Developing ASP.NET MVC Web Applications

Session: 14

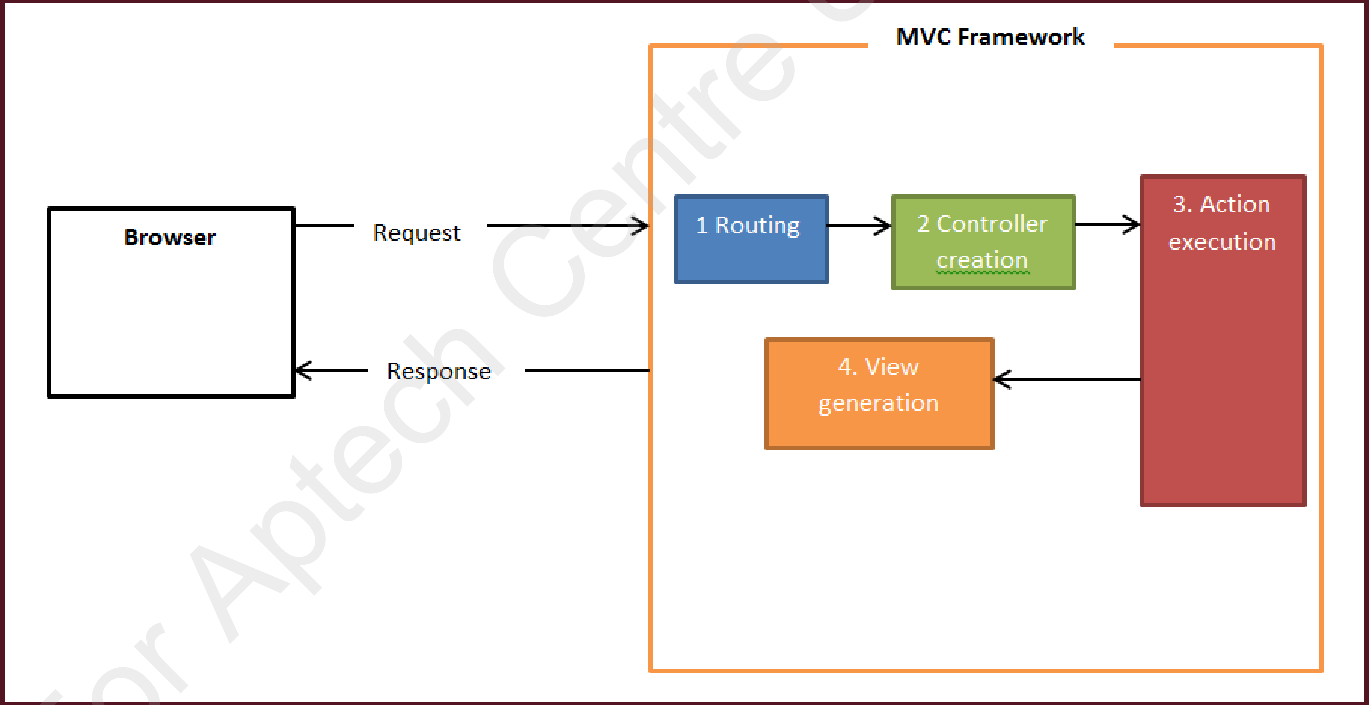
Advanced Concepts of ASP.NET MVC

Objectives

- Explain request handling
- Define and describe filters
- Explain Web API

Request Handling 1-2

- MVC Framework provides several components that together function to handle requests coming from a client.
- Following figure shows how MVC Framework performs request handling:



Request Handling 2-2

- In the preceding figure, the different steps to handle a request are as follows:
  - Routing: In this step, the Framework maps the URL of the incoming request with URL patterns present in the route table. If a matching pattern is found the request is forwarded to an IRouteHandler object.
  - Controller creation: In this step the MvcHandler object uses the IControllerFactory objects and creates a controller object that implements the IController interface.
  - Action execution: In this step the CreateActionInvoker() method of the controller object is called to obtain a ControllerActionInvoker object that determines and executes the action.
  - View generation: In this step a view engine is created that generates the view based on the ActionResult object. The view is then, sent as a response to the client.

Route Handler

- In an ASP.NET MVC application, when a request is received, the routing module matches the URL of the incoming request with route constraints defined for the application.
- Once a URL matches with a route constraint, the routing module returns a handler for this route.
- This handler is referred to as a route handler for the request and is an implementation of the IRouteHandler interface.
- The default route handler of MVC Framework is MvcRouteHandler.
- However, at times you might need to create custom route handler to handle requests.

HTTP Handler 1-3

- The HTTP handler handles HTTP requests that the route handler receives as a URL.
- For a request to an XML file, the HTTP handler will be responsible for loading the file and sending the content of the file as a response.
- To create an HTTP handler to process requests for XML files, you need to create a class that implements the IHttpHandler interface and override the ProcessRequest() method.
- This method accepts an HttpContext object that represents information about the HTTP request object.

