Web Service

Windows Programming Course

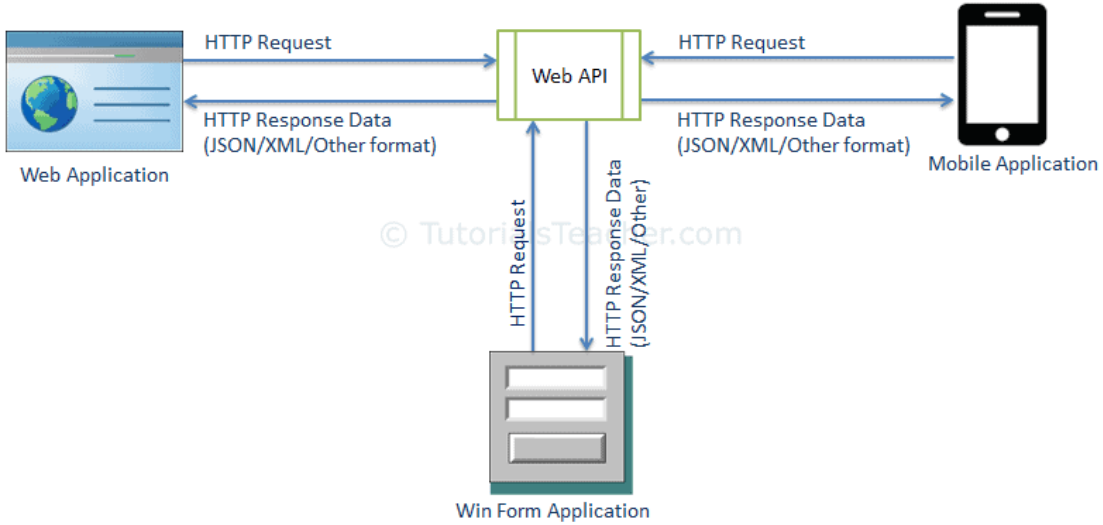
Agenda

1. Web Service Overview
2. Creating Web API
3. Consuming Web API
4. Testing Web API
5. Dependency Injection

1.
Web Service Overview

Web Service

A Web Service is a collection of protocols and standards used for exchanging data between applications or systems.



Type of Services

SOAP Web Services (Simple Object Access Protocol) is an application communication protocol for sending and receiving messages. It is based on XML.

REST Web Service (**Re**presentational **S**tate **T**ransfer) are based on the way how web works. REST is neither a standard nor a protocol. It is just an architectural style like say for example client-server architecture. It’s based on HTTP.

History of Web Service in .NET World

Windows Communication Foundation (WCF) was announced with .NET 3.0. The goal was to have one communication technology that is very flexible and fulfills all needs. However, WCF was initially based on SOAP => Too complex.

ASP.NET Web API was introduced in 2012 (version 1). It supports REST.

With the release of ASP.NET Core, the third major version of a Web API using an ASP.NET technology was released.

SOAP vs. REST Web Services

Feature	REST (Web API)	SOAP (WCF)
Protocol	HTTP	HTTP, TCP, UDP, NamedPipes, …
Message Format	Text (XML, Json)	XML
Service Interface	HTTP Verbs, Resource-based URLs	Service Contract, Action-based URLs
State Management	Stateless	Stateless or Stateful
Error Handling	Exceptions, HTTP Status	Faults, Behaviors

Resource-based vs. Action-based URLs

```
  Resource-based URL
-------------------------------
  GET    /accounts/application
  POST   /accounts
  GET    /queries/form
  POST   /queries
  GET    /order/123
  POST   /order/123/items
  DELETE /order/123
  GET    /order/123/invoice
  POST   /order/123/payments
  GET    /order/123/receipt
  POST   /sessions
  DELETE /user/123
  GET    /items/123/form
  PUT    /items/123
```

Web API

```
Action-based URL
-------------------------------
GET  /register
POST /register
GET  /catalog/search
POST /catalog/search
GET  /cart
POST /cart/add-item
POST /cart/empty
GET  /check-out
POST /check-out
GET  /thank-you
POST /sign-in
POST /admin/delete-user?id=123
GET  /catalog/edit?id=123
POST /catalog/edit?id=123
```

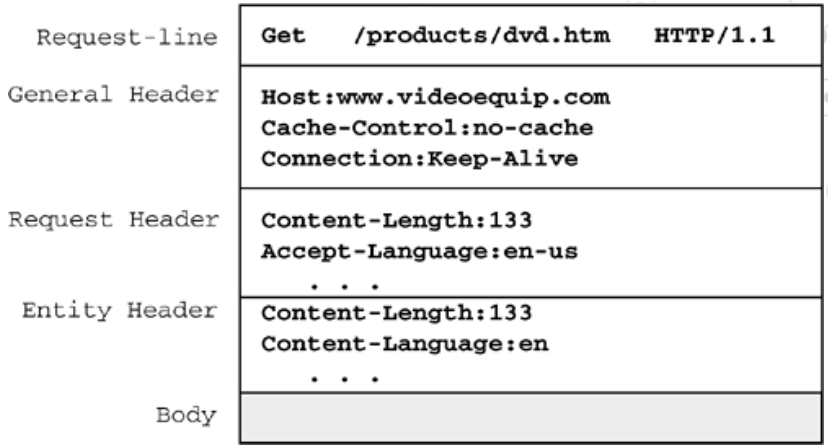
WCF

Features of REST

- Platform-independent (you don't care if the server is Unix, the client is a Mac, or anything else)
- Language-independent (C# can talk to Java, etc.)
- HTTP Based (especially HTTP Verb)
- Limited bandwidth and resources
- Totally stateless operations
- Caching
- Reach More Clients(Mobile , Tablet etc.)
- Lightweight hosting and scalable with Cloud.

