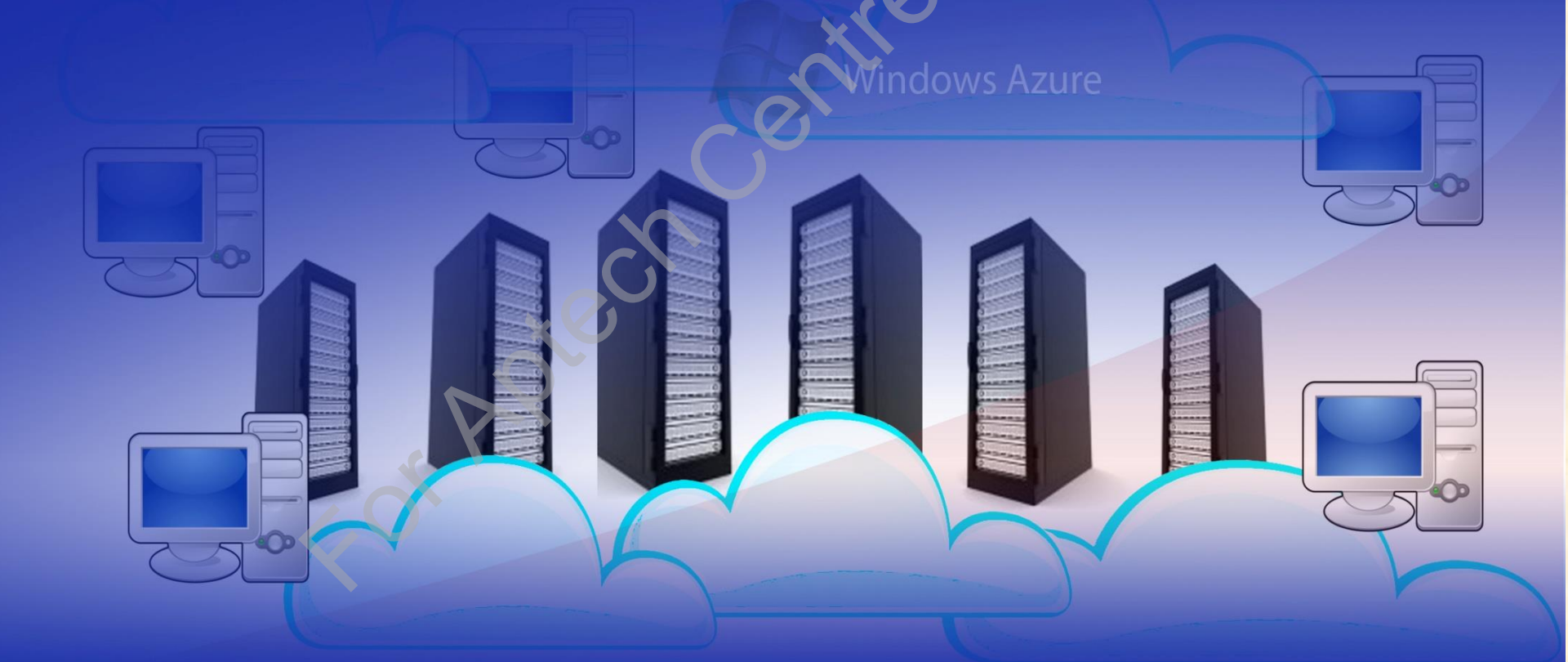
Enterprise Application Development Using Windows Azure and Web Services

Session 3

ASP.NET Web API Services



Learning Objectives

- Define and describe ASP.NET Web API
- Define and describe HTTP, REST, and Media Types
- Explain how to implement a ASP.NET Web API Service
- Explain how to secure a ASP.NET Web API Service

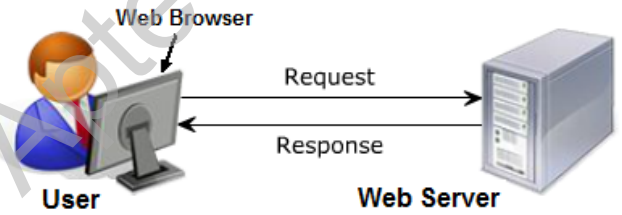
Introduction

ASP.NET Web API is a technology that allows you to create Web services that target diverse clients.

- ASP.NET Web API helps in handling client requests and sending back responses using content type, such as JavaScript Object Notation (JSON) or XML.

Evolution of ASP.NET Web API 1-6

- ASP.NET Web API is the result of:
  - The gradual evolution of traditional Web services.
- Both Web services and Web Applications:
  - Receive, process, and return appropriate responses.
- Web Applications:
  - A request/response interchange comprises a user communicating with the application through a client, such as a Web browser.



- Web Services:
  - A request/response interchange involves applications.

Evolution of ASP.NET Web API 2-6

- Consider an example:
  - An application provides payment information after a user checks out from an online shopping store application.
  - It accesses a Web service that a payment gateway service provider utilizes to process the payment.

Evolution of ASP.NET Web API 3-6

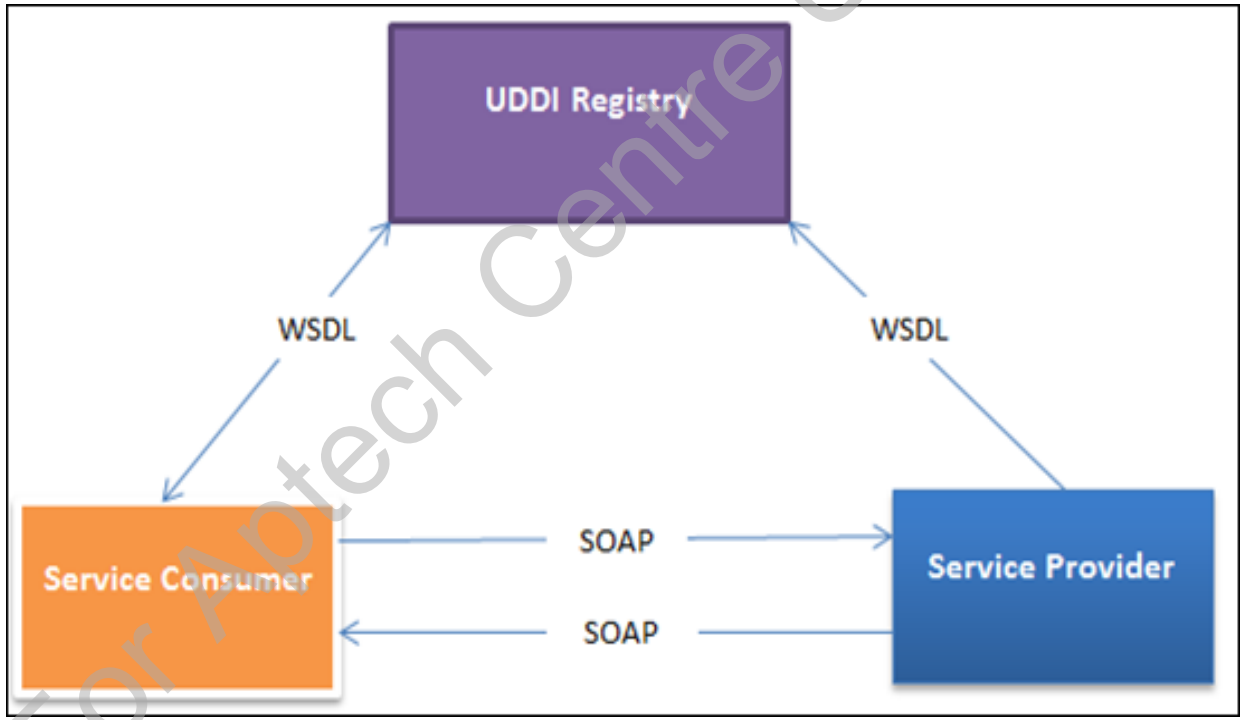
- To communicate with the payment gateway service provider, the shopping application and the provider both need to follow certain standards.
- These standards are also known as Web services standards.
- They include SOAP, Web Services Definition Language (WSDL), and Web Service Specifications (WS-*) that introduces formal service contracts.

