# Getting Started With ASP.NET Core for macOS, Linux and Windows

Microsoft® Most Valuable Professional

Agus Kurniawan

# Copyright

Getting Started with ASP.NET Core for macOS, Linux, and Windows

Agus Kurniawan

1st Edition, 2017

# Table of Contents

Copyright
Preface
1. Preparing Development Environment
    1.1 ASP.NET Core
    1.2 System Requirements
    1.3 Development Tools
2 Deploying ASP.NET Core
    2.1 Getting Started
    2.2 Deploying ASP.NET Core on macOS, Linux and Windows
    2.3 Testing
3. ASP.NET Core MVC Development
    3.1 Getting Started
    3.2 ASP.NET Core Template
    3.3 Building Your First ASP.NET Core MVC Application
        3.3.1 Creating A Project
        3.3.2 Build and Run the Program
    3.4 Building ASP.NET Core MVC Application
        3.4.1 Creating a Project
        3.4.2 Configuring ASP.NET Core MVC Project
        3.4.3 Writing Program
            3.4.3.1 Creating Model
            3.4.3.2 Creating Controller
            3.4.3.3 Creating View
        3.4.4 Testing
    3.5 What's Next?
4. ASP.NET Core API Development
    4.1 Getting Started

- 4.2 Building ASP.NET Core API Application
  - 4.2.1 Creating a Project
  - 4.2.2 Writing Program for ASP.NET Core API
  - 4.2.3 Testing
- 4.3 Client Application for ASP.NET Core API
  - 4.3.1 Creating Controller
  - 4.3.2 Creating View
- 4.4 Testing

5. ASP.NET Core and Angular 2
- 5.1 Getting Started
- 5.2 ASP.NET Core and Angular 2 Project Template
- 5.3 Run Demo: ASP.NET Core Angular SPA
  - 5.3.1 Creating a Project
  - 5.3.2 Running the Program
- 5.4 What's The Next?

6. ASP.NET Core, Entity Framework Core and PostgreSQL
- 6.1 Getting Started
- 6.2 PostgreSQL Database
- 6.3 Entity Framework Core
- 6.4 Demo Database Application
  - 6.4.1 Build PostgreSQL Database
  - 6.4.2 Creating a ASP.NET Core MVC Project
  - 6.4.3 Configure ASP.NET Core MVC Project
  - 6.4.4 Develop Data Access
  - 6.4.5 Develop ASP.NET Core MVC Controllers
  - 6.4.6 Develop ASP.NET Core MVC Views
  - 6.4.7 Testing

Source Code

Contact

# Preface

This book was written to help anyone want to get started with ASP.NET Core. It describes ASP.NET Core deploying and the basic elements of ASP.NET Core and shows how to work with several ASP.NET Core features.

Agus Kurniawan

Depok, January 2017

# 1. Preparing Development Environment

## 1.1 ASP.NET Core

ASP.NET Core is a lean and composable framework for building web and cloud applications. ASP.NET Core is fully open source and available on GitHub, http://github.com/aspnet/home .

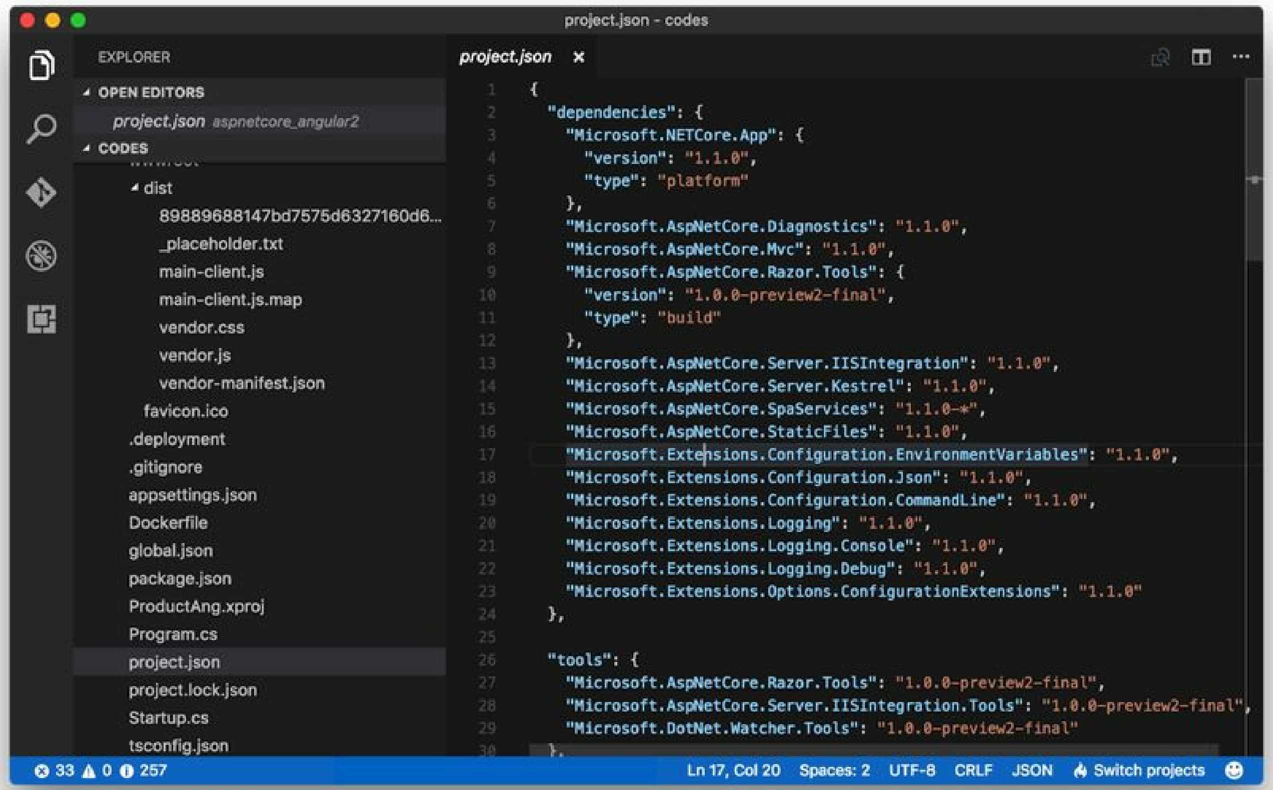
This book will focus on how to develop ASP.NET Core 1.1 on OS X, Linux and Windows platforms. We use ASP.NET Core 1.1 version.

## 1.2 System Requirements

For testing, I use Debian/Ubuntu Linux, macOS and Windows 10. You can use any Linux distribution which inheritance from Debian based.

## 1.3 Development Tools

In general, we can use any editor to write ASP.NET Core program. You use Sublime Text, http://www.sublimetext.com/ , to write ASP.NET Core codes. Microsoft also released Visual Studio Code and you can download it on https://code.visualstudio.com/ . In this book, I use Visual Studio Code to develop ASP.NET Core program.



```json
{
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.1.0",
      "type": "platform"
    },
    "Microsoft.AspNetCore.Diagnostics": "1.1.0",
    "Microsoft.AspNetCore.Mvc": "1.1.0",
    "Microsoft.AspNetCore.Razor.Tools": {
      "version": "1.0.0-preview2-final",
      "type": "build"
    },
    "Microsoft.AspNetCore.Server.IISIntegration": "1.1.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.1.0",
    "Microsoft.AspNetCore.SpaServices": "1.1.0-*",
    "Microsoft.AspNetCore.StaticFiles": "1.1.0",
    "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.1.0",
    "Microsoft.Extensions.Configuration.Json": "1.1.0",
    "Microsoft.Extensions.Configuration.CommandLine": "1.1.0",
    "Microsoft.Extensions.Logging": "1.1.0",
    "Microsoft.Extensions.Logging.Console": "1.1.0",
    "Microsoft.Extensions.Logging.Debug": "1.1.0",
    "Microsoft.Extensions.Options.ConfigurationExtensions": "1.1.0"
  },

  "tools": {
    "Microsoft.AspNetCore.Razor.Tools": "1.0.0-preview2-final",
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final",
    "Microsoft.DotNet.Watcher.Tools": "1.0.0-preview2-final"
  },
```

# 2 Deploying ASP.NET Core

In this chapter we will learn how to deploy ASP.NET Core.

## 2.1 Getting Started

In this section we start to deploy ASP.NET Core on your platform. In this book I use ASP.NET Core 1.1. Please follow installation steps on next section.

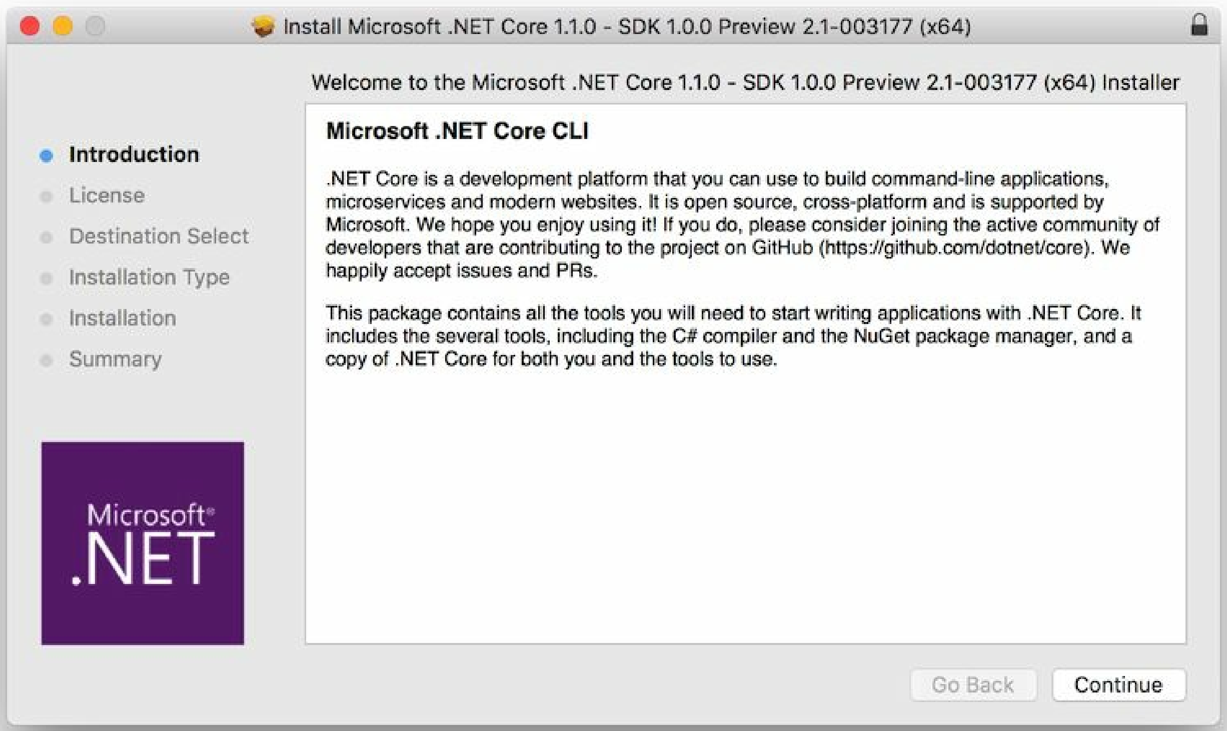
## 2.2 Deploying ASP.NET Core on macOS, Linux and Windows

To install ASP.NET Core on your platform, you should install .NET Core SDK. You can download it on https://www.microsoft.com/net/download/core.

Installation steps can be read in this website:

- macOS: https://www.microsoft.com/net/core#macos
- Linux: https://www.microsoft.com/net/core#linuxredhat ,
  Ubuntu: https://www.microsoft.com/net/core#linuxubuntu
- Windows: https://www.microsoft.com/net/core#windowsvs2015

You can see .NET Core SDK 1.1 on macOS.



Welcome to the Microsoft .NET Core 1.1.0 - SDK 1.0.0 Preview 2.1-003177 (x64) Installer

**Microsoft .NET Core CLI**

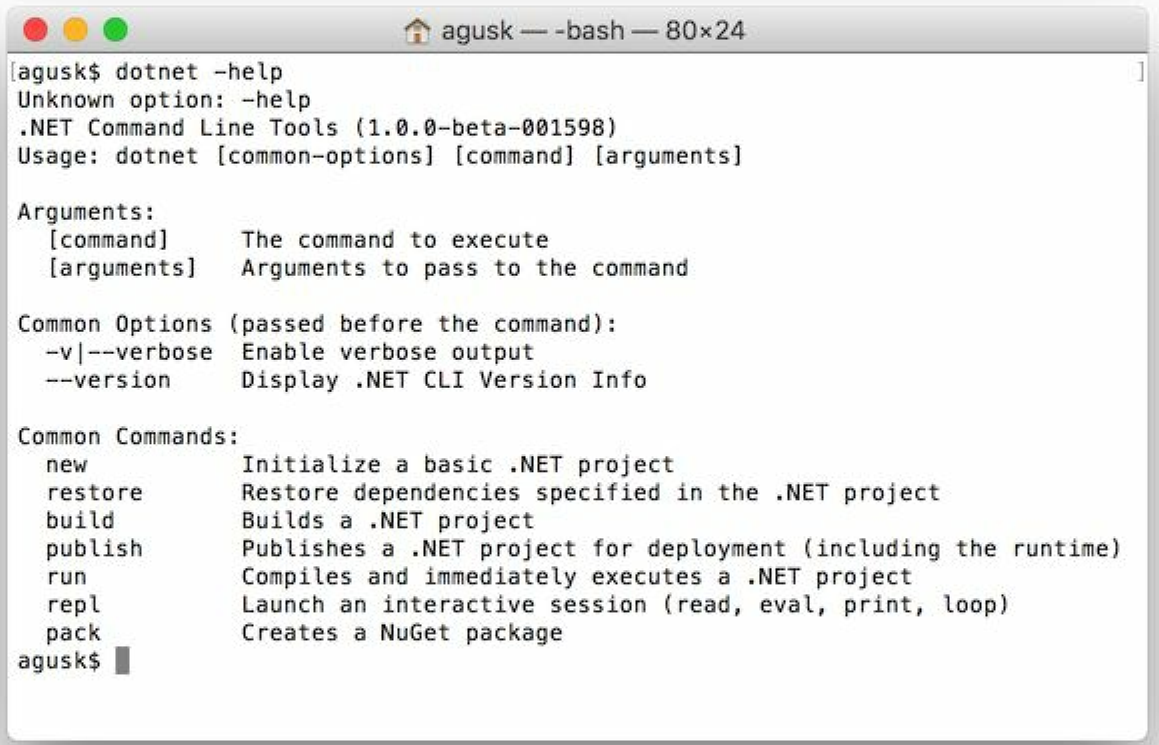
.NET Core is a development platform that you can use to build command-line applications, microservices and modern websites. It is open source, cross-platform and is supported by Microsoft. We hope you enjoy using it! If you do, please consider joining the active community of developers that are contributing to the project on GitHub (https://github.com/dotnet/core). We happily accept issues and PRs.

This package contains all the tools you will need to start writing applications with .NET Core. It includes the several tools, including the C# compiler and the NuGet package manager, and a copy of .NET Core for both you and the tools to use.

If done, you can test. Open Terminal and type this command.

```
$ dotnet -help
```

You should see a list of dotnet command.



```
agusk$ dotnet -help
Unknown option: -help
.NET Command Line Tools (1.0.0-beta-001598)
Usage: dotnet [common-options] [command] [arguments]

Arguments:
  [command]     The command to execute
  [arguments]   Arguments to pass to the command

Common Options (passed before the command):
  -v|--verbose  Enable verbose output
  --version     Display .NET CLI Version Info

Common Commands:
  new           Initialize a basic .NET project
  restore       Restore dependencies specified in the .NET project
  build         Builds a .NET project
  publish       Publishes a .NET project for deployment (including the runtime)
  run           Compiles and immediately executes a .NET project
  repl          Launch an interactive session (read, eval, print, loop)
  pack          Creates a NuGet package
agusk$
```

## 2.3 Testing

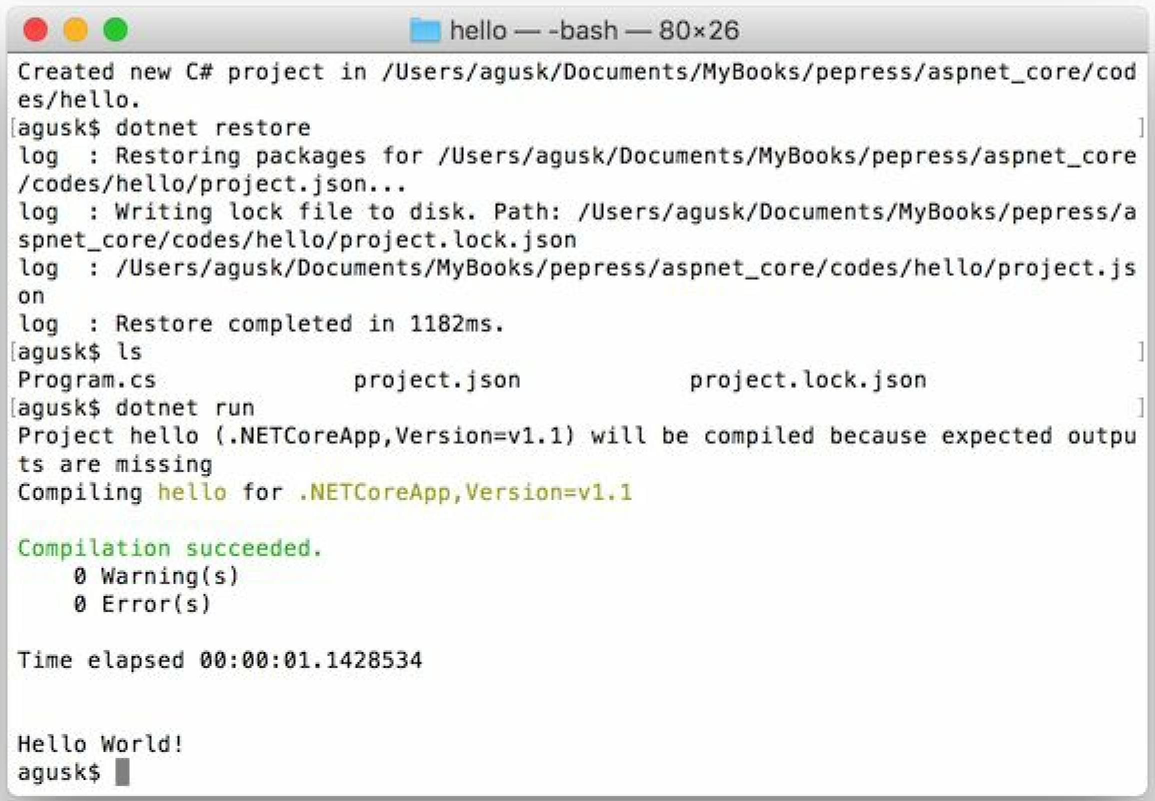
To test our .NET Core, we build a simple Console application. From Terminal, type these commands.

```
$ mkdir hello
$ cd hello
$ dotnet new
```

Now you can reload all required libraries for .NET Core application. Then, run the program.

```
$ dotnet restore
$ dotnet run
```

You should see "Hello World!" message on Terminal. You can see it in Figure below from my implementation.



```
Created new C# project in /Users/agusk/Documents/MyBooks/pepress/aspnet_core/cod
es/hello.
agusk$ dotnet restore
log  : Restoring packages for /Users/agusk/Documents/MyBooks/pepress/aspnet_core
/codes/hello/project.json...
log  : Writing lock file to disk. Path: /Users/agusk/Documents/MyBooks/pepress/a
spnet_core/codes/hello/project.lock.json
log  : /Users/agusk/Documents/MyBooks/pepress/aspnet_core/codes/hello/project.js
on
log  : Restore completed in 1182ms.
agusk$ ls
Program.cs              project.json            project.lock.json
agusk$ dotnet run
Project hello (.NETCoreApp,Version=v1.1) will be compiled because expected outpu
ts are missing
Compiling hello for .NETCoreApp,Version=v1.1

Compilation succeeded.
    0 Warning(s)
    0 Error(s)

Time elapsed 00:00:01.1428534


Hello World!
agusk$
```

# 3. ASP.NET Core MVC Development

In this chapter I'm going to explain how to work with ASP.NET Core MVC development.

