Web API Services

Session 4

Learning Objectives

- Web API and Cloud Computing
- 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

Characteristics of Distributed Applications

Distributed computing is a fundamental concept in modern technology that involves the use of multiple interconnected computers or nodes to work together as a unified system.

It is employed for various reasons, including:

- Scalability
- Fault Tolerance
- Performance Optimization
- Geographic Distribution
- Cost Efficiency
- Flexibility and Modularity
- Data Processing and Analytics

Layers of Distributed Applications

- Distributed applications typically consist of multiple logical layers that work together to provide the desired functionality.
- These layers can vary depending on the specific architecture and design choices.
- Some common logical layers found in distributed applications:
  1. Presentation Layer / UI Layer / View
  2. Application Layer / Business Logic Layer / Controller
  3. Data Layer / Model
  4. Communication Layer
  5. Integration Layer
  6. Infrastructure Layer

Storage Technologies

- Data can be represented in different models:
  - Relational (databases, tables and columns)
  - Hierarchical (XML, JSON)
  - Object oriented (entities in code)
- There are many types of data stores:
  - Relational databases
  - File-systems and Distributed file-systems
  - Distributed caches
  - No SQL databases/Azure Cosmos DB
  - Cloud-storage
  - In-memory stores

.NET Data Access Technologies

- .NET Core has a wide range of data access technologies:
  - System.IO
    - contains all the infrastructure required to access data persisted on a file system.
  - ADO.NET
    - is the basic SQL Data-Access technology.
  - Entity Framework
    - EF is an Object Relational Mapper (ORM) infrastructure. Applications use the object-oriented approach to represent data entities.
  - In-Memory Cache (System.Web)
    - ASP.NET Core introduces a powerful in-memory cache that can be used by any .NET Core application.
  - Microsoft Azure Redis Cache
    - It is a distributed in-memory store for .NET Core application, which negates the memory size limitation of in-memory caches by distributing cache objects over several servers.
- HTTP can also be used for accessing data:
  - Windows Azure Storage
- Declarative data queries written in C# using LINQ

Service Technology

- Services constitute a layer in application architecture, which exposes business logic capabilities to other application components to improve component modularity and reusability.
- Services are the core of distributed applications providing access to data and making it possible for users to interact with other applications.
- Separation of layers helps to ensure the existence of Single Responsibility Principle (SRP), making it possible to test each layer as an independent portion.

Web API

- A Web API, is a type of software interface that allows different software applications to communicate with each other over the internet.
- Web APIs use HTTP requests and responses to send and receive data, and typically use JSON or XML formats to structure the data.

Why Web API

- Web APIs can be used for:
  - **Integrating different software systems:** Web APIs allow different software applications to share data and functionality, making it easier to build complex systems.
  - **Creating mobile apps:** Mobile app developers can use Web APIs to access data and functionality from web services, making it easier to build cross-platform apps.
  - **Building web-based applications:** Web APIs can be used to build web-based applications that can access data and functionality from other systems.
  - **Developing IoT applications:** Web APIs can be used to connect IoT devices to the internet and allow them to communicate with other systems.

Web API and cloud computing

- Web API is widely used in cloud computing to provide a standardized interface for accessing cloud-based resources, such as storage, databases, and other services.
- The use of Web API in cloud computing allows developers to build powerful, scalable, and flexible applications that can be accessed from anywhere in the world, using a standardized interface that is easy to use and integrate with other services and resources.

Web API and cloud computing [Cont.…]

- **Cloud-based applications:** Web APIs can be used to expose functionality and data in cloud-based applications, making it easier for developers to build and deploy cloud-based applications that can be accessed from anywhere in the world.
- **Integration with third-party services:** Web APIs can be used to integrate cloud-based services with other third-party services, making it easier to develop applications that use a wide range of services and resources.
- **Data storage and retrieval:** Web APIs can be used to access data stored in cloud-based databases or storage services, making it easier for developers to build applications that store and retrieve data from the cloud.
- **Scalability and flexibility:** Using cloud-based resources in combination with Web APIs allows applications to scale up or down as needed, without requiring additional hardware or infrastructure.
- **Cross-platform compatibility:** Web APIs are platform-agnostic, meaning they can be used to build applications that can run on any device or platform with internet access.

SaaS and Web API

- Implementing Software as a Service (SaaS) using Web API involves building a cloud-based application that exposes its functionality and data through a set of standardized protocols and interfaces.
- Implementing SaaS using Web API involves building a cloud-based application that can be accessed by other software applications over the internet, providing a flexible, scalable, and cost-effective way to deliver software applications to users.

PaaS and Web API

- Implementing Platform as a Service (PaaS) involves **building** a cloud-based platform that provides developers with the tools and resources they need to build and deploy their applications.
- Implementing PaaS involves building a cloud-based platform that provides developers with the tools and resources they need to **build and deploy** their applications, providing a flexible, scalable, and cost-effective way to develop and deploy software applications.

IaaS and Web API

- Implementing Infrastructure as a Service (IaaS) using Web API involves **building a cloud-based infrastructure** that provides users with access to computing resources such as servers, storage, and networking resources.
- Implementing IaaS using Web API involves building a cloud-based infrastructure that can be accessed by other software applications over the internet, providing a flexible, scalable, and cost-effective way to access computing resources.

ASP.NET Web API

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

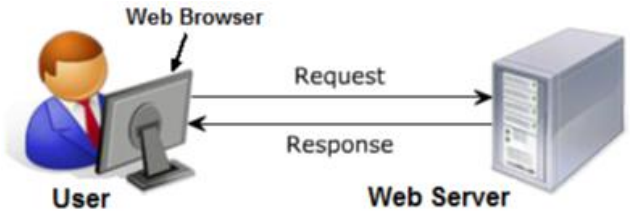
- ASP.NET Web API helps in handling client requests and sending back responses using content type, such as JavaScript Object Notation (JSON) or XML.

Cont….

- ASP.NET Web API is a framework for building HTTP services that can be consumed by a variety of clients, including web browsers, mobile apps, and IoT devices.
- The framework is built on top of the ASP.NET platform and provides a set of conventions and tools for building RESTful APIs that can handle HTTP requests and responses.

Evolution of ASP.NET Web API 1-6

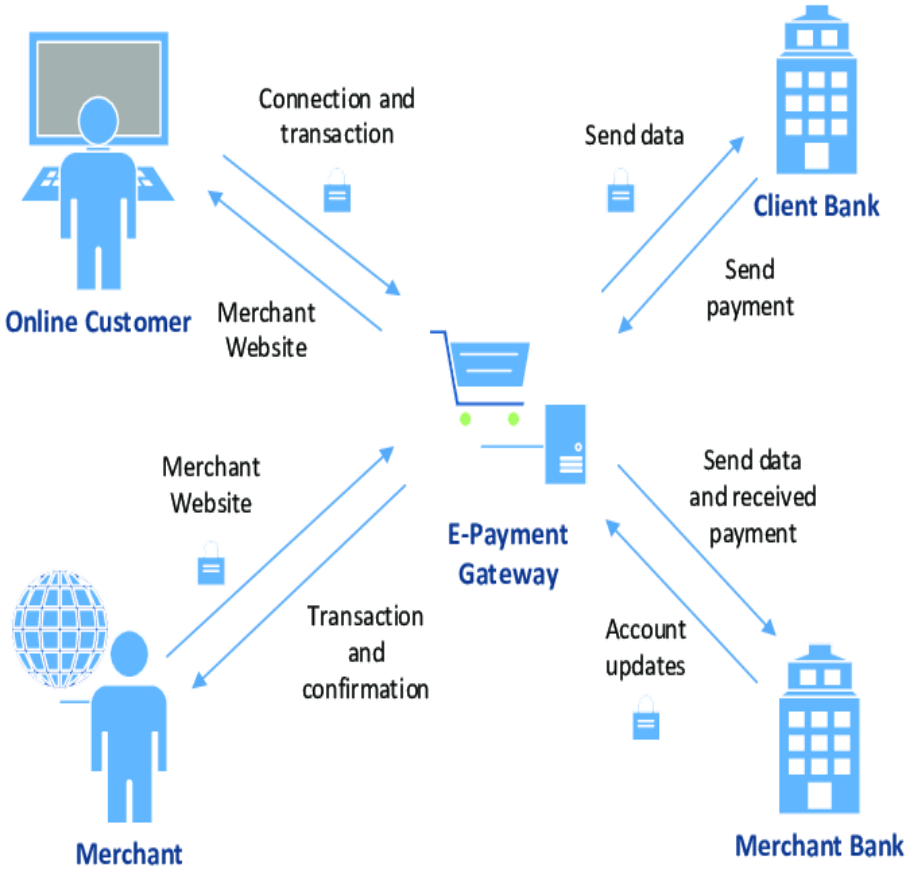
- ASP.NET Web API is the result of:
  - The gradual evolution of traditional Web services.
- Both Web services and Web Applications:
  - Receive, process, and return appropriate responses.
- Web Applications:
  - A request/response interchange comprises a user communicating with the application through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