HTTP Handler 2-3

- Following code snippet shows using the CustomHttpHandler class, that implements the IHttpHandler interface, which contains the ProcessRequest() method:

Snippet

```
public class CustomHttpHandler : IHttpHandler   {
public void ProcessRequest(HttpContext context) {
  RouteValueDictionary values =
   context.Request.RequestContext.RouteData.Values;
   string path = context.Server.MapPath("~/XML/" + values["name"] +
    ".xml");
  XDocument xdoc = XDocument.Load(path);
  context.Response.Write(xdoc);
  context.Response.Flush();
 }
 public bool IsReusable
  {
   get { return true; }
  }
 }
}
```

HTTP Handler 3-3

- In the preceding code:
  - The CustomHttpHandler class implements the IHttpHandler interface that contains the ProcessRequest() method. In this method the HttpContext.Request.RequestContext.RouteData.Values property is accessed to retrieve a RouteValueDictionary object that represents a collection of route values.
  - Then, the path to the requested URL is created and the Load() method of the XDocument class is called to load the XML file as an XDocument object that represents an XML document.
  - The Httpcontext.Response.Write() method is called to send the content of the XML document as response.

Filters

- In an ASP.NET MVC application there might be situations where you need to implement some functionality before or after the execution of an action method.
- In such situations, you need to use filters.
- While developing an ASP.NET MVC application, you can use filters at the following levels:
  - An action method: When you use filters in an action method, the filter will execute only when the associated action method is accessed.
  - A controller: When you use filters in controller, the filter will execute for all the actions methods defined in the controller.
  - Application: When you use filters in an application, the filter will execute for all the actions methods present in the application.
- The ASP.NET MVC Framework supports different types of filters, such as authorization, action, result, and exception filters.

Authorization Filters 1-3

- Authorization filters execute before an action method is invoked to make security decisions on whether to allow the execution of the action method or not.
- In ASP.NET MVC Framework, the AuthorizeAttribute class of the System.Web.Mvc namespace is an example of authorization filters.
- This class extends the FilterAttribute class and implements the IAuthorizationFilter interface.
- The authorization filter allows you to implement standard authentication and authorization functionality in your application.

Authorization Filters 2-3

- Following code snippet shows adding the Authorize attribute on the ViewPremiumProduct() method:

Snippet

```
[Authorize]
public ActionResult ViewPremiumProduct()
{
   return View();
}
```

- This code adds the [Authorize] attribute on the ViewPremiumProduct() action method.
- Once you have added the Authorize attribute on the ViewPremiumProduct() action method, the access to the corresponding view is restricted.
- Therefore, whenever any user tries to access this view, a page to authenticate the user is displayed.

Authorization Filters 3-3

- You can use the web.config file to specify the page to be displayed for user authentication.
- Following code snippet shows how to specify the page to be displayed for user authentication:

Snippet

```
<authentication mode="Forms">
  <forms loginUrl="~/Account/Login" timeout="1440" />
</authentication>
```

- In this code:
  - The <forms> element specifies the login URL for the application.
  - Whenever a user tries to access a restricted resource, the user is redirected to the login URL.
  - The timeout attribute of the <forms> element specifies the amount of time in minutes, after which the authentication cookie expires. It is set to 1440 minutes. Its default value is 30 minutes.

Action Filters 1-7

- MVC action filters enable you to execute some business logic before and after an action method executes.
- When you use an action filter, the filter receives the following notifications from the MVC Framework for each action method that the filter applies to:
  - The Framework is about to execute the action.
  - The Framework has completed executing the action.
- While creating an ASP.NET application, you can create your own action filter as per requirements.
- To use a custom action filter in an application, you first need to create it and then, apply it to one or more target actions.
- You can create a custom action filter by implementing the IActionFilter interface.

Action Filters 2-7

- Following table describes the methods of the IActionFilter interface:

Method	Description
void OnActionExecuting(ActionExecutingContext filterContext)	This method is called before an action method is invoked. The ActionExecutingContext object represents the filter context that you can use to access information about the current controller, HTTP context, request context, action result, and route data.
void OnActionExecuted (ActionExecutedContext filterContext)	This method is called after invocation of an action method. The ActionExecutedContext object represents the filter context that you can use to access information about the current controller, HTTP context, request context, action result, and route data.

Action Filters 3-7

- You can also extend the ActionFilterAttribute class to create a custom filter.
- This class implements the IActionFilter interface to provide a base class for filter attributes.
- To create a custom action filter that creates log messages, you need to create a class that extends from the ActionFilterAttribute class.
- Then, use the OnActionExecuting() method to specify the code to log messages.