## 3.1 Getting Started

In this chapter, we learn how to write a program for ASP.NET Core MVP from scratch.

## 3.2 ASP.NET Core Template

To simply our development, we can use ASP.NET Core template from yo. Firstly, your computer should already installed Node.js, https://nodejs.org/.

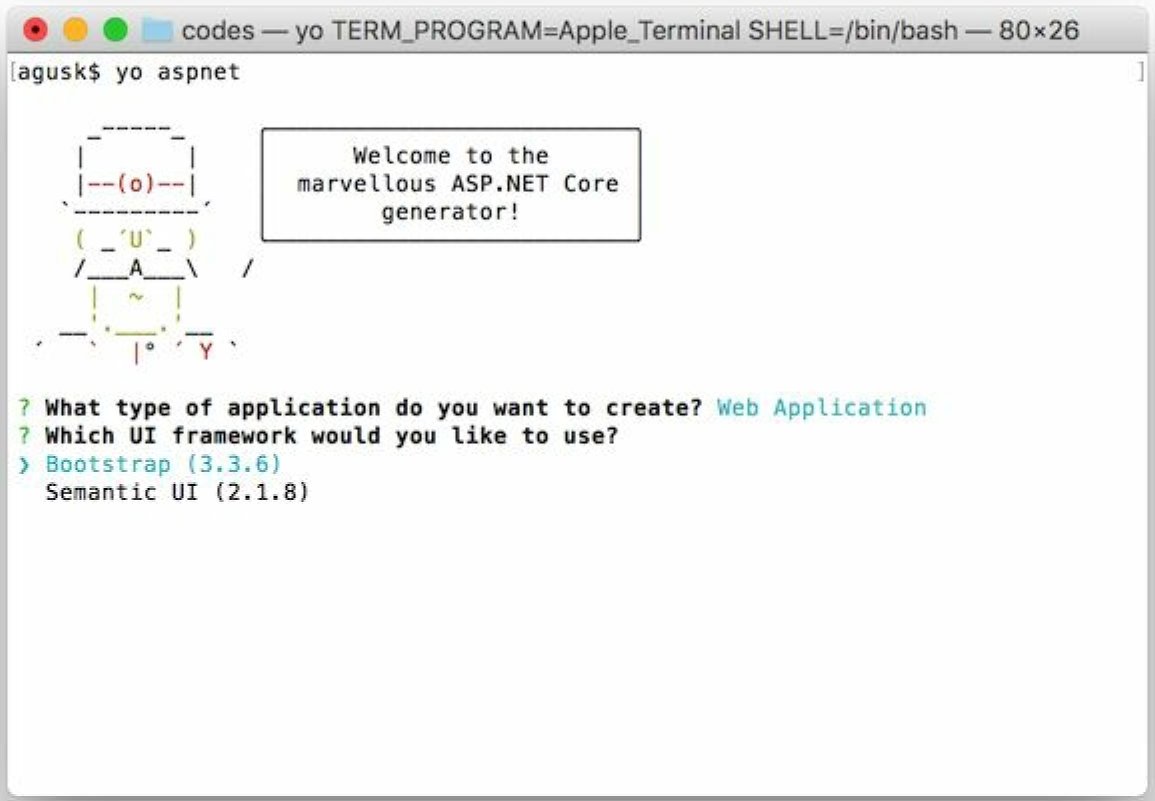
After installed Node.js, you can install yo and ASP.NET Core template. Type these commands.

```
$ npm install -g yo bower
$ npm install -g generator-aspnet
```

If succeed, you can test it. Open Terminal and type this command.

```
$ yo aspnet
```

You should see a list of template, shown in Figure below.



```
agusk$ yo aspnet

     _-----_
    |       |    ╭──────────────────────────╮
    |--(o)--|    │      Welcome to the      │
   `---------´   │  marvellous ASP.NET Core │
    ( _´U`_ )    │        generator!        │
    /___A___\   /╰──────────────────────────╯
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What type of application do you want to create? Web Application
? Which UI framework would you like to use?
❯ Bootstrap (3.3.6)
  Semantic UI (2.1.8)
```

Now you're ready to develop ASP.NET Core.

## 3.3 Building Your First ASP.NET Core MVC Application

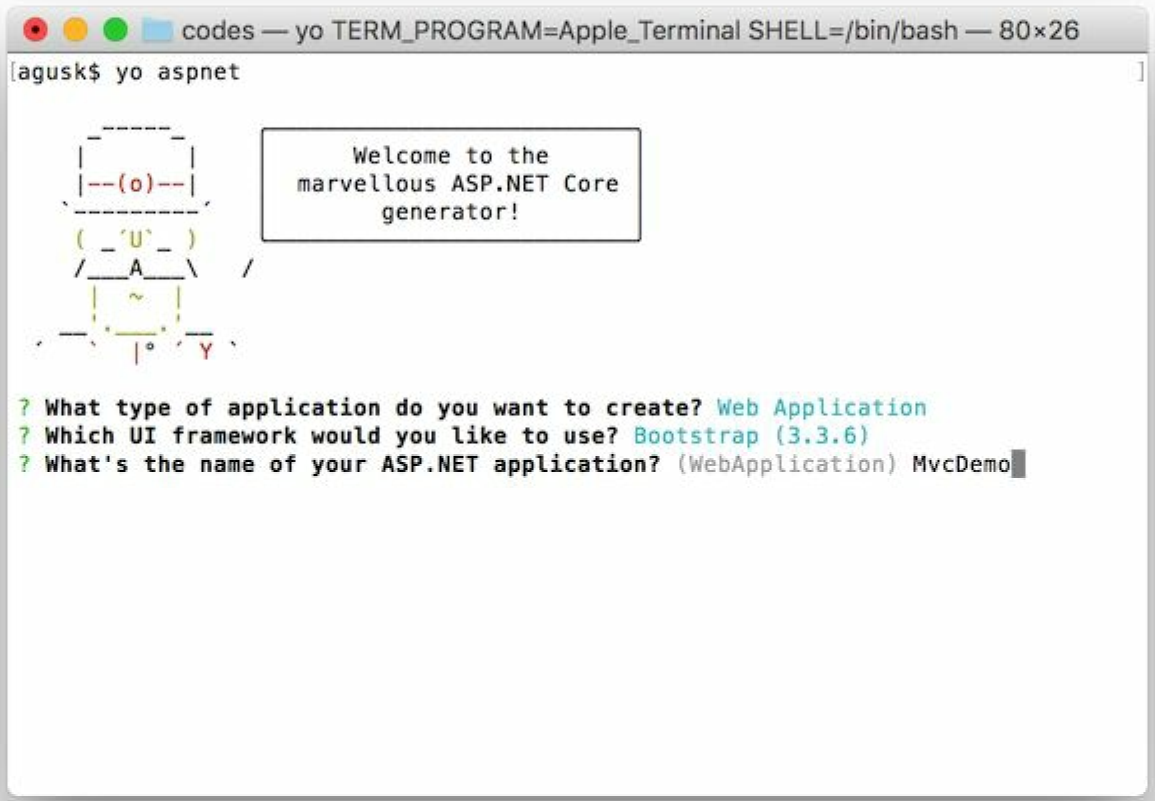
In this case, we build a simple program for ASP.NET Core MVC. We use the template from yo aspnet.

### 3.3.1 Creating A Project

In this section, we create a new project for ASP.NET Core MVC. Open Terminal or Command Promt. Then, you can type this command on Terminal.

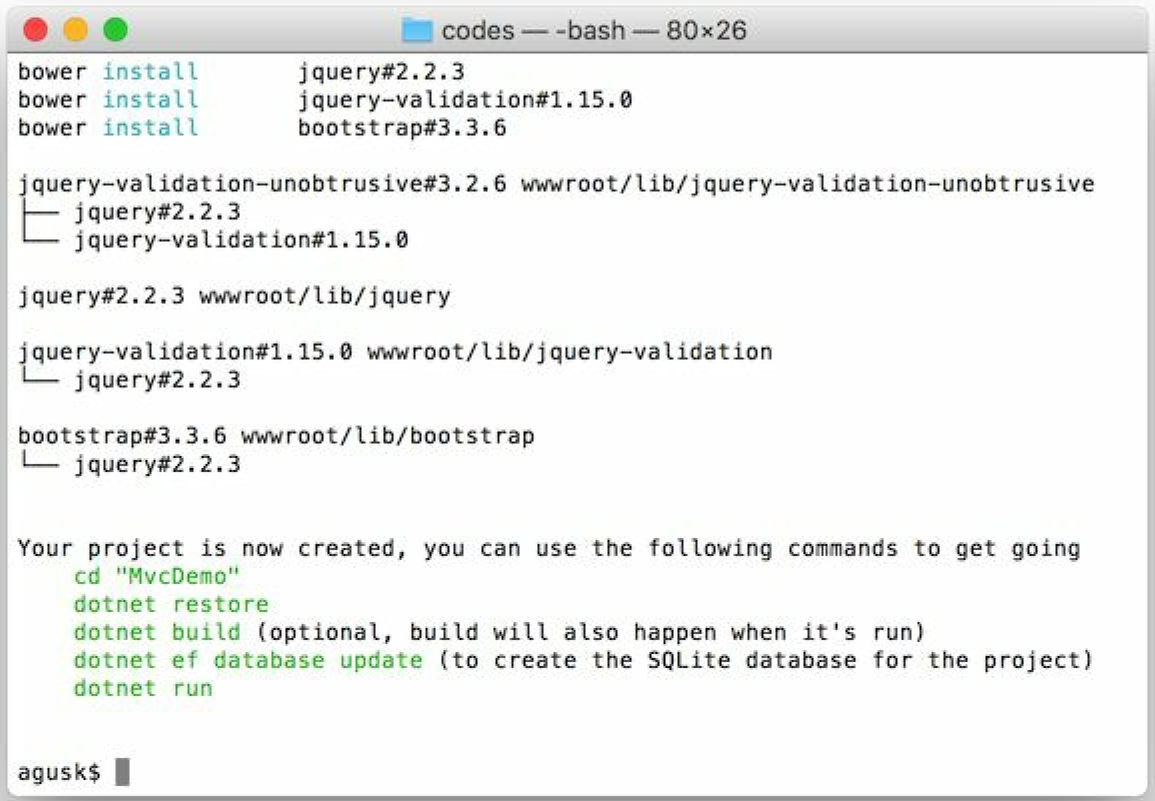
```
$ yo aspnet
```

Select Web Application with Bootstrap for UI framework. Given the project name, for instance, MvcDemo.


codes — yo TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash — 80×26

```
agusk$ yo aspnet

     _-----_     ╭──────────────────────────╮
    |       |    │      Welcome to the      │
    |--(o)--|    │ marvellous ASP.NET Core  │
   `---------´   │        generator!        │
    ( _´U`_ )    ╰──────────────────────────╯
    /___A___\   /
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What type of application do you want to create? Web Application
? Which UI framework would you like to use? Bootstrap (3.3.6)
? What's the name of your ASP.NET application? (WebApplication) MvcDemo
```

If succeed, you should this form.



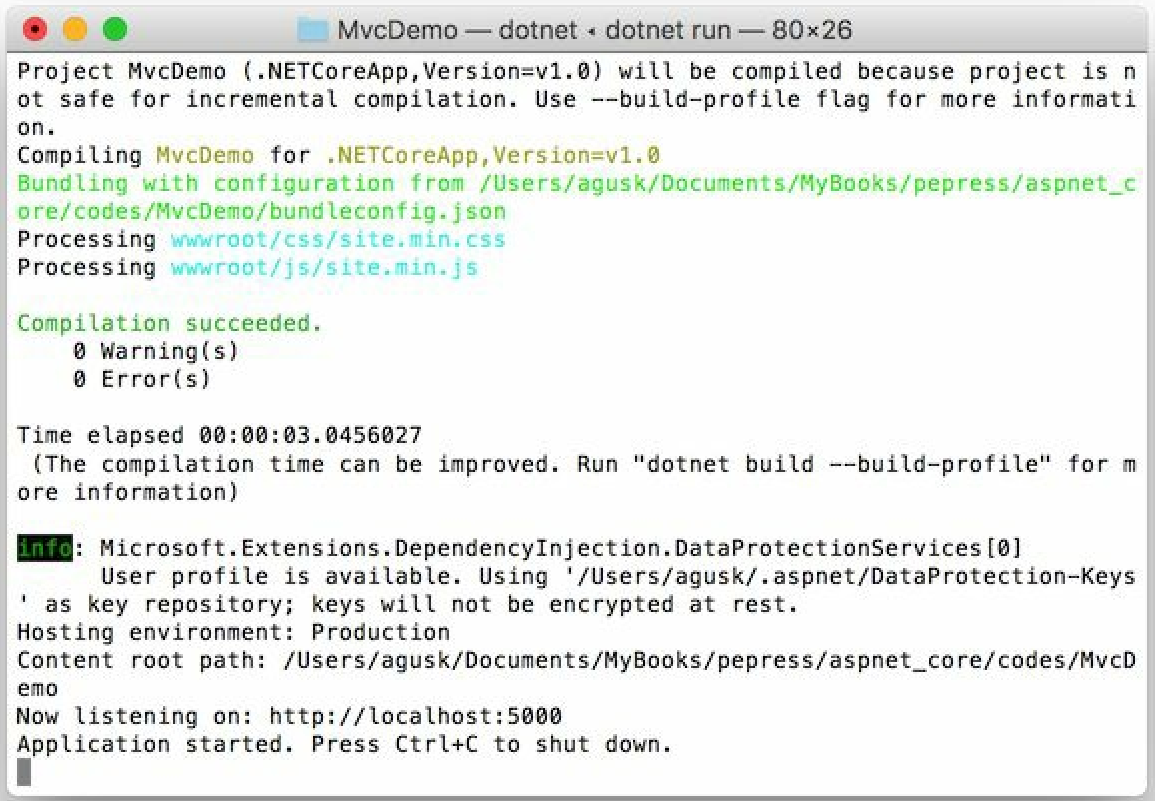
```
bower install       jquery#2.2.3
bower install       jquery-validation#1.15.0
bower install       bootstrap#3.3.6

jquery-validation-unobtrusive#3.2.6 wwwroot/lib/jquery-validation-unobtrusive
├── jquery#2.2.3
└── jquery-validation#1.15.0

jquery#2.2.3 wwwroot/lib/jquery

jquery-validation#1.15.0 wwwroot/lib/jquery-validation
└── jquery#2.2.3

bootstrap#3.3.6 wwwroot/lib/bootstrap
└── jquery#2.2.3


Your project is now created, you can use the following commands to get going
    cd "MvcDemo"
    dotnet restore
    dotnet build (optional, build will also happen when it's run)
    dotnet ef database update (to create the SQLite database for the project)
    dotnet run


agusk$ 
```

### 3.3.2 Build and Run the Program

Now you can test the program. Firstly, we load all required libraries using dotnet restore. Then, we execute the program using dotnet run.

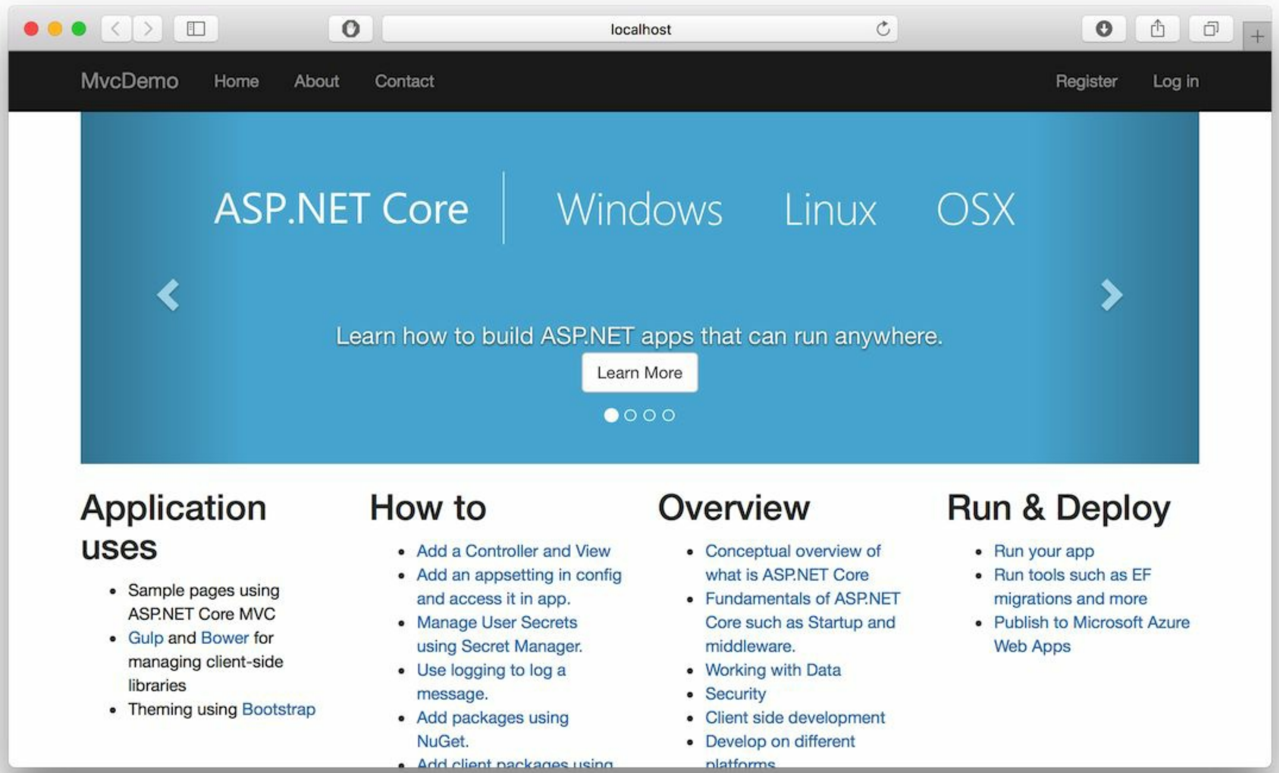
```
$ cd MvcDemo
$ dotnet restore
$ dotnet run
```