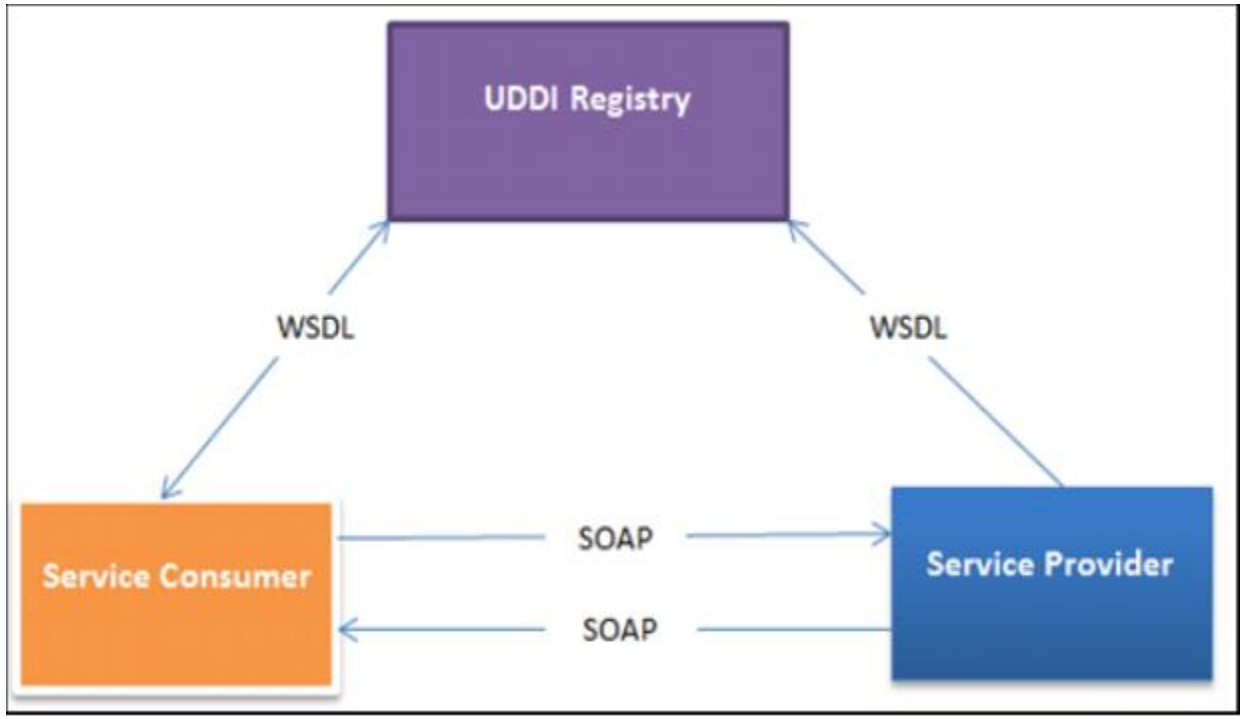
- To communicate with the payment gateway service provider, the shopping application and the provider both need to follow certain standards.
- These standards are also known as Web services standards.
- They include SOAP, Web Services Definition Language (WSDL), and Web Service Specifications (WS-*)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

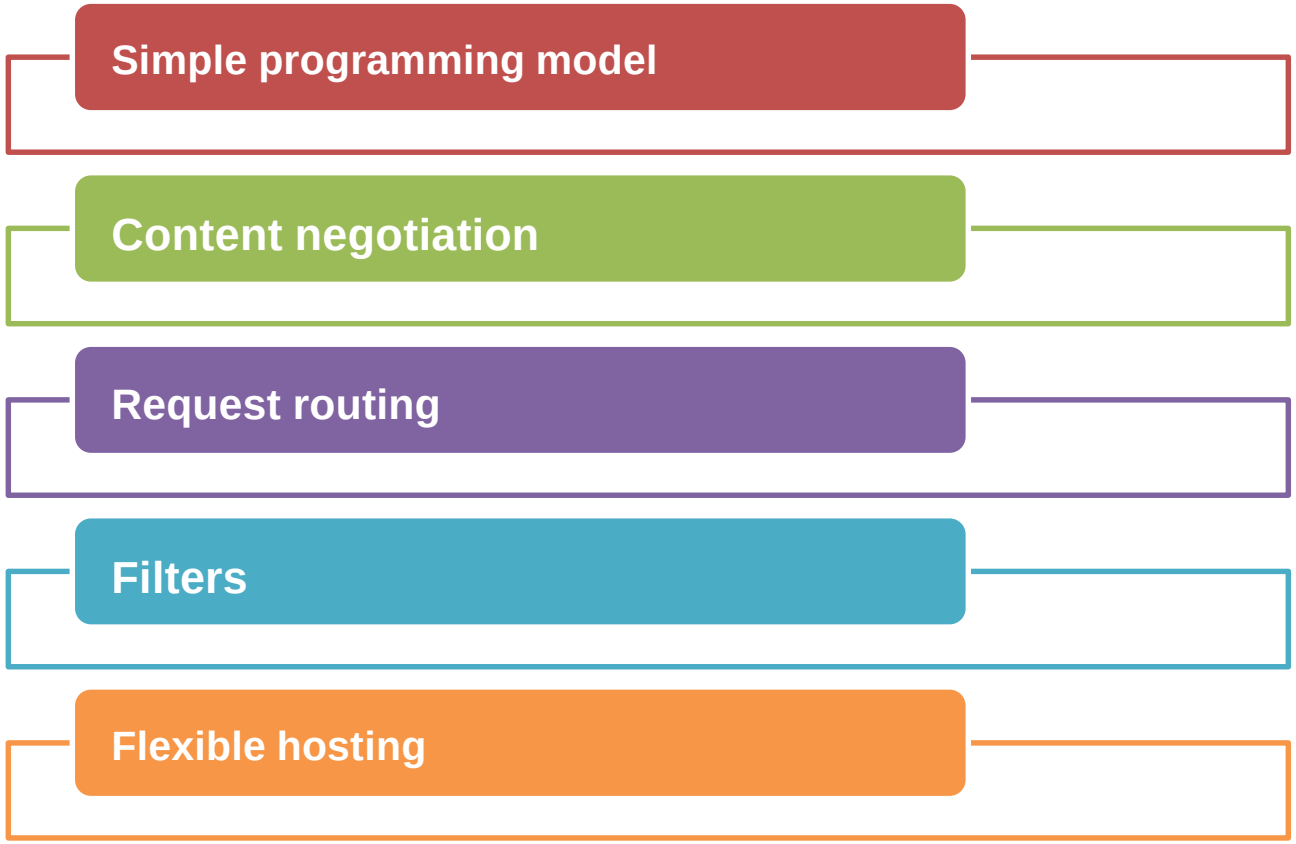


Evolution of ASP.NET Web API 6-6

- 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.
  - Defines a set of standards for securing web services, including encryption, digital signatures, and message integrity.
  - 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