Action Filters 4-7

- Following code shows the CustomActionFilterAttribute class:

Snippet

```
public class CustomActionFilterAttribute : ActionFilterAttribute  {
public override void OnActionExecuting(ActionExecutingContext
filterContext)
 {
   Debug.WriteLine("Action execution is about to start", "Action
Filter Log");
  }
. . .
}
```

- This code uses the OnActionExecuting() method and logs a message when the execution of the action method is about to start.
- Next, in the OnActionExecuted() method you can log a message to indicate that the execution of the action method has been completed.

Action Filters 5-7

- Following code shows the code of the OnActionExecuted() method:

Snippet

```
public override void OnActionExecuted(ActionExecutedContext
filterContext)
   {
      Debug.WriteLine("Action execution has completed", "Action
Filter Log");
   }
```

- This code uses the OnActionExecuted() method and creates a log message once the execution of the action method has been completed.
- Once you have created an action filter, you can apply it to the action method of a controller class.

Action Filters 6-7

- Following code shows how to apply the CustomActionFilter filter at the Index() action method:

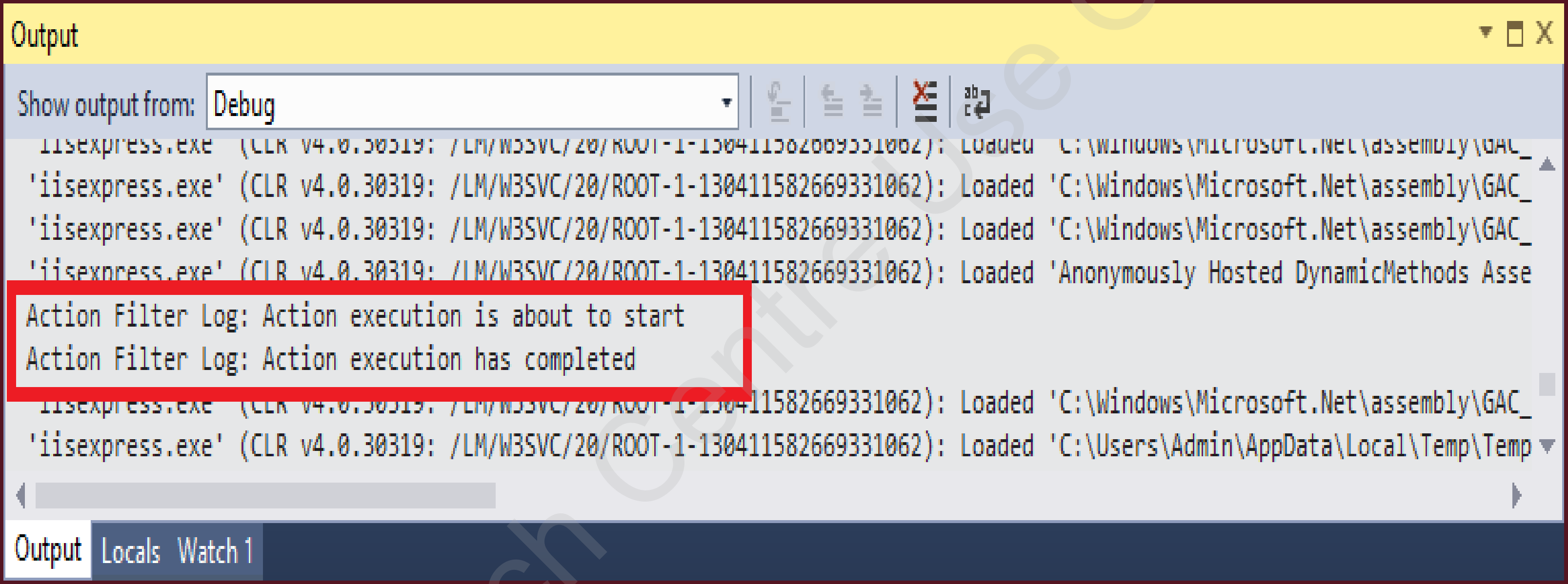
Snippet

```
public class HomeController : Controller
 {
      [CustomActionFilter]
      public ActionResult Index()
      {
           return View();
      }
 }
```

- This code applies the CustomActionFilter filter to the Index() action method of the Home controller.
- When you debug the application, the CustomActionFilter filter executes and logs the messages in the output window of Visual Studio 2013.

Action Filters 7-7

- Following figure shows the log messages in the Output window:



```
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/20/ROOT-1-130411582669331062): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/20/ROOT-1-130411582669331062): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/20/ROOT-1-130411582669331062): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/20/ROOT-1-130411582669331062): Loaded 'Anonymously Hosted DynamicMethods Asse
Action Filter Log: Action execution is about to start
Action Filter Log: Action execution has completed
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/20/ROOT-1-130411582669331062): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/20/ROOT-1-130411582669331062): Loaded 'C:\Users\Admin\AppData\Local\Temp\Temp
```

Result Filters 1-3

- A result filter operates on the result that an action returns.
- When you implement a result filter, the filter will receive the following notifications from the MVC Framework for each action method that the filter applies to:
  - The action method is about to execute the action result.
  - The action method has completed executing the action result.
- The OutputCacheAttribute class is one example of a result filter, which is used to mark an action method whose output will be cached.
- The OutputCache filter indicates the MVC Framework to cache the output from an action method.
- The same content can be reused to service subsequent requests for the same URL.