Evolution of ASP.NET Web API 4-6

- In traditional Web services, a service provider uses WSDL to publish how the service can be consumed.
- A WSDL document:
  - Describes the service endpoint where the service can be accessed.
  - Describes the service methods that consumers can call to avail the service.
  - Is published to a Universal Description, Discovery, and Integration (UDDI) registry.
- UDDI registry:
  - Can both be public or private to an organization.
  - Is a repository for publishing WSDL documents.
  - Is used by service consumers to access published WSDL documents in order to access a Web service.

Evolution of ASP.NET Web API 5-6

- Following figure shows the communication flow in a Web service:

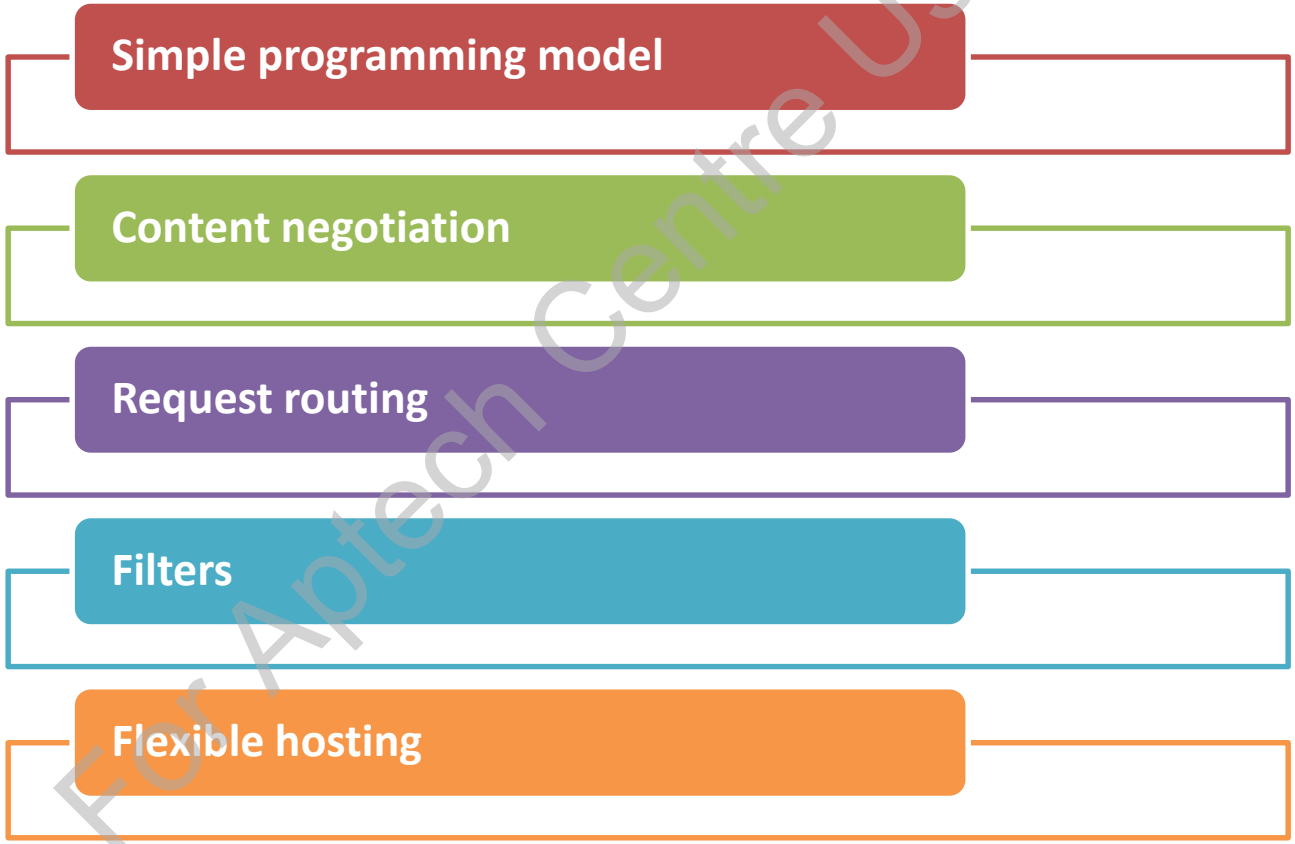


Evolution of ASP.NET Web API 6-6

- Gradually, several standards became part of the Web services stack.
- Consider an example of WS-Security:
  - Introduced an extension to SOAP for securely transmitting SOAP messages.
  - Introduced to manage trust relationships between entities involved in Web services communication. All the Web services standards became collectively known as WS-*.
  - Introduced RESTful services that are available over plain HTTP and do not require implementation of WS-*.

ASP.NET Web API Features 1-3

- Following are the key features of ASP.NET Web API:



ASP.NET Web API Features 2-3

Simple programming model

- Enables easy accessing and updating HTTP requests and responses in an ASP.NET Web API application.
- Uses the same programming model as used on the server and on the client.
- Involves any .NET application.

Content negotiation

- Provides support to enable the client and server to negotiate the data format that the service returns as a response.
- Supports JSON, XML, and Form URL-encoded formats.
- Provides flexibility by extending the default data format by adding your own formatters.
- Replaces the default content negotiation strategy with one of your own.

ASP.NET Web API Features 3-3

Request routing

- Provides a routing module for mapping request URLs to the correct controller action.
- Defines custom routes in a centralized location without any additional configurations.

Filters

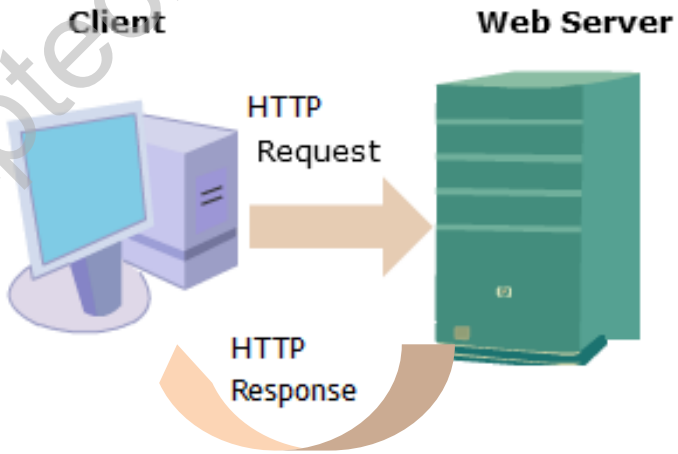
- Provides filters to perform certain logic just before and after an action method of an API controller is invoked.

Flexible hosting

- Provides the flexibility to be hosted with other .NET applications, such as ASP.NET MVC and ASP.NET Web Forms.
- Supports self-hosting, for applications hosted within a process that runs a console application.

HTTP Protocol

- HTTP protocol:
  - Communicates over the Web. When a URL in the Address bar of a browser is typed and submitted, an HTTP request is sent to an application running on a server that is represented by the URL.
  - Returns back a HTTP response. The browser on receiving the response displays the response to you.
  - Provides different types of request methods based on the type of operations that the request needs to make.



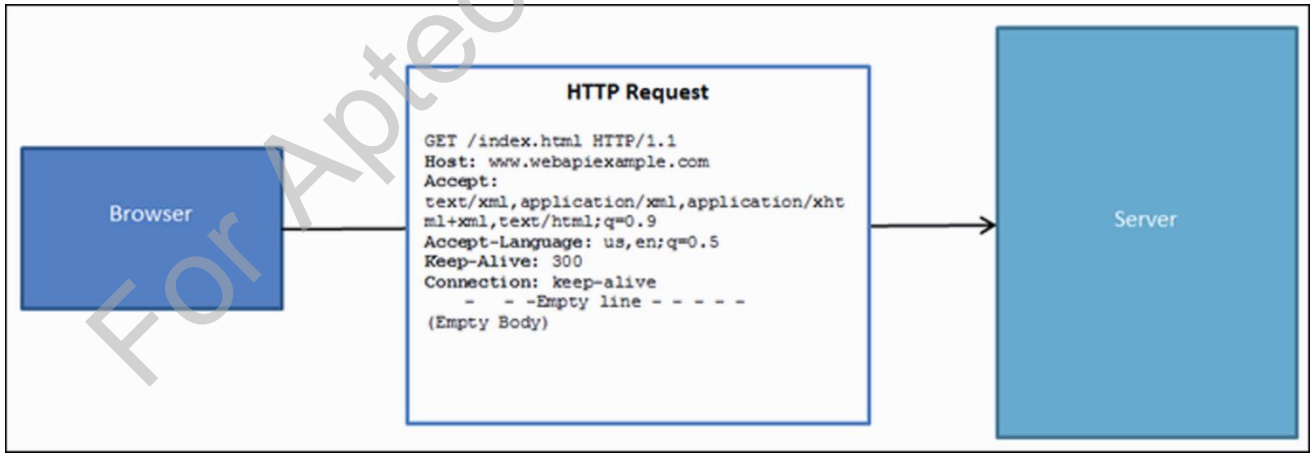
HTTP Methods

- The four main HTTP methods are as follows:

GET	POST	PUT	DELETE
• Requests the server to retrieves a resource.	• Requests the server that the target resource should process the data contained in the request.	• Requests the server to create or update a request.	• Requests the server to delete a resource.