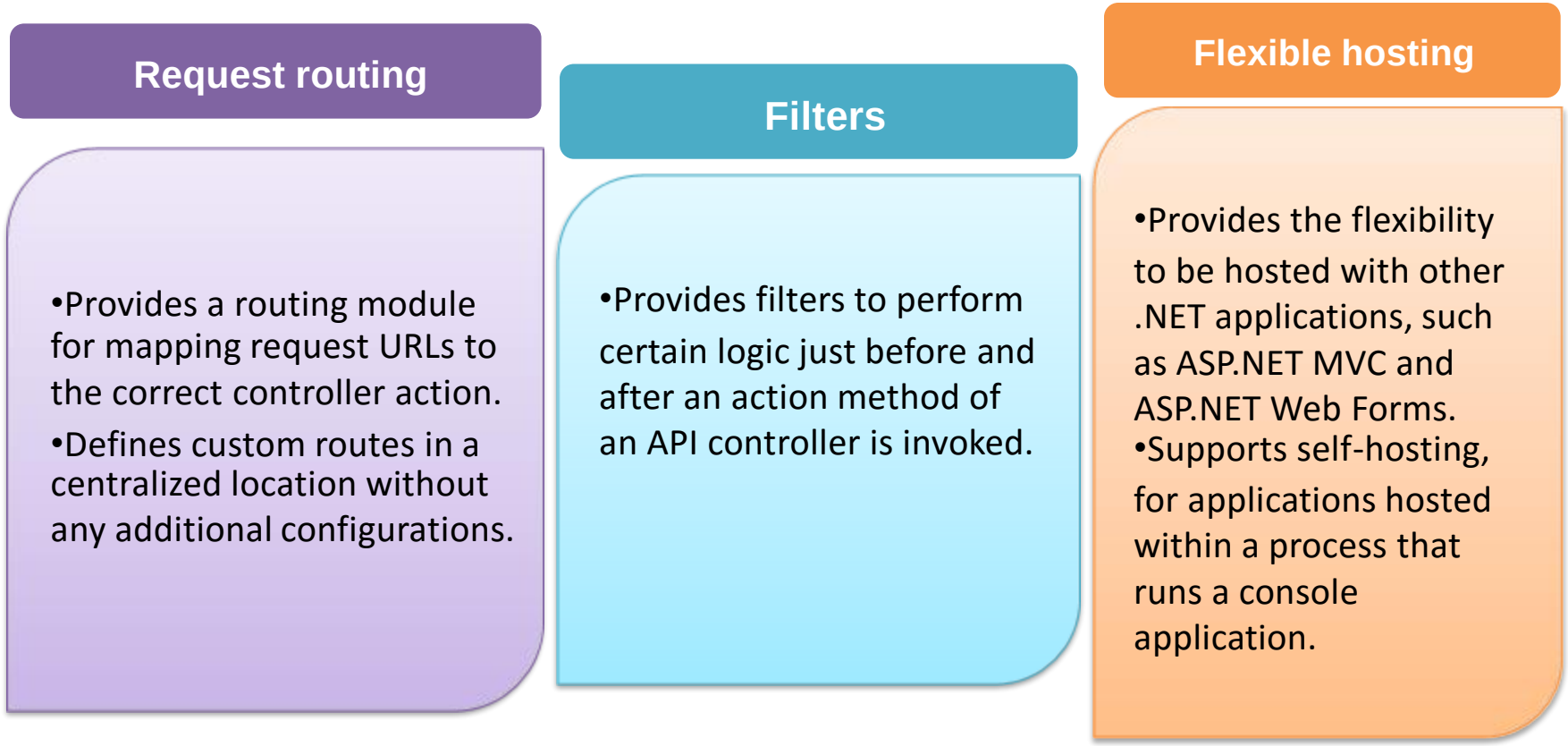
Simple programming model

The programming model of ASP.NET Web API is designed to be simple and easy to use, with a focus on building RESTful APIs that can handle HTTP requests and responses

Content negotiation

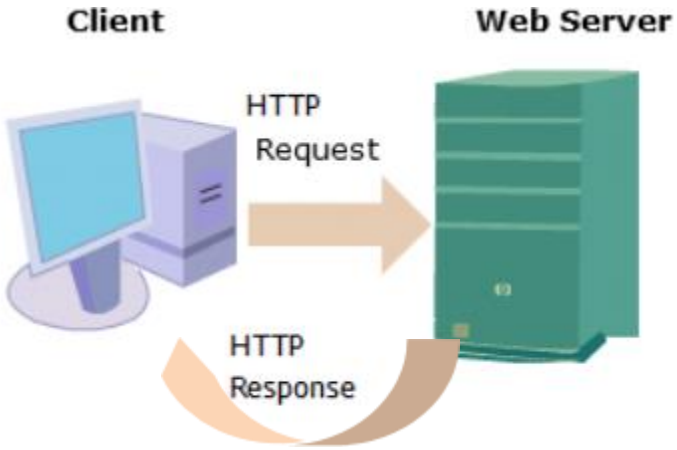
- Provides support to enable the client and server to negotiate the data format that the service returns as a response.
- Supports JSON, XML, and Form URL-encoded formats.
- Provides flexibility by extending the default data format by adding your own formatters.

ASP.NET Web API Features 3-3



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.

REST API

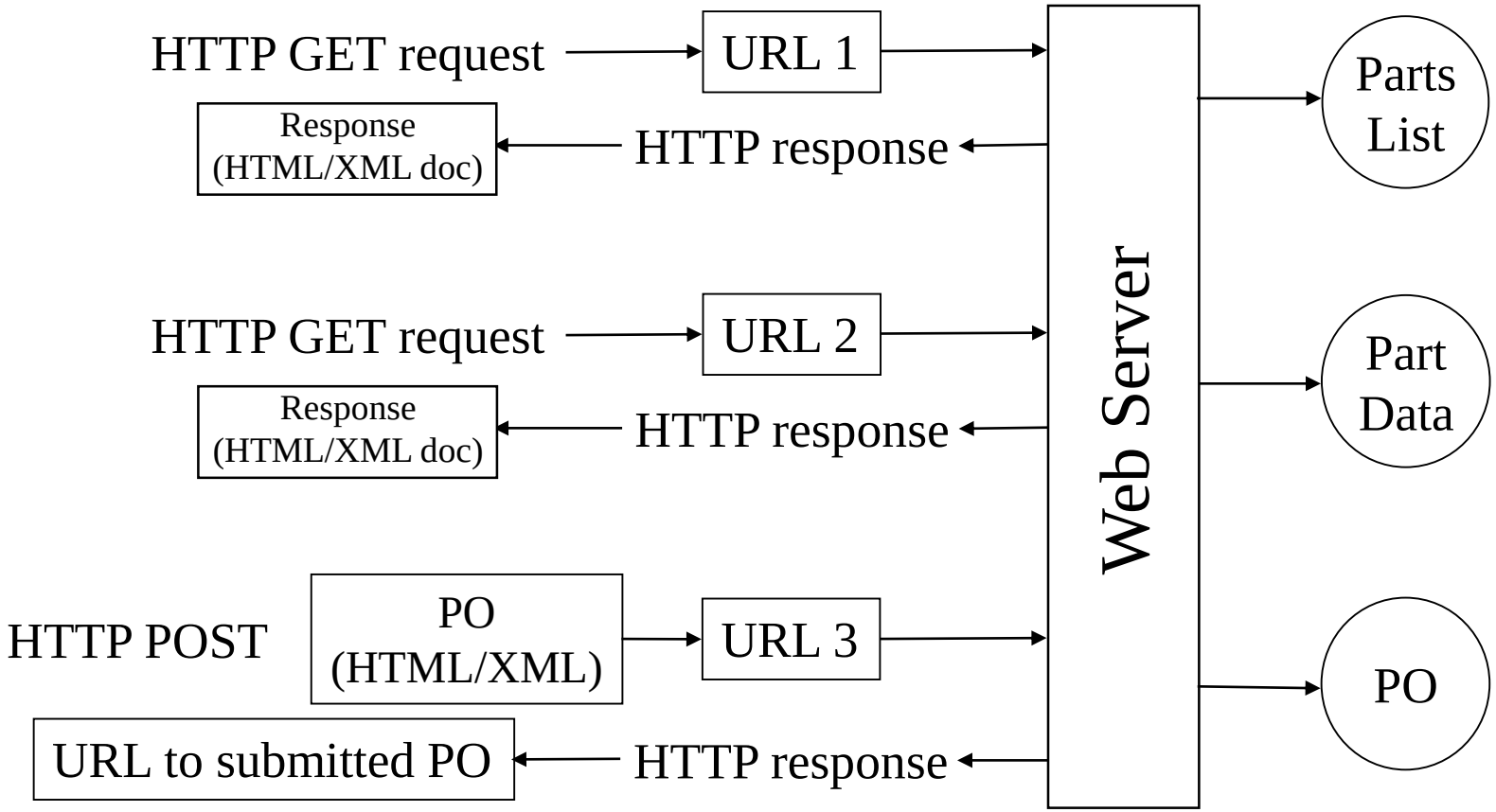
- “REST " was coined by Roy Fielding in his Ph.D. dissertation to describe a design pattern for implementing networked systems