Result Filters 2-3

- Caching action output can offer a significant increase in performance, because most of the time-consuming activities required to process a request are avoided.
- Following code snippet shows how to use the OutputCache attribute:

Snippet

```
public class HomeController : Controller   {
      [OutputCache]
       public ActionResult Index()   {
       //some code
      }
 }
```

- In this code, the [OutputCache] attribute is added to the Index() action method of the Home controller.
- You can specify the duration for which the output of the action should be cached by specifying a Duration property with the duration time in seconds.

Result Filters 3-3

- Following code shows specifying the Duration property:

Snippet

```
public class HomeController : Controller
   {
     [OutputCache(Duration=3)]
   public ActionResult Index()
   {
            //some code
     }
   }
```

- In this code, the Duration property of the OutputCache attribute is set to cache the output for 3 seconds.

Exception Filters 1-5

- In an ASP.NET MVC application, you can use exception filters to handle exceptions that the application throws at runtime.
- Exception filters are additional exception handling component of MVC Framework besides the built-in .NET Framework exception handling mechanism comprising try-catch block.
- The MVC Framework provides a built-in exception filter through the HandleError filter that the HandleErrorAttribute class implements.
- Like other filters, you can use the HandleError filter on an action method or a controller.
- The HandleError filter handles the exceptions that are raised by the controller actions and other filters applied to the action.
- This filter returns a view named Error.cshtml which by default is in the shared folder of the application.

Exception Filters 2-5

- Following code snippet shows the Error.cshtml view that displays an error message whenever an action with a HandleError filter throws an exception:

Snippet

```
@model System.Web.Mvc.HandleErrorInfo
@{
    ViewBag.Title = "Error";
}
<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your
request.</h2>
```

- This code displays an error message whenever an action with a HandleError filter throws an exception.
- To use the HandleError filter, you need to configure the web.config file by adding a customErrors attribute inside the <system.web> element.

Exception Filters 3-5

- Following code snippet shows how to enable custom errors in the web.config file:

Snippet

```
. . .
  <system.web>
. . .
    <customErrors mode="On" />
  </system.web>

. . .
```

- In this code, the On value of the mode attribute in the <customErrors> element enables exception handling using the HandleError filter.
- To test how the HandleError filter works, you can update the Index() action method of the Home controller to throw an exception and handle it using the HandleError filter.

Exception Filters 4-5

- Following code snippet shows how to use the HandleError filter on the Index() action method of the Home controller that throws an exception:

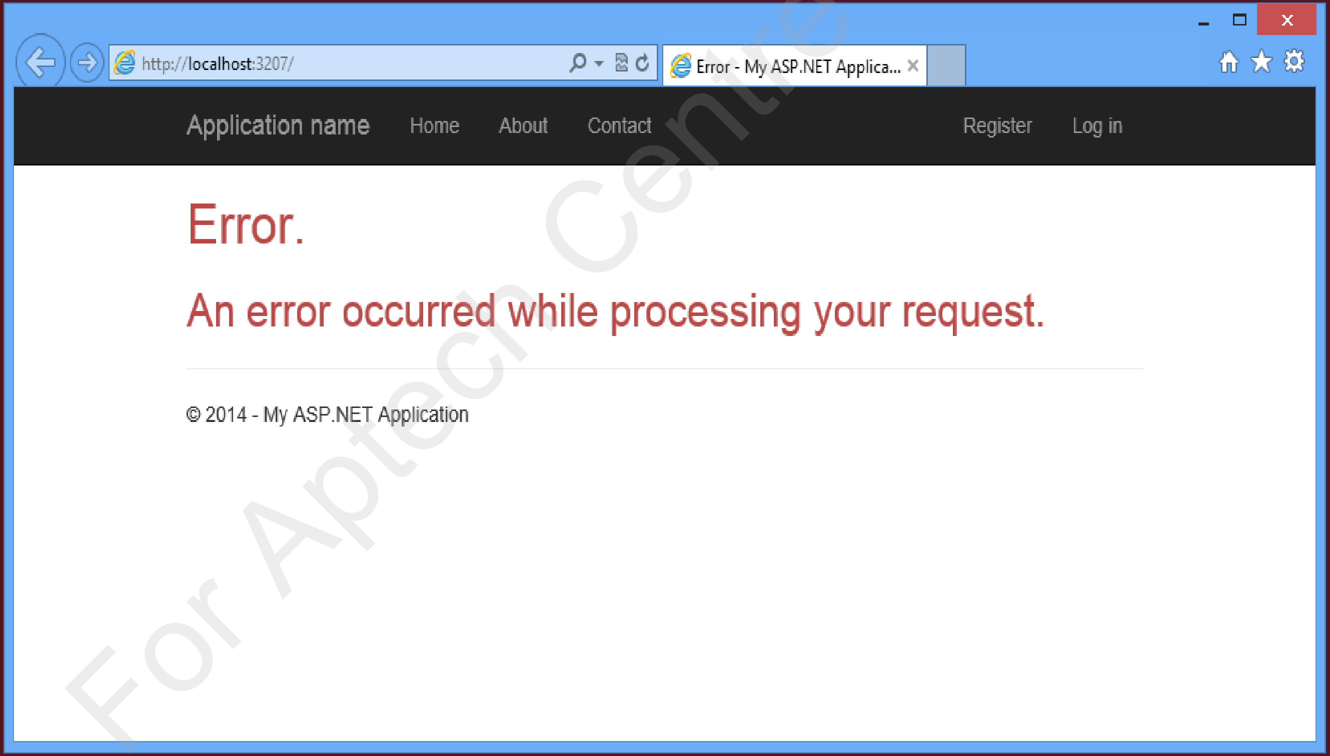
Snippet

```
public class HomeController : Controller
    {
       [HandleError]
         public ActionResult Index()
         {
             throw new DivideByZeroException();
             return View();
         }
. . .
}
```

