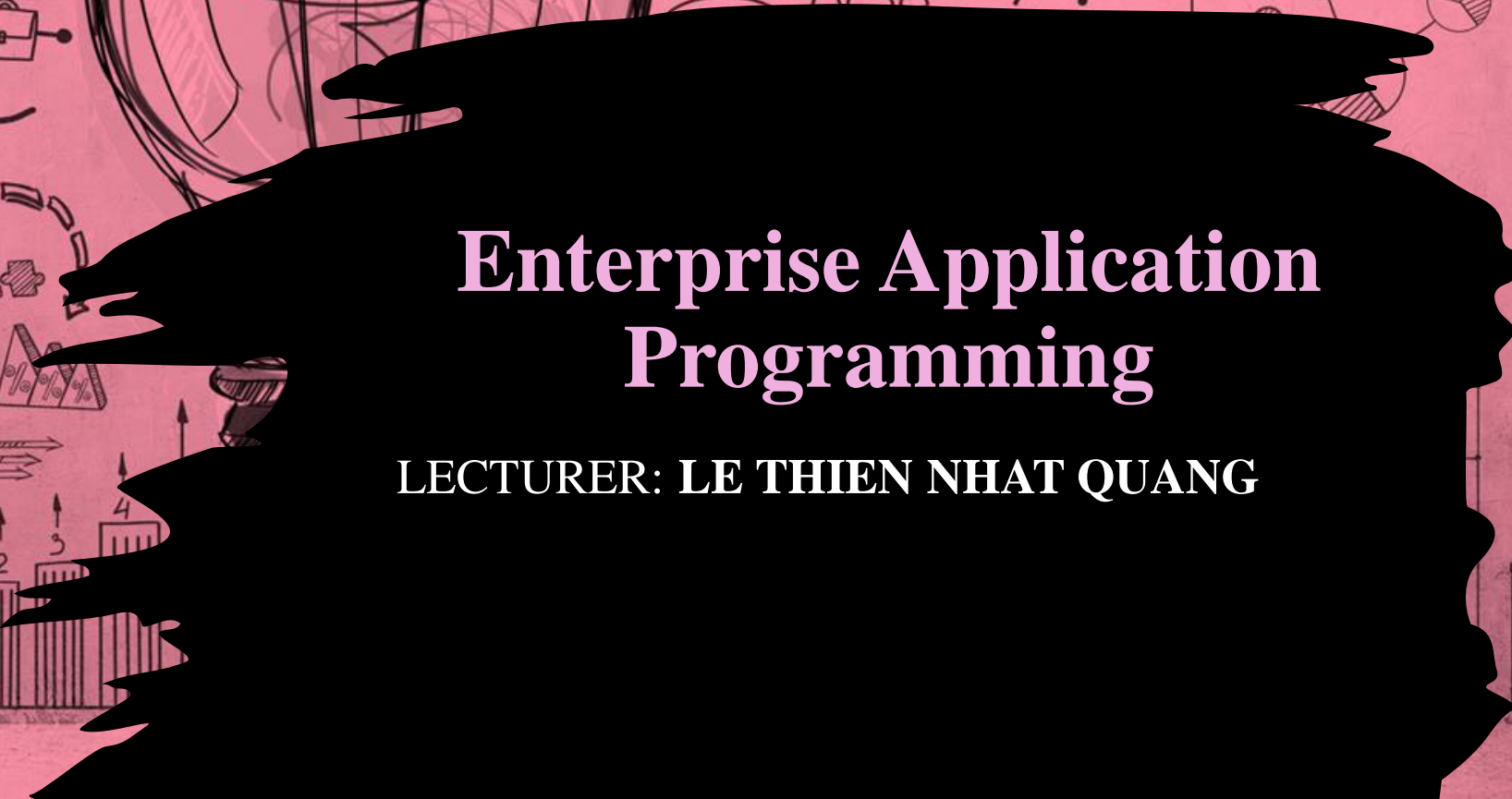


# Enterprise Application Programming

LECTURER: LE THIEN NHAT QUANG



# Enterprise Application Programming

LECTURER: LE THIEN NHAT QUANG

# So sánh API web ASP.NET vs WCF

API web	WCF
Nguồn mở và ships với .NET framework.	Ships với .NET framework
Chỉ hỗ trợ giao thức HTTP.	Hỗ trợ HTTP, TCP, UDP và giao thức truyền tải tùy chỉnh.
Bản đồ http phương thức động	Sử dụng các thuộc tính dựa trên mô hình lập trình
Sử dụng khái niệm định tuyến và điều khiển tương tự như ASP.NET MVC.	Sử dụng hợp đồng dịch vụ, vận hành và dữ liệu.
Không hỗ trợ nhắn tin và giao dịch đáng tin cậy.	Hỗ trợ nhắn tin và giao dịch đáng tin cậy.