Representational State Transfer

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

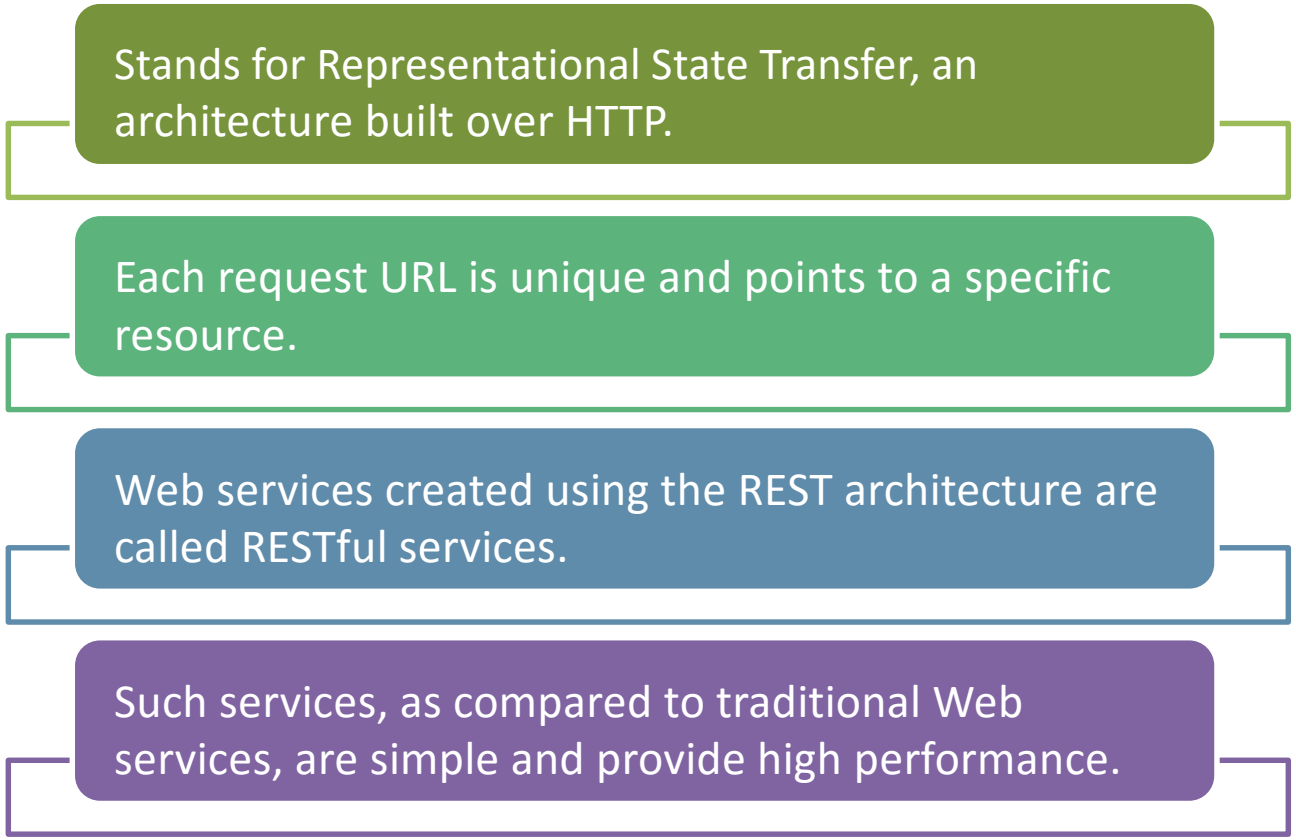
- Roy Fielding

The REST way of Designing the Web Services



REST 1-4

- REST:


Stands for Representational State Transfer, an architecture built over HTTP.
Each request URL is unique and points to a specific resource.
Web services created using the REST architecture are called RESTful services.
Such services, as compared to traditional Web services, are simple and provide high performance.

REST 2-4

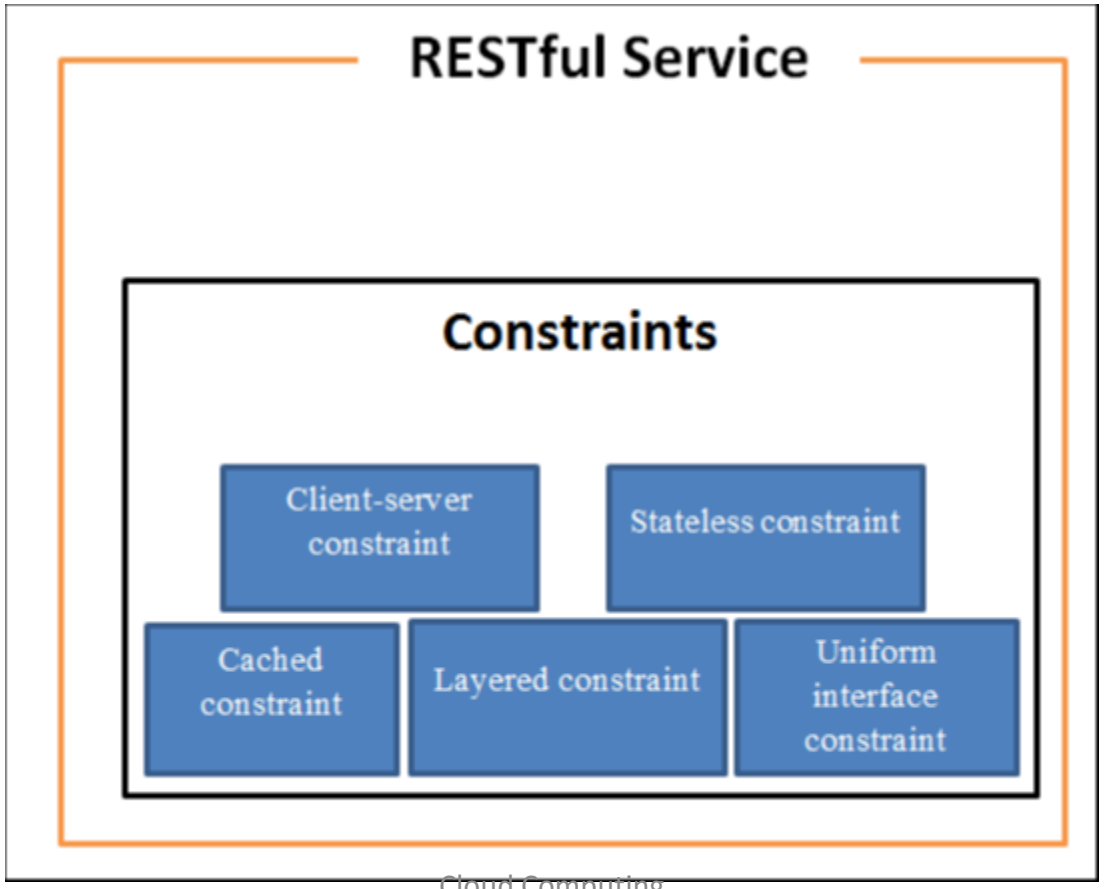
- To create RESTful services, you need to address the following basic design principles:

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.