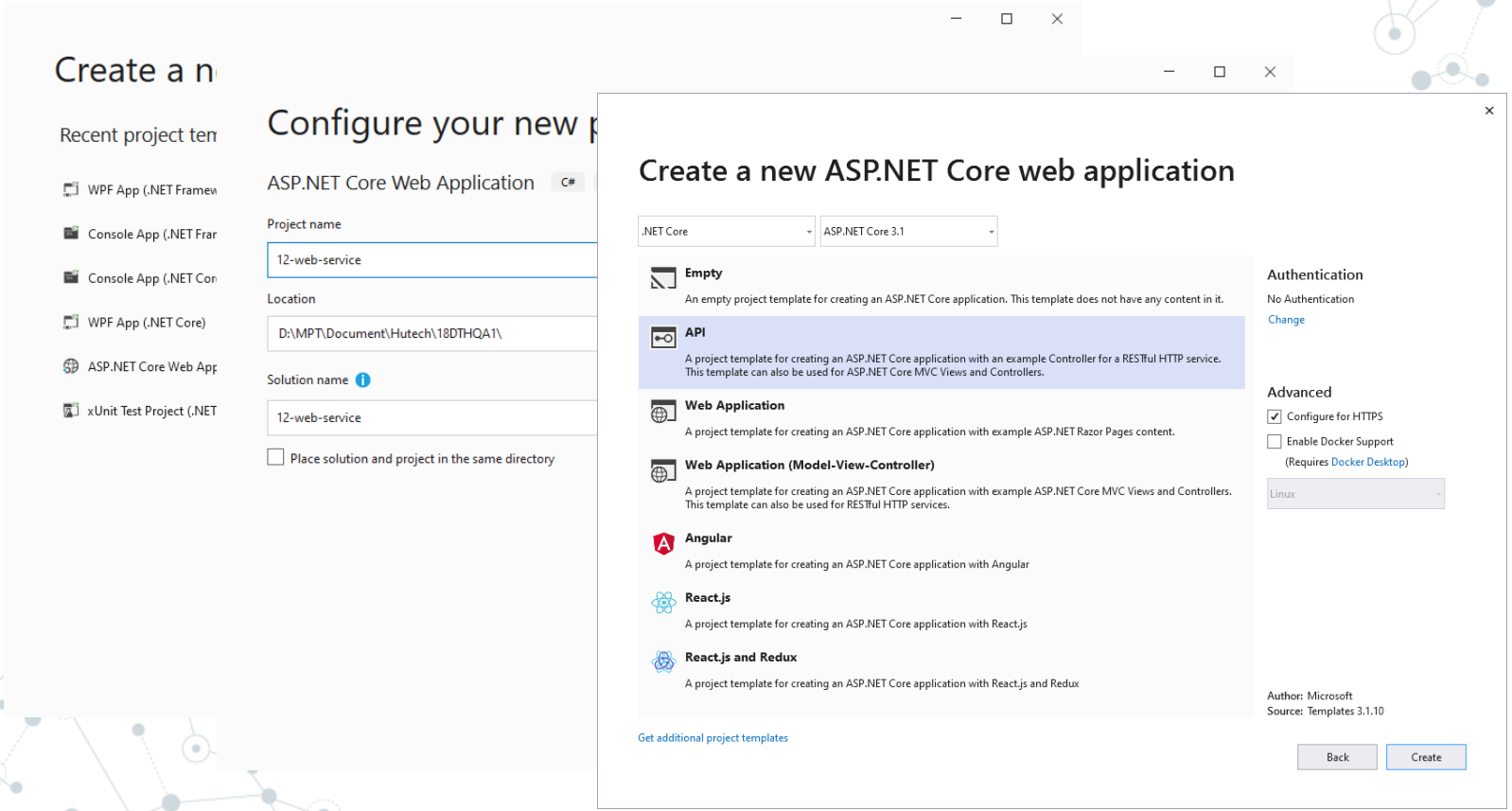
HTTP Methods (Verbs)

HTTP Method	Usage
GET	Retrieves data
POST	Inserts new record
PUT	Updates existing record
PATCH	Updates record partially
DELETE	Deletes record



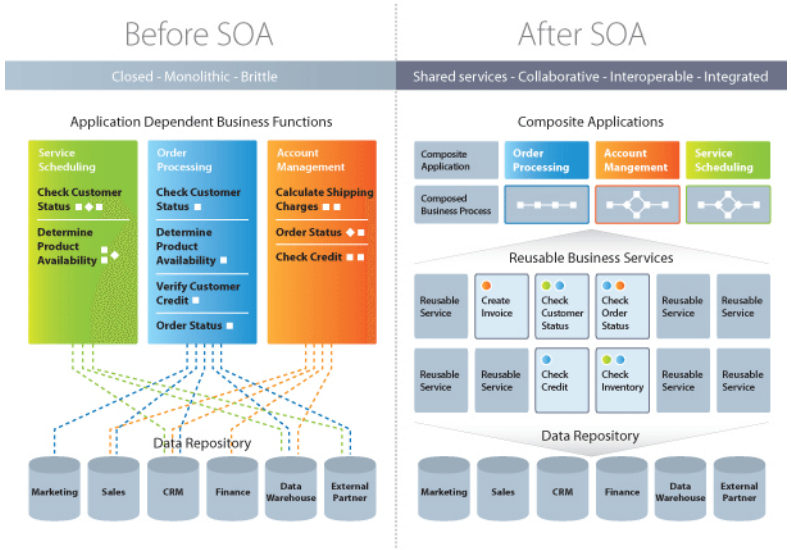
2.
Creating Web API

Creating a ASP.NET Core Web API



Service-Oriented Architecture (SOA)