API Web có thể được cấu hình bằng lớp httpConfiguration nhưng không có trong web.config.	Sử dụng web.config và các thuộc tính để định cấu hình một dịch vụ.
Lý tưởng để xây dựng các dịch vụ RESTful.	Hỗ trợ các dịch vụ RESTful nhưng có giới hạn.

# So sánh API web ASP.NET vs WCF

## Khi nào nên chọn WCF?

- Nếu sử dụng .NET Framework 3.5, API Web không hỗ trợ .NET 3.5 trở xuống.
- Nếu dịch vụ cần hỗ trợ nhiều giao thức như HTTP, TCP, Named pipe.
- Nếu muốn xây dựng dịch vụ với các tiêu chuẩn WS- \* như Nhắn tin đáng tin cậy, Giao dịch, Bảo mật tin nhắn.
- Nếu muốn sử dụng các mẫu trao đổi tin nhắn Yêu cầu-Trả lời, Một chiều và Song song

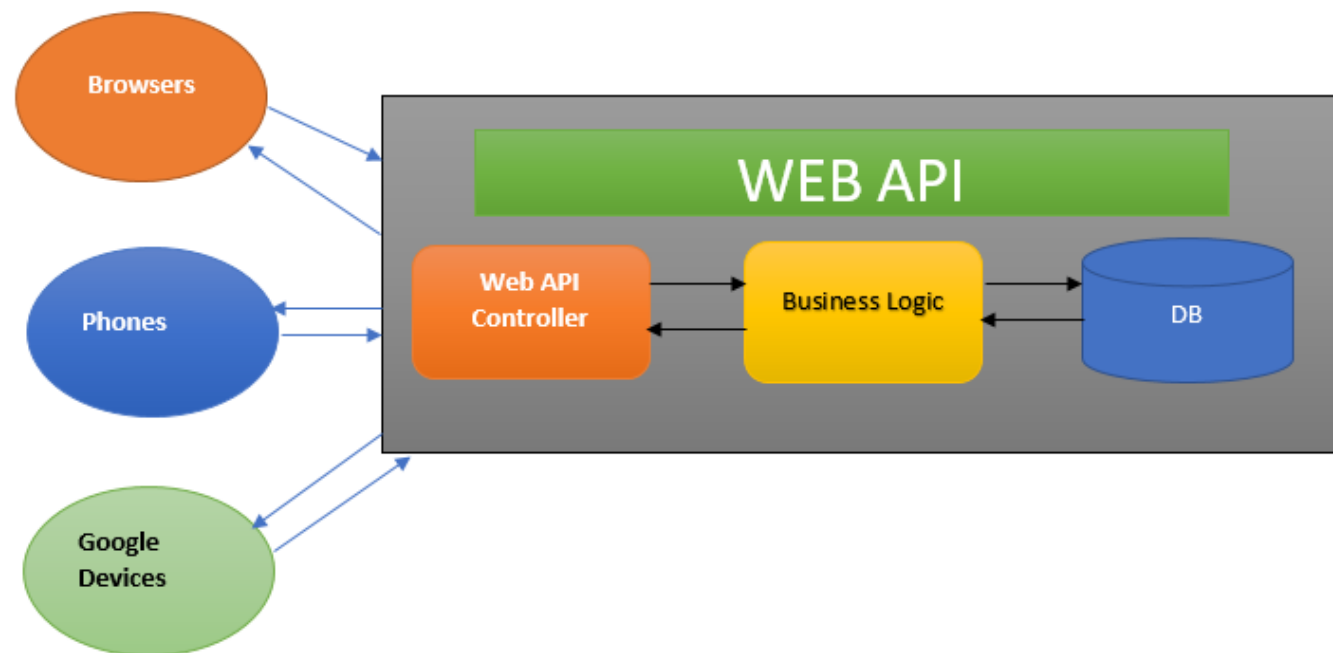
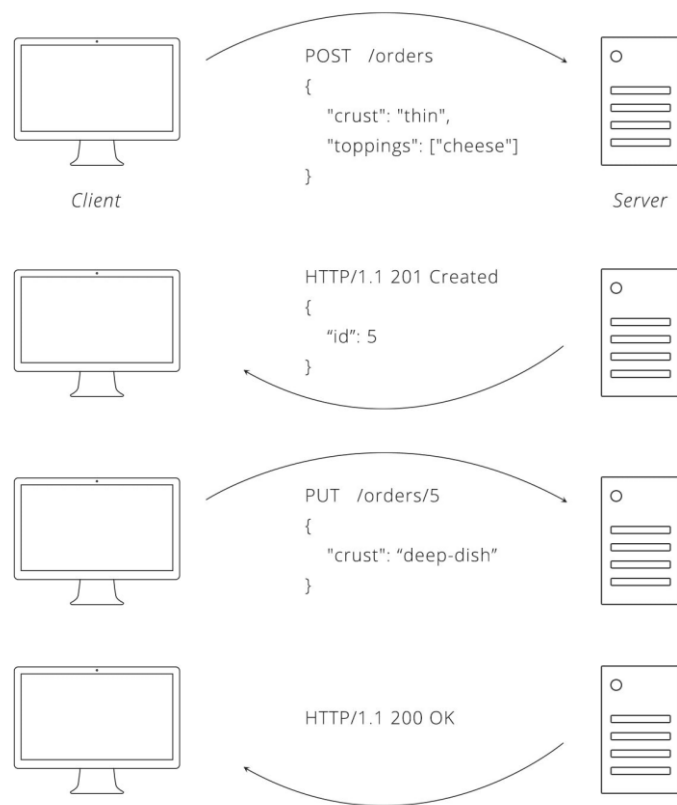
## Khi nào nên chọn API Web ASP.NET?