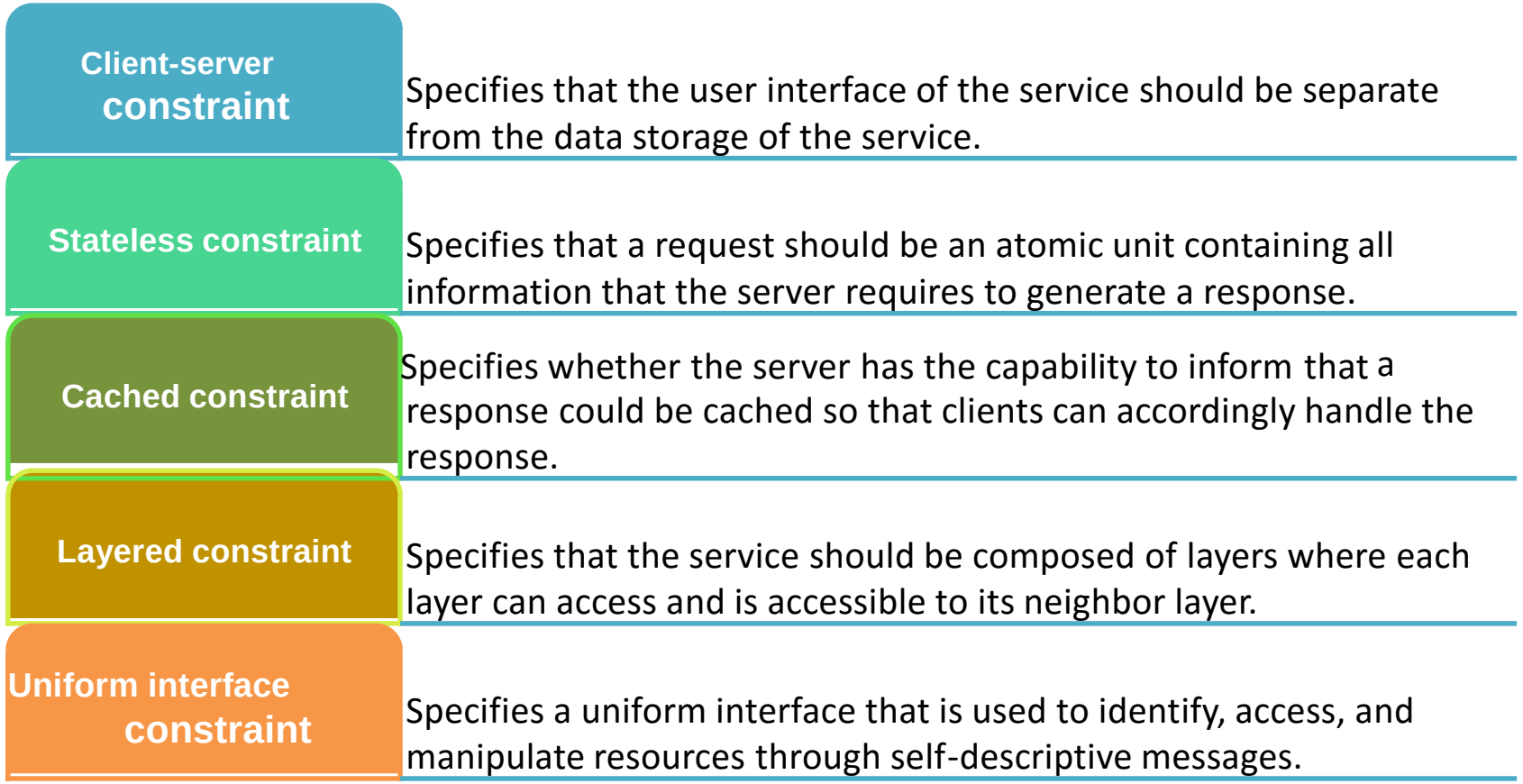
REST 3-4

- 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:



HTTP

Verb	URI or template	Use
POST	/order	Create a new order, and upon success, receive a Location header specifying the new order's URI.
GET	/order/{orderId}	Request the current state of the order specified by the URI.
PUT	/order/{orderId}	Update an order at the given URI with new information, providing the full representation.
DELETE	/order/{orderId}	Logically remove the order identified by the given URI.

Designing and implementing Web API Services

Designing and implementing Web API Services

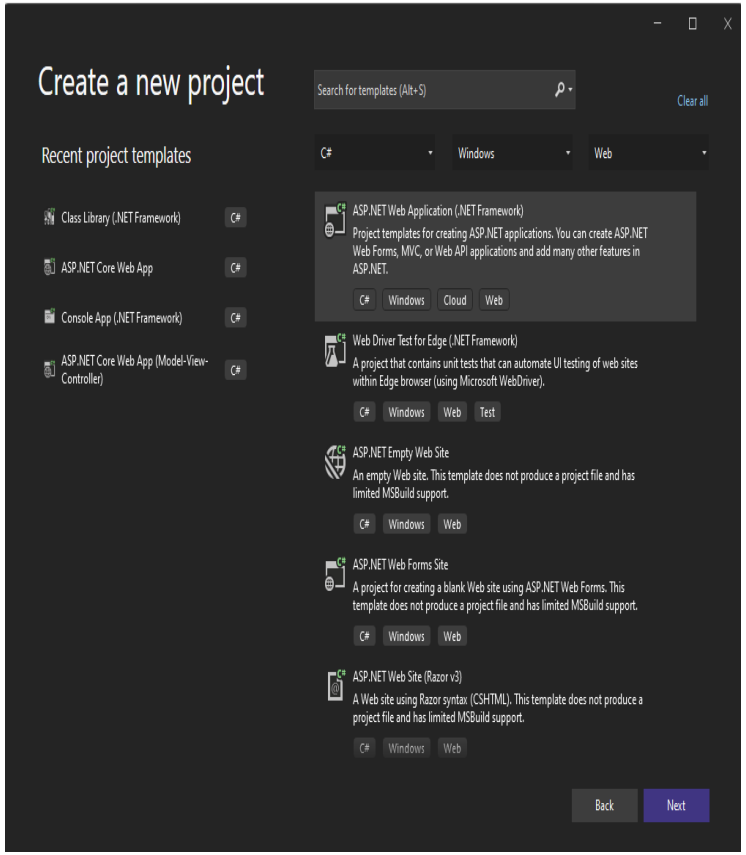
- Visual Studio provides templates and tools to simplify developing ASP.NET Web API services.

- When you create an ASP.NET Web API service in Visual Studio, 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

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

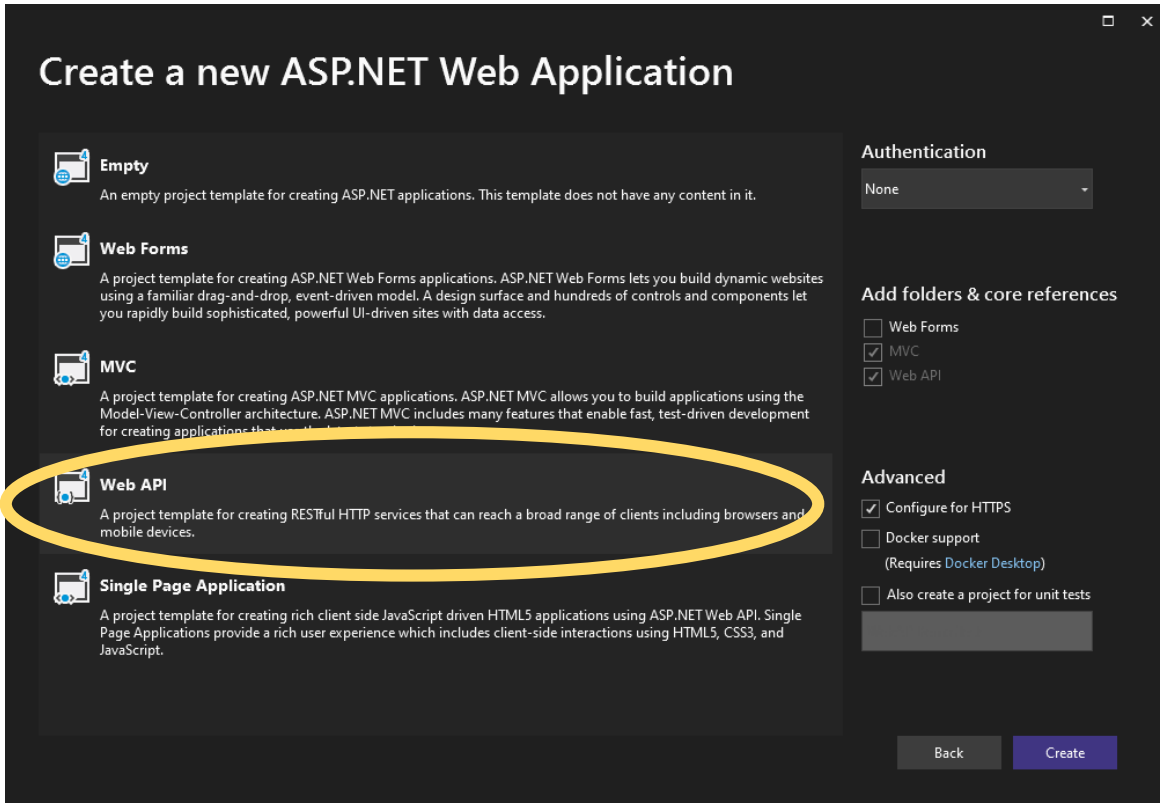
Step 1	• Open Visual Studio .
Step 2	• Click **File** → **New** → **Project** in Visual Studio .
Step 3	• In the **New Project** dialog box that appears, select **Web** under the **Installed** section, and then select the **ASP.NET Web Application** template.
Step 4	• Type **WebAPIDemo** in the **Name** 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

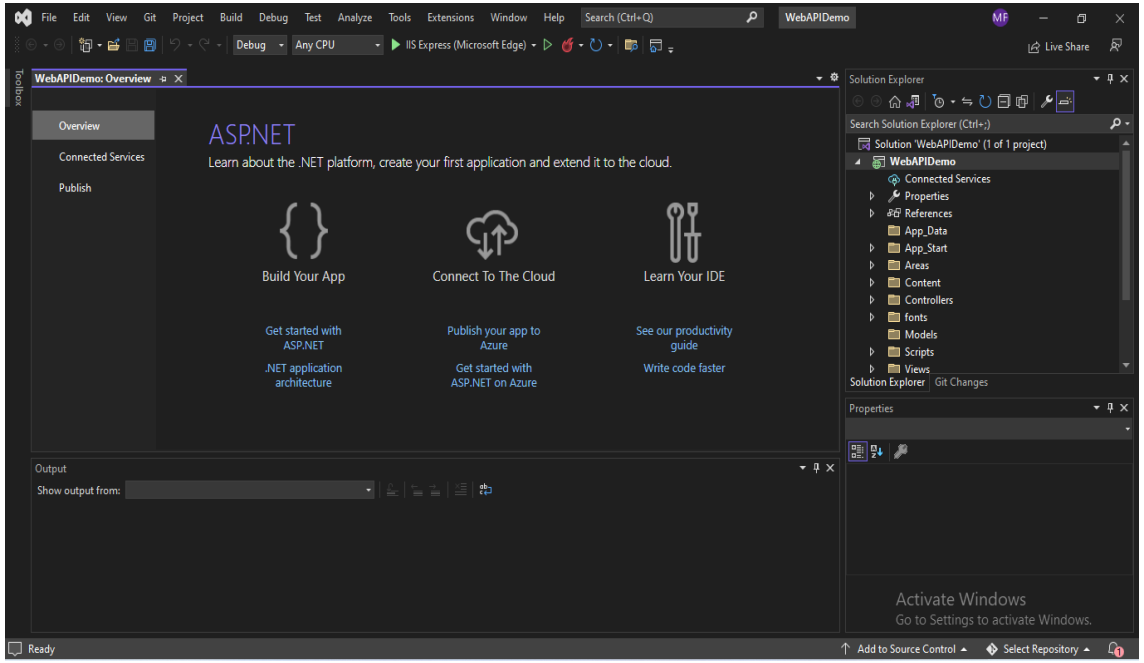
Step 7

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



Step 8

Click **Create**. Visual Studio 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