If success, you can see the program output.



```
Project MvcDemo (.NETCoreApp,Version=v1.0) will be compiled because project is n
ot safe for incremental compilation. Use --build-profile flag for more informati
on.
Compiling MvcDemo for .NETCoreApp,Version=v1.0
Bundling with configuration from /Users/agusk/Documents/MyBooks/pepress/aspnet_c
ore/codes/MvcDemo/bundleconfig.json
Processing wwwroot/css/site.min.css
Processing wwwroot/js/site.min.js

Compilation succeeded.
    0 Warning(s)
    0 Error(s)

Time elapsed 00:00:03.0456027
 (The compilation time can be improved. Run "dotnet build --build-profile" for m
ore information)

info: Microsoft.Extensions.DependencyInjection.DataProtectionServices[0]
      User profile is available. Using '/Users/agusk/.aspnet/DataProtection-Keys
' as key repository; keys will not be encrypted at rest.
Hosting environment: Production
Content root path: /Users/agusk/Documents/MyBooks/pepress/aspnet_core/codes/MvcD
emo
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Now you can open a browser and navigate to http://localhost:5000. You can see the form which is shown in Figure below.



Application uses

- Sample pages using ASP.NET Core MVC
- Gulp and Bower for managing client-side libraries
- Theming using Bootstrap

How to

- Add a Controller and View
- Add an appsetting in config and access it in app.
- Manage User Secrets using Secret Manager.
- Use logging to log a message.
- Add packages using NuGet.
- Add client packages using

Overview

- Conceptual overview of what is ASP.NET Core
- Fundamentals of ASP.NET Core such as Startup and middleware.
- Working with Data
- Security
- Client side development
- Develop on different platforms

Run & Deploy

- Run your app
- Run tools such as EF migrations and more
- Publish to Microsoft Azure Web Apps

## 3.4 Building ASP.NET Core MVC Application

The second demo is to build ASP.NET Core MVC application from scratch. We develop a simple entry form and utilize ASP.NET Core MVC session. The following is our scenario:

- User fills a form to create a new product
- User clicks Save button to save the data
- The data will be stored to a session stack
- Then, the data will be shown in another page

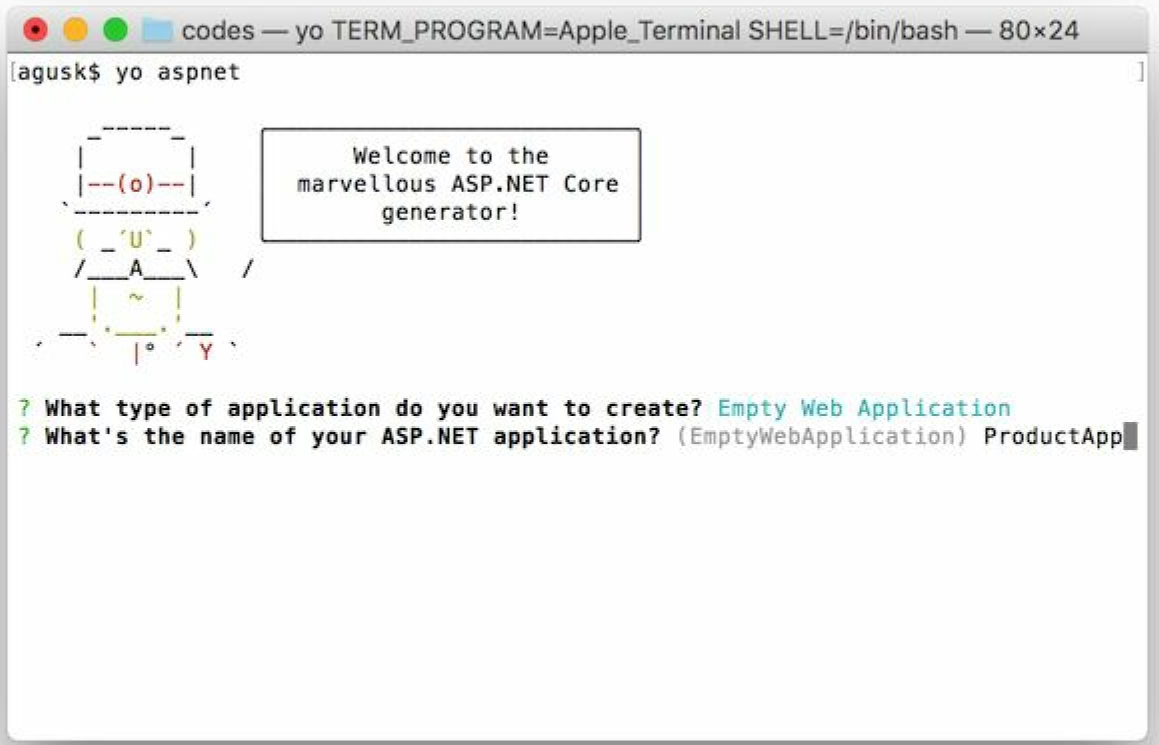
Let's start to develop.

### 3.4.1 Creating a Project

Firstly, we create a project by using yo aspnet. Open Terminal and type this command.

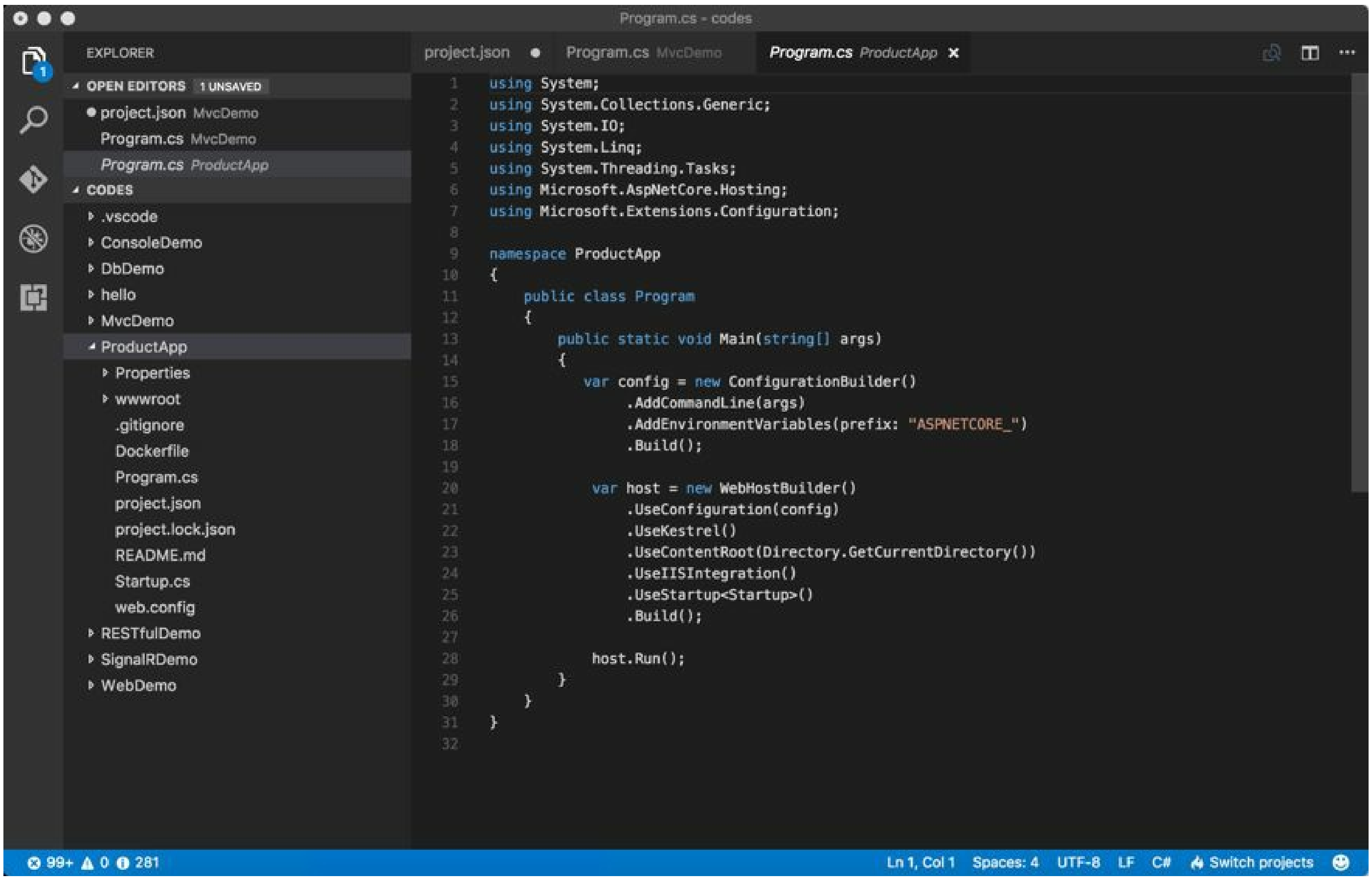
```
$ yo aspnet
```

Please select Empty Web Application. Given project name, ProductApp.



```
codes — yo TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash — 80×24
agusk$ yo aspnet

     _-----_
    |       |    ╭──────────────────────────╮
    |--(o)--|    │      Welcome to the      │
   `---------´   │ marvellous ASP.NET Core  │
    ( _´U`_ )    │        generator!        │
    /___A___\   /╰──────────────────────────╯
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What type of application do you want to create? Empty Web Application
? What's the name of your ASP.NET application? (EmptyWebApplication) ProductApp
```

You should select Empty Web Application. You can load this folder using Visual Studio Code.



```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;

namespace ProductApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var config = new ConfigurationBuilder()
                .AddCommandLine(args)
                .AddEnvironmentVariables(prefix: "ASPNETCORE_")
                .Build();

             var host = new WebHostBuilder()
                .UseConfiguration(config)
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .Build();

             host.Run();
        }
    }
}
```

Then, you create the following folder:

- Controllers
- Models
- Views

## 3.4.2 Configuring ASP.NET Core MVC Project

We need project a config file, called **project.json**, on the project root folder. It consists of required libraries. Please write this script for project.json file to enable MVC on ASP.NET Core.

```
{
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.1.0",
      "type": "platform"
    },
    "Microsoft.AspNetCore.Diagnostics": "1.1.0",
    "Microsoft.AspNetCore.Mvc": "1.1.0",
    "Microsoft.AspNetCore.Razor.Tools": {
      "version": "1.0.0-preview2-final",
      "type": "build"
    },
    "Microsoft.AspNetCore.Server.IISIntegration": "1.1.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.1.0",
    "Microsoft.AspNetCore.SpaServices": "1.1.0-*",
    "Microsoft.AspNetCore.StaticFiles": "1.1.0",
    "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.1.0",
    "Microsoft.Extensions.Configuration.Json": "1.1.0",
    "Microsoft.Extensions.Configuration.CommandLine": "1.1.0",
    "Microsoft.Extensions.Logging": "1.1.0",
    "Microsoft.Extensions.Logging.Console": "1.1.0",
    "Microsoft.Extensions.Logging.Debug": "1.1.0",
    "Microsoft.Extensions.Options.ConfigurationExtensions": "1.1.0"
  },

  "tools": {
    "Microsoft.AspNetCore.Razor.Tools": "1.0.0-preview2-final",
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final",
    "Microsoft.DotNet.Watcher.Tools": "1.0.0-preview2-final"
  },

  "frameworks": {
    "netcoreapp1.1": {
      "imports": [
        "dotnet5.6",
        "portable-net45+win8"
      ]
    }
  },

  "buildOptions": {
    "emitEntryPoint": true,
    "preserveCompilationContext": true,
    "compile": {
      "exclude": ["node_modules"]
    }
  },

  "runtimeOptions": {
    "configProperties": {
      "System.GC.Server": true
    }
  },

  "publishOptions": {
    "include": [
      "appsettings.json",
      "ClientApp/dist",
      "Views",
      "web.config",
      "wwwroot"
    ],
    "exclude": [
      "wwwroot/dist/*.map"
    ]
  },

  "scripts": {
    "prepublish": [
      "npm install",
      "node node_modules/webpack/bin/webpack.js --config webpack.config.vendor.js --env.prod",
      "node node_modules/webpack/bin/webpack.js --env.prod"
    ],
    "postpublish": [ "dotnet publish-iis --publish-folder %publish:OutputPath% --framework %pub
  },

  "tooling": {
    "defaultNamespace": "ProductAng"
  }
}
```

App will start by executing **Startup.cs** file. Create this file and write this code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

using Microsoft.AspNetCore.Mvc.ViewFeatures;
namespace ProductApp
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the conta
        // For more information on how to configure your application, visit https://go.microsof
        public void ConfigureServices(IServiceCollection services)
        {
            // add ASP.NET MVC service
            services.AddMvc();
            services.AddSingleton<ITempDataProvider, CookieTempDataProvider>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP reques
        public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory 
        {
            loggerFactory.AddConsole();

            // configure ASP.NET MVC Core
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
            }

            app.UseStaticFiles();
            // Add external authentication middleware below. To configure them please see https

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });


        }
    }
}
```

## 3.4.3 Writing Program

In this section, we write our ASP.NET Core MVC program.

### 3.4.3.1 Creating Model

A model holds a data. In our case, we create a module, called Product, which consists of four properties. Create a file, called **Product.cs**, and write this code.

```
using System.ComponentModel.DataAnnotations;

namespace ProductApp.Models
{
    public class Product
    {
        [Required]
        [MinLength(4)]
        [Display( Name = "Name" )]
        public string Name { get; set; }
        [Display( Name = "Product Code" )]
        public string ProductCode { get; set; }
        [Display( Name = "Quantity" )]
        public int Quantity { get; set; }
        [Display( Name = "Is Discount" )]
        public bool IsDiscount { get; set; }
    }
}
```

Save this file into Models folder.

### 3.4.3.2 Creating Controller

Now we create MVC controller, called Home controller. Create a file, called **HomeController.cs**, and write this code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

using ProductApp.Models;

namespace ProductApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
        public IActionResult Create()
        {
            var product = new Product();
            return View(product);
        }
        [HttpPost]
        public IActionResult Create(Product obj)
        {
            // do database processing
            ///////

            // for testing, save it into a session
            TempData["name"] = obj.Name;
            TempData["code"] = obj.ProductCode;
            TempData["quantity"] = obj.Quantity;
            TempData["discount"] = obj.IsDiscount.ToString();

            return RedirectToAction("Save");
        }
        public IActionResult Save()
        {

            var product = new Product();
            product.Name = (string)TempData["name"];
            product.ProductCode = (string)TempData["code"];
            product.Quantity = (int)TempData["quantity"];
            product.IsDiscount = Convert.ToBoolean((string)TempData["discount"]);

            return View(product);
        }


    }
}
```

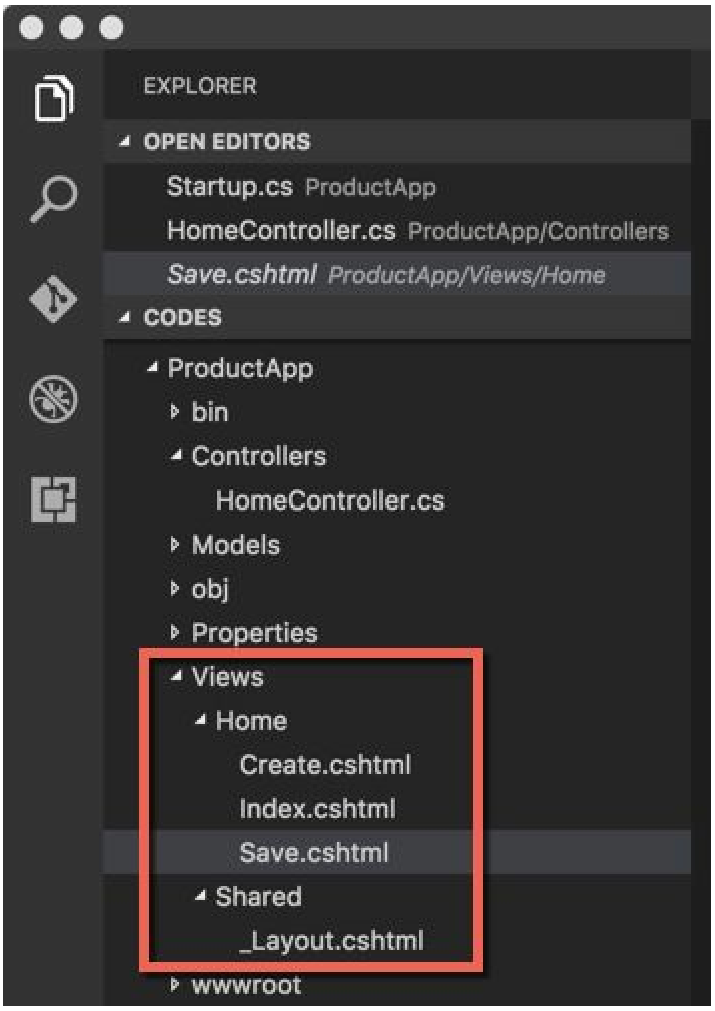
### 3.4.3.3 Creating View

We create three page files as follows:

- Index.cshtml
- Create.cshtml
- Save.cshtml
- _Layout.cshtml

You create a folder, Home, inside Views folder. Then, add Index.cshtml, Create.cshtml, and Save.cshtml files. Furthermore, create a folder, Shared. Then, add _Layout.cshtml file inside the folder.

You can see the view structure in Figure below.



We also create a layout, called **_Shared.cshtml**. This is used for a view template. Write this script.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET MVC Core Application</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.r
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossor
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target='
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a href="/">Home</a></li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <footer>
```

```
            <p>&copy; @DateTime.Now.Year - My ASP.NET MVC Core Application</p>
        </footer>
    </div>
</body>
</html>
```

The following is a script for **Index.cshtml** file.

```
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "Home Page";

}

<div class="jumbotron">
    <h1>ASP.NET Core MVC Demo</h1>
    <p class="lead"> Filling and submitting a form.</p>
    <p><a href="/Home/Create" class="btn btn-primary btn-large">Create &raquo;</a></p>
</div>
```

The following is a script for **Create.cshtml** file.

```
@using ProductApp.Models
@model Product
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "Create Product";
}

<div class="jumbotron">
    <h1>ASP.NET MVC Core</h1>
    <p class="lead">Create a new product. Fill this form. Then, click Save button if done.</p>

</div>
@using ( Html.BeginForm() )
    {
        <fieldset>
                <legend>Create a new Product</legend>

                <div class="editor-label">
                        @Html.LabelFor(model => model.Name)
                </div>
                <div class="editor-field">
                        @Html.TextBoxFor(model => model.Name)
                </div>
                <br/>

                <div class="editor-label">
                        @Html.LabelFor(model => model.ProductCode)
                </div>
                <div class="editor-field">
                        @Html.TextBoxFor( model => model.ProductCode )
                </div>
                <br/>
                <div class="editor-label">
                        @Html.LabelFor(model => model.Quantity)
                </div>
                <div class="editor-field">
                        @Html.TextBoxFor( model => model.Quantity )
                </div>
                <br/>

                <div class="editor-label">
                        <div style="float: left;">
                                @Html.CheckBoxFor(model => model.IsDiscount)
                        </div>
                        @Html.LabelFor(model => model.IsDiscount)
                </div>
                <br/>

                <input type="submit" value="Save" />

        </fieldset>
    }
<p><br></p>
```

The following is a script for **Save.cshtml** file.

```
@model ProductApp.Models.Product
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "Reading Session";
}

<div class="jumbotron">
    <h1>ASP.NET Core MVC</h1>
    <p class="lead">Reading data from ASP.NET Core MVC session.</p>
    <p><a href="/Home" class="btn btn-primary btn-large">Back to home &raquo;</a></p>
</div>
<div>
    <p>Name: @(Model != null ? Model.Name : "")</p>
    <p>Product Code: @(Model != null ? Model.ProductCode : "")</p>
    <p>Quantity: @(Model != null ? Model.Quantity : 0)</p>
    <p>Is discount: @(Model != null ? Model.IsDiscount : false)</p>
</div>
<p><br></p>
```

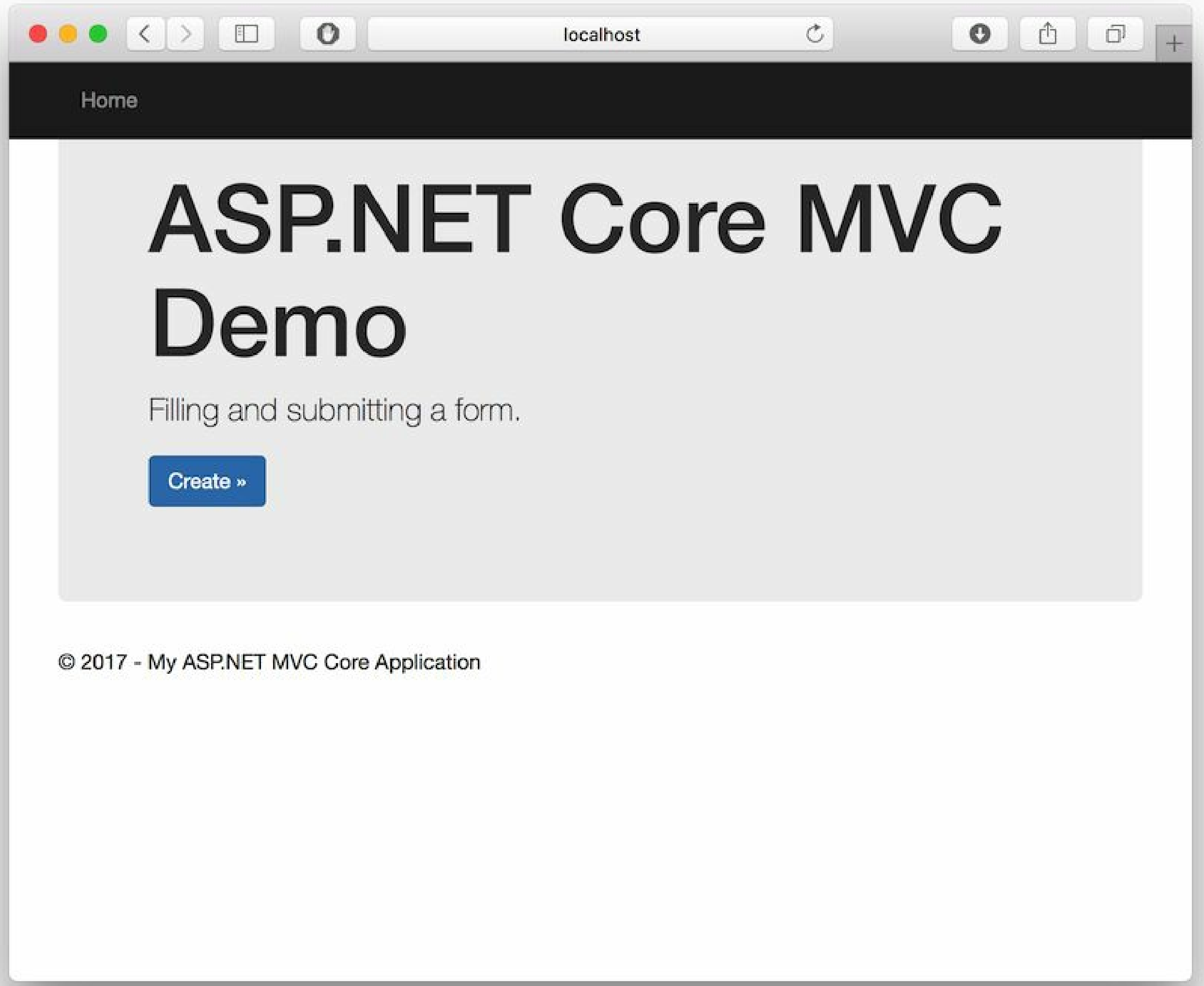
Save all files and store them into Views folder. See project structure.