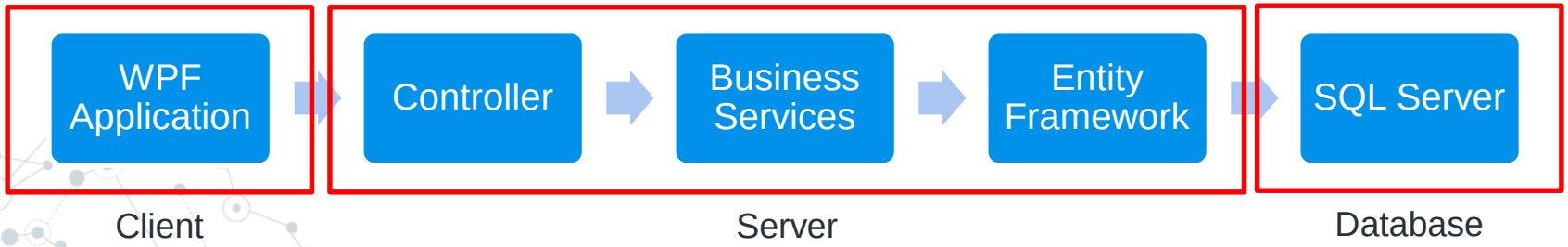
Service-Oriented Architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.



Architecture for SIM (the Final Project)

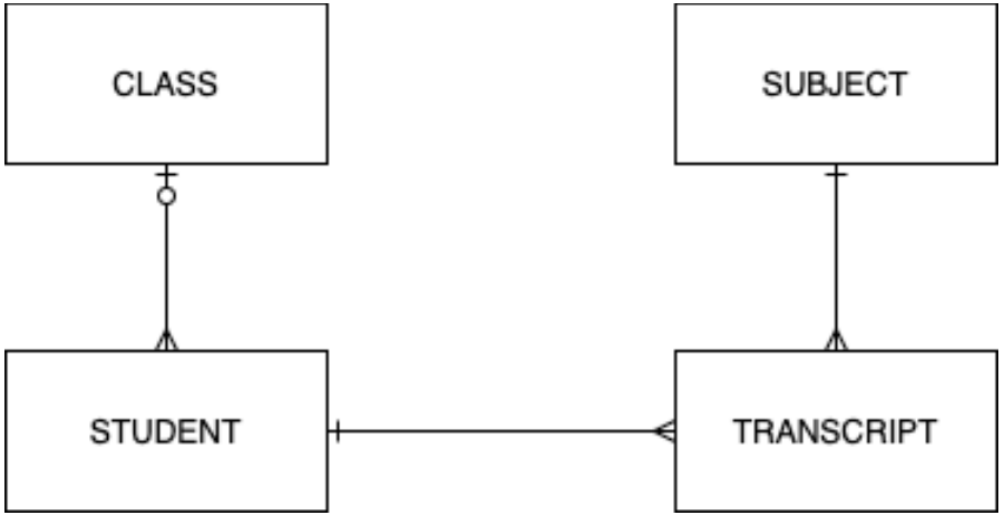
3 tier architecture:

- Client: WPF Application
- Server: Web Service. There are 3 layers:
  - Controller
  - Service
  - Repository
  - Entity Framework
- Database: SQL Server



Defining a Model

Define class that represents the data to return and change:



Creating a Service


```
public interface IStudentService
{
  int Add(Student student);

  IEnumerable<Student> Search(SearchStudentCriteria criteria);

  void Update(Student student);

  void Remove(int id);
}
```

Creating a Controller


```
public class ValuesController : ApiController
{
    // GET api/values
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // GET api/values/5
    public string Get(int id)
    {
        return "value";
    }

    // POST api/values
    public void Post([FromBody]string value)
    {
    }

    // PUT api/values/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE api/values/5
    public void Delete(int id)
    {
    }
}
```

Web API controller Base class

Handles Http GET request

http://localhost:1234/api/values

Handles Http GET request with query string

http://localhost:1234/api/values?id=1

Handles Http POST request

http://localhost:1234/api/values

Handles Http Put request

http://localhost:1234/api/values?id=1

Handles Http DELETE request

http://localhost:1234/api/values?id=1

Parameter Binding

HTTP Method	Query string	Request Body
GET	Primitive Type Complex Type	NA
POST	Primitive Type	Complex Type
PUT	Primitive Type	Complex Type
PATCH	Primitive Type	Complex Type
DELETE	Primitive Type Complex Type	NA