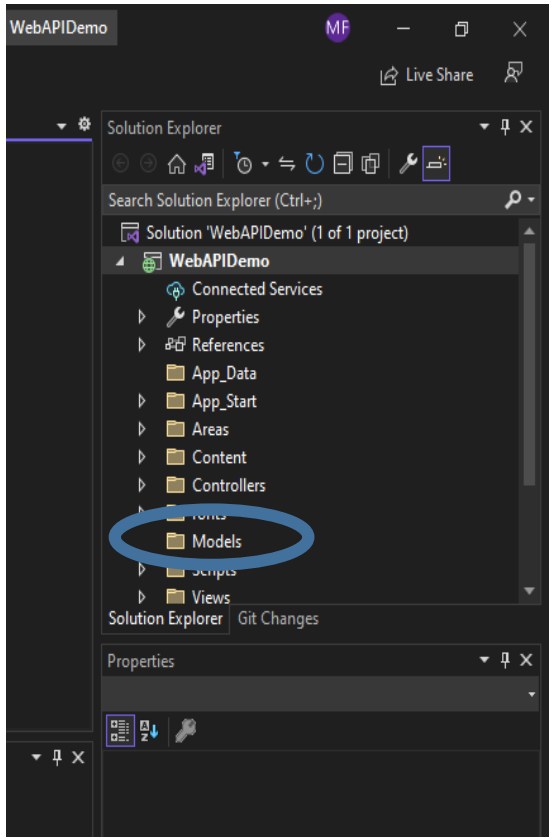
- To create a model in Visual Studio, you need to perform the following steps:

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.

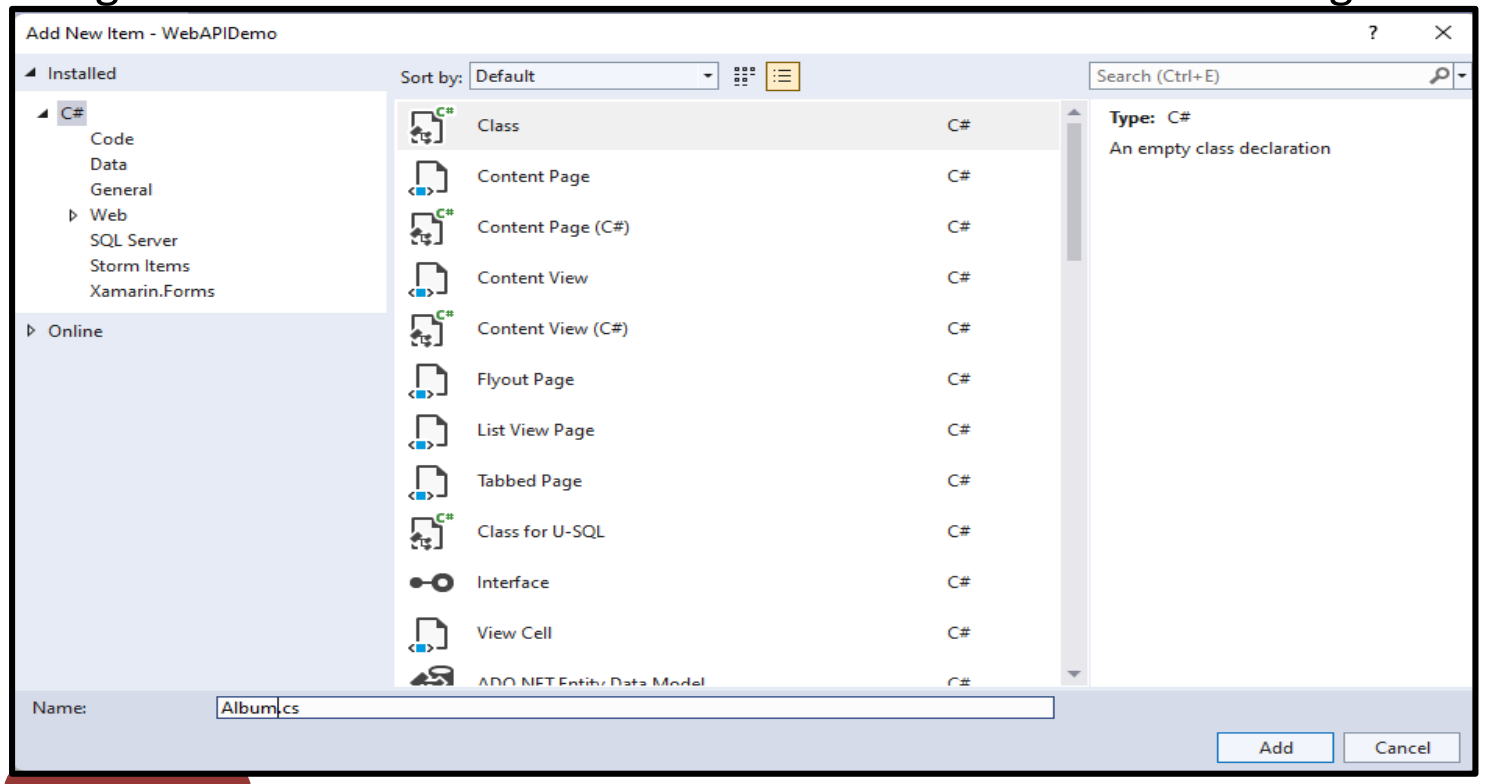
Step 2

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:

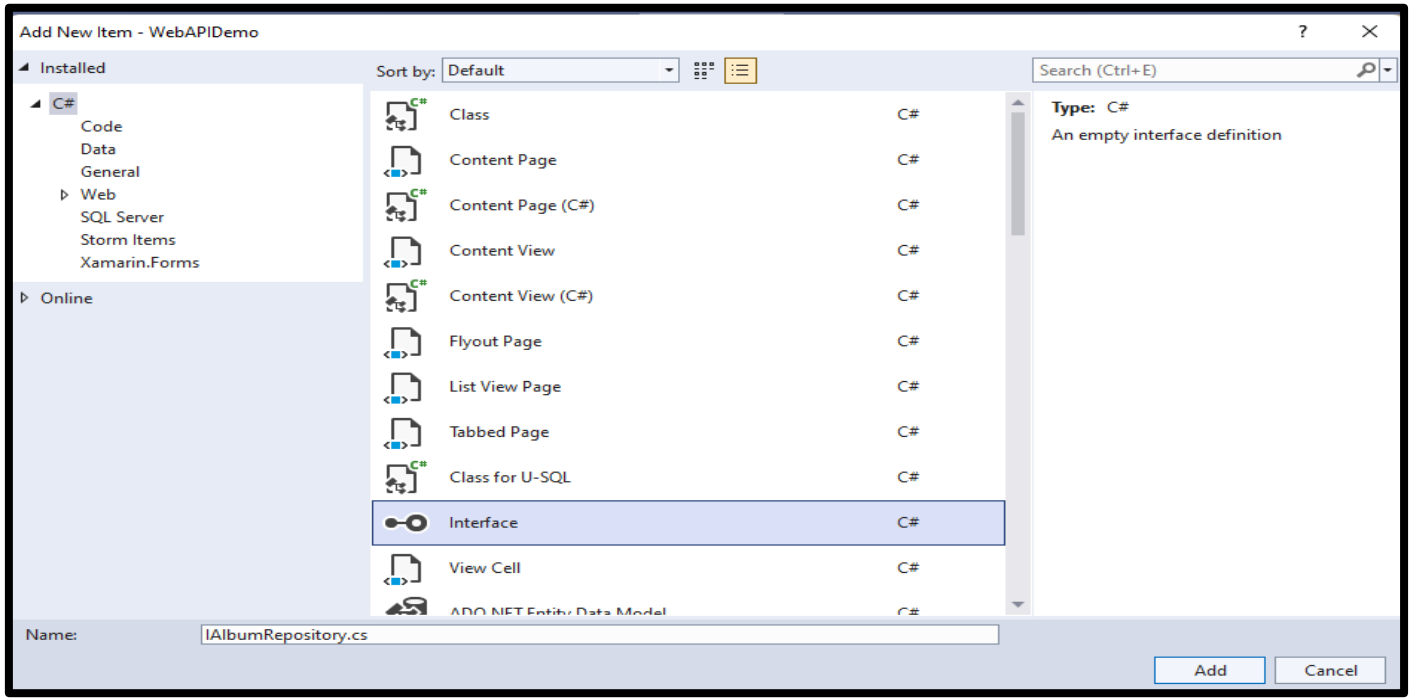
Step 1: Right-click the **Models** folder in Solution Explorer and select **Add → New Item**. The **WebAPIDemo** dialog box is displayed.

