**In this video we will discuss**

* Why is versioning required in Web API
* Different options available to version Web API services
* How to version a Web API service using URI's

**Why is versioning required in Web API?**

* Once a Web API service is made public, different client applications start using your Web API services.
* As the business grows and requirements change, we may have to change the services as well, but the changes to the services should be done in way that does not break any existing client applications.
* This is when Web API versioning helps. We keep the existing services as is, so we are not breaking the existing client applications, and develop a new version of the service that new client applications can start using.

**Different options available to version Web API services :** Versioning can be implemented using  
1. URI's  
2. Query String  
3. Version Header  
4. Accept Header  
5. Media Type   
  
**How to version a Web API service using URI's :** In this video we will discuss versioning using URI's. In our upcoming videos we will discuss the rest of the versioning options. Let us understand versioning using URI's with an example.   
  
Version 1 of Student class (StudentV1) has just 2 properties (Id & Name). 

public class StudentV1

{

    public int Id { get; set; }

    public string Name { get; set; }

}

Consider the following StudentsV1Controller : This is our Version 1 controller.

public class StudentsV1Controller : ApiController

{

    List<StudentV1> students = new List<StudentV1>()

    {

        new StudentV1() { Id = 1, Name = "Tom"},

        new StudentV1() { Id = 2, Name = "Sam"},

        new StudentV1() { Id = 3, Name = "John"},

    };

    public IEnumerable<StudentV1> Get()

    {

        return students;

    }

    public StudentV1 Get(int id)

    {

        return students.FirstOrDefault(s => s.Id == id);

    }

}

In **WebApiConfig.cs** file we have the following route.

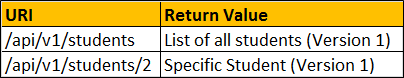
config.Routes.MapHttpRoute(

    name: "Version1",

    routeTemplate: "api/v1/Students/{id}",

    defaults: new { id = RouteParameter.Optional, controller = "StudentsV1" }

);

Clients of our Version1 service can use the following URLs to get either the list of all students or a specific student by Id. At the moment, the student objects the service return has Id and Name properties.   
   
  
Now let's say the business requirements have changed and some new clients want want FirstName and LastName properties instead of Name. If we change the existing Version 1 service, it will break all the existing client applications. So there is a need to create Version 2 of the service which the new client applications can use.   
  
All the existing client application will continue to work the same way as before and they have 2 options now. If they do not want to make changes to their application they can continue to use our version 1 of the service or they can use version 2 by modifying their application code. The important point is with versioning we are not breaking the existing clients and at the same time we are also satisfying the new client requirements.  
  
**Here are the steps to create Version 2 of our service**  
  
**Step 1 :**Add a class file in the Models folder. Name it StudentV2. Copy and paste the following code. Notice, instead of Name we have FirstName and LastName properties.

public class StudentV2

{

    public int Id { get; set; }

    public string FirstName { get; set; }

    public string LastName { get; set; }

}

**Step 2 :** Add a new Web API 2 empty controller to the controllers folder. Name it "StudentsV2Controller". Copy and paste the following code. Notice "StudentsV2Controller" returns "StudentsV2" object that has FirstName and LastName instead of Name property. 

public class StudentsV2Controller : ApiController

{

    List<StudentV2> students = new List<StudentV2>()

    {

        new StudentV2() { Id = 1, FirstName = "Tom", LastName = "T"},

        new StudentV2() { Id = 2, FirstName = "Sam", LastName = "S"},

        new StudentV2() { Id = 3, FirstName = "John", LastName = "J"}

    };

    public IEnumerable<StudentV2> Get()

    {

        return students;

    }

    public StudentV2 Get(int id)

    {

        return students.FirstOrDefault(s => s.Id == id);

    }

}

**Step 3 :** Include the following route in WebApiConfig.cs file.  

config.Routes.MapHttpRoute(

    name: "Version2",

    routeTemplate: "api/v2/Students/{id}",

    defaults: new { id = RouteParameter.Optional, controller = "StudentsV2" }

);

So at the moment in WebApiConfig.cs file we have 2 routes as shown below. Notice the route template, for each of the routes. As you might have guessed   
Version 1 clients use "/api/v1/Students/{id}" route and  
Version 2 clients use "/api/v2/Students/{id}" route 

config.Routes.MapHttpRoute(

    name: "Version1",

    routeTemplate: "api/v1/Students/{id}",

    defaults: new { id = RouteParameter.Optional, controller = "StudentsV1" }

);

config.Routes.MapHttpRoute(

    name: "Version2",

    routeTemplate: "api/v2/Students/{id}",

    defaults: new { id = RouteParameter.Optional, controller = "StudentsV2" }

);

So at this point we have versioning in place and if we navigate to   
/api/v1/students - We get students with Id and Name properties  
/api/v2/students - We get students with Id, FirstName and LastName properties  
  
At the moment we are using **convention-based routing** to implement versioning. We can also use Attribute Routing instead of convention-based routing to implement versioning. Use the [Route] attribute on methods in StudentsV1Controller and StudentsV2Controller as shown below.

public class StudentsV1Controller : ApiController

{

    [Route("api/v1/students")]

    public IEnumerable<StudentV1> Get() {...}

    [Route("api/v1/students/{id}")]

    public StudentV1 Get(int id) {...}

}

public class StudentsV2Controller : ApiController

{

    [Route("api/v2/students")]

    public IEnumerable<StudentV2> Get() {...}

    [Route("api/v2/students/{id}")]

    public StudentV2 Get(int id) {...}

}

Since we are using Attribute Routing we can safely comment the following 2 route templates in WebApiConfig.cs file

//config.Routes.MapHttpRoute(

//    name: "Version1",

//    routeTemplate: "api/v1/Students/{id}",

//    defaults: new { id = RouteParameter.Optional, controller = "StudentsV1" }

//);

//config.Routes.MapHttpRoute(

//    name: "Version2",

//    routeTemplate: "api/v2/Students/{id}",

//    defaults: new { id = RouteParameter.Optional, controller = "StudentsV2" }

//);

At this point, build the solution and test the application. It should work exactly the same way as before.