- Nếu bạn đang sử dụng .NET framework 4.0 trở lên.
- Nếu bạn muốn xây dựng một dịch vụ chỉ hỗ trợ giao thức HTTP.
- Để xây dựng các dịch vụ dựa trên HTTP RESTful.
- Nếu quen thuộc với ASP.NET MVC.

# Đặc điểm của Web API ASP.NET

- Là nền tảng lý tưởng để xây dựng các dịch vụ RESTful.
- Được xây dựng trên nền tảng ASP.NET và hỗ trợ đường dẫn yêu cầu / phản hồi ASP.NET
- Ánh xạ các động từ HTTP thành tên phương thức.
- Hỗ trợ các định dạng khác nhau của dữ liệu phản hồi. Hỗ trợ tích hợp cho định dạng JSON, XML, BSON.
- Có thể được lưu trữ trong IIS, Tự lưu trữ hoặc máy chủ web khác hỗ trợ .NET 4.0+.
- Khung API Web ASP.NET bao gồm HttpClient mới để giao tiếp với máy chủ API Web. HttpClient có thể được sử dụng trong phía máy chủ ASP.MVC, ứng dụng Windows Form, ứng dụng Console hoặc các ứng dụng khác.

# Sự tương tác giữa Server và Client (use API)





# Tổng quan về Restful API

- Http Method gồm có 9 loại nhưng RESTful chỉ sử dụng 4 loại phổ biến
- GET (SELECT): Trả về một Resource hoặc một danh sách Resource.
- POST (CREATE): Tạo mới một Resource.
- PUT (UPDATE): Cập nhật thông tin cho Resource.
- DELETE (DELETE): Xoá một Resource.
  - => Tương ứng với cái tên thường gọi là CRUD (Create, Read, Update, Delete)

# Nguyên tắc thiết kế Restful

Khi chúng ta gửi 1 request tới 1 API nào đó thì sẽ có vài status code để nhận biết như sau:

- **200 OK** – Trả về thành công cho tất cả phương thức
- **201 Created** – Trả về khi một Resource được tạo thành công.
- **204 No Content** – Trả về khi Resource xóa thành công.
- **304 Not Modified** – Client có thể sử dụng dữ liệu cache.
- **400 Bad Request** – Request không hợp lệ
- **401 Unauthorized** – Request cần có auth.
- **403 Forbidden** – bị từ chối không cho phép.
- **404 Not Found** – Không tìm thấy resource từ URI.
- **405 Method Not Allowed** – Phương thức không cho phép với user hiện tại.
- **410 Gone** – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
- **415 Unsupported Media Type** – Không hỗ trợ kiểu Resource này.
- **422 Unprocessable Entity** – Dữ liệu không được xác thực.
- **429 Too Many Requests** – Request bị từ chối do bị giới hạn.