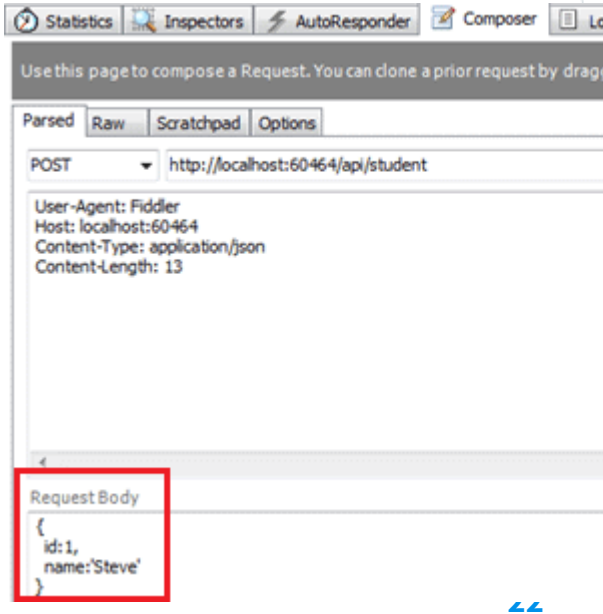
Parameter Binding – GET Method with Primitive Parameter

```
public class StudentController : ApiController {
  public Student Get(int id) {
  }
}

// http://localhost/api/student?id=1
// http://localhost/api/student?ID=1
```

Parameter Binding – POST Method with Primitive Parameter

By default, Web API gets the value of a primitive parameter from the query string and complex type parameter from the request body.

```
public class Student {
  public int Id { get; set; }
  public string Name { get; set; }
}

public class StudentController : ApiController {
  public Student Post(Student stud) {
  }
}
```



Parameter Binding – [FromUri] and [FromBody]

Use [FromUri] attribute to force Web API to get the value of complex type from the query string and [FromBody] attribute to get the value of primitive type from the request body, opposite to the default rules..

```
public class StudentController : ApiController {
  public Student Post([FromBody]string name) {
  }
  public Student Post([FromUri]Student student) {
  }

}
```

Action Return Type

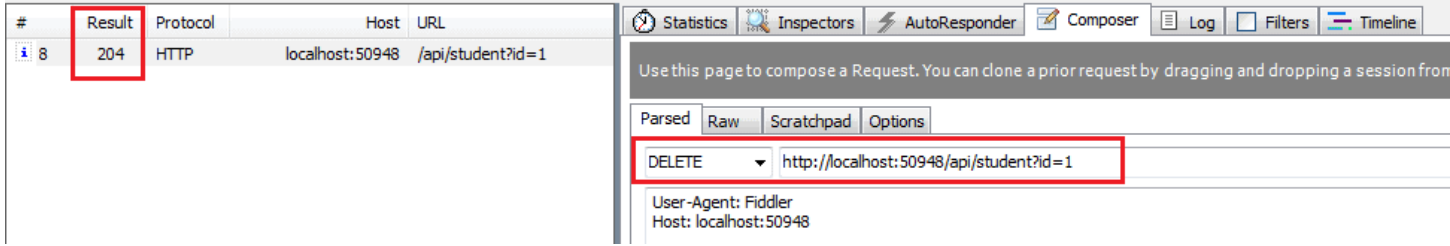
The Web API action method can have following return types.

- Void
- Primitive type or Complex type
- HttpResponseMessage
- IHttpActionResult

Action Return Type – Void

It's not necessary that all action methods must return something. It can have void return type. E.g.:

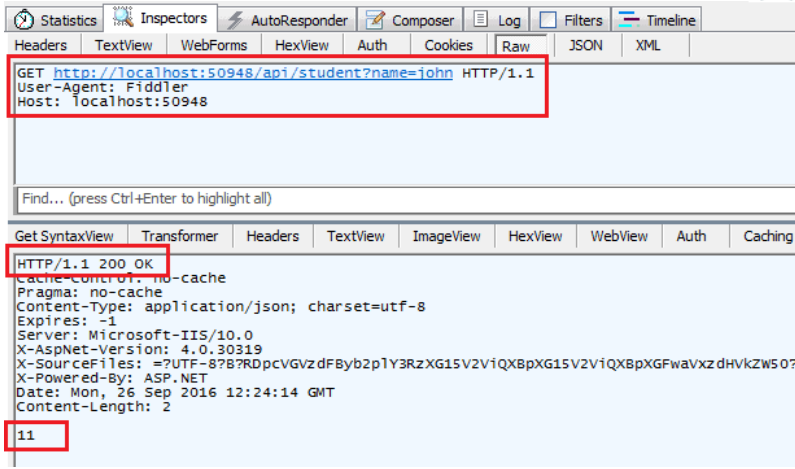
```
public class StudentController : ApiController {
  public void Delete(int id) {
    DeleteStudentFromDB(id);
  }
}
```



Action Return Type – Primitive type or Complex type

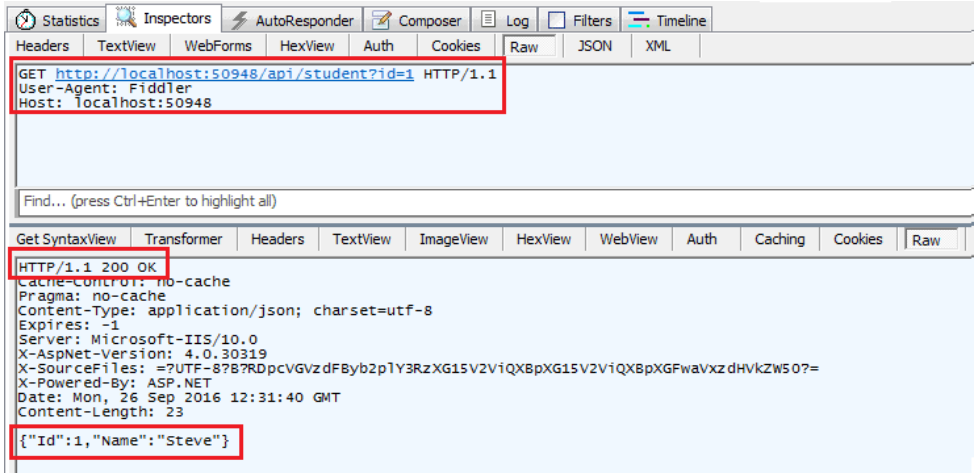
An action method can return primitive or other custom complex types as other normal methods. E.g.:

```
public class Student {
    public int Id { get; set; }
    public string Name { get; set; }
}
public class StudentController : ApiController {
  public int GetId(string name) {
    int id = GetStudentId(name);
    return id;
  }
}
```



Action Return Type – Primitive type or Complex type (cont.)