HTTP Request 1-4

- HTTP Request:
  - Irrespective of the method used in an HTTP request, all HTTP requests:
    - Have a standard format.
    - Contains request headers and a request body.
    - Provides information about the request, such as the HTTP method, the URL of the request, and the type of content that the request expects as response. The request body can be empty or can contain data.
- Following figure shows a HTTP request sent from a browser to a resource hosted on a server:



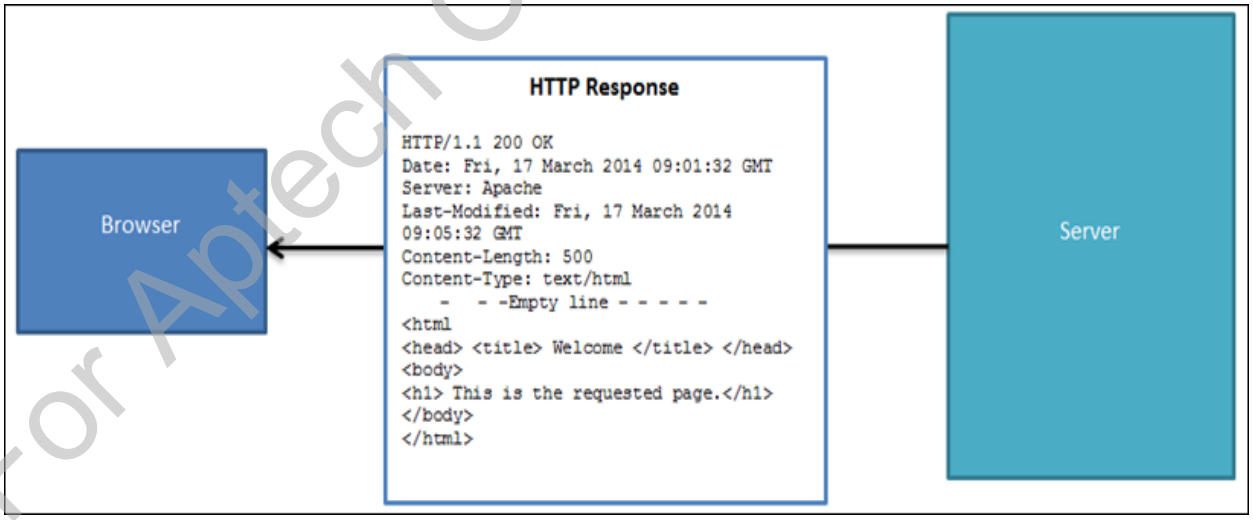
HTTP Request 2-4

- The first line of the HTTP request specifies GET as the HTTP method and the resource that the request attempts to access.
- The first line is followed by these HTTP request headers:

Host	Specifies the domain name of the application whose resource is being accessed.
Accept	Specifies the content type, also known as MIME type that the request expects as response.
Accept-Language	Specifies the preferred language of the client.
Connection	Specifies whether the server should use the same connection for HTTP communication instead of creating a new connection for each new request.
KeepAlive	Specifies the duration in seconds for which the server should use the same connection for HTTP communication.

HTTP Request 3-4

- HTTP Response:
  - Whenever a server receives an HTTP request, it sends back an HTTP response.
  - An HTTP response contains response headers and a response body.
- Following figure shows a HTTP response sent from a server to a browser:



HTTP Request 4-4

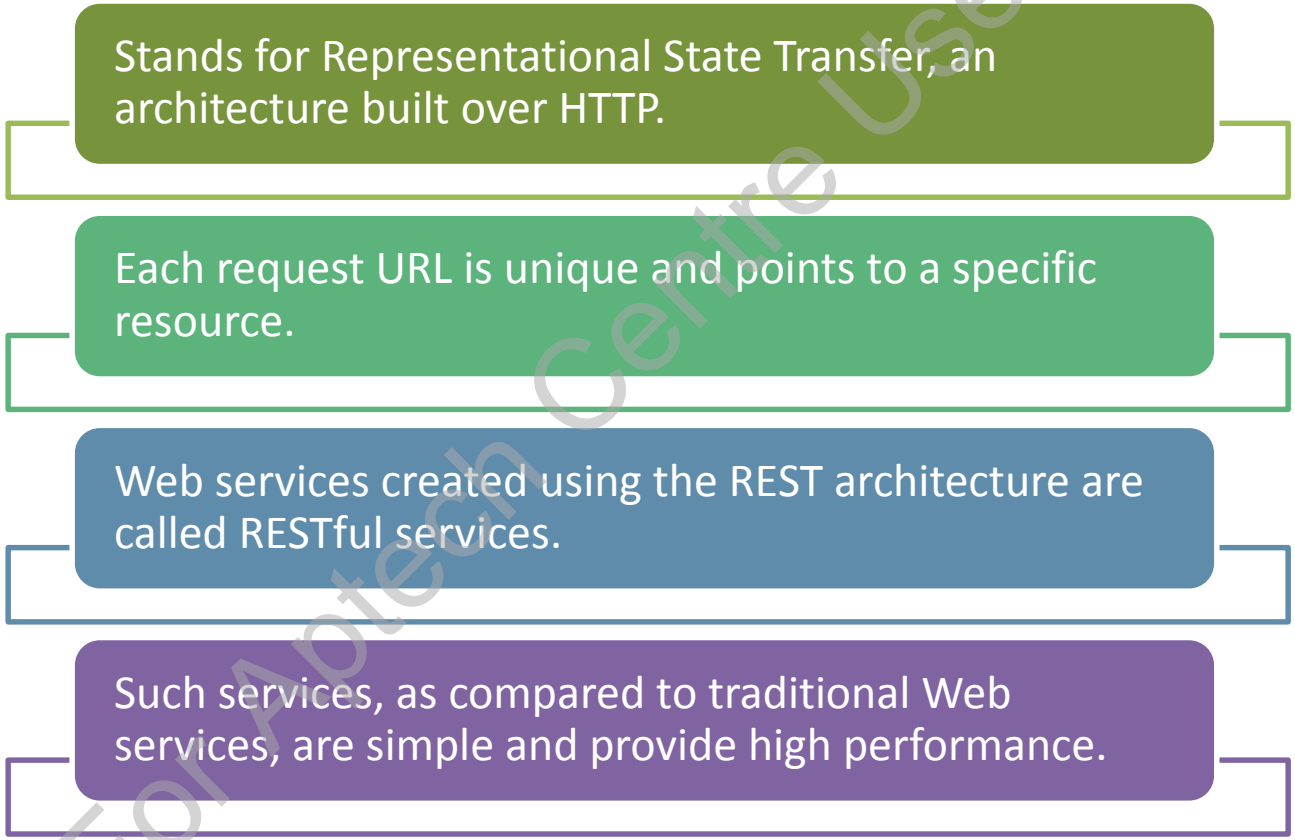
- The first line of the HTTP response specifies a HTTP status code.
- The **200 OK** status code indicates that the request has succeeded.
- The first line is followed by these key HTTP response headers:
  - **Date**: Specifies the date and time when the response is being sent from the server.
  - **Server**: Specifies the server that is handling the request response exchange.
  - **Last-Modified**: Specifies the date and time at which the resource was last modified.
  - **Content-Length**: Specifies the size of the response body, in decimal number of octets.
  - **Content-Type**: Specifies the type of content that the response contains. In this example, the `text/html` value specifies the browser known to render the response body as HTML.