- This code applies the HandleError filter to the Index() action method of the Home controller. The Index() action method throws an exception of type DivideByZeroException.

Exception Filters 5-5

- When you access the Index() action method of the Home controller, the HandleError filter will catch the exception of type DivideByZeroException and display the Error.cshtml view.
- Following figure shows the Error.cshtml view:



Error.

An error occurred while processing your request.

© 2014 - My ASP.NET Application

Introduction to Web API 1-2

- Web API is a Framework that allows you to easily create Web services that provide an API for a wide range of clients, such as browsers and mobile devices using the HTTP protocol.
- Using Web API you can display data based on the client requests.
- You can use Web API to develop HTTP based services where client can make a GET, PUT, POST, and DELETE requests and access the response provided by Web API.
- Web API allows you to add special controllers, known as API Controller.

Introduction to Web API 2-2

- The main characteristics of an API controller are as follows:
  - The action methods return model instead of the ActionResult, objects.
  - The action methods selected based on the HTTP method used in the request.
- The model objects that are returned from an API controller action method are encoded as JSON and sent to the client.
- In addition, as API controllers deliver Web data services, so they do not support views, layouts to generate HTML for browsers to display.

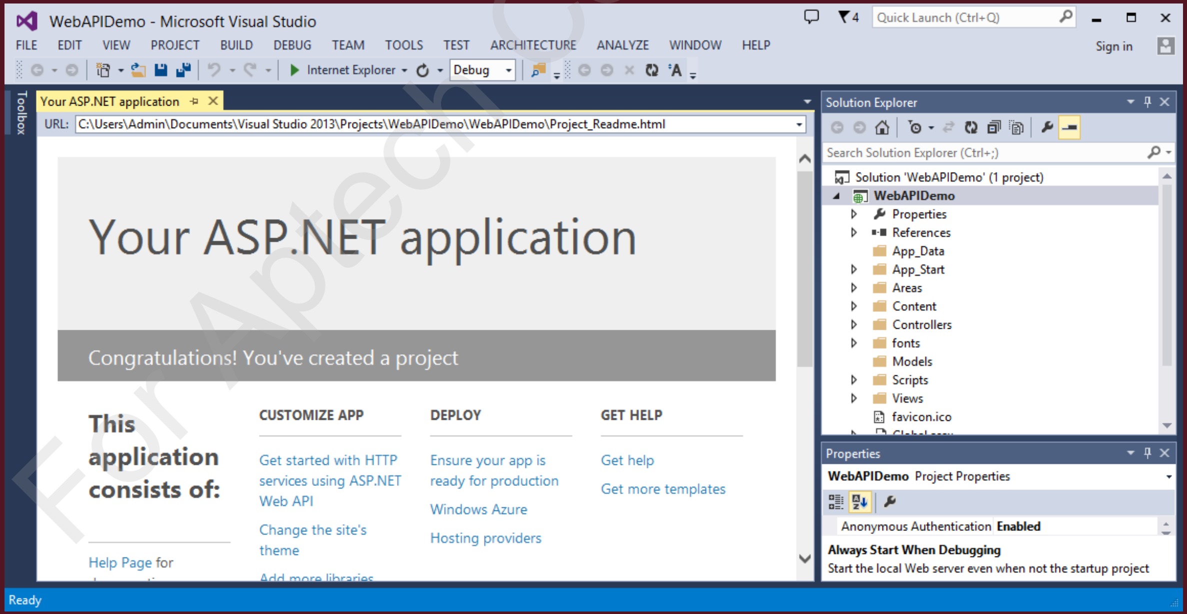
Creating a Web API Application 1-7

- A Web API application is just a regular MVC Framework application with the addition of a special kind of controller.
- Visual Studio 2013 allows you to create and build Web API application.
- To create a new Web API project in Visual Studio 2013, you need to perform the following steps:
  - Open Visual Studio 2013.
  - Click File→New→Projects menu options in the menu bar of Visual Studio 2013.
  - In the New Project dialog box that appears, select Web under the Installed section and then, select the ASP.NET Web Application template.
  - Type WebAPIDemo in the Name text field.
  - Click the Browse button and specify the location where the application has to be created.