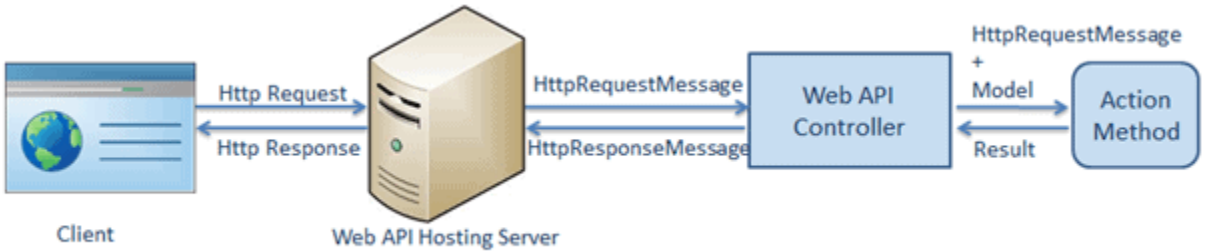
```
public class StudentController : ApiController {
  public Student GetStudent(int id) {
    var student = GetStudentFromDB(id);
    return student;
  }
}
```



Action Return Type – HttpResponseMessage

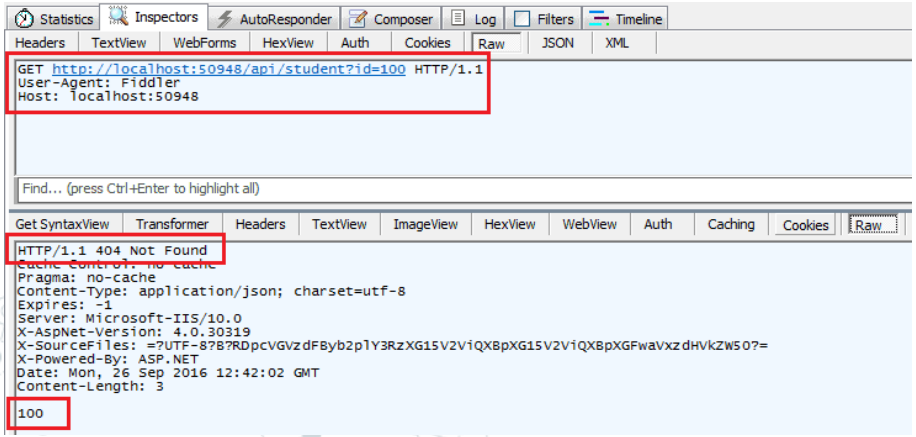
Web API controller always returns an object of HttpResponseMessage to the hosting infrastructure.

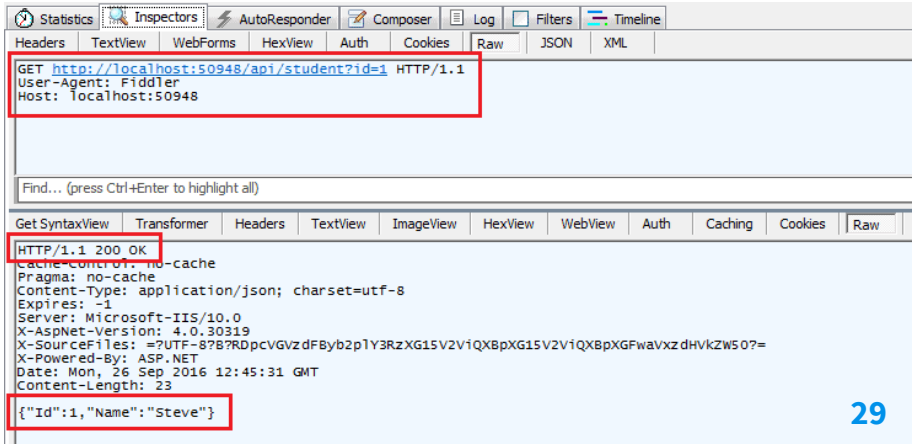
The advantage of sending HttpResponseMessage from an action method is that you can configure a response your way: the status code, content or error message (if any) as per your requirement.



Action Return Type – HttpResponseMessage (cont.)

```
public HttpResponseMessage Get(int id) {
    Student stud = GetStudentFromDB(id);
    if (stud == null) {
        return Request.CreateResponse(HttpStatusCode.NotFound, id);
    }
    return Request.CreateResponse(HttpStatusCode.OK, stud);
}
```





Action Return Type – IHttpActionResult

The IHttpActionResult was introduced in Web API 2 (.NET 4.5). An action method in Web API 2 can return an implementation of IHttpActionResult class.

You can create your own class that implements IHttpActionResult or use various methods of ApiController class that returns an object that implement the IHttpActionResult.

Action Return Type – IHttpActionResult (cont.)