HTTP Response

```
HTTP/1.1 200 OK
Date: Fri, 17 March 2014 09:01:32 GMT
Server: Apache
Last-Modified: Fri, 17 March 2014
09:05:32 GMT
Content-Length: 500
Content-Type: text/html
    -   - -Empty line - - - - -
<html
<head> <title> Welcome </title> </head>
<body>
<h1> This is the requested page.</h1>
</body>
</html>
```

REST 1-4

- REST:



REST 2-4

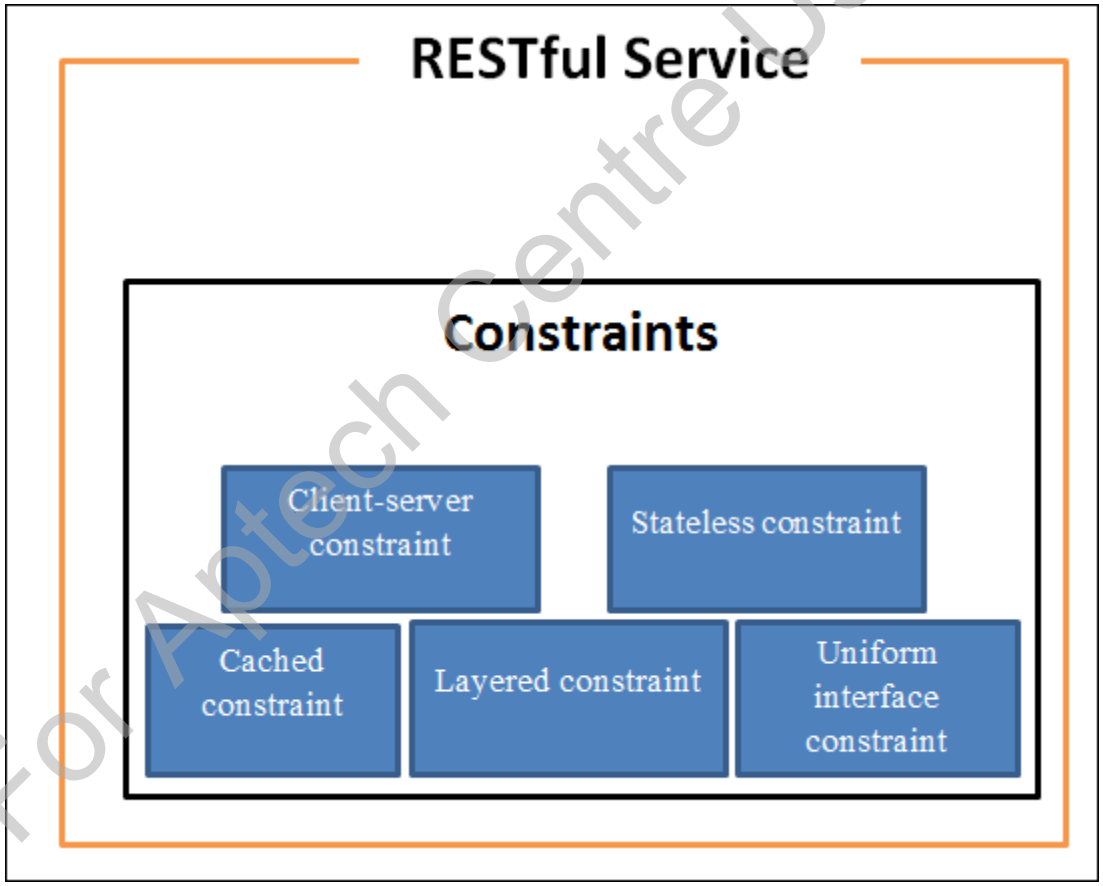
- To create RESTful services, you need to address the following basic design principles:

Explicitly and consistently use the HTTP methods to interact with services. For example, use the GET HTTP method to retrieve a resource use GET and the POST HTTP method to create a resource.	HTTP is stateless and therefore, each HTTP request should be created with all the information required by the server to generate the response.	URLs should only be used to access resources of a RESTful service. These URLs should be consistent and intuitive.	The XML and JSON data format must be supported for request/response exchange between clients and the RESTful service.

- For a Web service to qualify as a RESTful service, the service has to conform to certain defined constraints.

REST 3-4

- Following figure illustrates the mandatory constraints for a Web service to qualify as a RESTful service:



REST 4-4

- The various terms shown in the figure are explained as follows:

Client-server constraint	Specifies that the user interface of the service should be separate from the data storage of the service.
Stateless constraint	Specifies that a request should be an atomic unit containing all information that the server requires to generate a response.
Cached constraint	Specifies whether the server has the capability to inform that a response could be cached so that clients can accordingly handle the response.
Layered constraint	Specifies that the service should be composed of layers where each layer can access and is accessible to its neighbor layer.
Uniform interface constraint	Specifies a uniform interface that is used to identify, access, and manipulate resources through self-descriptive messages.

Media Types 1-3

- Media Types:
  - Is also known as content type, a standard to identify the type of data being exchanged over the Internet.
  - Are used by browsers to specify the type of data they expect as response from the server.

  - A particular media type is identified using an identifier based on the Multipurpose Internet Mail Extensions (MIME) specification. This identifier is known as MIME type.
  - An HTTP request uses the Accept HTTP request header to specify the MIME type that the request expects as a response.
  - In an ASP.NET Web API service, XML and JSON are used as the data formats to deliver service responses.

Media Types 2-3

- Following code snippet shows a response of an ASP.NET Web API service that sends a response in XML format:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8"?>
<Product xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Id>36</Id>
<Name>Laptop</Name>
<Category>Electronics</Category>
</Product>
```

- This code shows a response whose MIME type is `application/xml`, as indicated by the Content-Type response header. The response body contains the information of a specific product in XML format.
- ASP.NET Web API service uses the `JsonMediaTypeFormatter` to send response in JSON format.

Media Types 3-3

- Following code snippet shows a response of an ASP.NET Web API service that sends a response in JSON format:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8

{"Id":"Name","Category":"36","Laptop":"Electronics"}
```

- This code shows a response whose MIME type is `application/json`, as indicated by the Content-Type response header. The response body contains the information of a specific product in JSON format.

Designing and Implementing an ASP.NET Web API Service

- Visual Studio 2013 provides templates and tools to simplify developing ASP.NET Web API services.

- When you create an ASP.NET Web API service in Visual Studio 2013, the Integrated Development Environment (IDE):
  - Creates a skeleton application with a default directory structure.
  - Contains the basic ASP.NET Web API components, such as a controller, route configurations, and the reference libraries.
  - Adds the required services based on the application requirements.
  - Debugs and tests the application before finally hosting it to a production server.

Creating an ASP.NET Web API Application 1-2

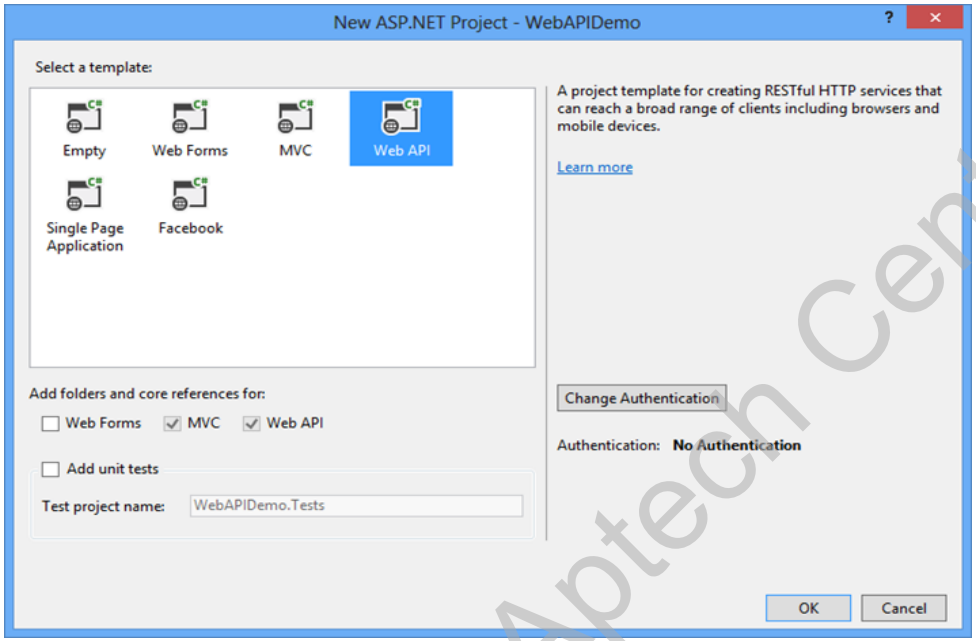
- To create a new ASP.NET Web API application in Visual Studio 2013, you need to perform the following steps:

Step 1	• Open Visual Studio 2013.
Step 2	• Click **File** → **New** → **Project** in Visual Studio 2013.
Step 3	• In the **New Project** dialog box that appears, select **Web** under the **Installed** section, and then select the **ASP.NET Web Application** template.
Step 4	• Type **WebAPIDemo** in the **Name** text field.
Step 5	• Click **Browse** in the dialog box and specify the location where the application has to be created.
Step 6	• Click **OK**. The **New ASP.NET Project – WebAPIDemo** dialog box is displayed.