Creating a Web API Application 2-7

- Click OK. The New ASP.NET Project – WebAPIDemo dialog box is displayed.
- Select Web API under the Select a template section of the New ASP.NET Project – WebAPIDemo dialog box.
- Click OK. Visual Studio 2013 displays the newly created Web API application.

- Following figure shows the newly created Web API application in Visual Studio 2013:



URL: C:\Users\Admin\Documents\Visual Studio 2013\Projects\WebAPIDemo\WebAPIDemo\Project_Readme.html

Your ASP.NET application

Congratulations! You've created a project

This application consists of:

Help Page for

CUSTOMIZE APP
Get started with HTTP services using ASP.NET Web API
Change the site's theme
Add more libraries

DEPLOY
Ensure your app is ready for production
Windows Azure
Hosting providers

GET HELP
Get help
Get more templates

Solution 'WebAPIDemo' (1 project)
- WebAPIDemo
  - Properties
  - References
  - App_Data
  - App_Start
  - Areas
  - Content
  - Controllers
  - fonts
  - Models
  - Scripts
  - Views
  - favicon.ico

WebAPIDemo Project Properties
Anonymous Authentication Enabled
Always Start When Debugging
Start the local Web server even when not the startup project

Creating a Web API Application 3-7

- Once you have created the Web API application, you need to create a model.
- To create a model in Visual Studio 2013, you need to perform the following steps:
  - Right-click the Model folder in the Solution Explorer window and select Add→Class from the context menu that arrears. The Add New Item – WebAPIDemo dialog box is displayed.
  - Type Product.cs in the Name text field of the Add New Item – WebAPIDemo dialog box.
  - Click Add. The Code Editor displays the newly created Product class.
  - In Code Editor add the code to the Product class to represent a product.

Creating a Web API Application 4-7

- Following code shows the Product class:

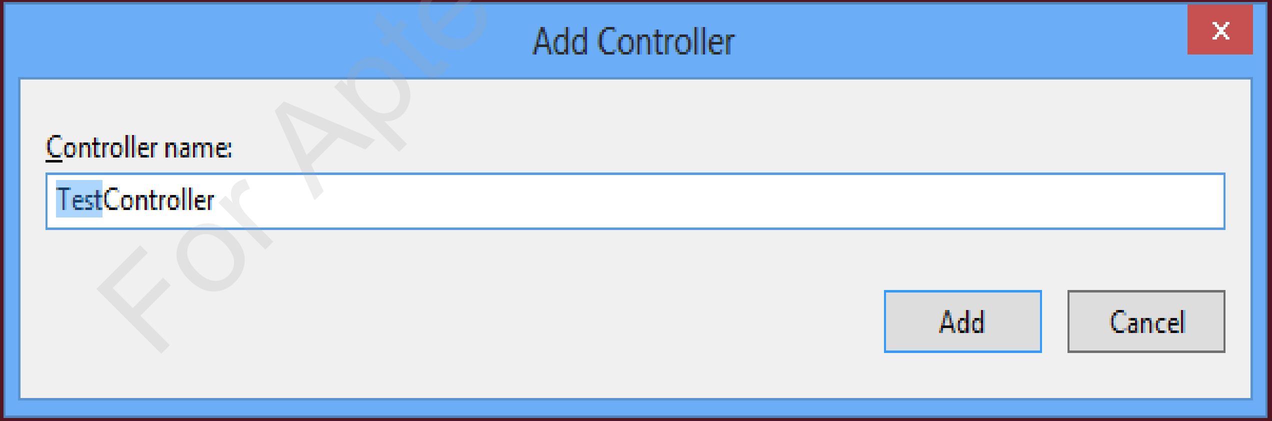
Snippet

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
namespace WebAPIDemo.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Category { get; set; }
        public string Name { get; set; }
    }
}
```

- This code declares three variables named Id, Category, and Name along with the get and set methods.
- Once you have created the model named, Product.cs you can add a Web API controller to the application.

Creating a Web API Application 5-7

- In Visual Studio 2013, to create a Web API controller, you need to perform the following steps:
  - Right-click the Controllers folder in the Solution Explorer window and select Add→Controller from the context menu that arrears. The Add Scaffold dialog box is displayed.
  - Select the Web API 2 Controller – Empty template in the Add Scaffold dialog box.
  - Click Add. The Add Controller dialog box is displayed.
  - Type TestController in the Controller name field.
- Following figure shows the Add Controller dialog box:



Controller name: TestController

Creating a Web API Application 6-7

- Click Add. The Code Editor displays the newly created TestController controller class.
- In the TestController controller class, initialize a Product array with Product objects. Then, create a Get() method to return the Product array

- Following code shows the TestController controller class:

Snippet