## 3.4.4 Testing

Now you are ready to execute. Type these command to load required libraries and execute the program.

```
$ dotnet restore
$ dotnet build
$ dotnet run
```

Open a browser and navigate to http://localhost:5000 . Click **Create** button.

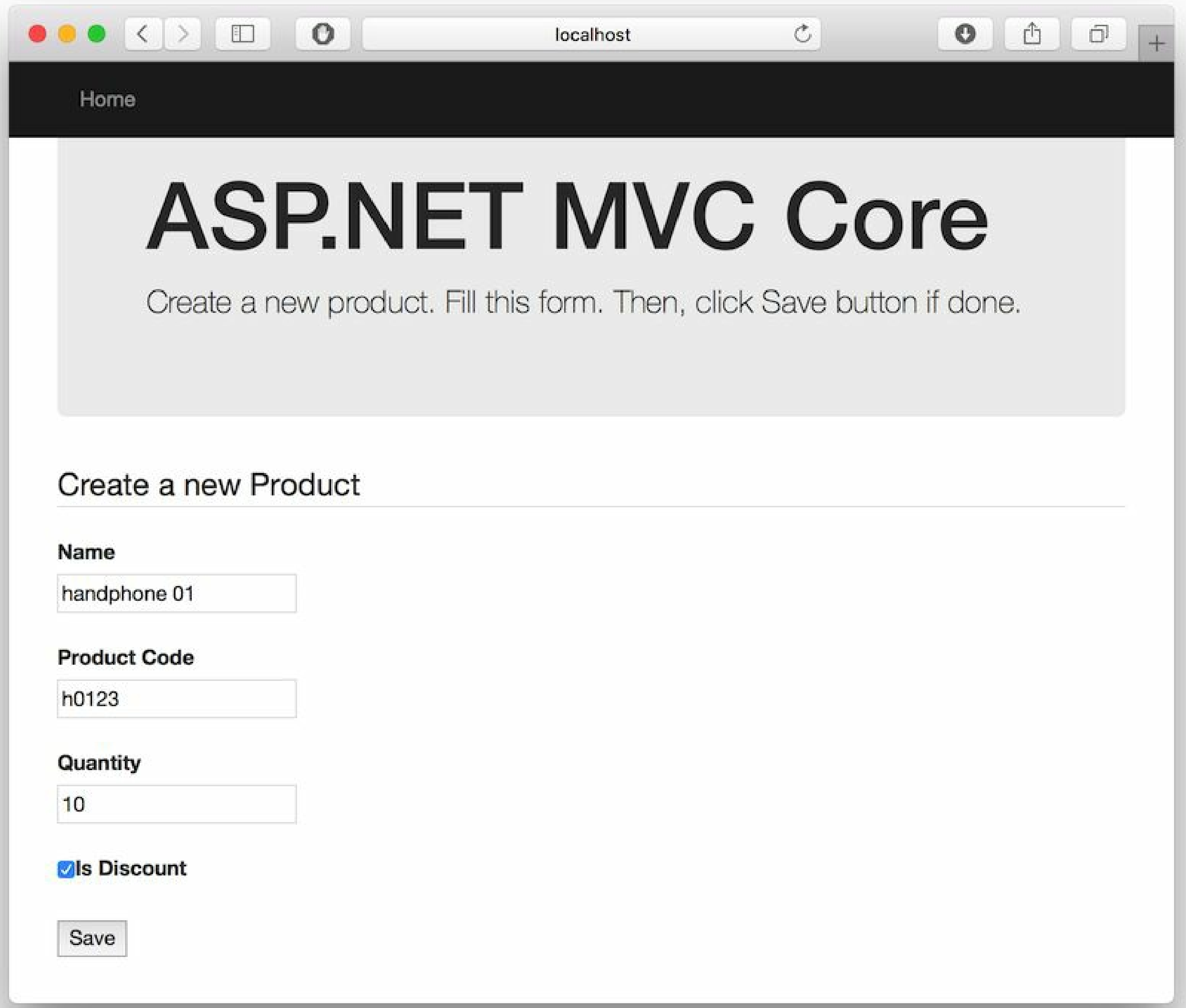


ASP.NET Core MVC Demo

Filling and submitting a form.

© 2017 - My ASP.NET MVC Core Application

Fill all fields. If done, click **Save** button.


localhost

Home

ASP.NET MVC Core

Create a new product. Fill this form. Then, click Save button if done.

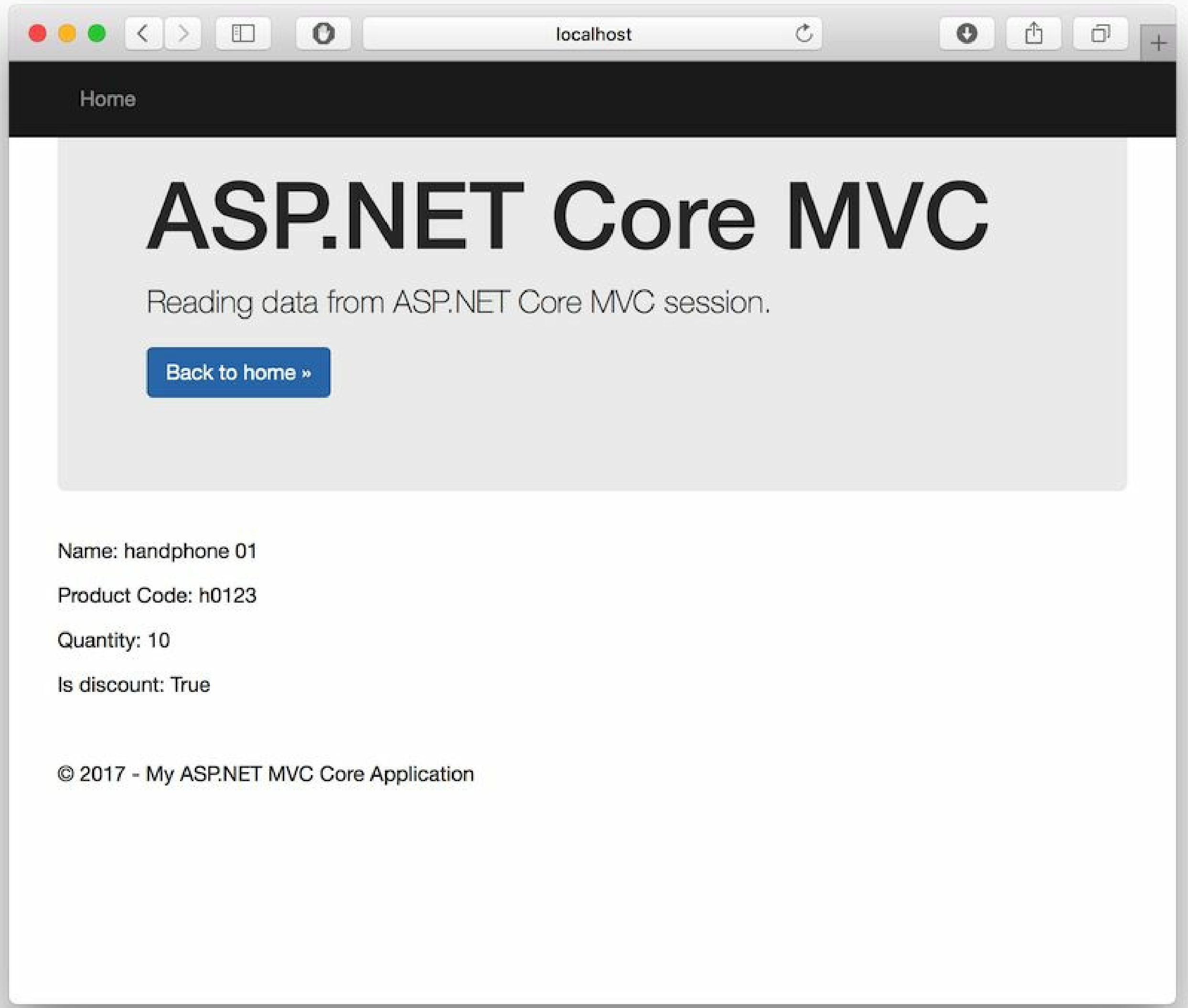
Create a new Product

| | |
|---|---|
| Name | handphone 01 |
| Product Code | h0123 |
| Quantity | 10 |

[✓] Is Discount

Save

Our filled form will be shown in the next page. It uses ASP.NET session to keep our filled data.



ASP.NET Core MVC

Reading data from ASP.NET Core MVC session.

Name: handphone 01

Product Code: h0123

Quantity: 10

Is discount: True

© 2017 - My ASP.NET MVC Core Application

## 3.5 What's Next?

I recommend you to read some tutorials about ASP.NET MVC from books or website in order to improve your development skill.

# 4. ASP.NET Core API Development

In this chapter I'm going to explain how to work with ASP.NET Core API development.

## 4.1 Getting Started

In this chapter, we learn how to write a program for ASP.NET Core API from scratch. ASP.NET Core API provides JSON output so we also build ASP.NET Core to consume this API (RESTful).

Let's start.

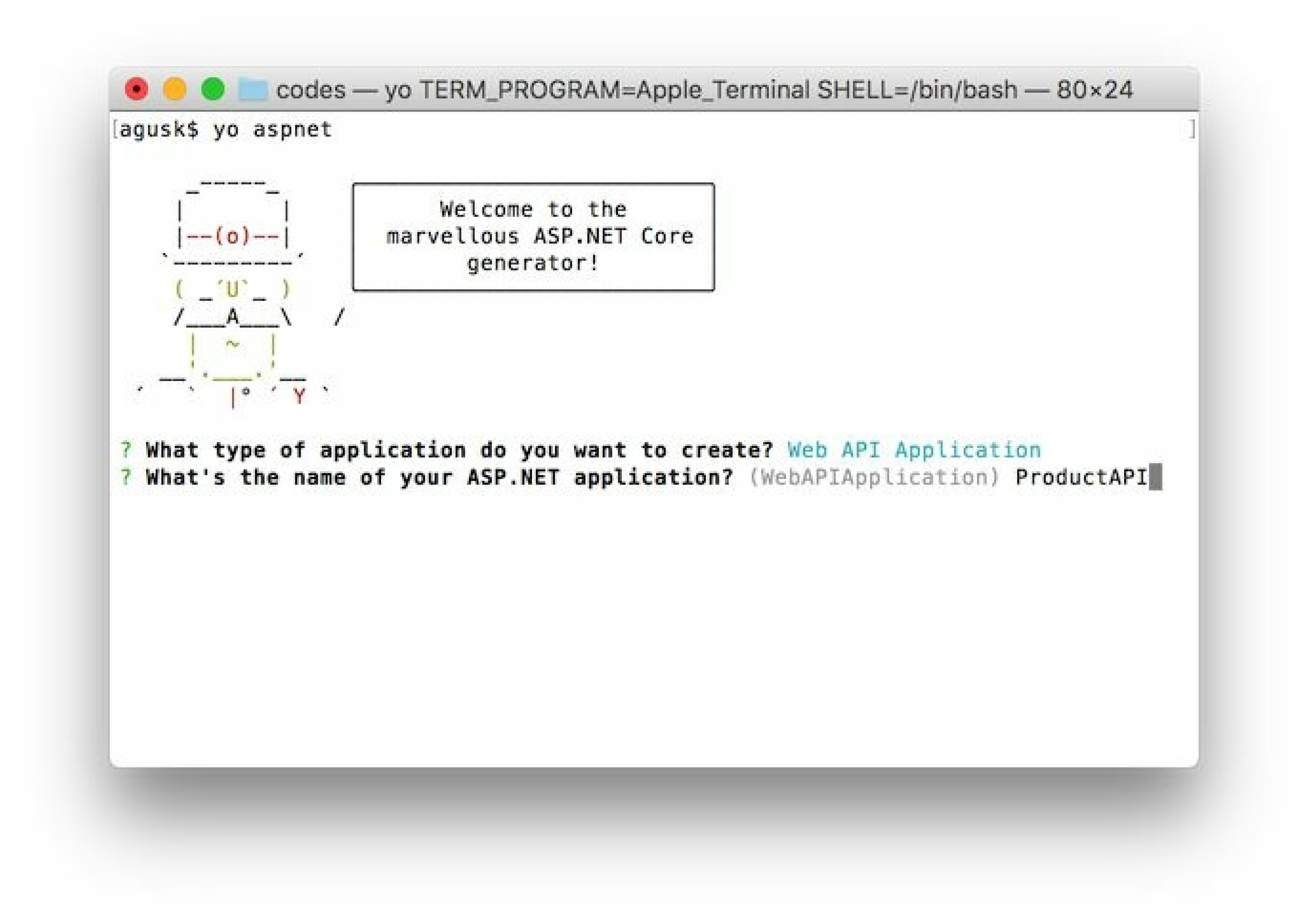
## 4.2 Building ASP.NET Core API Application

In this demo, we build ASP.NET Core API which provides RESTfull data, product data. The RESTfull app will be consumed by ASP.NET Core MVC. We build both applications in one project.

Let's start to develop.

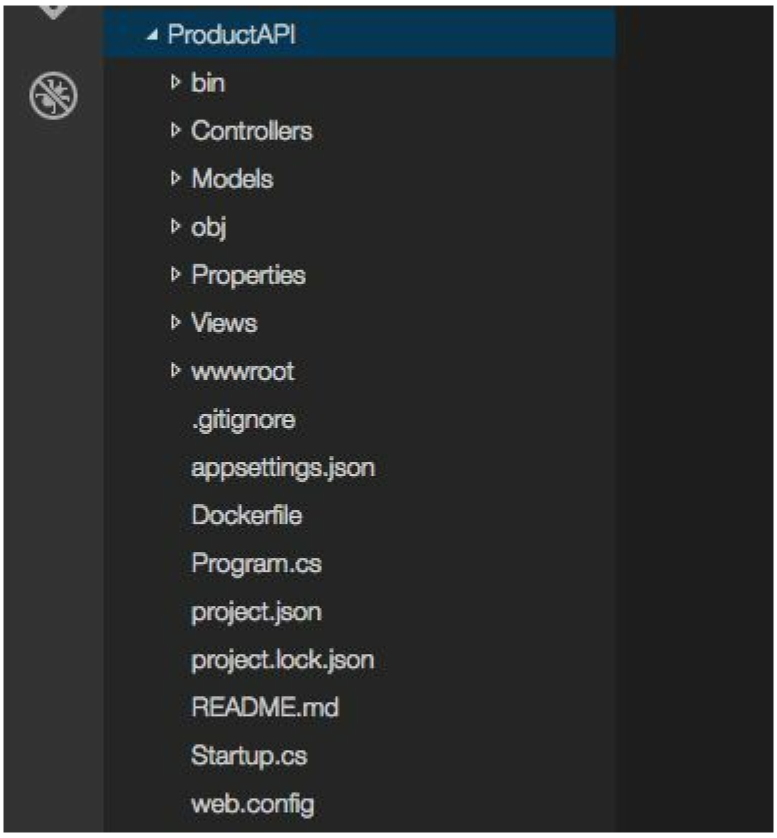
### 4.2.1 Creating a Project

Using yo aspnet, you can create a project. Select Web API Application and given the project name, ProductAPI.


codes — yo TERM_PROGRAM=Apple_Terminal SHELL=/bin/bash — 80×24

```
agusk$ yo aspnet

     _-----_
    |       |    ╭──────────────────────────╮
    |--(o)--|    │      Welcome to the      │
   `---------´   │ marvellous ASP.NET Core  │
    ( _´U`_ )    │        generator!        │
    /___A___\   /╰──────────────────────────╯
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What type of application do you want to create? Web API Application
? What's the name of your ASP.NET application? (WebAPIApplication) ProductAPI
```

You can see the project structure on the Figure below.



```
▴ ProductAPI
  ▹ bin
  ▹ Controllers
  ▹ Models
  ▹ obj
  ▹ Properties
  ▹ Views
  ▹ wwwroot
    .gitignore
    appsettings.json
    Dockerfile
    Program.cs
    project.json
    project.lock.json
    README.md
    Startup.cs
    web.config
```

We need project a config file, called **project.json**, on the project root folder. It consists of required libraries. Please modify this script for project.json file.