ApiController Method	Description
BadRequest()	Creates a BadRequestResult object with status code 400.
Created()	Creates a CreatedNegotiatedContentResult with status code 201 Created.
InternalServerError()	Creates an InternalServerErrorResult with status code 500 Internal server error.
NotFound()	Creates a NotFoundResult with status code404.
Ok()	Creates an OkResult with status code 200.
ResponseMessage()	Creates a ResponseMessageResult with the specified HttpResponseMessage.
StatusCode()	Creates a StatusCodeResult with the specified http status code.
Unauthorized()	Creates an UnauthorizedResult with status code 401.

Data Formats

Media type (aka MIME type) specifies the format of the data as type/subtype e.g. text/html, text/xml, application/json, image/jpeg etc.

In HTTP request, MIME type is specified in the request header using:

- The **Accept** attribute specifies the format of response data which the client expects.
- The **Content-Type** attribute specifies the format of the data in the request body.

Data Formats (cont.)

HTTP GET Request:

```
GET http://localhost:60464/api/student HTTP/1.1
User-Agent: Fiddler
Host: localhost:1234
Accept: application/json
```

HTTP POST Request:

```
POST http://localhost:60464/api/student?age=15 HTTP/1.1
User-Agent: Fiddler
Host: localhost:60464
Content-Type: application/json
Content-Length: 13

{
  id:1,
  name:'Steve'
}
```

Web API Filters

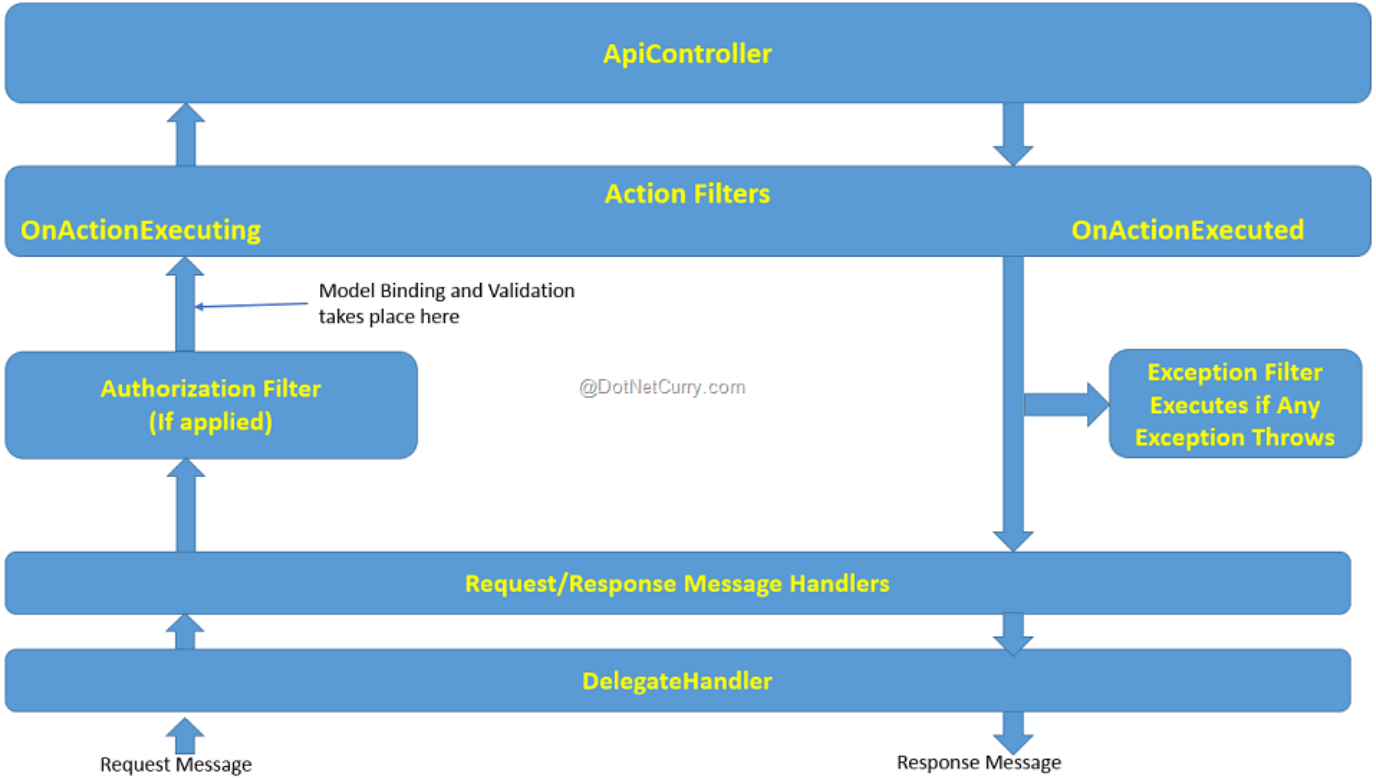
Web API includes filters to add extra logic before or after action method executes.

Filters can be used to provide cross-cutting features such as logging, exception handling, performance measurement, authentication and authorization.

Filters are actually attributes that can be applied on the Web API controller or one or more action methods.

Every filter attribute class must implement IFilter interface

Web API Filters (cont.)



Web API Filters (cont.)

The following table lists important interfaces and classes that can be used to create Web API filters.

