Save code**.
2. Open your MVC project in Visual Studio.
3. Include the generated code files to your MVC project.
4. Add a view model for products.



```
public class ProductListItemViewModel
{
    public readonly ProductPrice PriceDetail;
    public readonly string Name;
    public readonly string ImagePath;
    public readonly string PublicStatusName;
    public readonly bool Available;
    public readonly int ProductPageID;
    public readonly string ProductPageAlias;

    /// <summary>
    /// Creates a model from an item from a product listing.
    /// </summary>
    /// <param name="productPage">Product's page.</param>
    /// <param name="priceDetail">Price of the product.</param>
    /// <param name="publicStatusName">Display name of the product's public
status.</param>
    public ProductListItemViewModel(SKUTreeNode productPage, ProductPrice
priceDetail, string publicStatusName)
    {
        // Sets the page information
        Name = productPage.DocumentName;
        ProductPageID = productPage.NodeID;
        ProductPageAlias = productPage.NodeAlias;

        // Sets the SKU information
        ImagePath = productPage.SKU.SKUImagePath;
        Available = !productPage.SKU.SKUSellOnlyAvailable || productPage.SKU.
SKUAvailableItems > 0;
        PublicStatusName = publicStatusName;

        // Sets the price
        PriceDetail = priceDetail;
    }
}
```

5. Add a new controller for each product page type you intend to display. The controller's action retrieves products of the specific product page type and displays them. The following example uses the **LearningProductType** as an example.



```
public class LearningProductTypeController : Controller
{
    private readonly string siteName = SiteContext.CurrentSiteName;
    private readonly IShoppingService shoppingService;
    private readonly IPricingService pricingService;

    /// <summary>
    /// Constructor.
    /// You can use a dependency injection container to initialize the
services.
    /// </summary>
    public LearningProductTypeController()
    {
        shoppingService = new ShoppingService();
        pricingService = new PricingService();
    }

    /// <summary>
    /// Displays a product listing page of the class's product page type.
    /// </summary>
    public ActionResult Index()
    {
        // Gets products of the product page type (via the generated page
type code)
        List<LearningProductType> products = LearningProductTypeProvider.
GetLearningProductTypes()
            .LatestVersion(false)
            .Published(true)
            .OnSite(siteName)
            .Culture("en-US")
            .CombineWithDefaultCulture()
            .WhereTrue("SKUEnabled")
            .OrderByDescending("SKUInStoreFrom")
            .ToList();

        // Displays the action's view with an initialized view model
        return View(products.Select(
            product => new ProductListItemViewModel(
                product,
                pricingService.CalculatePrice(product.SKU, shoppingService.
GetCurrentShoppingCart()),
                product.Product.PublicStatus?.PublicStatusDisplayName))
            );
    }
}
```



We recommend using a [dependency injection container](#) to initialize service instances. When configuring the lifetime scope for *ShoppingService* and *PricingService*, create a shared instance (singleton) for all requests.

6. Add a view that sets the appearance of the product listing.



```
@using Kentico.Ecommerce

<h2>Product listing of the LearningProductType</h2>

<div>
    @foreach (ProductListItemViewModel product in Model)
    {
        /* Creates a hyperlink to the product controller to display the product
        detail page */
        <a href="@Url.RouteUrl("Product", new {id = product.ProductPageID,
productAlias = product.ProductPageAlias})">
            <h3>@product.Name</h3>

            /* Displays information about the product's public status */
            @if (!string.IsNullOrEmpty(product.PublicStatusName))
            {
                <span>@product.PublicStatusName</span>
            }

            /* Displays the product's image */
            @if (!string.IsNullOrEmpty(product.ImagePath))
            {
                
            }

            /* Displays the product's other properties */

            @{
                ProductPrice price = product.PriceDetail;
                Currency currency = price.Currency;
            }
            <div>
                @if (!product.Available)
                {
                    <span>Out of stock</span>
                }

                <span>@currency.FormatPrice(price.Price)</span>

                @if (price.ListPrice > price.Price)
                {
                    <s>@currency.FormatPrice(price.ListPrice)</s>
                }
            </div>
        </a>
    }
</div>
```



### Changing image size

To change the size of products' images, use the **ImageUrl** HTML helper method from the [Kentico.Content.Web.Mvc integration package](#). The system then automatically resizes the image based on the entered size while keeping the image's aspect ratio.

Parameters:

- *productImagePath* – a string representing a path to the image that you want to display
- *SizeConstrant* – a *SizeConstrant* object consisting of:
  - *Width* – you can enter the required width in pixels
  - *Height* – you can enter the required height in pixels
  - OR—
  - *Size* – you can enter both width and height in pixels – if used, this does not keep the aspect ratio
  - OR—
  - *MaxWidthOrHeight* – larger images are scaled down, smaller images are not scaled up

```
@Url.Kentico().ImageUrl(productImagePath, SizeConstraint.Size(width, height))
```

For example:

```

```

Visitors can now browse through products in the product listing and get to product detail pages by clicking on some product.