Creating an ASP.NET Web API Application 2-2

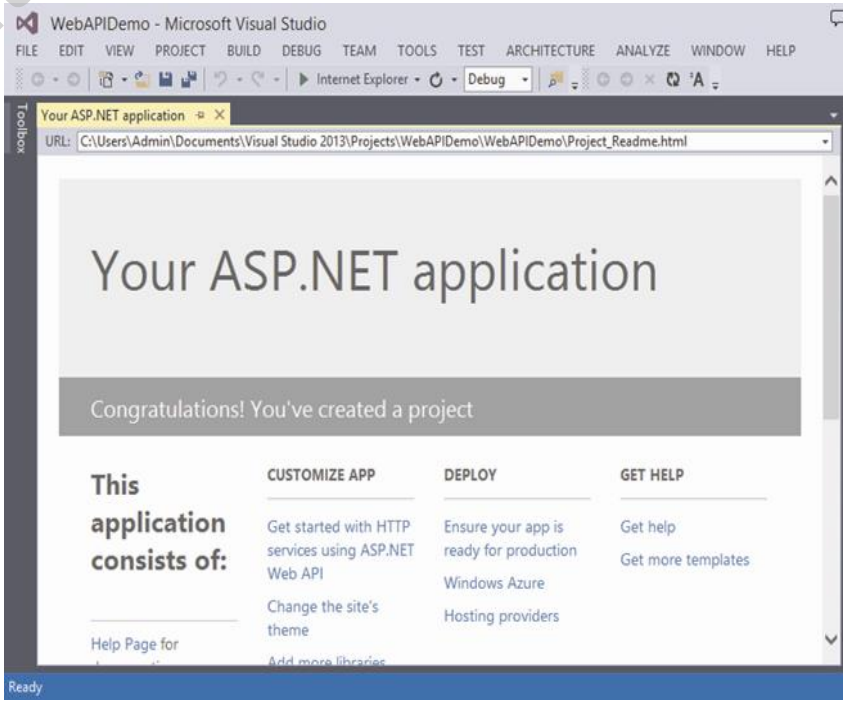
Step 7

Select **Web API** under the **Select a template** section of the **New ASP.NET Project –WebAPIDemo** dialog box.



Step 8

Click **OK**. Visual Studio 2013 displays the newly created ASP.NET Web API application.



Adding a Model 1-4

- Once you have created the ASP.NET Web API application, you need to create a model.

- A model in an ASP.NET Web API service represents application specific data.

  - For example, if you are creating a service for online social integration, your service will typically contain a Profile model to represent profile information of user, a Login model to represent the login information of users, and a Post model to represent information that you post online.

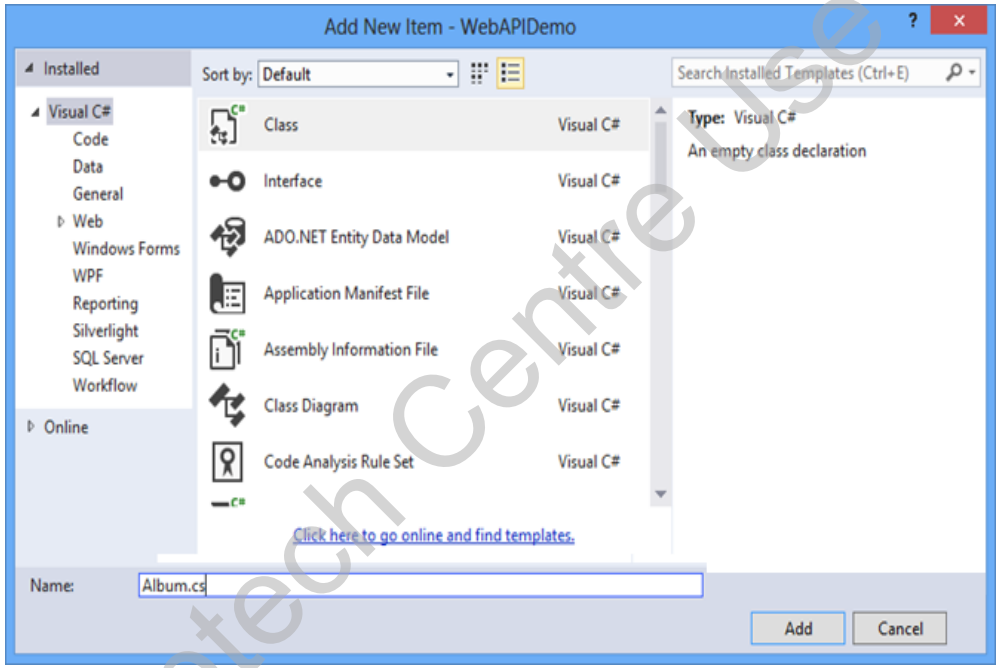
Adding a Model 2-4

- To create a model in Visual Studio 2013, you need to perform the following steps:

Step 1	Right-click the **Models** folder in the Solution Explorer window and select **Add** → **Class** from the context menu that appears. The **Add New Item – WebAPIDemo** dialog box is displayed.
Step 2	Type **Album.cs** in the **Name** text field of the **Add New Item – WebAPIDemo** dialog box.

Adding a Model 3-4

- Figure shows **the Add New Item – WebAPIDemo** dialog box:



Step 3

Click **Add**. The Code Editor displays the newly created **Album** class.