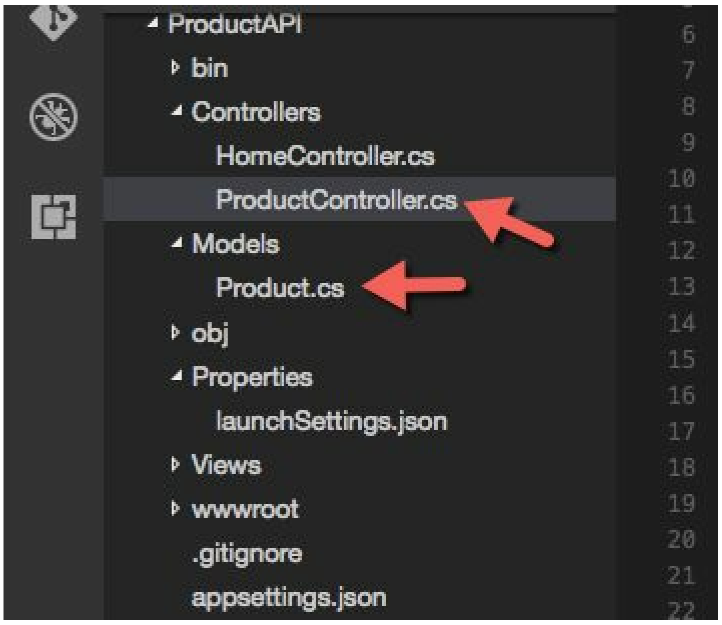
```
{
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.1.0",
      "type": "platform"
    },
    "Microsoft.AspNetCore.Mvc": "1.1.0",
    "Microsoft.AspNetCore.Routing": "1.1.0",
    "Microsoft.AspNetCore.Server.IISIntegration": "1.1.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.1.0",
    "Microsoft.Extensions.Configuration.EnvironmentVariables": "1.1.0",
    "Microsoft.Extensions.Configuration.FileExtensions": "1.1.0",
    "Microsoft.Extensions.Configuration.Json": "1.1.0",
    "Microsoft.Extensions.Configuration.CommandLine": "1.1.0",
    "Microsoft.Extensions.Logging": "1.1.0",
    "Microsoft.Extensions.Logging.Console": "1.1.0",
    "Microsoft.Extensions.Logging.Debug": "1.1.0",
    "Microsoft.Extensions.Options.ConfigurationExtensions": "1.1.0"
  },

  "tools": {
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.1.0-preview4-final"
  },

  "frameworks": {
    "netcoreapp1.1": {
      "imports": [
        "dotnet5.6",
        "portable-net45+win8"
      ]
    }
  },

  "buildOptions": {
    "emitEntryPoint": true,
    "preserveCompilationContext": true
  },

  "runtimeOptions": {
    "configProperties": {
      "System.GC.Server": true
    }
  },

  "publishOptions": {
    "include": [
      "wwwroot",
      "**/*.cshtml",
      "appsettings.json",
      "web.config"
    ]
  },

  "scripts": {
    "postpublish": [ "dotnet publish-iis --publish-folder %publish:OutputPath% --framework %pub
  },

  "tooling": {
    "defaultNamespace": "ProductAPI"
  }
}
```

App will start by executing **Startup.cs** file. Modify this file and write this code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace ProductAPI
{
    public class Startup
    {
        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
                .AddEnvironmentVariables();
            Configuration = builder.Build();
        }

        public IConfigurationRoot Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the conta
        public void ConfigureServices(IServiceCollection services)
        {
            // Add framework services.
            services.AddMvc();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP reques
        public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory
        {
            loggerFactory.AddConsole(Configuration.GetSection("Logging"));
            loggerFactory.AddDebug();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```

## 4.2.2 Writing Program for ASP.NET Core API

The next step is to write RESTfull API on ASP.NET Core MVC. You create a controller, ProductController.cs, a model, Product.cs. You can see the project structure as follows.



The following is codes for Product model.

```
using System.ComponentModel.DataAnnotations;

namespace ProductAPI.Models
{
    public class Product
    {
        [Required]
        [MinLength(4)]
        [Display( Name = "Name" )]
        public string Name { get; set; }
        [Display( Name = "Product Code" )]
        public string ProductCode { get; set; }
        [Display( Name = "Quantity" )]
        public int Quantity { get; set; }
        [Display( Name = "Is Discount" )]
        public bool IsDiscount { get; set; }
    }
}
```

In Product controller, we expose Get method which generate a random list for Product. The following is code implementation.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

using ProductAPI.Models;
namespace ProductAPI.Controllers
{
    public class ProductController : Controller
    {
        // GET a list of product
        [HttpGet]
        public List<ProductAPI.Models.Product> Get()
        {
            // generate product
            List<ProductAPI.Models.Product> list = new List<ProductAPI.Models.Product>();

            for(int i=0;i<10;i++){
                Product o = new Product();
                o.Name = String.Format("Product {0}",i);
                o.ProductCode = String.Format("PR{0}",i);
                o.Quantity = i;
                o.IsDiscount = true;

                list.Add(o);
            }

            return list;
        }

    }
}
```

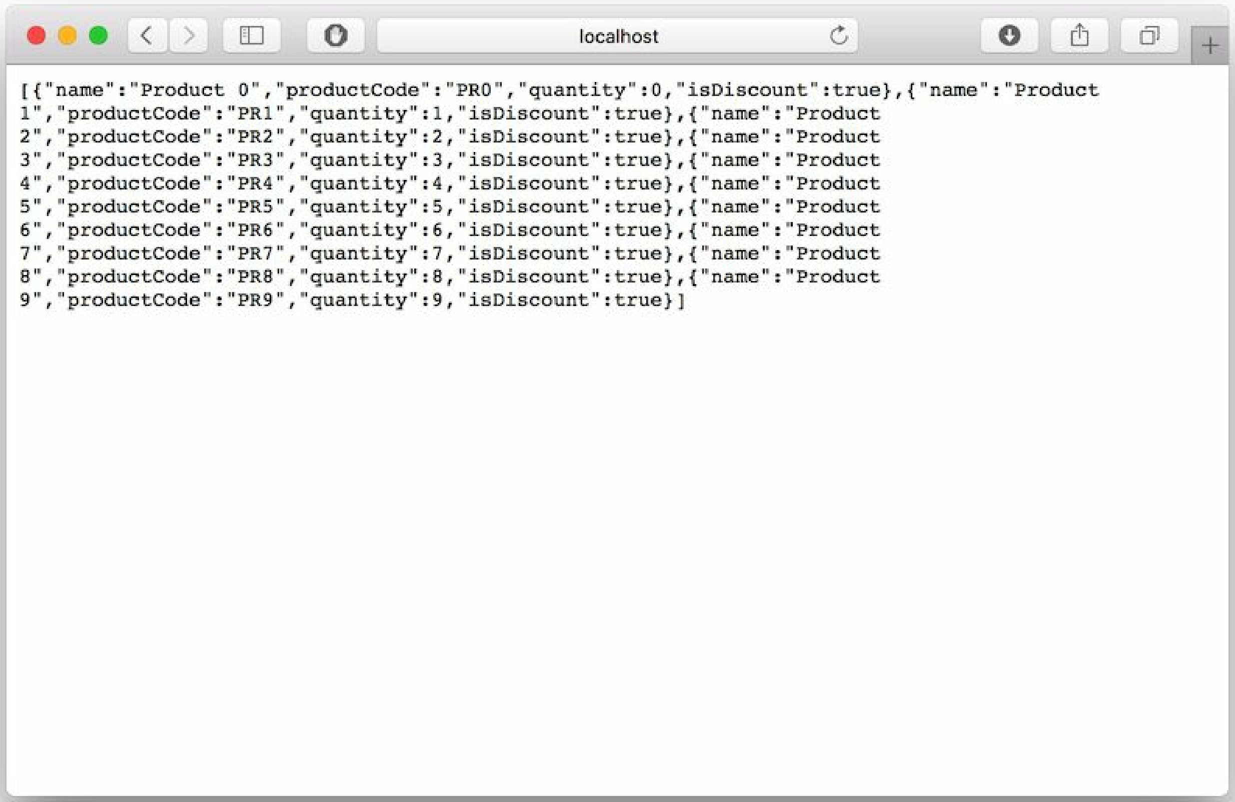
Save all codes.

### 4.2.3 Testing

Now we can test our ASP.NET Core API. On our project folder, navigate your Terminal and type these commands.

```
$ dotnet restore
$ dotnet build
$ dotnet run
```

If done, you can open a browser and navigate to http://localhost:5000/product/get . You should see a list of product. You can see my program output in Figure below.



```
[{"name":"Product 0","productCode":"PR0","quantity":0,"isDiscount":true},{"name":"Product
1","productCode":"PR1","quantity":1,"isDiscount":true},{"name":"Product
2","productCode":"PR2","quantity":2,"isDiscount":true},{"name":"Product
3","productCode":"PR3","quantity":3,"isDiscount":true},{"name":"Product
4","productCode":"PR4","quantity":4,"isDiscount":true},{"name":"Product
5","productCode":"PR5","quantity":5,"isDiscount":true},{"name":"Product
6","productCode":"PR6","quantity":6,"isDiscount":true},{"name":"Product
7","productCode":"PR7","quantity":7,"isDiscount":true},{"name":"Product
8","productCode":"PR8","quantity":8,"isDiscount":true},{"name":"Product
9","productCode":"PR9","quantity":9,"isDiscount":true}]
```

Now your ASP.NET Core API is running well.

In the next section, we build ASP.NET Core MVC to consume RESTfull API.

## 4.3 Client Application for ASP.NET Core API

In this section, we write our ASP.NET Core MVC program. We use HTML5 on View to access RESTfill. We use jQuery.

### 4.3.1 Creating Controller

Firstly, we create MVC controller, called Home controller. Create a file, called **HomeController.cs**, and write this code.

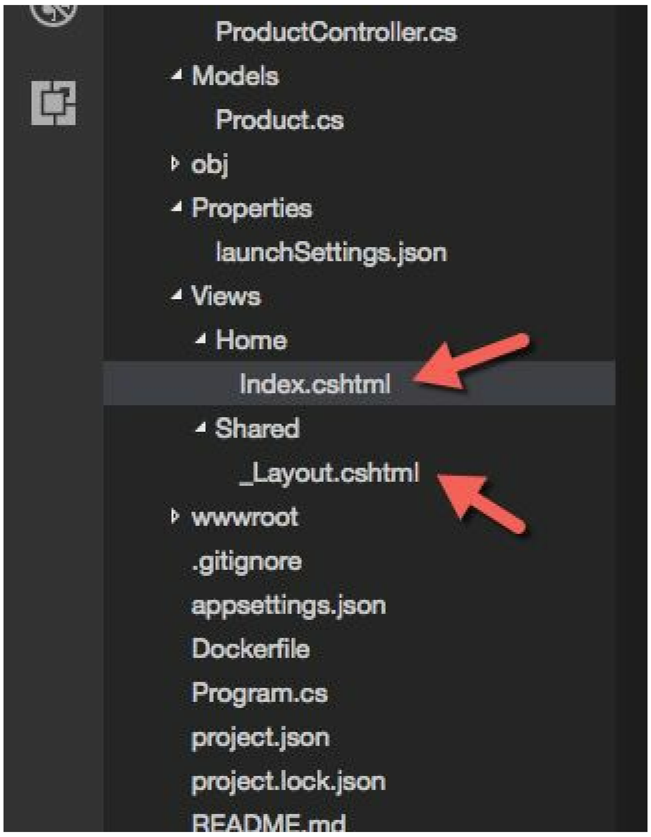
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace ProductAPI.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

    }
}
```

Next, you should create views.

## 4.3.2 Creating View

We create two pages for view, _Layout.cshtml for template and Index.cshtml for our UI.

You can see the view structure in Figure below.



We also create a layout, called **_Shared.cshtml**. Write this script.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET MVC Core Application</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossor

    <style>
        table { margin: 1em; border-collapse: collapse; }
        td, th { padding: .3em; border: 1px #ccc solid; }
        thead { background: #70ff45; }
        .even{
            background-color: #beffca;
        }
        .odd{
            background-color: #f0fff6;
        }
    </style>
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target='
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a href="/">Home</a></li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET MVC Core Application</p>
        </footer>
    </div>
</body>
</html>
```

The following is a script for **Index.cshtml** file.

```
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "Home Page";

}
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<div class="jumbotron">
    <h1>ASP.NET Core MVC Demo</h1>
    <p class="lead"> Access ASP.NET Core API.</p>
    <p><a href="#" id="get-data" class="btn btn-primary btn-large">Get JSON data &raquo;</a></p
    <table id="myTable">
        <thead>
            <tr>
                <th>Product Name</th>
                <th>Code</th>
                <th>Quantity</th>
                <th>Is Discount</th>
            </tr>
        </thead>
        <tbody >
        </tbody>
    </table>
</div>

<script>
$(document).ready(function () {
  $('#get-data').click(function () {
    var showData = $('#show-data');

    $.getJSON('/product/get', function (data) {
      console.log(data);

      var tableRef = document.getElementById('myTable').getElementsByTagName('tbody')[0];
      var newRow   = tableRef.insertRow(tableRef.rows.length);

      var items = '';
      for (var i = 0; i < data.length; i++) {
          var newRow   = tableRef.insertRow(tableRef.rows.length);

          var newCell1  = newRow.insertCell(0);
          var newText1  = document.createTextNode(data[i].name);
          newCell1.appendChild(newText1);
          var newCell2  = newRow.insertCell(1);
          var newText2  = document.createTextNode(data[i].productCode);
          newCell2.appendChild(newText2);
          var newCell3  = newRow.insertCell(2);
          var newText3  = document.createTextNode(data[i].quantity);
          newCell3.appendChild(newText3);
          var newCell4  = newRow.insertCell(3);
          var newText4  = document.createTextNode(data[i].isDiscount);
          newCell4.appendChild(newText4);

      }


    });

    showData.text('Loading the JSON file.');
  });
});
</script>
```

Save all files and store them into Views folder.

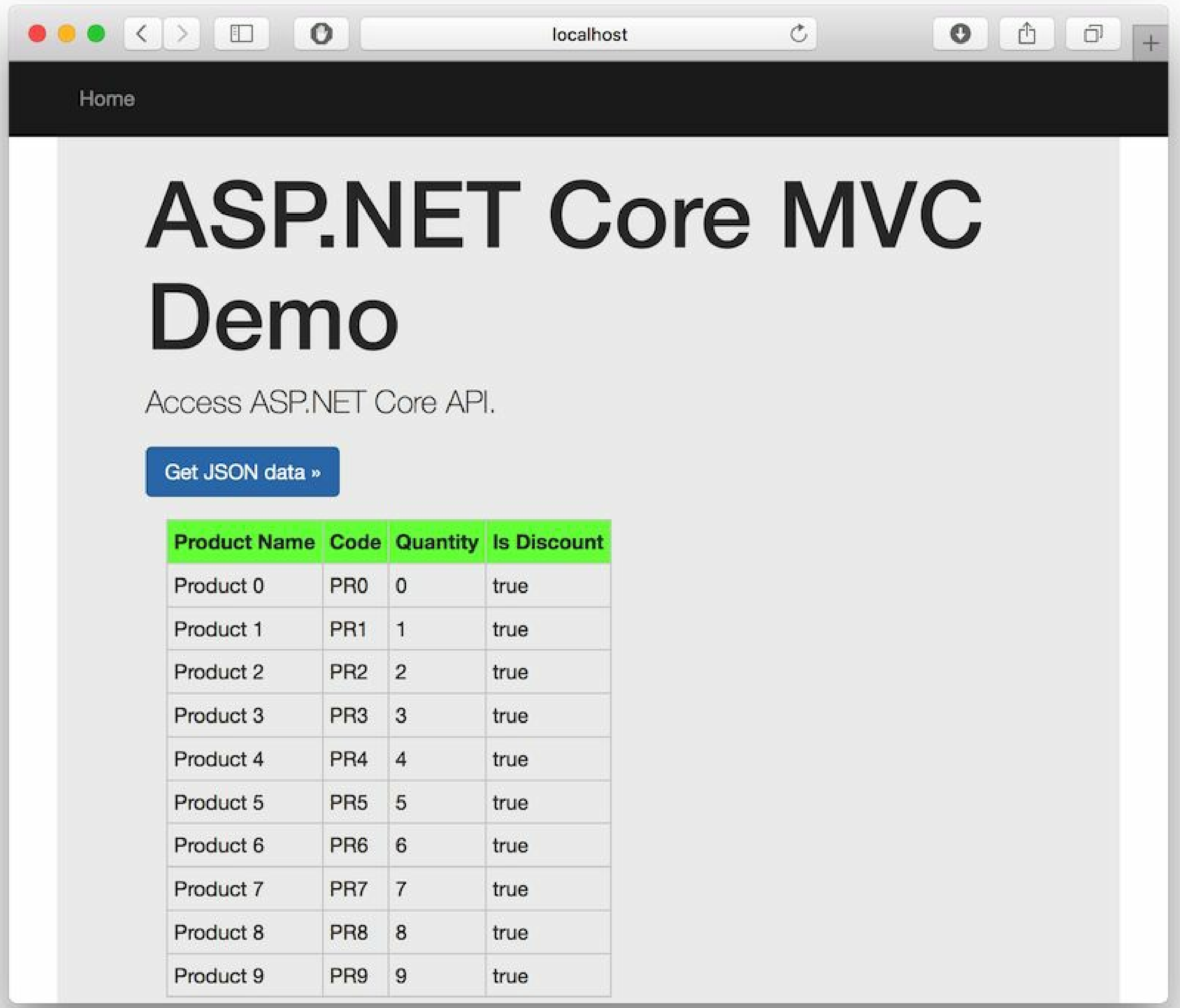
## 4.4 Testing

Now you are ready to execute. Type these command to load required libraries and execute the program.

```
$ dotnet restore
$ dotnet build
$ dotnet run
```

Open a browser and navigate to http://localhost:5004 . Click **Get JSON data** button. You computer should connected to Internet because our jQuery file is download. You can download this file and put it in local.

The following is program output.



ASP.NET Core MVC Demo

Access ASP.NET Core API.

Get JSON data »

| Product Name | Code | Quantity | Is Discount |
|---|---|---|---|
| Product 0 | PR0 | 0 | true |
| Product 1 | PR1 | 1 | true |
| Product 2 | PR2 | 2 | true |
| Product 3 | PR3 | 3 | true |
| Product 4 | PR4 | 4 | true |
| Product 5 | PR5 | 5 | true |
| Product 6 | PR6 | 6 | true |
| Product 7 | PR7 | 7 | true |
| Product 8 | PR8 | 8 | true |
| Product 9 | PR9 | 9 | true |

# 5. ASP.NET Core and Angular 2

In this chapter I'm going to explain how to work with ASP.NET Core MVC and Angular 2.

## 5.1 Getting Started

In this section, we develop a simple app to work with ASP.NET RESTful and AngularJS.

AngularJS is an open-source JavaScript framework, maintained by Google, that assists with running single-page applications. Further information about AngularJS, you can visit official website on http://angularjs.org/ .

In this chapter, we explore to integrate Angular 2 into ASP.NET Core.

## 5.2 ASP.NET Core and Angular 2 Project Template

You can learn Angular 2 on the official website, https://angular.io. We develop Angular 2 using JavaScript and TypeScript.

The idea to integrate ASP.NET Core MVC and Angular 2 is to merge the both projects, ASP.NET Core and Angular 2. Fortunately, we can use aspnetcore spa template, https://github.com/aspnet/JavaScriptServices.

You can install this template by typing this command on Terminal.