# Thiết kế Result API

- **Bước 1:** Tạo Project Web API với tên Writeup\_WebAPI
- **Bước 2:** Ý tưởng là mình sẽ tạo 1 class Model để xử lý trong phần API rồi được Controller gọi lên để hiển thị ra View. Tạo class **Users** ở trong **Folder Model**.

```
public class Users {  
    public string Username { get; set; }  
    public string Password { get; set; }  
    public string Fullname { get; set; }  
    public bool IsActive { get; set; }  
}
```



# Thiết kế Result API - 2

- **Bước 3:** Tạo Controller của WebAPI. Tạo Controller Web API là **UserController**. Thêm 1 hàm AllUser để tạo ra 1 Danh sách User để test.

```
private List<Users> AllUser() {  
    List<Users> list = new List<Users>();  
    for (int i = 1; i < 6; i++)        // Tạo ra 6 User  
    {  
        Users u = new Users()    // Tạo ra user mới  
        {  
            Username = $"user {i}", Password = $"password {i}", Fullname = $"fullname {i}", IsActive = true  
        };  
        list.Add(u);                // Thêm vào danh sách User  
    }  
    return list;  
}
```

# Thiết kế Result API - 3

- **Bước 4:** Setup Uri của API bằng cách vào Folder AppStart/WebApiConfig.cs và sửa lại đường routeTemplate là "**api/{controller}/{action}/{id}**"
- Thêm vào đoạn code Json Formatter bên dưới để định dạng dữ liệu trả về là dạng json.

# Thiết kế Result API - 4

```
public static void Register(HttpConfiguration config)
{
    // Web API configuration and services
    // Web API routes
    config.MapHttpAttributeRoutes();

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

    config.Formatters.JsonFormatter.SupportedMediaTypes // Thêm vào
        .Add(new MediaTypeHeaderValue("text/html"));
}
```

# Thiết kế Result API - 5

- **Bước 5:** Tạo API. Thực hiện trả về là **HttpResponseMessage** để xác định được StatusCode trả về là bao nhiêu hoặc có thể trả về kèm với Json báo lỗi bên trong Content của HttpClient.

## 5.1. GET METHOD: PHẦN USERCONTROLLER

- Thêm đoạn code thực hiện phương thức GET để lấy tất cả User

# Thiết kế Result API - 6

```
//GET api/User/GetAllUser  
[HttpGet] //Cho phép truy cập với phương thức là GET  
public HttpResponseMessage GetAllUser()  
{  
    var list = AllUser();  
    if (list != null)  
        return Request.CreateResponse(HttpStatusCode.OK, list);  
    else return Request.CreateResponse(HttpStatusCode.NotFound);  
}
```

- Uri của website sẽ là phần *host:port + /api/{tên controller}/{action của controller}/{tham số nếu có}* như phần cấu hình ở file Config.
- Khi truy cập vào địa chỉ */api/User/GetAllUser* chúng ta sẽ thu được 5 User đã tạo dưới dạng **Json**

# Thiết kế Result API - 7

## 5.2. POST METHOD - PHẦN USERCONTROLLER

- Ở đây mình sử dụng POSTMAN test thử, hoặc có thể vào <https://reqbin.com/> để trực tiếp sử dụng trên web không cần tải về
- Thêm đoạn code dùng để POST dữ liệu tải lên.
- Function này dùng để tạo thêm 1 user và trả về danh sách user đã được thêm vào phần Content của HttpResponseMessage



# Thiết kế Result API - 8

```
// POST api/Users/CreateNew
[HttpPost]
public HttpResponseMessage CreateNew(Users u)
{
    try
    {
        var list = AllUser();
        list.Add(u);    // Thêm User đã được truyền tham số User u
        return Request.CreateResponse(HttpStatusCode.OK, list);
    }
    catch (Exception)
    {
        return Request.CreateResponse(HttpStatusCode.BadRequest);
    }
}
```