Adding a Model 4-4

Step 4 In Code Editor, add the code shown in the following code snippet to the **Album** class to represent a product.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
namespace WebAPIDemo.Models
{
    public class Album
    {
    public int Id { get; set; }
    public string Name { get; set; }
    public string Genre { get; set; }
    public decimal Price { get; set; }
    }
}
```

This code declares variables named `Id`, `Name`, `Genre`, and `Price` along with the `get` and `set` methods.

Adding a Repository 1-8

- In an ASP.NET Web API service-based application, a repository is:
  - A data source that stores the data of the application
  - An in-memory application object
  - An XML file
  - A separate Relational Database Management System (RDBMS)
  - A cloud-base storage system

- For the **WebAPIDemo** project, create an in-memory application object as a repository to store a collection of albums. Perform the following steps to create a repository in Visual Studio 2013:

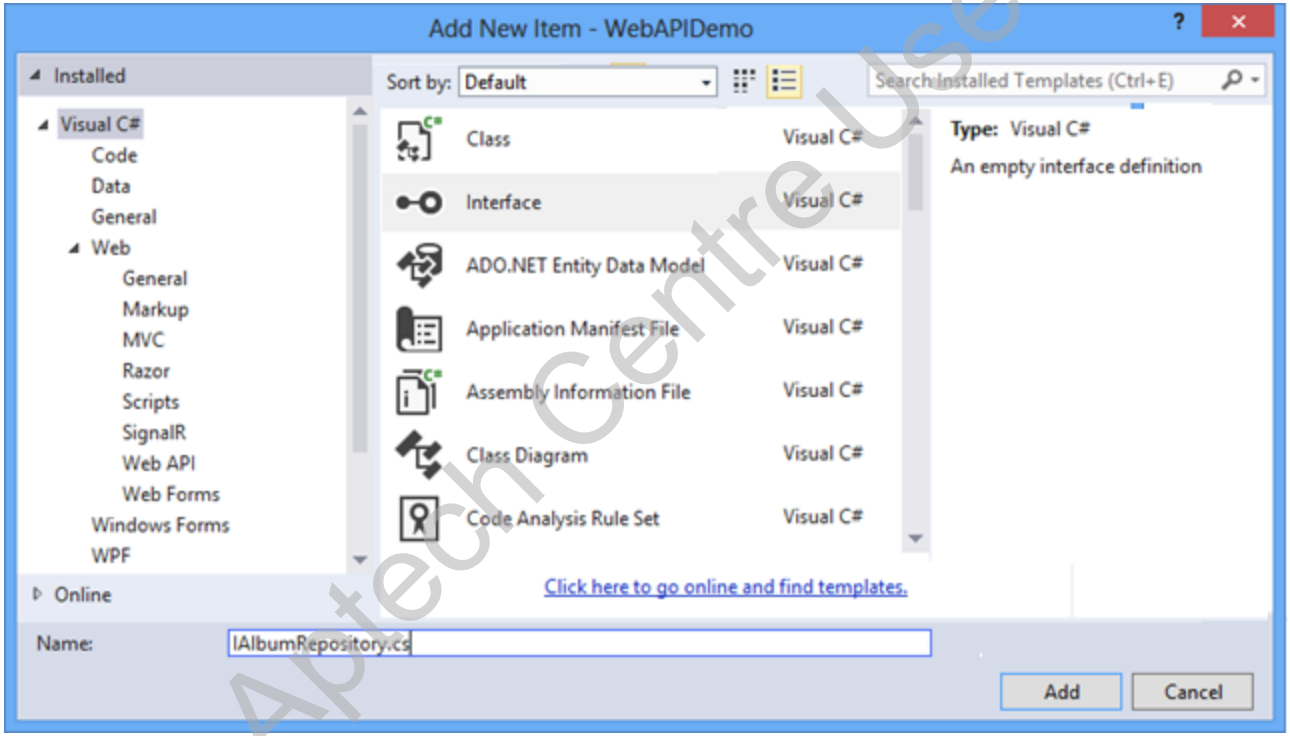
Step 1: Right-click the **Models** folder in Solution Explorer and select **Add → New Item**. The **WebAPIDemo** dialog box is displayed.

Step 2: Select **Visual C#** in the **Templates** pane, and **Interface** on the right of the **Add New Item-WebAPIDemo** dialog box.

Step 3: Type **IAlbumRepository** in the **Name** field.

Adding a Repository 2-8

- Figure shows the process of adding the interface.



Step 4: Click **Add**. The Code Editor displays the newly created **IAlbumRepository** repository.

Adding a Repository 3-8

Step 5: In Code Editor, add the following code to the **IAlbumRepository** repository.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WebAPIDemo.Models
{
interface IAlbumRepository
    {
IEnumerable<Album>GetAll();
        Album Get(int id)
        Album Add(Album item);
void Remove(int id);
bool Update(Album item);
    }
}
```

Adding a Repository 4-8

- The code creates an `IAlbumRepository` interface and declares the following methods:

`GetAll()`	`Get(int id)`	`Add(Album item)`	`Remove(int id`	`Update(Album item)`
• An implementation of this method should return an `IEnumerable<Album>` object that contains details of all the albums.	• An implementation of this method should return an `Album` object of the specified Id passed as parameters to the method.	• An implementation of this method should add a new `Album` object to the `AlbumRepository` object. Once added, this method should return the new `Album` object.	• An implementation of this method should remove an `Album` object specified by the Id passed as parameter from the `AlbumRepository` object.	• An implementation of this method should update the `AlbumRepository` object with the `Album` object passed as parameter.

Adding a Repository 5-8

Step 6: Similarly, create another class named `AlbumRepository` in the **Models** folder.

Step 7: In the Code Editor, add the code to the `AlbumRepository` class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace WebAPIDemo.Models
{
  public class AlbumRepository : IAlbumRepository
  {
   private List<Album> albums = new List<Album>();
    private int _nextId = 1;
    public AlbumRepository() {
```

Adding a Repository 6-8

```
    Add(new Album{Name="The Band", Genre="Classic Rock",
        Price=150});
    Add(new Album {Name="The Blueprint", Genre="HipHop",
        Price=200});
    Add(new Album {Name="Unconditional", Genre="Hard Rock",
       Price=175 });
     }
public IEnumerable<Album>GetAll()
{
  return albums;
}
public Album Get(int id)
{
    return albums.Find(p =>p.Id == id);
 }
public Album Add(Album item)
{
if (item == null)
```

Adding a Repository 7-8

```
{
   throw new ArgumentNullException("item");
}
item.Id = _nextId++;
albums.Add(item);
return item;
}
public void Remove(int id)
{
   albums.RemoveAll(p =>p.Id == id);
}
public bool Update(Album item)
{
if (item == null)
{
    throw new ArgumentNullException("item");
 }
```

Adding a Repository 8-8