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 :

Step 1

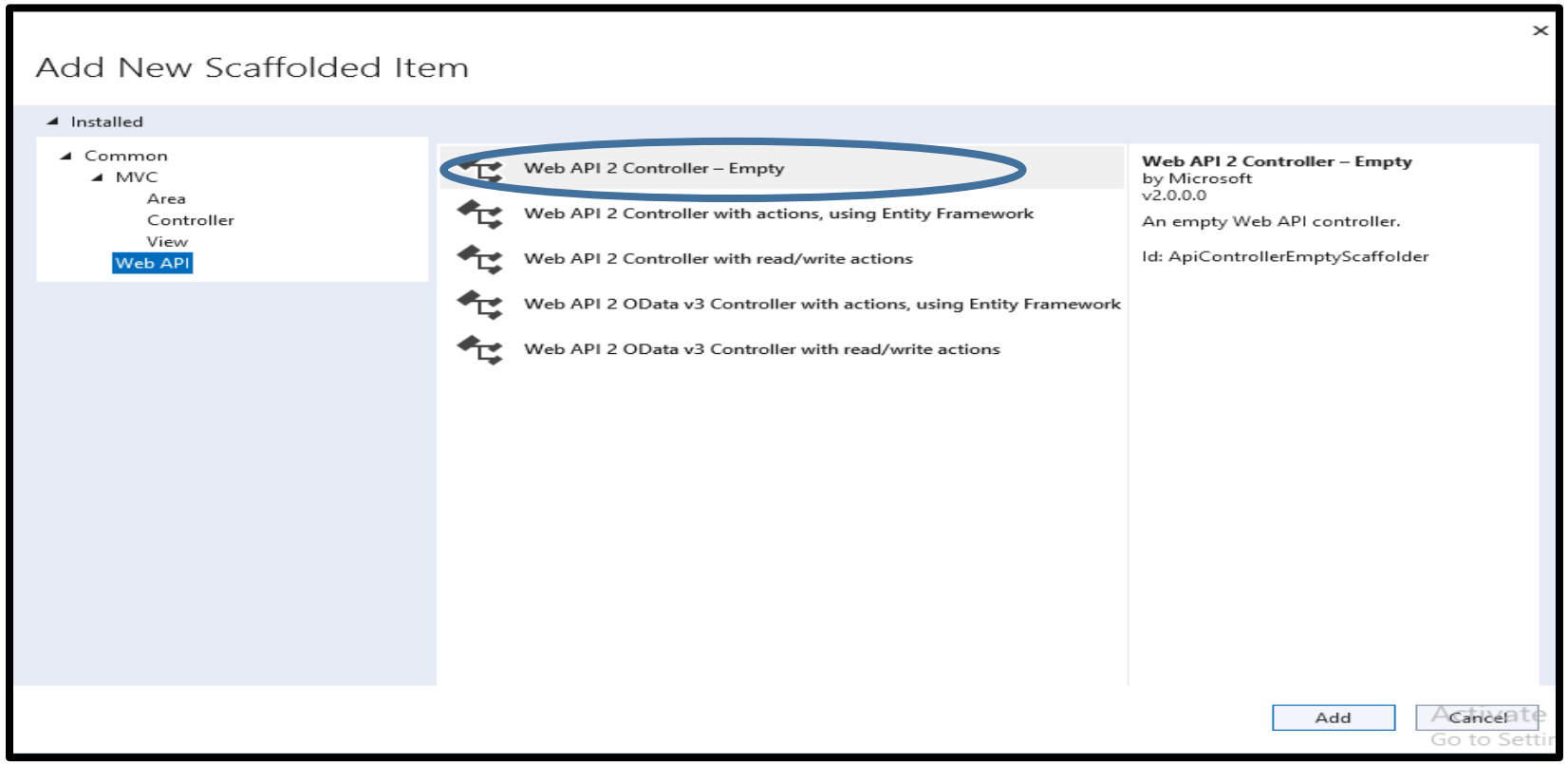
- Right-click the **Controllers** folder in the Solution Explorer window and select **Add → Controller** from the context menu that appears. The **Add Scaffold** dialog box is displayed.

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

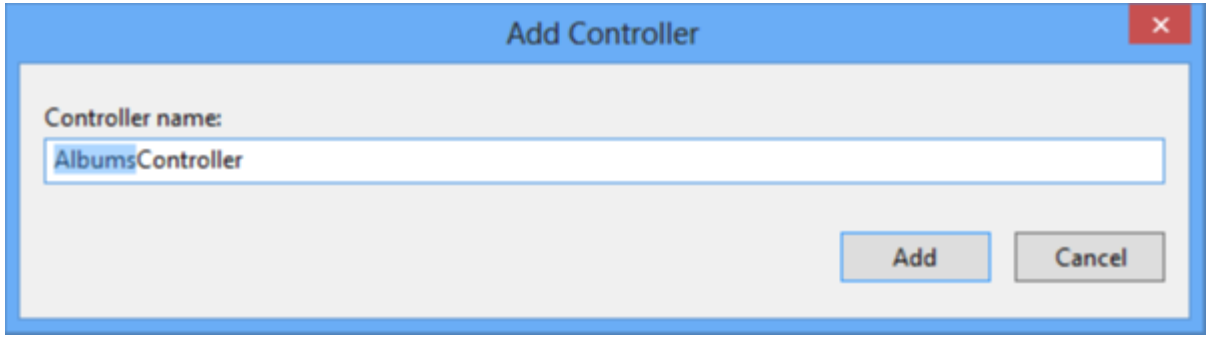
Step 3

- 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 readonly IAlbumRepository albumRepository = 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 ));
    }
```

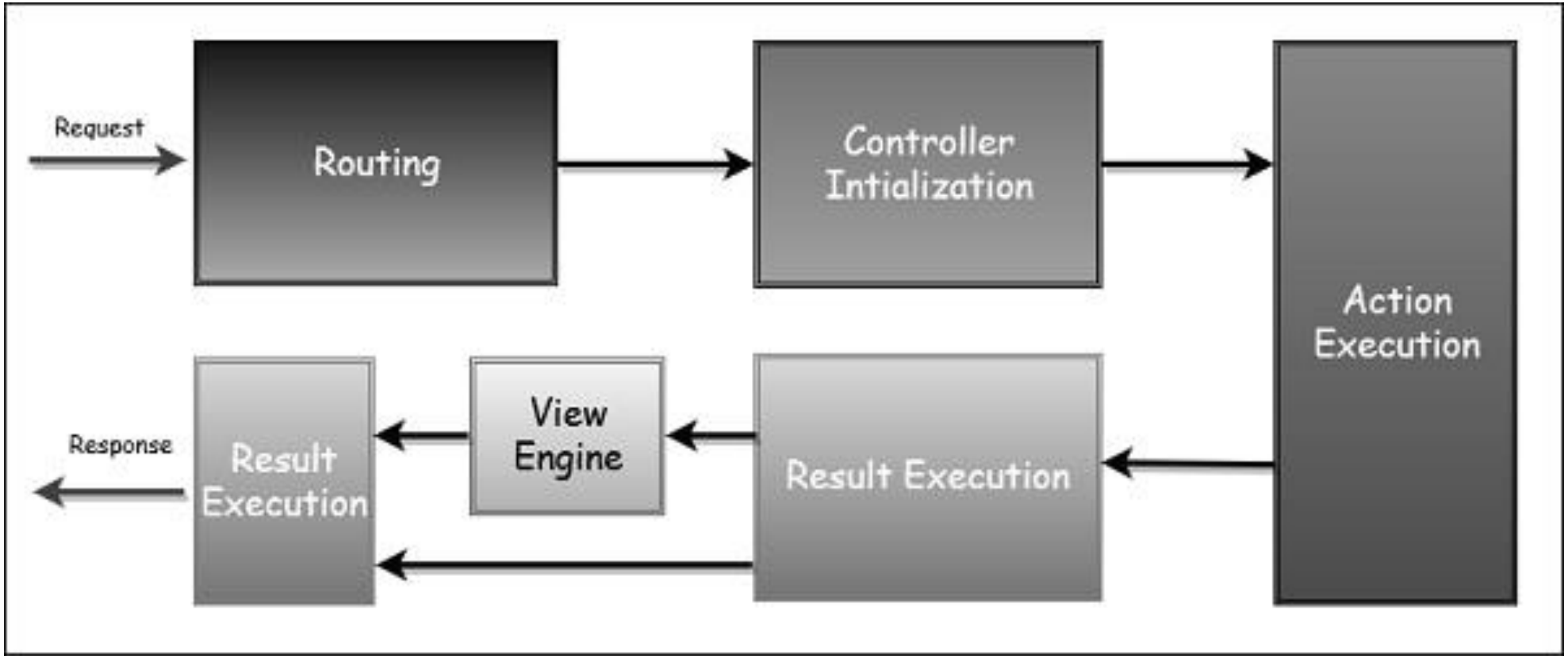
Adding an ASP.NET Web API Controller 8-9