# Thiết kế Result API - 9

## **TIẾN HÀNH GỬI POST REQUEST**

- Có thể gửi post ở dạng json hoặc urlencoded
- Dữ liệu trả về đã thêm user “....” vào danh sách

https://localhost:44313/Album/Album

Save

v

✎

💬

GET

v

https://localhost:44313/Album/Album

Send

v

Params

Authorization

Headers (8)

Body ●

Pre-request Script

Tests

Settings

Cookies

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON v

Beautify

```
1 {
2   ... "Id": 4,
3   ... "Name": "The Band",
4   ... "Genre": "Classic Rock",
5   ... "Price": 150.0
6 }
```

Body Cookies Headers (10) Test Results



200 OK

3 ms

627 B



Save as Example

...

Pretty

Raw

Preview

Visualize

JSON

v



```
16   "Name": "Unconditional",
17   "Genre": "Hard Rock",
18   "Price": 175.0
19 },
20 {
21   "Id": 4,
22   "Name": "The Band",
23   "Genre": "Classic Rock",
24   "Price": 150.0
25 }
26 ]
```

# Thiết kế Result API - 10

## 5.3. PUT METHOD

- Thêm 1 hàm dùng để update thông tin user dùng Method là PUT

### TIẾN HÀNH GỬI PUT REQUEST

- Cập nhật thông tin của user 1 bằng cách nhập đúng username cần update, dữ liệu update sẽ là các thông tin còn lại.
- Dữ liệu trả về user 1 đã được cập nhật password, fullname và isactive

# Thiết kế Result API - 11

```
// PUT api/Users/UpdateUser
[HttpPut]
public HttpResponseMessage UpdateUser(Users u) {
    try {
        var list = AllUser();    // Lấy index User thông qua username
        int index = list.FindIndex(p => p.Username == u.Username);
        list[index] = u;        // Update user
        return Request.CreateResponse(HttpStatusCode.OK, list);
    }
    catch (Exception) {
        return Request.CreateResponse(HttpStatusCode.BadRequest);
    }
}
```

# BÀI 1

## Thiết kế Result API - 12

The screenshot displays a REST client interface with a PUT request to `https://localhost:44398/api/User/UpdateUser`. The request body is a JSON object with the following fields:

```
{  1  {  2    "username": "user 1",  3    "password": "password update",  4    "fullname": "full name update",  5    "isactive": false  6  }
```

The response is a 200 OK status with a response time of 17.13 s and a body size of 836 B. The response body is a JSON array of four user objects, displayed in a "Pretty" format:

```
1  [  2    {  3      "Username": "user 1",  4      "Password": "password update",  5      "Fullname": "full name update",  6      "IsActive": false  7    },  8    {  9      "Username": "user 2", 10      "Password": "password 2", 11      "Fullname": "fullname 2", 12      "IsActive": true 13    }, 14    { 15      "Username": "user 3", 16      "Password": "password 3", 17      "Fullname": "fullname 3", 18      "IsActive": true 19    }, 20    { 21      "Username": "user 4", 22      "Password": "password 4", 23      "Fullname": "fullname 4", 24      "IsActive": true 25    }, 26  ]
```



# Thiết kế Result API - 13

## 5.4. DELETE METHOD

- Thêm vào hàm **DeleteUser** bằng cách truyền vào User và so sánh username.

### TIẾN HÀNH GỬI DELETE REQUEST

- Dữ liệu trả về user 2 đã bị Delete ra khỏi danh sách

# Thiết kế Result API - 14

```
// DELETE api/Users/DeleteUser
[HttpDelete]
public HttpResponseMessage DeleteUser (Users u) {
    try {
        var list = AllUser();
        int index = list.FindIndex(p => p.Username == u.Username);
        list.RemoveAt(index);
        return Request.CreateResponse(HttpStatusCode.OK, list);
    }
    catch (Exception) {
        return Request.CreateResponse(HttpStatusCode.BadRequest);
    }
}
```

# Thiết kế Result API - 15

The screenshot displays a REST client interface with a DELETE request and its response.