```
int index = albums.FindIndex(p =>p.Id == item.Id);
if (index == -1)
{
    return false;
}
     albums.RemoveAt(index);
     albums.Add(item);
     return true;
     }
     }
}
```

- In this code, the `AlbumRepository` class implements the `IAlbumRepository` interface that it created.
- For each of the methods declared in the `IAlbumRepository` interface, the `AlbumRepository` class provides implementation to retrieve, add, and delete albums that the `Album` model represents.

Adding an ASP.NET Web API Controller 1-9

- After creating the model named, `Album` and the repository implementation that acts as a data source for `Album` objects, add an ASP.NET Web API controller to the application.

- The ASP.NET Web API controller is a class that handles HTTP requests from the client.

- This class extends the `ApiController` class and provides methods that are invoked for various types of requests, such as `GET`, `POST`, `PUT`, and `DELETE`.

Adding an ASP.NET Web API Controller 2-9

- Following steps help to create an ASP.NET Web API controller in Visual Studio 2013:

Step 1

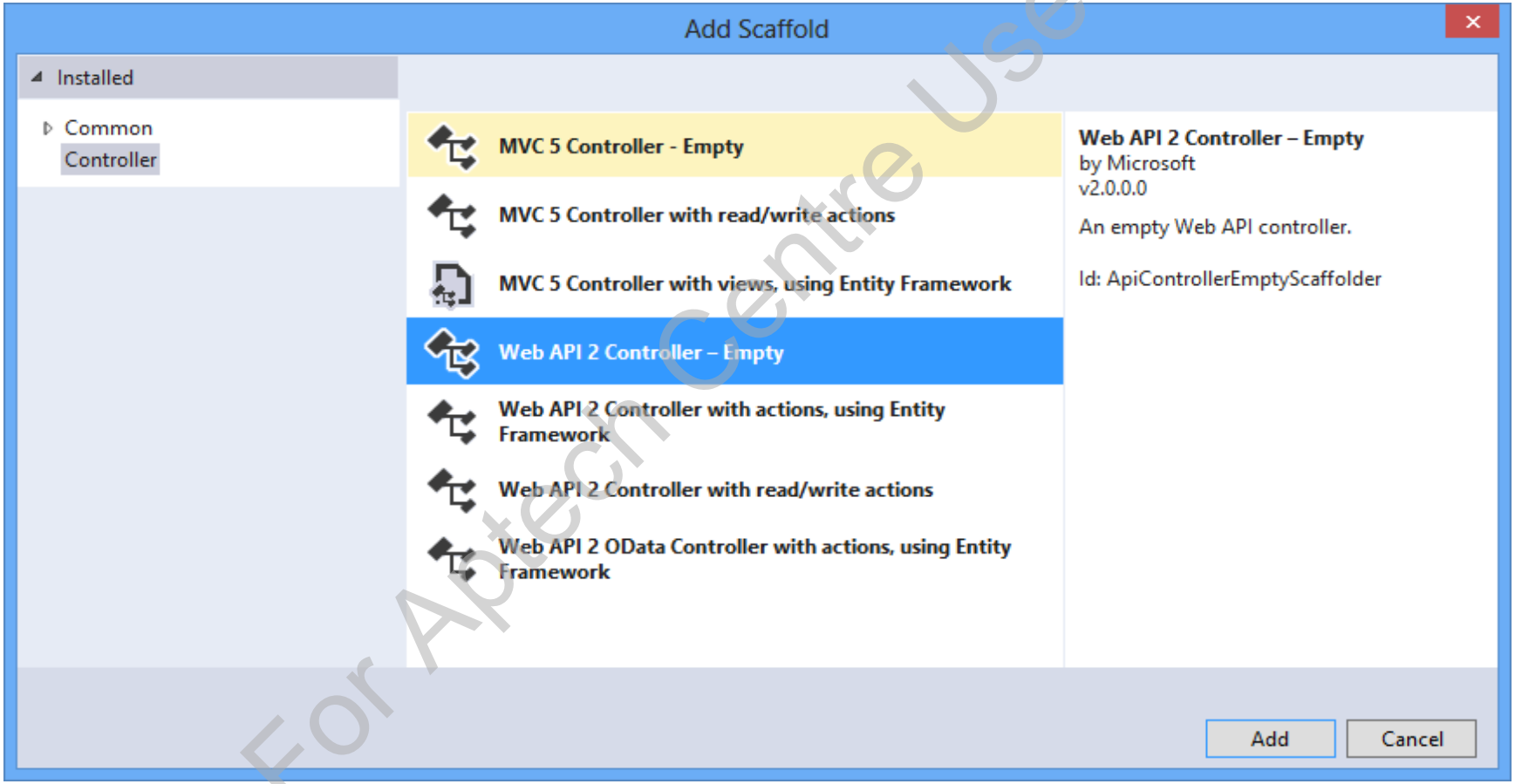
- Right-click the **Controllers** folder in the Solution Explorer window and select **Add → Controller** from the context menu that appears. The **Add Scaffold** dialog box is displayed.

Step 2

- Select the **Web API 2 Controller – Empty** template in the **Add Scaffold** dialog box.

Adding an ASP.NET Web API Controller 3-9

- Following figure shows the **Add Scaffold** dialog box:



Adding an ASP.NET Web API Controller 4-9

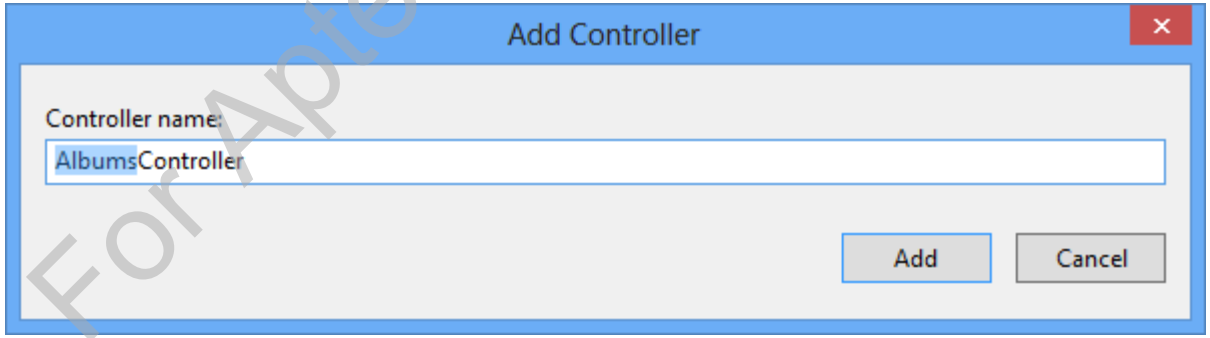
Step 3

- Click **Add**. The **Add Controller** dialog box is displayed.

Step 4

- Type **AlbumsController** in the **Controller** name field.

❑ Following figure shows the **Add Controller** dialog box:



Adding an ASP.NET Web API Controller 5-9

Step 5

- Click **Add**. The Code Editor displays the newly created `AlbumsController` controller class.

Step 6

- In the `AlbumsController` controller class, add the HTTP methods to retrieve, add, update, and delete albums that the `AlbumRepository` object contains.

Adding an ASP.NET Web API Controller 6-9

- Following code shows the `AlbumsController` class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using WebAPIDemo.Models;

namespace WebAPIDemo.Controllers
{
  public class AlbumsController : ApiController
  {
    static readonlyIAlbumRepositoryalbumRepository = new
    AlbumRepository();
    public IEnumerable<Album> Get()
    {
```

Adding an ASP.NET Web API Controller 7-9

```
    return albumRepository.GetAll();
}

public Album Get(int id)
{
    Album album = albumRepository.Get(id);
    if (album == null)
    {
    throw new HttpResponseException(HttpStatusCode.NotFound);
    }
    return album;
}
public IEnumerable<Album> GetAlbumByGenre(string genre)
{
    return albumRepository.GetAll().Where(
    p =>string.Equals(p.Genre,genre,
      StringComparison.OrdinalIgnoreCase ));
}
```

Adding an ASP.NET Web API Controller 8-9

```
    public string Post(Album album)
    {
          album = albumRepository.Add(album);
          return "Album added successfully";
    }
    public void Put(int id, Album album)
    {
         album.Id = id;
         albumRepository.Update(album);
    }
    public void Delete(int id)
    {
        Album album = albumRepository.Get(id);
        albumRepository.Remove(id);
    }
  }
}
```

Adding an ASP.NET Web API Controller 9-9

- In this code:
  - The `AlbumsController` class extends the `ApiController` class.
  - The `Get()` method accesses the album repository to return all albums as an `IEnumerable<Album>` object. The `Get(int id)` method accesses the album repository to return an album with the specified Id as an `Album` object.
  - The `GetAlbumByGenre(string genre)` method returns all albums of the specified genre as an `IEnumerable<Album>` object.
  - The `Post(Album album)` method adds the `Album` object passed as parameter to the album repository. The `Put(int id, Album album)` method updates an album in the album repository based on the specified id.
  - The `Delete(int id)` method deletes an album from the album repository based on the specified id.

Defining Routes 1-3

- After creating the ASP.NET Web API controller, register it with the ASP.NET routing Framework.
- When the Web API application receives a request, the routing framework tries to match the Uniform Resource Identifier (URI) against one of the route templates defined in the `WebApiConfig.cs` file.
- If no route matches, the client receives a 404 error.
- When an ASP.NET Web API application is created 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 Routes 2-3

- Following code shows the default route configuration of the ASP.NET Web API application:

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

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

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

- 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
- Use the `RouteTable.MapHttpRoute()` method to configure additional routes of an ASP.NET Web API application.

Defining Routes 3-3

- To configure a new route in the `WebApiConfig.cs` file, refer the following code:

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

- In this code, a route named, `AlbumWebApiRoute` is created with the route pattern `album/{controller}/{id}`.

Assessing the Application 1-2

- After creating the controller class and configuring the routing of the ASP.NET Web API application, access it from a browser using the following steps:

Step 1:

Click **Debug** → **Start Without Debugging**.

Step 2:

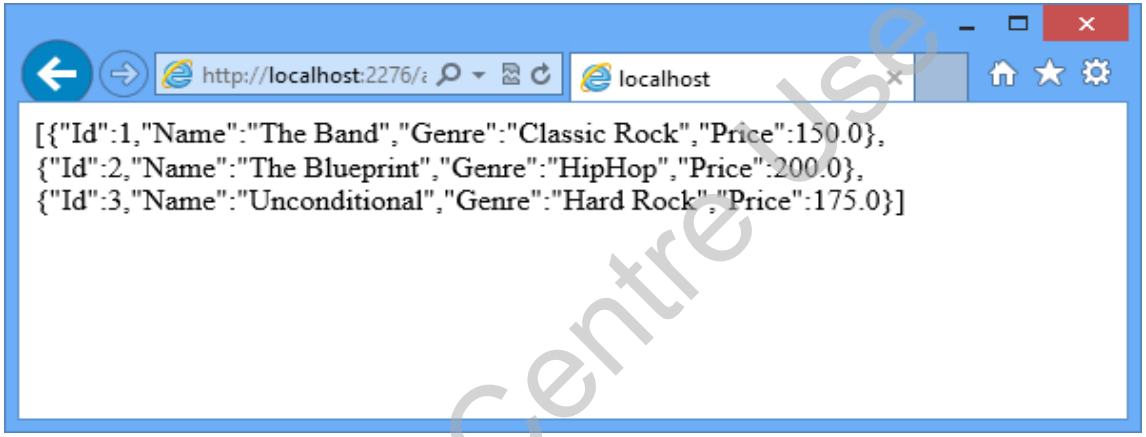
Type the following URL in the address bar of the browser:

```
http://localhost:2276/album/albums
```

This URL retrieves details of all the albums.