```
public class TestController : ApiController {
 Product[] products = new Product[]  {
   new Product {Id = 1, Name = "Product 1", Category= "Category 
1"},
   new Product {Id = 2, Name = "Product 2", Category= "Category 
1"},
   new Product {Id = 3, Name = "Product 3", Category= "Category 
2"},
     };
   public IEnumerable<Product> Get()
     {
        return products;
     }
 }
```

Creating a Web API Application 7-7

- The preceding code:
  - Defines a Product array that contains three Product objects.
  - A Get() method is defined that will process GET requests that arrives to the controller.
  - The Get() method returns the Product array.

Defining Routing in Web API 1-4

- Once you have created the Web API controller, you need to register it with the ASP.NET routing Framework.
- When the Web API application receives a request, the routing Framework tries to match the URI against one of the route templates defined in the WebApiConfig.cs file. If no route matches, the client receives a 404 error.
- When you create a Web API application in Visual Studio 2013, by default the IDE configures the route of the application in the WebApiConfig.cs file under the App_Start folder.

Defining Routing in Web API 2-4

- Following code snippet shows the default route configuration of the Web API application:

Snippet

```
config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
```

- This code shows the default route configuration for a Web API application.
- This route configuration contains a route template specified by the routeTemplate attribute that defines the following pattern:

`"api/{controller}/{id}"`

Defining Routing in Web API 3-4

- In the preceding route pattern:
  - api: Is a literal path segment
  - {controller}: Is a placeholder for the name of the controller to access
  - {id}: Is an optional placeholder that the controller method accepts as parameter
- You can use the RouteTable.MapHttpRoute() method to configure routes of a Web API application.
- For example, consider the following URL: http://localhost:9510/web/Test
- Assuming that the preceding URL needs to access the TestController controller class of the WebAPIDemo application, you need to configure a route in the WebApiConfig.cs file.

Defining Routing in Web API 4-4

- Following code snippet shows how to configure a route:

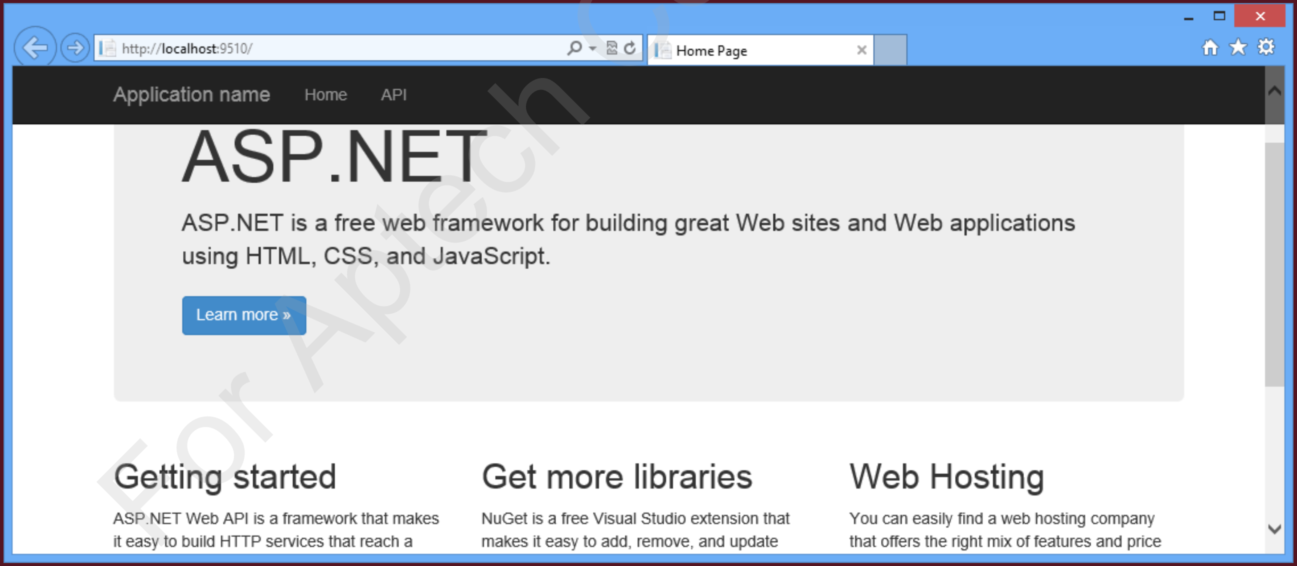
Snippet