```
    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

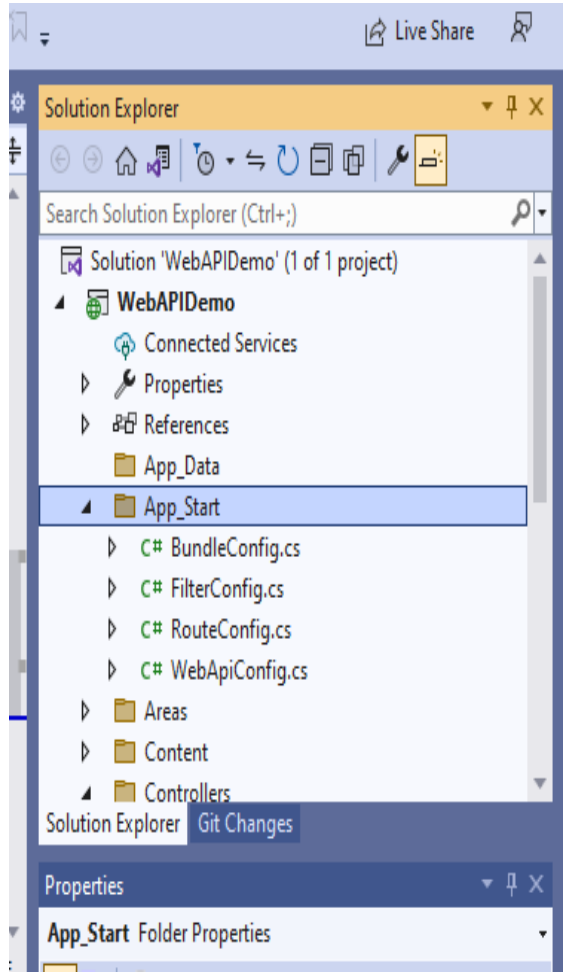
- In this code:
  - The `AlbumsController` class extends the `ApiController` class.
  - The `Get()` method accesses the album repository to return all albums as an `IEnumerable <Album>` object. The `Get(int id)` method accesses the album repository to return an album with the specified Id as an `Album` 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.

Steps that the components of the ASP.NET MVC Framework performs while handling an incoming request:



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 by default, the IDE configures the route of the application in the **`WebApiConfig.cs`** file under the **`App_Start`** folder.



Defining Routes 2-3

- config.Routes.MapHttpRoute is a method in ASP.NET Web API used to configure the routing for HTTP requests to Web API controllers.

```
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 defines the URL pattern to match: `"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

Defining Routes 3-3

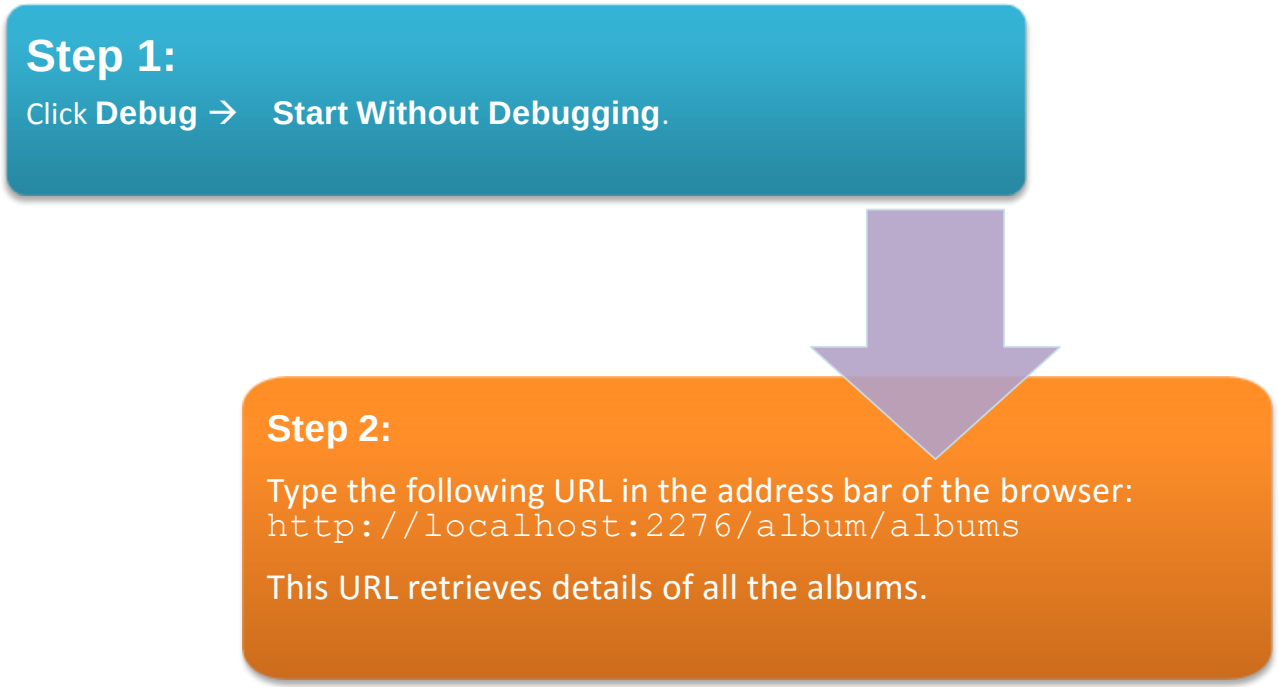
- To configure a new route in the `WebApiConfig.cs` file, refer the following code:

```
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}`.

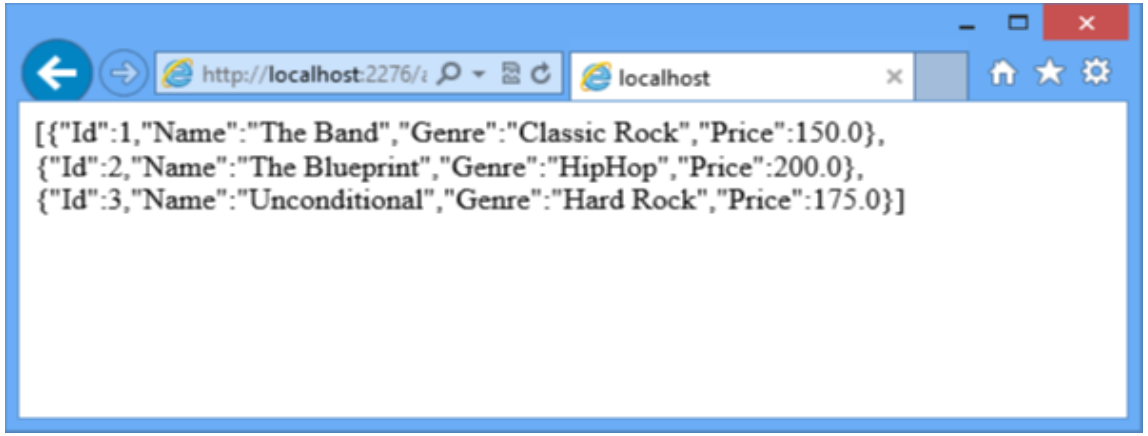
Assessing the Application 1-2

- 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:



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
```

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.