Assessing the Application 2-2

- The browser displays the album details as shown in the following figure:



- To access a specific album, based on the Id, type the following URL in the address bar of the browser:

```
http://localhost:2276/album/albums/1
```

Securing an ASP.NET Web API Service 1-2

- Securing an ASP.NET Web API service involves two key security concepts:

Authentication

- This is the process of identifying an individual, usually based on the username and password provided by the user.
- An ASP.NET Web API service does not provide any new authentication mechanism since that is already taken care of by the Web application that the Web API service is part of.

Authorization

- This is the process of either allowing or denying an authenticated user access to a restricted resource. For example, consider an ASP.NET Web API controller that contains two GET methods: `GetNewsHeadlines()` and `GetNews()` that allow a user to retrieve news headlines and complete news respectively.
- Authorization can be enforced using the **Authorize** action filter.

Securing an ASP.NET Web API Service 2-2

- Following code shows using the **Authorize** filter:

```
[Authorize]
public Album GetNews(id)
{
   News news= newsRepository.GetNews(id);
if (news == null)
     {
throw new
HttpResponseException(HttpStatusCode.NotFound);
     }
return news;
}
```

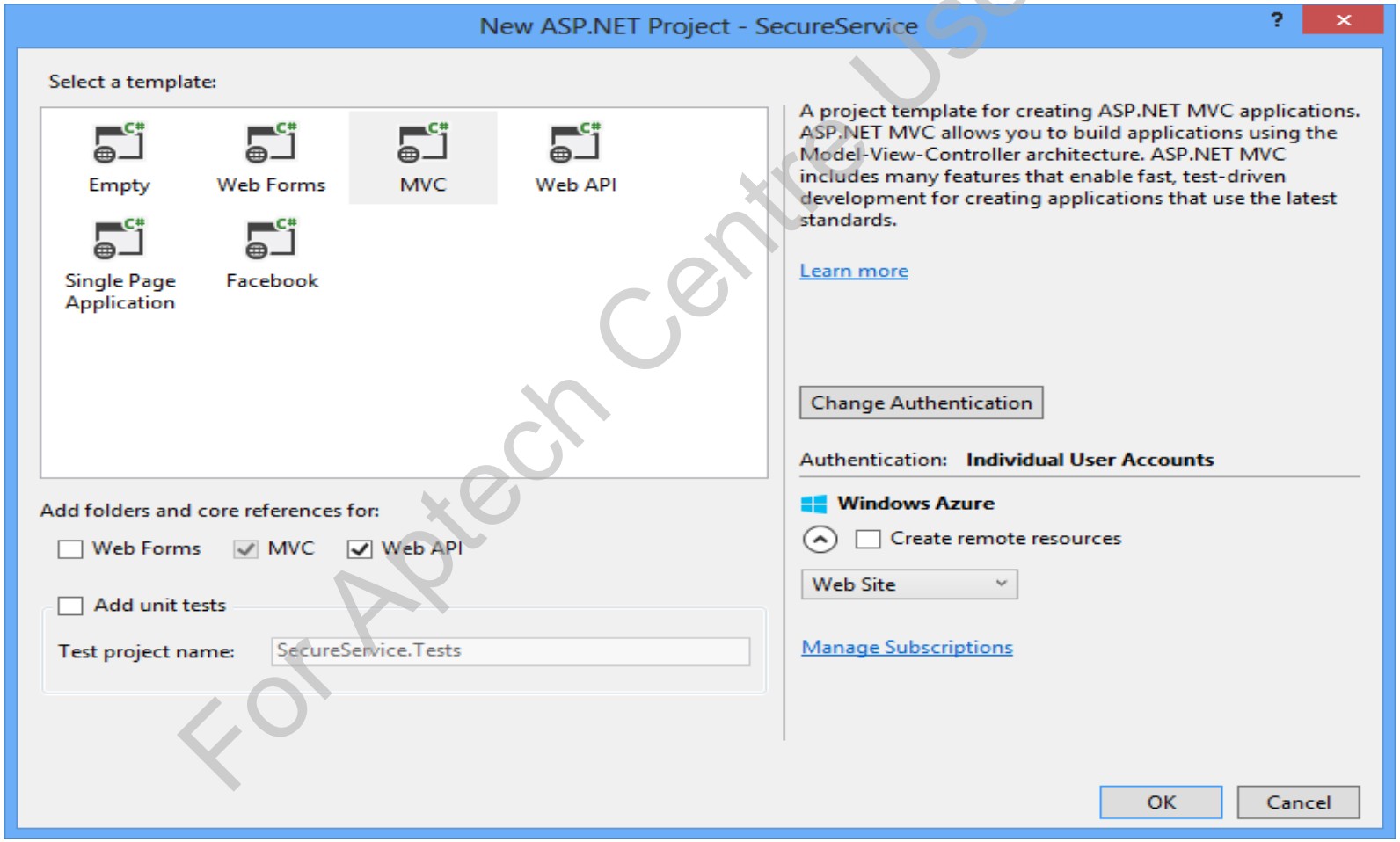
Authentication and Authorization in an ASP.NET Web API Service 1-5

- Following are the steps to secure an ASP.NET Web API service:

Step 1	• Open Visual Studio 2013.
Step 2	• Select **File** → **New** → **Project**. The **New Project** dialog box is displayed.
Step 3	• Select **Web** under the **Installed** section and then select the **ASP.NET Web Application** template.
Step 4	• In the **Name** text field, type **SecureService** and click **OK**. The **New ASP.NET Project – Secure Service** dialog box is displayed.
Step 5	• From the **Select a template** section, select **MVC** and then, select the **Web API** check box.

Authentication and Authorization in an ASP.NET Web API Service 2-5

- Following figure shows the **New ASP.NET Project – Secure Service** dialog box:



Authentication and Authorization in an ASP.NET Web API Service 3-5

Step 6

- Click **OK**. Visual Studio 2013 creates the project and adds an **AccountController** controller class to the project. The **AccountController** class is responsible for implementing authentication in the application.

Step 7

- In the Solution Explorer window, right-click the **Controllers** folder and select **Add → Web API Controller Class (v2)**. The **Specify Name for Item** dialog box is displayed.

Step 8

- Type **ValuesController** in the **Item name** text box.

Step 9

- Click **OK**. Visual Studio 2013 adds an API controller named **ValuesController** in the **Controllers** directory.

Step 10

- To test the Web API Service, select **Debug → Start Without Debugging**. The browser displays the home page of the application.

Authentication and Authorization in an ASP.NET Web API Service 4-5

Step 11
- Type the following URL in the address bar of the browser to access the ASP.NET Web API service: `http://localhost:3476/api/values`

Step 12
- Close the browser.

Step 13
- In Visual Studio 2013, double-click **ValuesController.cs** file under the **Controllers** directory to open in Code Editor.

Step 14
- Add the **Authorize** attribute to the Get() method.

Step 15
- To test whether the Get() method of the Web API Service requires authorization, select **Debug → Start** without Debugging. The browser displays the home page of the application.

Step 16
- Type the following URL at the address bar of the browser to access the ASP.NET Web API service: `http://localhost:3476/api/values`

Authentication and Authorization in an ASP.NET Web API Service 5-5

- Register with the application and then log in to access the Web API service.
- Steps to register with the application are:

Step 1

- Click the **Register** hyperlink. The **Register** page is displayed.

Step 2

- Type **Andy** in the **User name** text box and **pass@123** in the **Password** and **Confirm Password** text boxes.

Step 3

- Click **Register**. Once you have registered with the application, type the following URL to access the ASP.NET Web API service:
  `http://localhost:3476/api/values`

Summary

- ASP.NET Web API is a .NET Framework technology to create Web services for different types of clients.
- ASP.NET Web API is an implementation of RESTful service that removes the complexities of creating Web services by relying purely on HTTP.
- HTTP is the standard protocol for communication over the Web.
- REST is a service architecture where each request URL is unique and points to a specific resource.
- Media type is a standard to identify the type of data being exchanged over the Internet between the browser and the server.
- In an ASP.NET Web API service, the routing framework is responsible to match request URI against one of the route templates defined in the WebApiConfig.cs file.
- Authentication and authorization are two mechanisms to secure ASP.NET Web API services.