Filter Type	Interface	Class	Description
Simple Filter	IFilter		Defines the methods that are used in a filter
Action Filter	IActionFilter	ActionFilterAttribute	Used to add extra logic before or after action methods execute.
Authentication Filter	IAuthenticationFilter		Used to force users or clients to be authenticated before action methods execute.
Authorization Filter	IAuthorizationFilter	AuthorizationFilterAttribute	Used to restrict access to action methods to specific users or groups.
Exception Filter	IExceptionFilter	ExceptionFilterAttribute	Used to handle all unhandled exception in Web API.

Web API Filters – Log Filter Example

```
public class LogAttribute : ActionFilterAttribute {

  public override void OnActionExecuting(HttpActionContext actionContext) {
    Trace.WriteLine(string.Format("Action Method {0} executing at {1}",
actionContext.ActionDescriptor.ActionName, DateTime.Now.ToShortDateString()),
"Web API Logs");
  }

  public override void OnActionExecuted(HttpActionExecutedContext
actionExecutedContext) {
    Trace.WriteLine(string.Format("Action Method {0} executed at {1}",
actionExecutedContext.ActionContext.ActionDescriptor.ActionName,
DateTime.Now.ToShortDateString()), "Web API Logs");
  }
}
```

Web API Filters – Log Filter Example

```
[Log]
public class StudentController : ApiController {
  public StudentController() {
  }

  public Student Get()
  {
    // Provide implementation
  }
}
```

CURD Operations

Follow the tutorial here to generate Web API Controller with CURD operations

3.
Consuming Web API

HttpClient

The HttpClient class instance acts as a session to send HTTP requests.

An HttpClient instance is a collection of settings applied to all requests executed by that instance.

In addition, every HttpClient instance uses its own connection pool, isolating its requests from requests executed by other HttpClient instances.

HttpClient is intended to be instantiated once and re-used throughout the life of an application. Instantiating an HttpClient class for every request will exhaust the number of sockets available under heavy loads.

Methods

Using the following method to send requests:

- GetAsync
- GetByteArrayAsync
- GetStreamAsync
- GetStringAsync
- PostAsync
- PutAsync
- DeleteAsync
- SendAsync

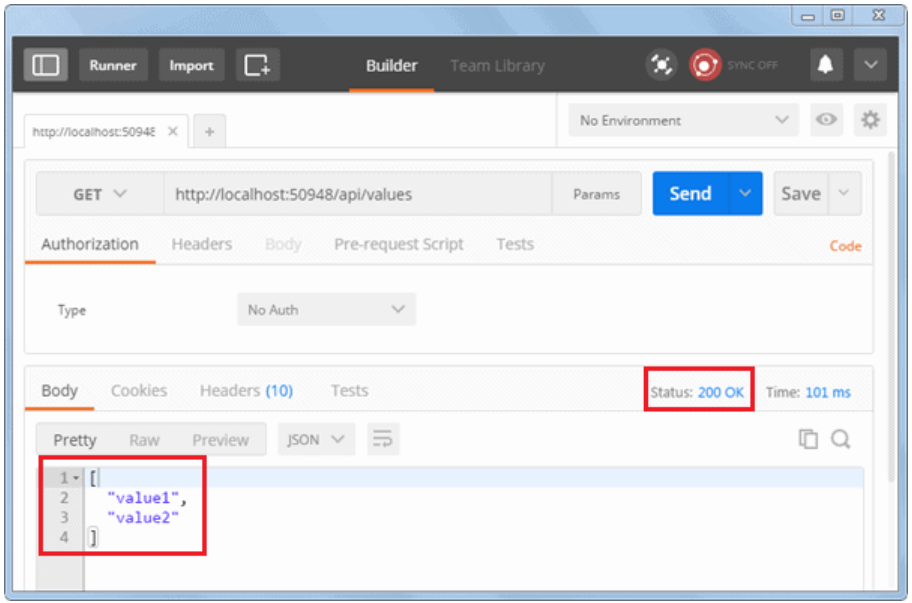
Consume GET/POST/PUT/DELETE methods

Follow the tutorial here to consume API Methods.

4.
Testing Web API

Postman

Postman is a free API debugging tool.



Swagger

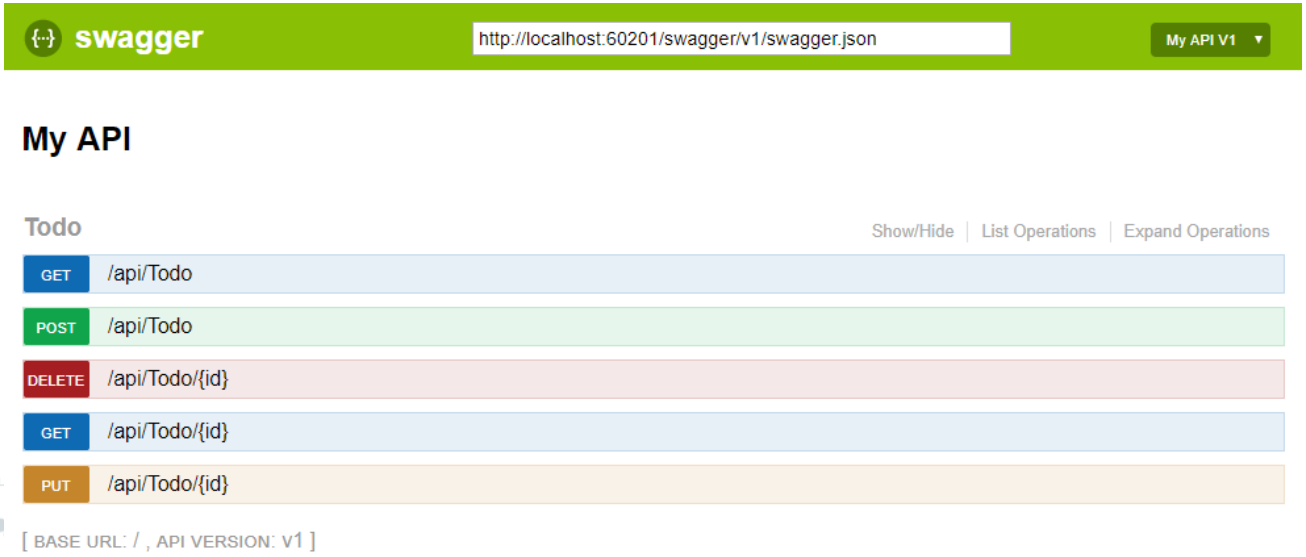
Swagger supports to generate useful document (OpenAPI specification) and help pages for Web API.