```
$ npm install -g yo generator-aspnetcore-spa
```

If done, you can build a sample project from this project template.

## 5.3 Run Demo: ASP.NET Core Angular SPA

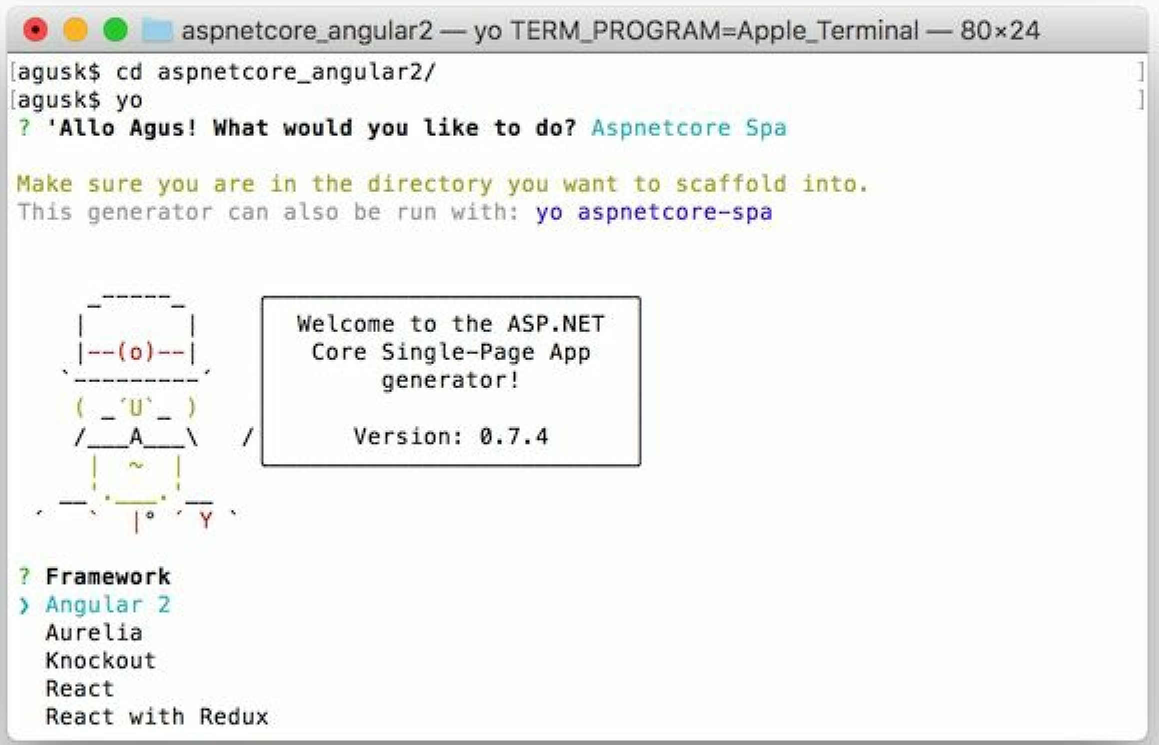
In this section, we run program sample from aspnetcore spa template.

### 5.3.1 Creating a Project

You can create ASP.NET Core with Angular 2 by typing this command.

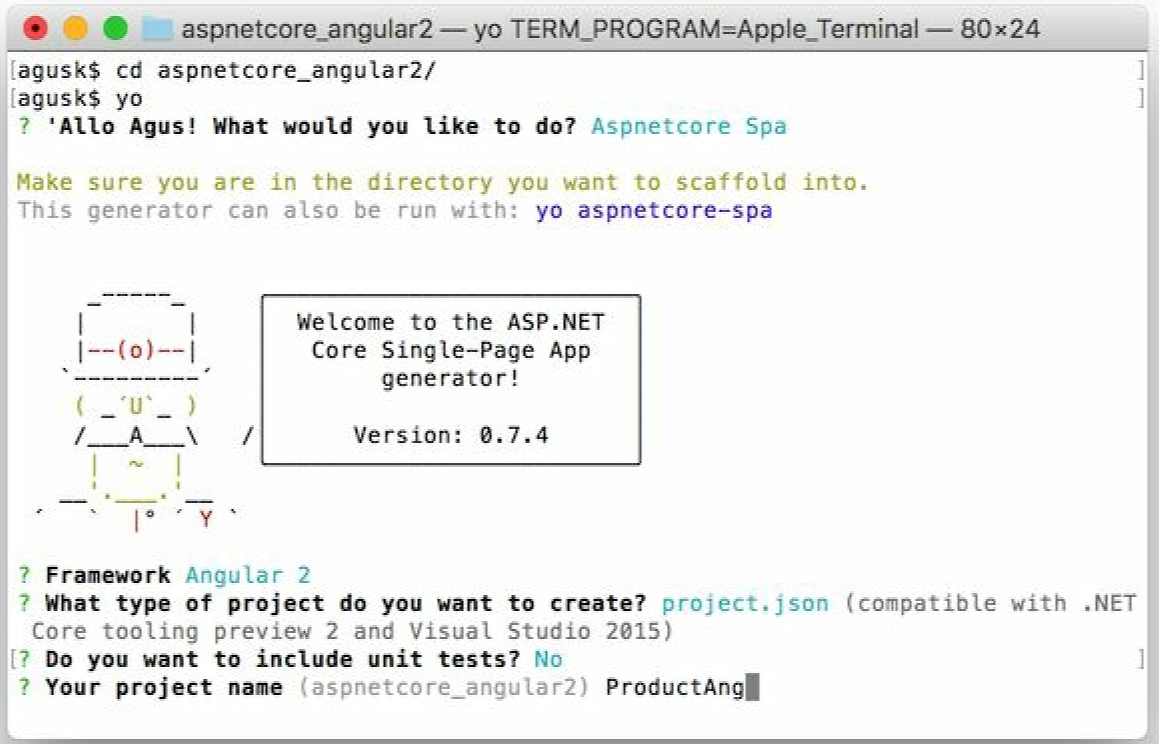
```
$ yo aspnetcore-spa
```

Select Angular 2 as framework.



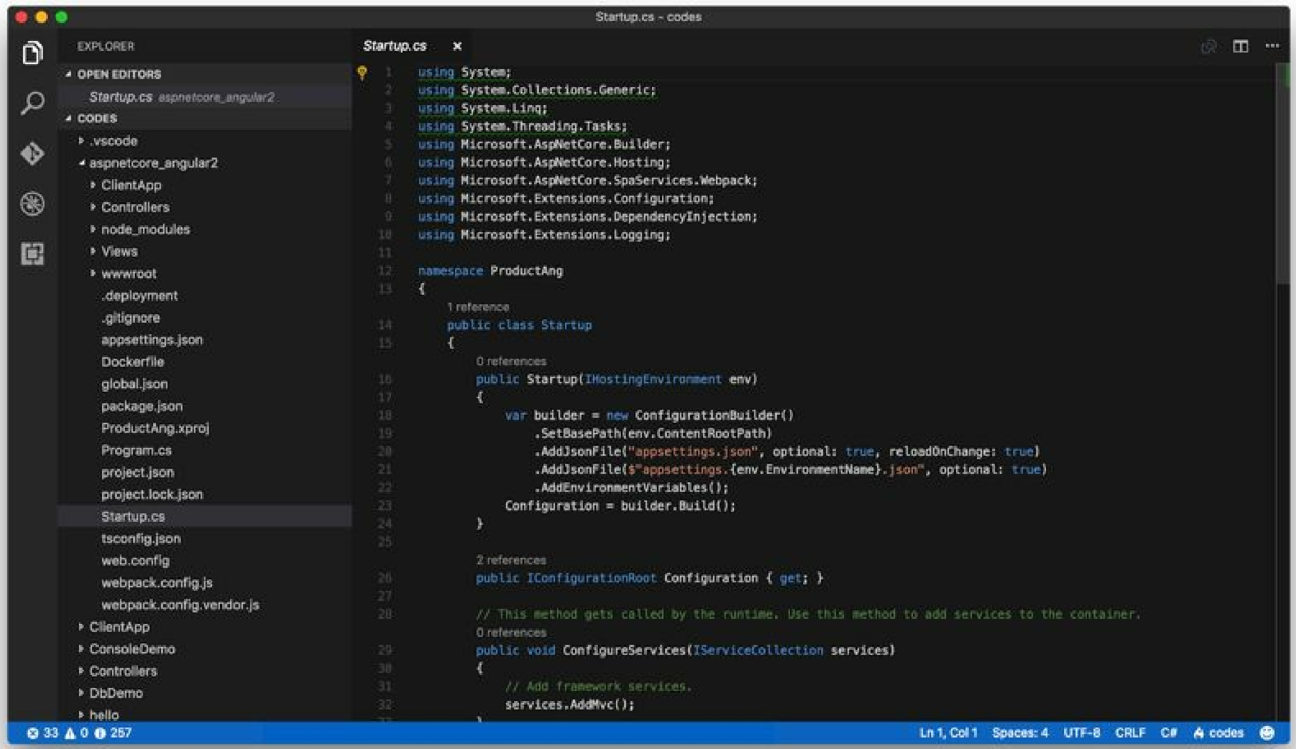
```
[agusk$ cd aspnetcore_angular2/
[agusk$ yo
? 'Allo Agus! What would you like to do? Aspnetcore Spa

Make sure you are in the directory you want to scaffold into.
This generator can also be run with: yo aspnetcore-spa


     _-----_     ╭──────────────────────────╮
    |       |    │  Welcome to the ASP.NET  │
    |--(o)--|    │   Core Single-Page App   │
   `---------´   │        generator!        │
    ( _´U`_ )    │                          │
    /___A___\   /│      Version: 0.7.4      │
     |  ~  |     ╰──────────────────────────╯
   __'.___.'__
 ´   `  |° ´ Y `

? Framework
❯ Angular 2
  Aurelia
  Knockout
  React
  React with Redux
```

Given a project name, called ProductAng.



```
[agusk$ cd aspnetcore_angular2/
[agusk$ yo
? 'Allo Agus! What would you like to do? Aspnetcore Spa

Make sure you are in the directory you want to scaffold into.
This generator can also be run with: yo aspnetcore-spa


     _-----_     ╭──────────────────────────╮
    |       |    │  Welcome to the ASP.NET  │
    |--(o)--|    │   Core Single-Page App   │
   `---------´   │        generator!        │
    ( _´U`_ )    │                          │
    /___A___\   /│      Version: 0.7.4      │
     |  ~  |     ╰──────────────────────────╯
   __'.___.'__
 ´   `  |° ´ Y `

? Framework Angular 2
? What type of project do you want to create? project.json (compatible with .NET
 Core tooling preview 2 and Visual Studio 2015)
[? Do you want to include unit tests? No
? Your project name (aspnetcore_angular2) ProductAng
```

If done, you can see the complete project. You can open it using Visual Studio Code.



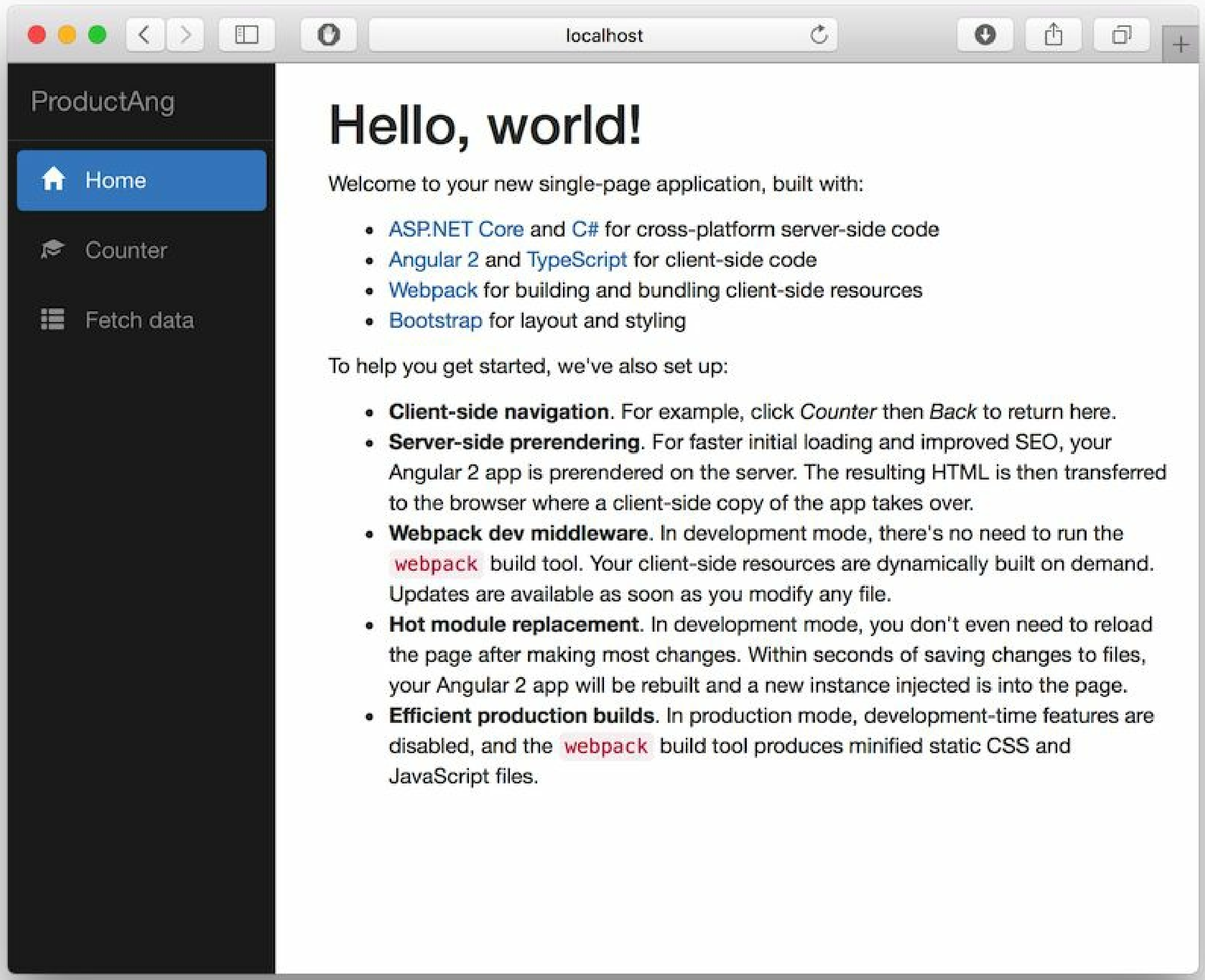
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.SpaServices.Webpack;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace ProductAng
{
    1 reference
    public class Startup
    {
        0 references
        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
                .AddEnvironmentVariables();
            Configuration = builder.Build();
        }

        2 references
        public IConfigurationRoot Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            // Add framework services.
            services.AddMvc();
```

## 5.3.2 Running the Program

To run the program, you do it as usual. Open Terminal and navigate to project folder. Then, type these commands.

```
$ dotnet restore
$ dotnet build
$ dotnet run
```

You open a browser and navigate to http://localhost:5000/. You should SPA program with Angular 2.



Hello, world!

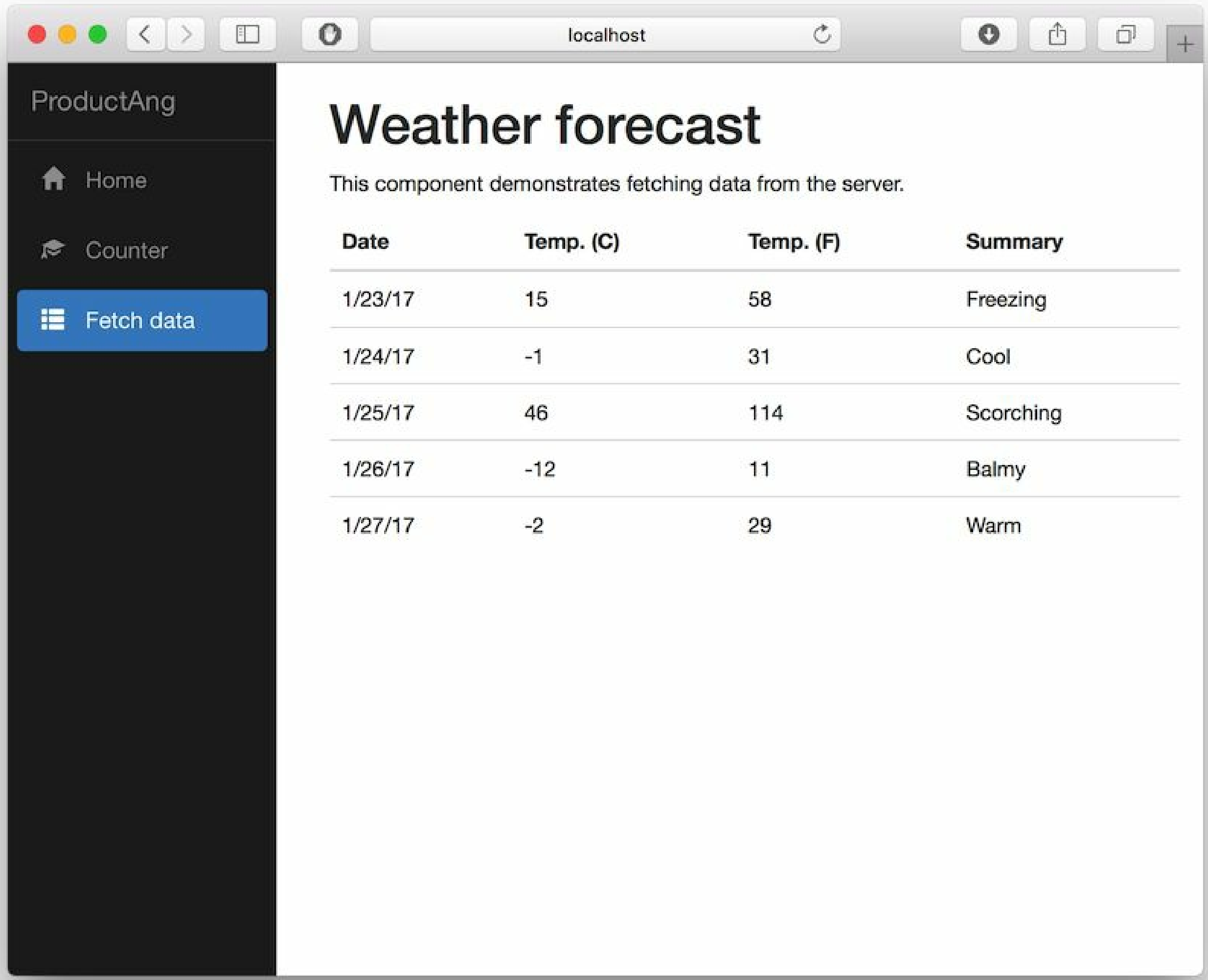
Welcome to your new single-page application, built with:

- ASP.NET Core and C# for cross-platform server-side code
- Angular 2 and TypeScript for client-side code
- Webpack for building and bundling client-side resources
- Bootstrap for layout and styling

To help you get started, we've also set up:

- **Client-side navigation**. For example, click *Counter* then *Back* to return here.
- **Server-side prerendering**. For faster initial loading and improved SEO, your Angular 2 app is prerendered on the server. The resulting HTML is then transferred to the browser where a client-side copy of the app takes over.
- **Webpack dev middleware**. In development mode, there's no need to run the `webpack` build tool. Your client-side resources are dynamically built on demand. Updates are available as soon as you modify any file.
- **Hot module replacement**. In development mode, you don't even need to reload the page after making most changes. Within seconds of saving changes to files, your Angular 2 app will be rebuilt and a new instance injected is into the page.
- **Efficient production builds**. In production mode, development-time features are disabled, and the `webpack` build tool produces minified static CSS and JavaScript files.

You can click menu Fetch data to see Angular 2 feed remote data.



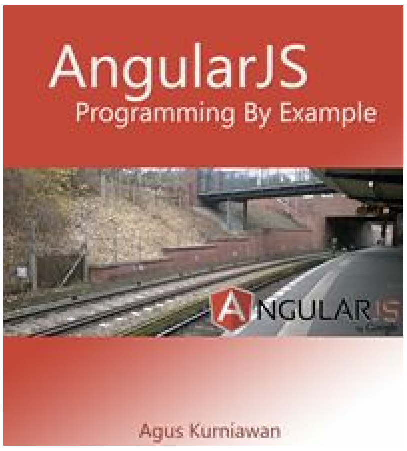
# Weather forecast

This component demonstrates fetching data from the server.

| Date | Temp. (C) | Temp. (F) | Summary |
|---|---|---|---|
| 1/23/17 | 15 | 58 | Freezing |
| 1/24/17 | -1 | 31 | Cool |
| 1/25/17 | 46 | 114 | Scorching |
| 1/26/17 | -12 | 11 | Balmy |
| 1/27/17 | -2 | 29 | Warm |

## 5.4 What's The Next?

You can do more practices for AngularJS by reading several tutorial in Internet or books. I also wrote a book about AngularJS Programming by Example. This book helps you to learn about AngularJS programming. Please visit my blog, http://blog.aguskurniawan.net/post/AngularJS.aspx . This book uses the first version of AngularJS. Currently, I'm working on my new book for Angular 2.



