**SnipCart intergration with .NET Core and VS Code**

**Goal** : The goal of this document is to list the steps to integrate a functional cart system into any website using SnipCart.

**First things first: Setting up our development environment**

* Install Visual Studio Code: <https://code.visualstudio.com/>
* Install .NET Core SDK 1.0 RC3 build 004530: <https://github.com/dotnet/core/blob/master/release-notes/rc3-download.md>
* Install Node: <https://nodejs.org/en/>
* Working directory: aspnetcoreapp/snipcart
* Get node packages: npm install –g yo bower grunt-cli gulp generator-aspnet
* Project creation with Yeoman: yo aspnet > Web Application Basic [without Membership and Authorization] > Bootstrap 3.3.7 > snipcart
* Project build with CLI (will run at http://localhost:5000):
  + cd snipcart
  + dotnet restore
  + dotnet build
  + dotnet run
* Get VS Code extensions: C# and .Net Core Project Manager (Nuget)

At this point, everything should be working and our development environment is finally set up.

Let the fun begin!

**Reflecting on life (not really): What kind of data will our website display?**

While starting a new development project is always very exciting, it’s also crucial to kick things off on the right foot.

A question that has to be answered early on is: *What kind of data is our website going to serve*? Even though the chances of getting the answer exactly right on our first shot is extremely slim, only having a general idea will help us go a long way.

Since the sole purpose of this document is to integrate an e-commerce shopping cart to a website, we can say with confidence that the chances of us using some kind of **product** are pretty good.

Knowing that we will be dealing with products in our project, now is the time to determine what information we will need to have displayed to the user. Usually, products will have these:

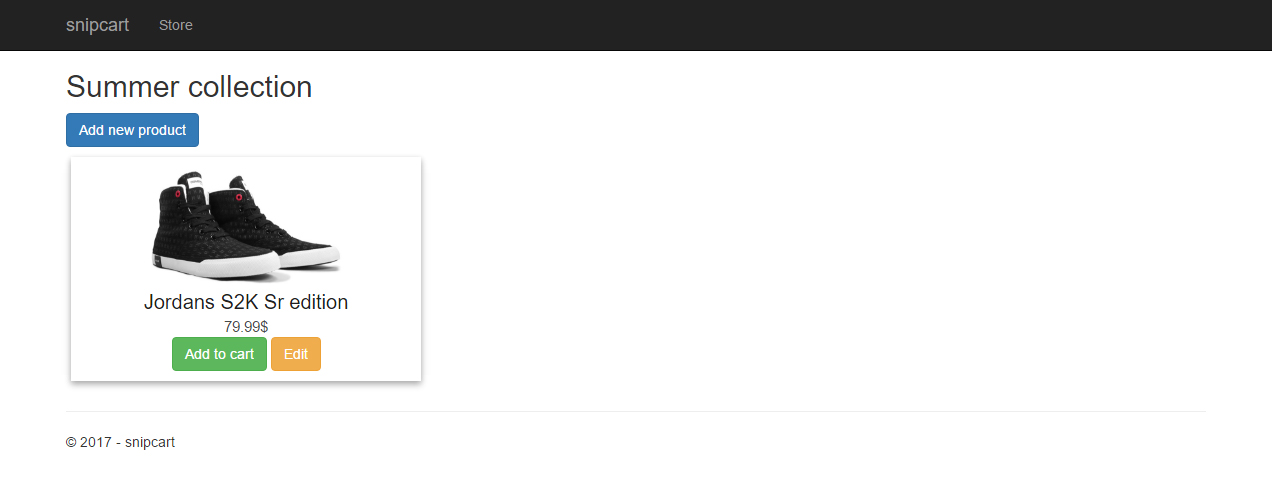
* A title
* A description
* A price
* Picture

Now that we know some of the info that every product will have, we can proceed to prototyping our storefront.

**Prototyping a nice static storefront**

Knowing that our website is simple and will only be showing some data about products, namely a title, description, price and one picture, we can quickly and efficiently start prototyping our storefront with dummy data.

We will also strategically insert placeholder buttons in the page to add and edit (and delete, which will appear when the edit window will open) products.



**Creating the model**

As we know, we will only be needing a single model for this project, which will be named “Product”. To set this up, we will create a new folder in the root of the project named “Models”, which we will then create a Product.cs file within it.

Our model will then need these attributes:

* Id (int)
* Title (string)
* Description (string)
* Price (double)
* Image (string)

Here is the model in code:

using System.Collections.Generic;

namespace snipcart.Models

{

public class Product

{

public int Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public double Price { get; set; }

public string Image { get; set; }

public List<Product> Products { get; set; }

}

}

**EntityFrameWork Core In-Memory database**

For simplicity, we will be using an in-memory database, which will simulate calls to a real EntityFrameWork database.

To set it up, let’s create a file called InMemoryDB.cs within the Models folder and insert the following code:

using Microsoft.EntityFrameworkCore;

using snipcart.Models;

namespace snipcart

{

public class InMemoryDB: DbContext

{

public InMemoryDB (DbContextOptions<InMemoryDB> options)

: base(options)

{

}

public DbSet<Product> Products { get; set; }

}

}

Next, in the Startup.cs file of the root of the project, use our In-Memory database by adding the following code to the top of the file:

using snipcart;

Now, let’s tell our project to use this service by altering the Configure method to the following:

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

loggerFactory.AddConsole(Configuration.GetSection("Logging"));

loggerFactory.AddDebug();

var context = app.ApplicationServices.GetService<InMemoryDB>();

AddTestData(context);

…

… And by adding the AddTestData() method at the end like so:

private static void AddTestData(InMemoryDB context)

{

var testProd1 = new snipcart.Models.Product

{

Id = 1,

Title = "Jordans S2K Sr edition",

Description = "The Jordans S2K Sr edition is the best bang for your buck.",

Price = 97.99,

Image = "http://simpleproductphotography.com/wp-content/uploads/2016/06/huf-converse-product-red-skidgrip-1.jpg"

};

var testProd2 = new snipcart.Models.Product

{

Id = 2,

Title = "Lamborghini Huracan",

Description = "The Lamborghini Huracan is definitely the best supercar for the money.",

Price = 278999.99,

Image = "http://1.bp.blogspot.com/-Gaj30dheGzE/VfGQL2uD0\_I/AAAAAAAAWJ4/IOomh6RXDpY/w800/lambo-huracan-roadster-rendering-ts-4.jpg"

};

context.Products.Add(testProd1);

context.Products.Add(testProd2);

context.SaveChanges();

}

In a nutshell, the code that we’ve just added in the Startup.cs file will do the following when the project starts up:

* Tell our project we want to use the snipcart namespace
* Tell our project to use the InMemoryDB service, which is our In-Memory database
* Create two dummy entries in our In-Memory database

At this point, we now have access to the InMemoryDB and its contents (exciting!).