JSON Sao chép


```
{
  "openapi": "3.0.1",
  "info": {
    "title": "API V1",
    "version": "v1"
  },
  "paths": {
    "/api/Todo": {
      "get": {
        "tags": [
          "Todo"
        ],
        "operationId": "ApiTodoGet",
        "responses": {
          "200": {
            "description": "Success",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/ToDoItem"
                  }
                }
```

Swagger (cont.)

Swagger UI offers a web-based UI that provides information about the service, using the generated OpenAPI specification.



Swagger – Setup for Web API (.NET Framework)

Following steps to setup Swagger for Web API

- Add a NuGet Package: Swashbuckle
- Once you have installed Swashbuckle to your project you can find a SwaggerConfig file in App_Start.
- Run the project and add */Swagger* in url.

E.g. http://localhost:5543/swagger

Swagger – Setup for ASP.NET Core Web API

Follow the tutorial here to setup Swagger for ASP.NET Core Web API

C# Sao chép

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));
    services.AddControllers();

    // Register the Swagger generator, defining 1 or more Swagger documents
    services.AddSwaggerGen();
}
```

In the `Startup.Configure` method, enable the middleware for serving the generated JSON document and the Swagger UI:

C# Sao chép

```
public void Configure(IApplicationBuilder app)
{
    // Enable middleware to serve generated Swagger as a JSON endpoint.
    app.UseSwagger();

    // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
    // specifying the Swagger JSON endpoint.
    app.UseSwaggerUI(c =>
    {
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
    });

    app.UseRouting();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

5.
Dependency Injection

The Problem

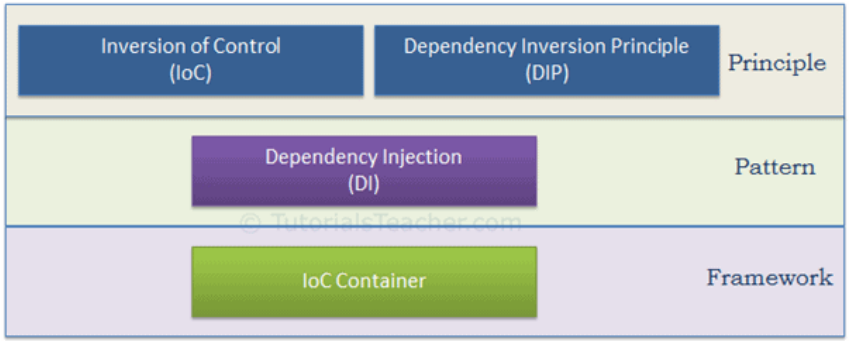
In the example, StudentService depend on ServiceA, ServiceB, ServiceC and ServiceD
=> Have to instantiate all dependent services to instantiate Student Service.

```
public class StudentService {
  public void StudentService (IServiceA srvA, IServiceB srvB, IServiceC srvC,
IServiceD srvD)
}
// Instantiate Student Service
var srvA = new ServiceA();
var srvB = new ServiceB();
var srvC = new ServiceC();
var srvD = new ServiceD();
var studentSrv = new StudentService(srvA, srvB, srvC, srvD);
```

Dependency Injection

Dependency Injection (DI) is a design pattern that helps you write code that is loosely coupled and achieves the important principle of Inversion of Control (IoC).

Using DI, we move the creation and binding of the dependent objects outside of the class that depends on them.



DI in Web API (.NET Framework)

There are many IoC containers available for dependency injection such as Ninject, Unity, Castle Widsor, Autofac etc.

Following the tutorial here to setup Ninject for ASP.NET Web API.

DI in Web API (.NET Core)

ASP.NET Core supports the dependency injection (DI) software design pattern.

Follow the tutorial here to configure DI in ASP.NET Core.

The important point to remember is Service Lifetimes:

- **Transient**: Create each time it will be requested.
- **Scoped**: Create an instance of a service for each client request.
- **Singleton**: Create one instance of the service

Reference

https://www.tutorialsteacher.com/webapi/web-api-tutorials

Review Questions

1. What are different types of Web Services?
2. What is REST Web Services?
3. What is the purpose of HTTP Verb in REST based webservices?
4. What is the difference between PUT and POST operations?
5. What is the purpose of HTTP Status Code?
6. What tools are used to test a web service?

Demo

- Create Web Service (.NET Framework)
  - Add Swagger
  - Test with Postman
- Create Web Service (.NET Core)
  - Add Swagger
  - Add Service
  - Demo DI
- Final project
  - Demo a flow from client to server
- Linq
- Q&A

Thanks!

Any questions?

You can find me at:

tranminhphuoc@gmail.com