# 6. ASP.NET Core, Entity Framework Core and PostgreSQL

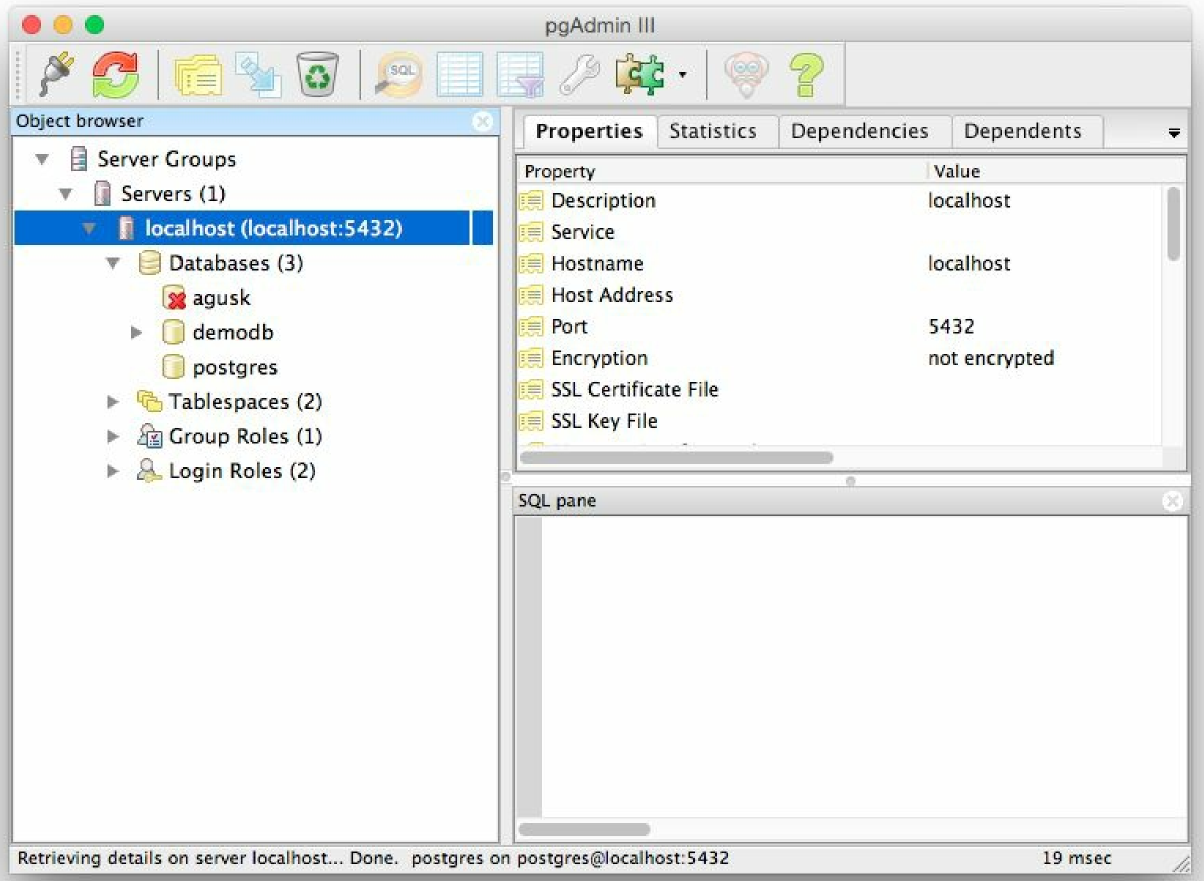
In this chapter I'm going to explain how to work with ASP.NET Core MVC, Entity Framework Core and PostgreSQL.

## 6.1 Getting Started

In this chapter, we learn how to work with ASP.NET Core MVC to access database, PostgreSQL, using Entity Framework Core.

## 6.2 PostgreSQL Database

In this chapter, we use PostgreSQL for database server. You can install it based on your platform on https://www.postgresql.org.



| Property | Value |
|---|---|
| Description | localhost |
| Service | |
| Hostname | localhost |
| Host Address | |
| Port | 5432 |
| Encryption | not encrypted |
| SSL Certificate File | |
| SSL Key File | |

```
Retrieving details on server localhost... Done.   postgres on postgres@localhost:5432   19 msec
```

## 6.3 Entity Framework Core

If you have experiences in Entity Framework (EF) library that is used to manage data processing for several database engines, now Microsoft provides EF for .NET Core. It's called EF Core. Further information about EF Core, you can visit the official website on https://github.com/aspnet/EntityFramework.

## 6.4 Demo Database Application

In this section, we will build a simple ASP.NET Core MVC to access PostgreSQL through Entity Framework Core.

Let’s start!.

### 6.4.1 Build PostgreSQL Database

Firstly, you create a database, for instance, demodb. Then, create a table for our testing. The following is a script for creating a table, product.

```
CREATE TABLE public.product
(
  id integer NOT NULL DEFAULT nextval('product_id_seq'::regclass),
  quantity integer,
  is_discount boolean,
  name text,
  product_code text
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.product
  OWNER TO postgres;
```

You may create a login user for accessing this database.

### 6.4.2 Creating a ASP.NET Core MVC Project

We can create ASP.NET Core MVC project using yo aspnet. You can use Web Application template.

### 6.4.3 Configure ASP.NET Core MVC Project

The next step is to load .NET Framework Core for PostgreSQL into our project. You add these libraries into project.json file under dependencies section.

```
    "Npgsql.EntityFrameworkCore.PostgreSQL": "1.1.0",
    "Microsoft.EntityFrameworkCore.Design": {
      "version": "1.1.0",
      "type": "build"
    },
```

Then, you can load required libraries by typing this command on Terminal.

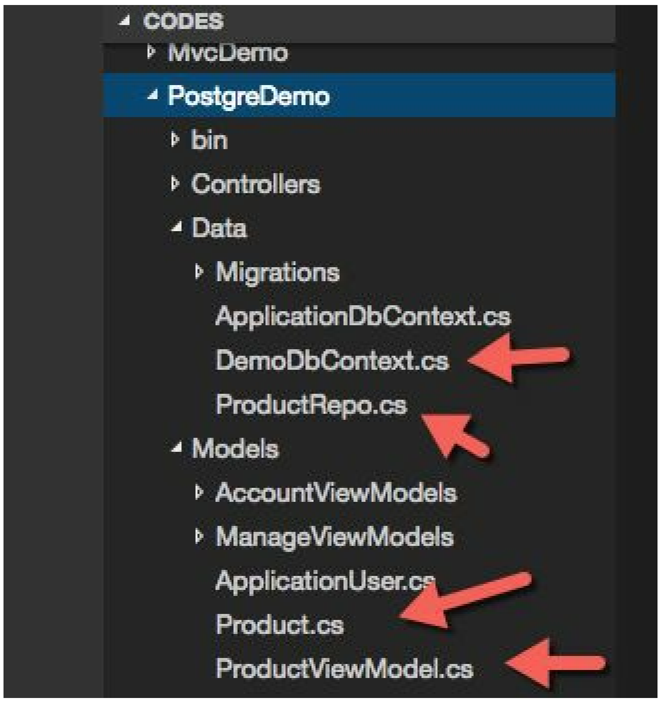
```
$ dotnet restore
```

Then, we add our database connection string for PostgreSQL. We add it on appsettings.json file. We add DbContextSettings. You can change values for your PostgreSQL.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=PostgreDemo.db"
  },
  "DbContextSettings" :{
    "ConnectionString" : "User ID=agusk;Password=agusk;Host=localhost;Port=5432;Database=demodb
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  }
}
```

## 6.4.4 Develop Data Access

After we loaded EF Core inside our project, now we develop classes to access PostgreSQL. We create two Model and Data objects. You can see them in Figure below.



Product is a data model which holds data from database table in PostgreSQL. It's mapping from database table. The following is implementation of Product.

```
using System.ComponentModel.DataAnnotations.Schema;

namespace PostgreDemo.Models
{
    [Table("product")]
    public class Product
    {
        [Column("id")]
        public int Id { get; set; }
        [Column("name")]
        public string Name { get; set; }
        [Column("product_code")]
        public string ProductCode { get; set; }
        [Column("quantity")]
        public int Quantity { get; set; }
        [Column("is_discount")]
        public bool IsDiscount { get; set; }
    }
}
```

ProductViewModel is model object which is used to display data on View of ASP.NET Core MVC.

```
using System.ComponentModel.DataAnnotations;

namespace PostgreDemo.Models
{
    public class ProductViewModel
    {
        [Display( Name = "Id" )]
        public int Id { get; set; }
        [Required]
        [MinLength(4)]
        [Display( Name = "Name" )]
        public string Name { get; set; }
        [Display( Name = "Product Code" )]
        public string ProductCode { get; set; }
        [Display( Name = "Quantity" )]
        public int Quantity { get; set; }
        [Display( Name = "Is Discount" )]
        public bool IsDiscount { get; set; }
    }
}
```

Now we also create a database context for EF Core. We define DemoDbContext for a database context from PostgreSQL.

```
using Microsoft.EntityFrameworkCore;
using PostgreDemo.Models;


namespace PostgreDemo.Data
{
    public class DemoDbContext : DbContext
    {
        public DemoDbContext(DbContextOptions<DemoDbContext> options) : base(options) { }

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

To manipulate Product table, we define ProductRepo class. We create two methods, Save() and GetProductByCode(), in our object.

```
using System.Linq;
using PostgreDemo.Models;

namespace PostgreDemo.Data
{
    public class ProductRepo
    {
        private readonly DemoDbContext _databaseContext;

        public ProductRepo(DemoDbContext databaseContext)
        {
            _databaseContext = databaseContext;
        }

        public Product GetProductByCode(string code)
        {
            return _databaseContext.Products.SingleOrDefault(x => x.ProductCode == code);
        }

        public void Save(Product product)
        {
            var productFromDb = _databaseContext.Products.SingleOrDefault(x => x.Id == product.
            if(productFromDb != null)
            {
                productFromDb = product;
            }
            else
            {
                _databaseContext.Products.Add(product);
            }
            _databaseContext.SaveChanges();
        }
    }
}
```

These objects will be called from Controller.

### 6.4.5 Develop ASP.NET Core MVC Controllers

We create Product Controller to handle our View and push data to UI. The following is ProductController class implementation.

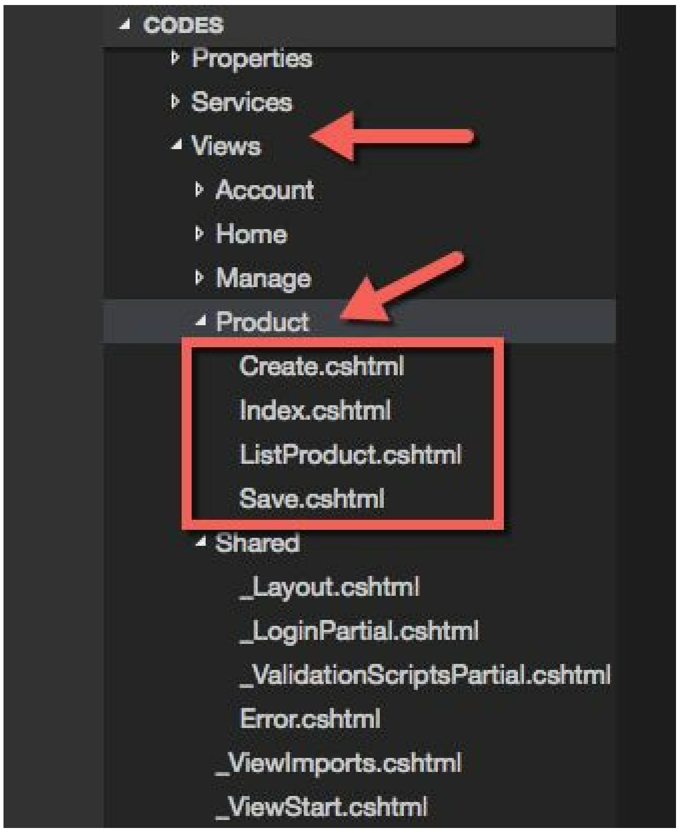
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

using PostgreDemo.Data;
using PostgreDemo.Models;

namespace PostgreDemo.Controllers
{
    public class ProductController : Controller
    {
        private readonly ProductRepo _productRepository;
        public ProductController(DemoDbContext databaseContext)
        {
            _productRepository = new ProductRepo(databaseContext);
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Create()
        {
            return View();
        }
        [HttpPost]
        public IActionResult Create(ProductViewModel obj)
        {
            // do database processing
            ///////
            Product o = new Product();
            o.Id = obj.Id;
            o.Name = obj.Name;
            o.ProductCode = obj.ProductCode;
            o.Quantity = obj.Quantity;
            o.IsDiscount = obj.IsDiscount;

            _productRepository.Save(o);

            return RedirectToAction("Save");
        }
        public IActionResult Save()
        {
            return View();
        }

        public IActionResult ListProduct()
        {
            return View(new ProductViewModel());
        }
        [HttpPost]
        public IActionResult ListProduct(string code)
        {
            Product obj = _productRepository.GetProductByCode(code);
            ProductViewModel o = new ProductViewModel();
            if(o!=null)
            {
                o.Id = obj.Id;
                o.Name = obj.Name;
                o.ProductCode = obj.ProductCode;
                o.Quantity = obj.Quantity;
                o.IsDiscount = obj.IsDiscount;
            }

            return View(o);
        }


    }
}
```

To activate EF Core on our project, we add a database context in ConfigureServices() in Startup.cs file. Add these codes.

```
        public void ConfigureServices(IServiceCollection services)
        {
            // Add framework services.
            services.AddDbContext<ApplicationDbContext>(options =>
                options.UseSqlite(Configuration.GetConnectionString("DefaultConnection")));

            services.AddIdentity<ApplicationUser, IdentityRole>()
                .AddEntityFrameworkStores<ApplicationDbContext>()
                .AddDefaultTokenProviders();

            services.AddMvc();

            // postgresql
            var connectionString =  Configuration["DbContextSettings:ConnectionString"];
            services.AddDbContext<DemoDbContext>(opts => opts.UseNpgsql(connectionString));

            // Add application services.
            services.AddTransient<IEmailSender, AuthMessageSender>();
            services.AddTransient<ISmsSender, AuthMessageSender>();
        }
```

## 6.4.6 Develop ASP.NET Core MVC Views

After developed Controller, we develop our Views. We Create Product in Views. Then, create four view files which are shown in Figure below.



The following is implementation of Index.cshtml file.

```
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "Home Page";

}

<div class="jumbotron">
    <h1>ASP.NET Core MVC Demo</h1>
    <p class="lead"> Filling and submitting a form.</p>
    <p><a href="/Home/Create" class="btn btn-primary btn-large">Create &raquo;</a></p>
</div>
```

The following is implementation of Create.cshtml file.

```
@using PostgreDemo.Models
@model ProductViewModel
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "Create Product";
}

<div class="jumbotron">
    <h1>ASP.NET Core MVC and PostgreSQL</h1>
    <p class="lead">Create a new product. Fill this form. Then, click Save button if done.</p>

</div>
@using ( Html.BeginForm() )
    {
        <fieldset>
                <legend>Create a new Product</legend>
                <div class="editor-label">
                        @Html.LabelFor(model => model.Name)
                </div>
                <div class="editor-field">
                        @Html.TextBoxFor(model => model.Name)
                </div>
                <br/>

                <div class="editor-label">
                        @Html.LabelFor(model => model.ProductCode)
                </div>
                <div class="editor-field">
                        @Html.TextBoxFor( model => model.ProductCode )
                </div>
                <br/>
                <div class="editor-label">
                        @Html.LabelFor(model => model.Quantity)
                </div>
                <div class="editor-field">
                        @Html.TextBoxFor( model => model.Quantity )
                </div>
                <br/>

                <div class="editor-label">
                        <div style="float: left;">
                                @Html.CheckBoxFor(model => model.IsDiscount)
                        </div>
                        @Html.LabelFor(model => model.IsDiscount)
                </div>
                <br/>

                <input type="submit" value="Save" />

        </fieldset>
    }
<p><br></p>
```

The following is implementation of Save.cshtml file.

```
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "Saved Data";
}

<div class="jumbotron">
    <h1>ASP.NET Core MVC and PostgreSQL</h1>
    <p class="lead">Product has saved into PostgreSQL.</p>
    <p><a href="/Product" class="btn btn-primary btn-large">Back to home &raquo;</a></p>
</div>
<p><br></p>
```

The following is implementation of ListProduct.cshtml file.

```
@using PostgreDemo.Models
@model ProductViewModel
@{
    Layout = "/Views/Shared/_Layout.cshtml";
    ViewBag.Title = "List Product";
}
```

```
<div class="jumbotron">
    <h1>ASP.NET Core MVC and PostgreSQL</h1>
    <p class="lead">Fill product code. Then, click Search button if done.</p>
</div>
@using ( Html.BeginForm() )
    {
        <div class="editor-label">
                <label>Product Code</label>
        </div>
        <div class="editor-field">
                <input type="text" id="code" name="code">
        </div>
        <br/>
        <input type="submit" value="Search" />
    }
<p><br></p>
<fieldset>
        <legend>Product Information</legend>

        <div class="editor-label">
                @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
                @Html.TextBoxFor(model => model.Name)
        </div>
        <br/>

        <div class="editor-label">
                @Html.LabelFor(model => model.ProductCode)
        </div>
        <div class="editor-field">
                @Html.TextBoxFor( model => model.ProductCode )
        </div>
        <br/>
        <div class="editor-label">
                @Html.LabelFor(model => model.Quantity)
        </div>
        <div class="editor-field">
                @Html.TextBoxFor( model => model.Quantity )
        </div>
        <br/>

        <div class="editor-label">
                <div style="float: left;">
                        @Html.CheckBoxFor(model => model.IsDiscount)
                </div>
                @Html.LabelFor(model => model.IsDiscount)
        </div>
</fieldset>
<p><br></p>
```

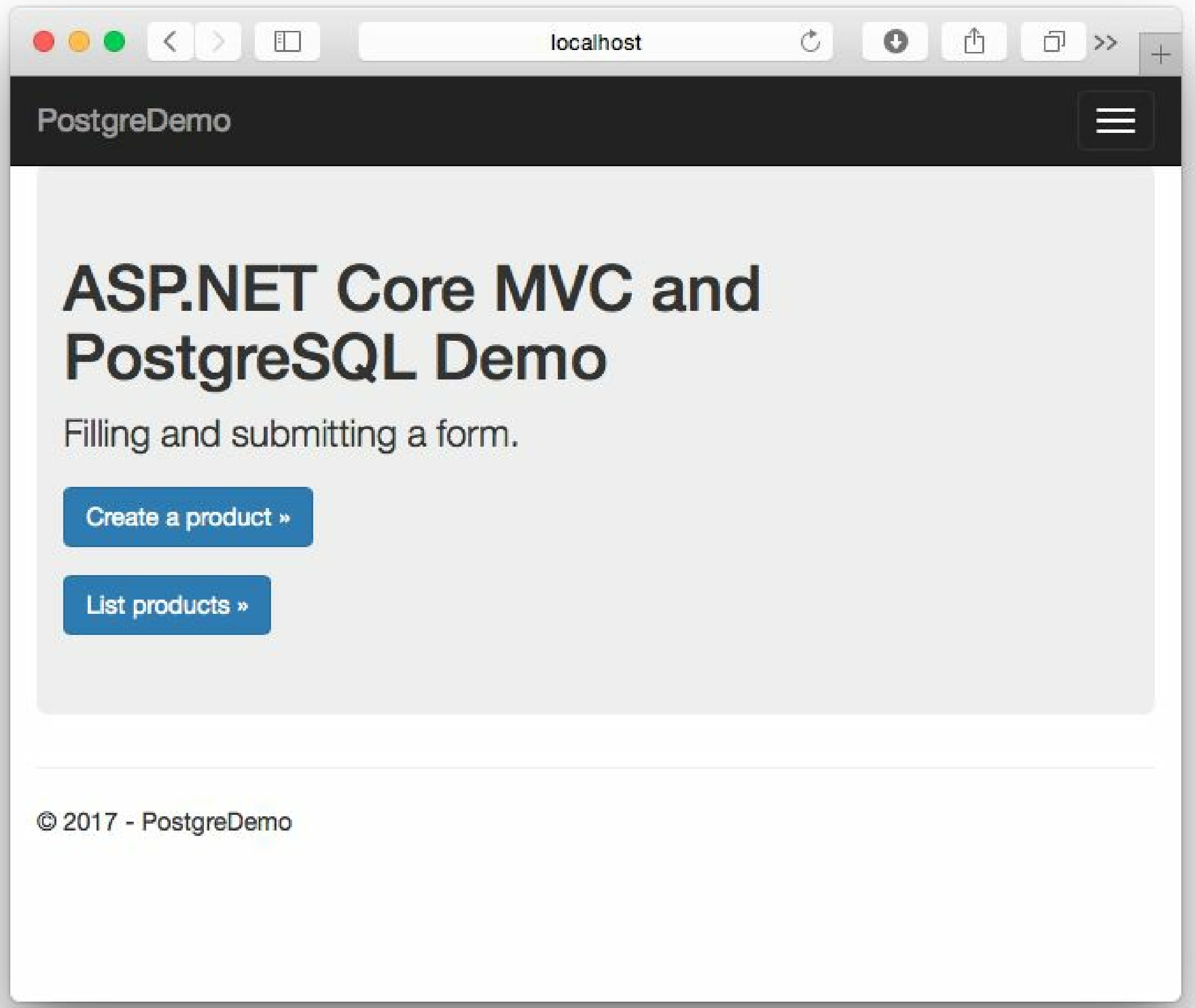
Seve All files.

### 6.4.7 Testing

Now you can the program.