```
config.Routes.MapHttpRoute(
      name: "TestDefaultApi",
      routeTemplate: "web/{controller}/{id}",
      defaults: new { id = RouteParameter.Optional }
    );
```

- In this code, a route named TestDefaultApi is created with the route pattern web/{controller}/{id}.

Accessing an ASP.NET Web API Application 1-2

- Once you have created the controller class and configured the routing of the Web API application, you can access it over the browser.
- For that, you need to perform the following steps:
  - Click Debug→Start Without Debugging. The browser window displays the Home page of the application.
- Following figure shows the Home page of the application:



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.

Getting started

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a

Get more libraries

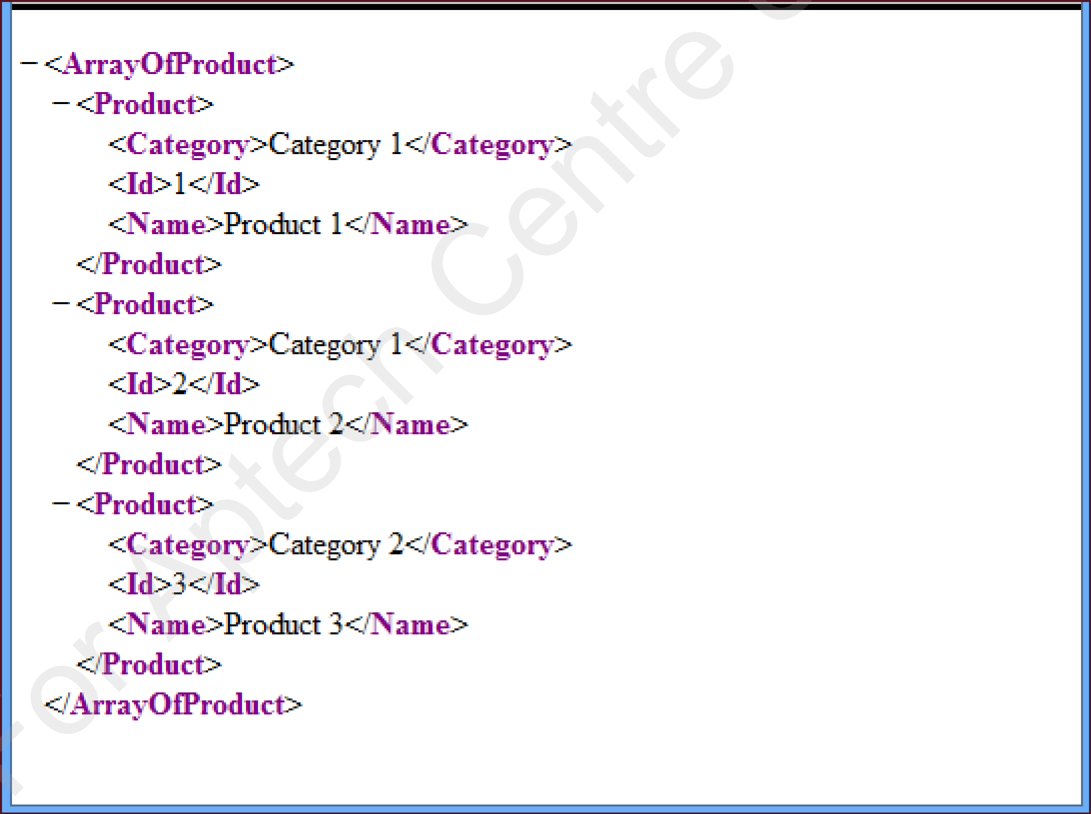
NuGet is a free Visual Studio extension that makes it easy to add, remove, and update

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price

Accessing an ASP.NET Web API Application 2-2

- Type the following URL in the address bar of the browser: http://localhost:9510/web/Test
- Browser displays the product details, as shown in the following figure:



```
- <ArrayOfProduct>
  - <Product>
      <Category>Category 1</Category>
      <Id>1</Id>
      <Name>Product 1</Name>
    </Product>
  - <Product>
      <Category>Category 1</Category>
      <Id>2</Id>
      <Name>Product 2</Name>
    </Product>
  - <Product>
      <Category>Category 2</Category>
      <Id>3</Id>
      <Name>Product 3</Name>
    </Product>
  </ArrayOfProduct>
```

Summary

- MVC Framework provides several components that together function to handle requests coming from a client.
- In an ASP.NET MVC application, when a request is received, the routing module matches the URL of the incoming request with route constraints defined for the application.
- The HTTP handler handles HTTP requests that the route handler receives as a URL.
- Authorization filters execute before an action method is invoked to make security decisions on whether to allow the execution of the action method.
- MVC action filters enable you to execute some business logic before and after an action method executes.
- Exception filters are additional exception handling component of MVC Framework besides the built-in .NET Framework exception handling mechanism comprising of try-catch block.
- Web API is a Framework that allows you to easily create Web services that provide an API for a wide range of clients, such as browsers and mobile devices using the HTTP protocol.