```
$ dotnet restore
$ dotnet build
$ dotnet run
```

Open a browser and navigate to http://lcoalhost:5000/product.



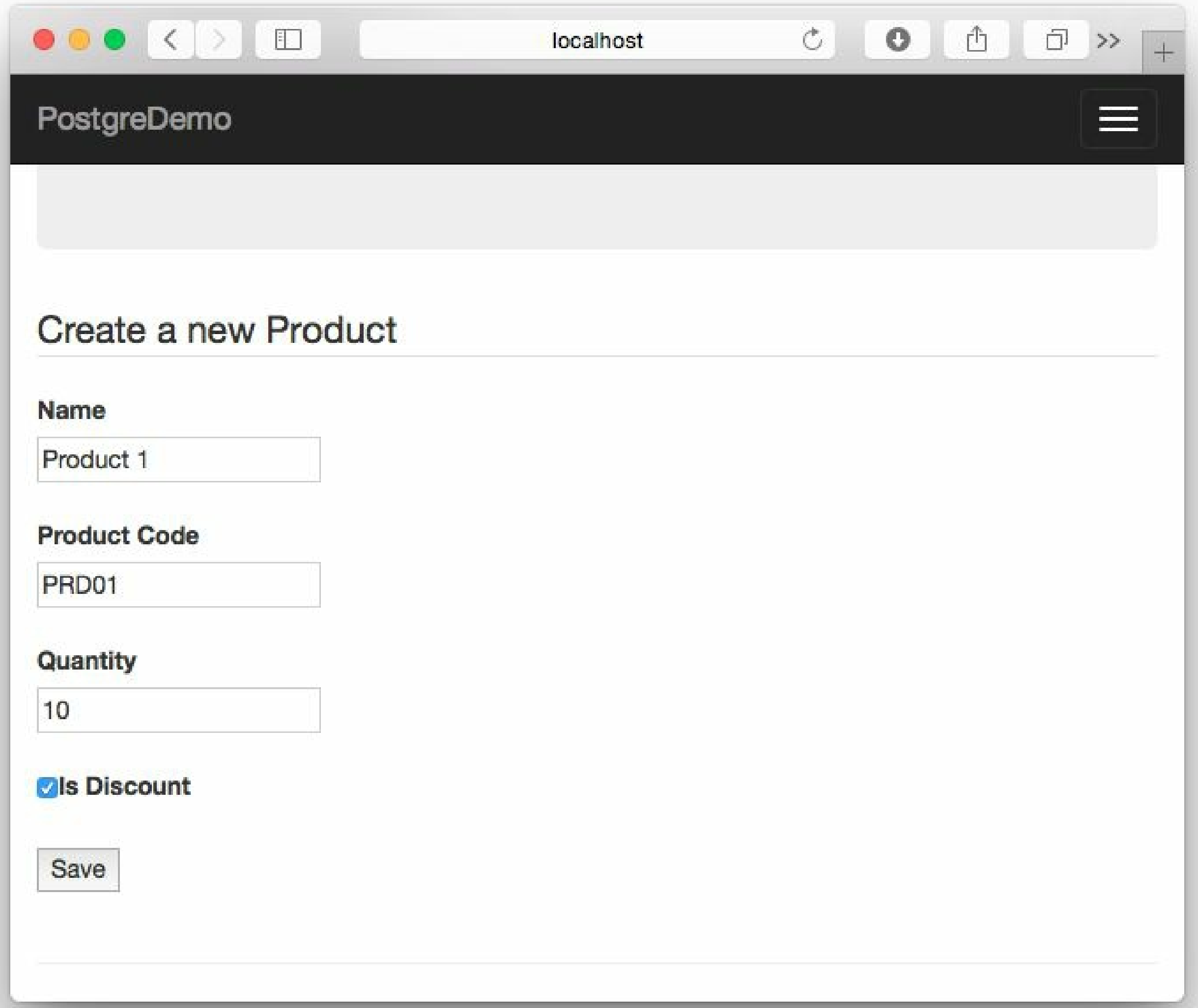
ASP.NET Core MVC and PostgreSQL Demo

Filling and submitting a form.

© 2017 - PostgreDemo

Then, click Create a product button.

Fill all fields for creating a new product.



Create a new Product

Name

Product 1

Product Code

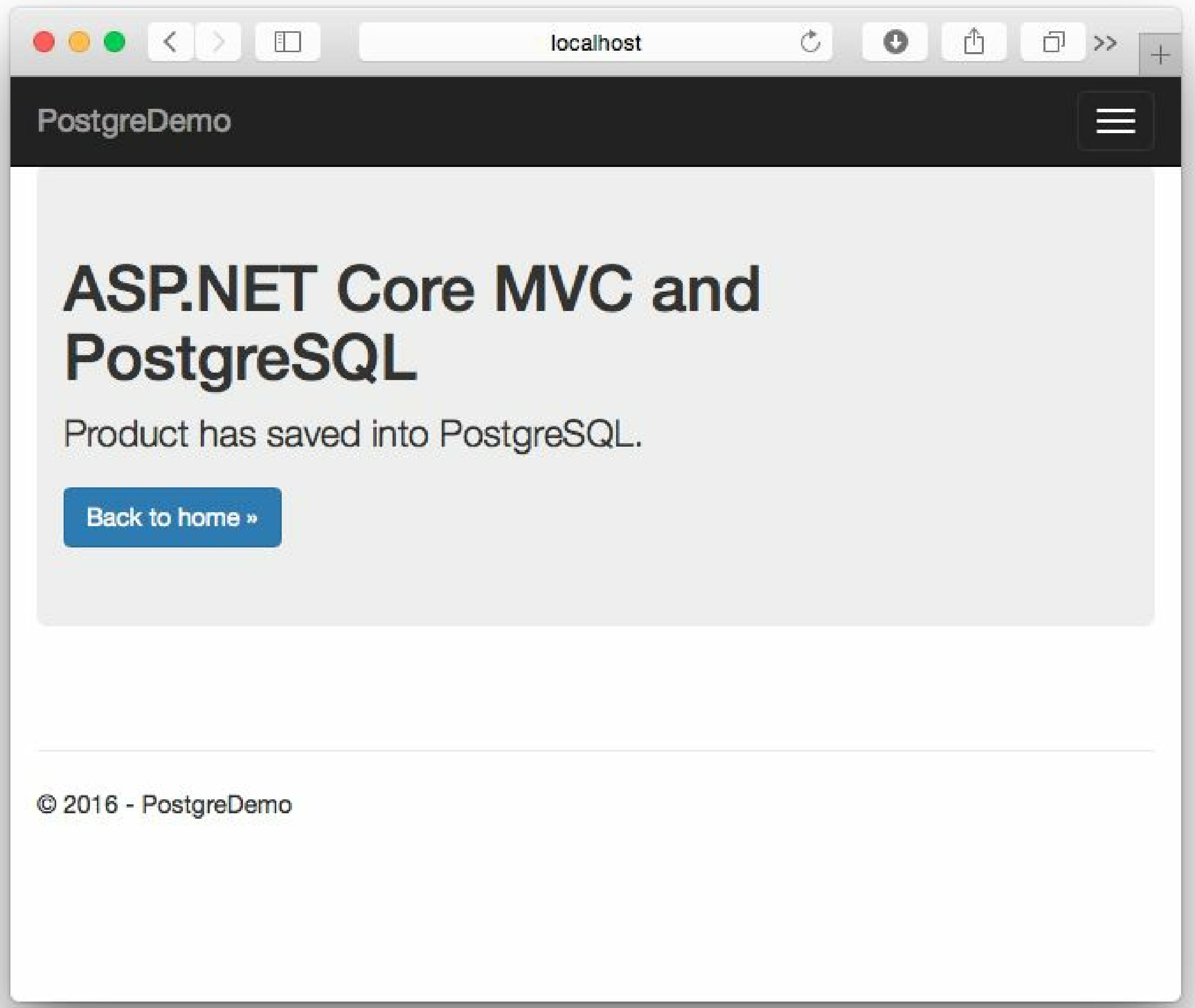
PRD01

Quantity

10

☑ Is Discount

If done, click Save button. If succeed, you should see the following form.


localhost

PostgreDemo

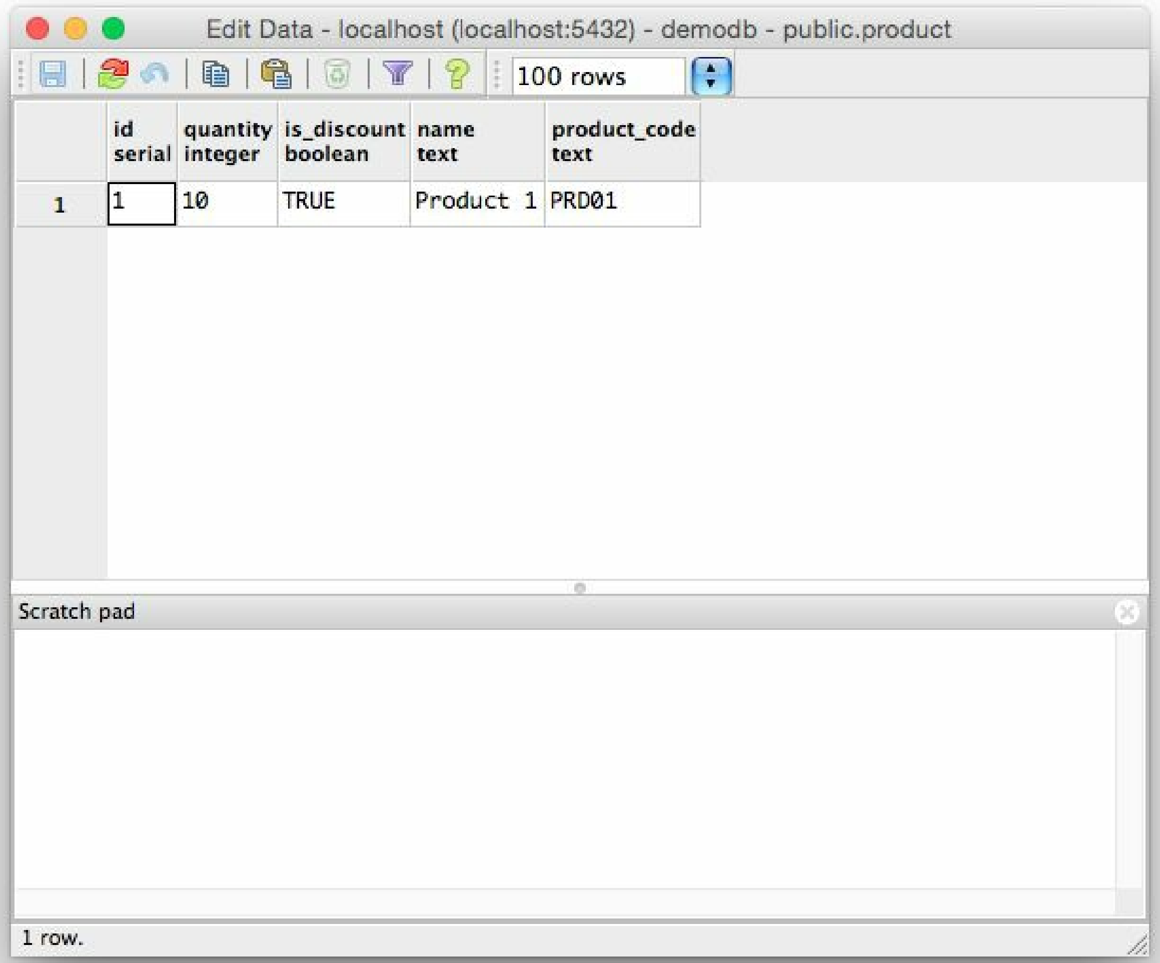
ASP.NET Core MVC and PostgreSQL

Product has saved into PostgreSQL.

Back to home »

© 2016 - PostgreDemo

You can verify the data in PostgreSQL manager.


Edit Data - localhost (localhost:5432) - demodb - public.product

100 rows

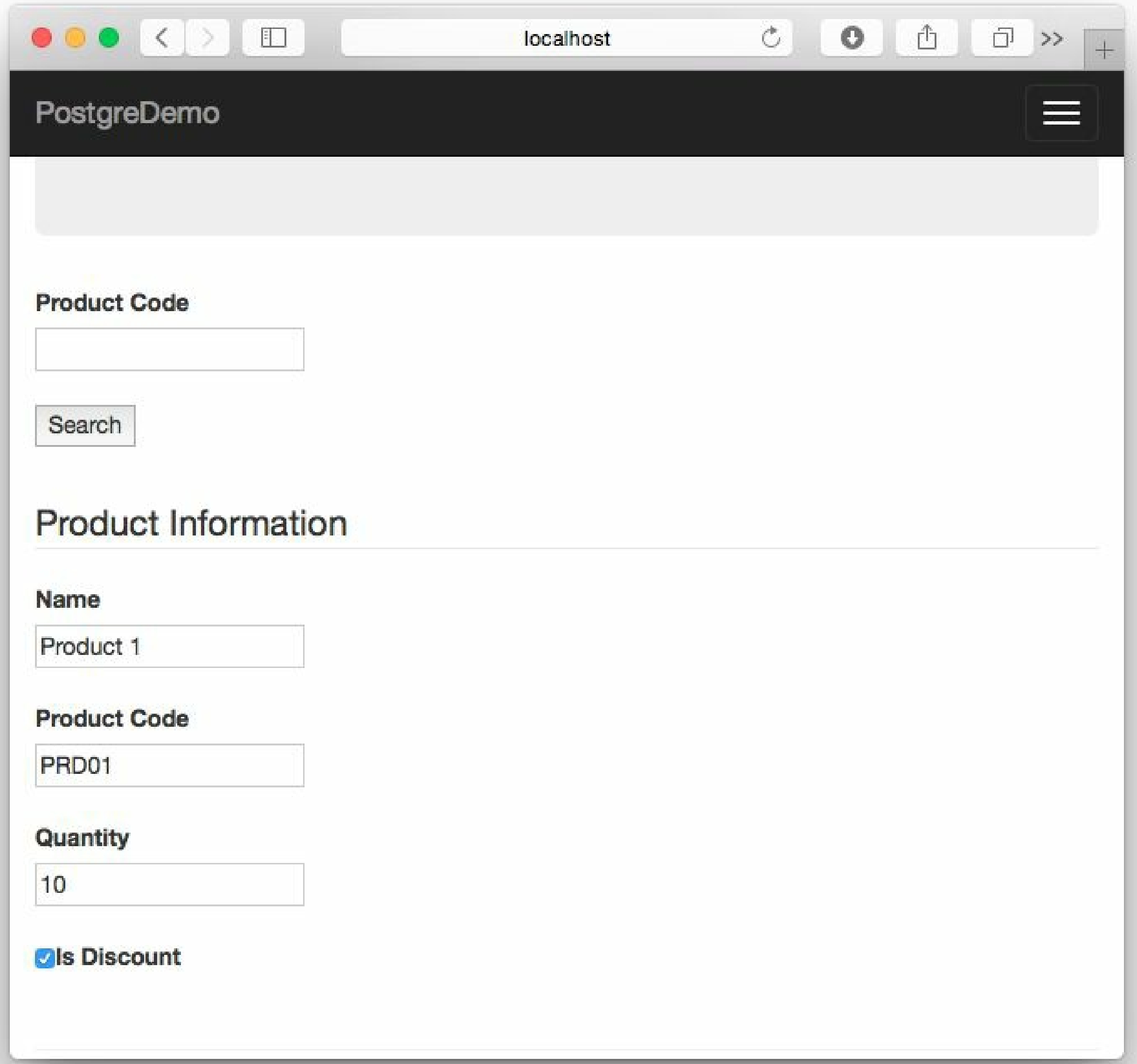
| | id serial | quantity integer | is_discount boolean | name text | product_code text |
|---|---|---|---|---|---|
| 1 | 1 | 10 | TRUE | Product 1 | PRD01 |

Scratch pad

1 row.

For testing of listing data, you can click List products button in home form. Fill a product code that you inserted a product before. If done, click Search button.

You should see a product with a product code which you entry.



localhost

PostgreDemo

Product Code

Search

Product Information

Name

Product 1

Product Code

PRD01

Quantity

10

☑ Is Discount

# Source Code

Source code can be downloaded on http://www.aguskurniawan.net/book/aspnetcore26st.zip.

# Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: http://blog.aguskurniawan.net.

# Table of Contents

Copyright 2

Preface 5

1. Preparing Development Environment 6
   - 1.1 ASP.NET Core 6
   - 1.2 System Requirements 6
   - 1.3 Development Tools 6

2 Deploying ASP.NET Core 8
   - 2.1 Getting Started 8
   - 2.2 Deploying ASP.NET Core on macOS, Linux and Windows 8
   - 2.3 Testing 10

3. ASP.NET Core MVC Development 12
   - 3.1 Getting Started 12
   - 3.2 ASP.NET Core Template 12
   - 3.3 Building Your First ASP.NET Core MVC Application 13
     - 3.3.1 Creating A Project 13
     - 3.3.2 Build and Run the Program 15
   - 3.4 Building ASP.NET Core MVC Application 17
     - 3.4.1 Creating a Project 17
     - 3.4.2 Configuring ASP.NET Core MVC Project 19
     - 3.4.3 Writing Program 21
       - 3.4.3.1 Creating Model 22
       - 3.4.3.2 Creating Controller 22
       - 3.4.3.3 Creating View 23
     - 3.4.4 Testing 26
   - 3.5 What's Next? 29

4. ASP.NET Core API Development 30
   - 4.1 Getting Started 30
   - 4.2 Building ASP.NET Core API Application 30
     - 4.2.1 Creating a Project 30
     - 4.2.2 Writing Program for ASP.NET Core API 33
     - 4.2.3 Testing 35
   - 4.3 Client Application for ASP.NET Core API 36
     - 4.3.1 Creating Controller 36
     - 4.3.2 Creating View 37

4.4 Testing 39
5. ASP.NET Core and Angular 2 41
5.1 Getting Started 41
5.2 ASP.NET Core and Angular 2 Project Template 41
5.3 Run Demo: ASP.NET Core Angular SPA 41
5.3.1 Creating a Project 41
5.3.2 Running the Program 43
5.4 What's The Next? 45
6. ASP.NET Core, Entity Framework Core and PostgreSQL 46
6.1 Getting Started 46
6.2 PostgreSQL Database 46
6.3 Entity Framework Core 47
6.4 Demo Database Application 47
6.4.1 Build PostgreSQL Database 47
6.4.2 Creating a ASP.NET Core MVC Project 47
6.4.3 Configure ASP.NET Core MVC Project 47
6.4.4 Develop Data Access 48
6.4.5 Develop ASP.NET Core MVC Controllers 50
6.4.6 Develop ASP.NET Core MVC Views 52
6.4.7 Testing 55
Source Code 61
Contact 62