**Request:**

- Method: DELETE
- URL: `https://localhost:44398/api/User/DeleteUser`
- Body (JSON):

```
{  "username": "user 2",  "password": "password",  "fullname": "fullname",  "isactive": true}
```

**Response:**

- Status: 200 OK
- Time: 7 ms
- Size: 738 B
- Body (JSON):

```
[  {    "Username": "user 1",    "Password": "password 1",    "Fullname": "fullname 1",    "IsActive": true  },  {    "Username": "user 3",    "Password": "password 3",    "Fullname": "fullname 3",    "IsActive": true  },  {    "Username": "user 4",    "Password": "password 4",    "Fullname": "fullname 4",    "IsActive": true  },  {    "Username": "user 5",    "Password": "password 5",    "Fullname": "fullname 5",    "IsActive": true  }]
```

# Thiết kế Result API - 16

- **Bước 6:** Gọi API Lên Controller. Sau khi đã tạo xong 1 API đơn giản, chúng ta sẽ gọi API lên controller để hiển thị lên View giao diện
- Ở đây, chúng ta phải sử dụng 1 class gọi Request để có thể gọi API lên được, với **HttpClient**.
- Mượn tạm HomeController để hiển thị trên trang Home
- Chúng ta sẽ sử dụng phương thức async/await để có thể xử lý bất đồng bộ cho HttpClient

# Thiết kế Result API - 17

```
public async Task<ActionResult> Index(Users user) {
    ViewBag.Title = "Home Page";
    var list = await GetAllUser();
    if (list != null)                // Nếu list user khác null thì trả về View có chứa list
    return View(list);
    return View(null);
}

private async Task<List<Users>> GetAllUser() // Hàm Gọi API trả về list user
{
    string baseUrl = Request.Url.Scheme + "://" + Request.Url.Authority + Request.ApplicationPath.TrimEnd('/') + "/";
    // Lấy base uri của website
    using (var httpClient = new HttpClient()) {
        HttpResponseMessage res = await httpClient.GetAsync(baseUrl + "api/User/GetAllUser");
        if (res.StatusCode == System.Net.HttpStatusCode.OK) {
            List<Users> list = new List<Users>();
            list = res.Content.ReadAsAsync<List<Users>>().Result; return list;
        }
    }
    return null;
}
```

# Thiết kế Result API - 18

```
@{ ViewBag.Title = "All User"; }  
@model  
IEnumerable<Writeup_WebAPI.Models.Users>  
<div class="container">  
<h1>@ViewBag.Title</h1>  
@{  
    int i = 1;  
    foreach (var item in Model) {  
        <h4>User @i</h4>  
        <h5>@item.Username</h5>  
        <h5>@item.Password</h5>  
        <h5>@item.Fullname</h5>  
        <h5>@item.IsActive</h5>  
        i++;  
    }  
}  
</div>
```



# ASP.NET Web API sử dụng SQL Server

- **Bước 1:** Tạo cơ sở dữ liệu cần thiết
  - Tạo cơ sở dữ liệu có tên **WEBAPI\_DB**
  - Sau đó tạo bảng **Employees** và chèn vào đó một chút dữ liệu để test dễ dàng hơn.

ID	FirstName	LastName	Gender	Salary
1	Pranaya	Rout	Male	60000
2	Anurag	Mohanty	Male	45000
3	Preety	Tiwari	Female	45000
4	Sambit	Mohanty	Male	70000
5	Shushanta	Jena	Male	45000
6	Priyanka	Dewangan	Female	30000
7	Sandeep	Kiran	Male	45000
8	Shudhanshu	Nayak	Male	30000
9	Hina	Sharma	Female	35000
10	Preetiranjana	Sahoo	Male	80000

## ASP.NET Web API sử dụng SQL Server - 2

- **Bước 2:** Tạo một dự án API Web ASP.NET mới với tên **EmployeeService**. Chọn mẫu dự án **WEB API** và nhấp vào nút **OK**.
  - Nhấp chuột phải vào thư mục **Models** và sau đó chọn Add - New Item và từ các “Add New Item” cửa sổ chọn “Data” từ khung bên trái và sau đó là chọn **ADO.NET Entity Data Model** từ phần panel ở giữa, đặt tên **EmployeeDataMode** và nhấn Add. Chọn tùy chọn **EF Designer from database** và nhấn Next.
  - Sửa đổi Chuỗi kết nối là **EmployeeDBContext** và nhấp vào nút Next.
  - Trên màn hình tiếp theo, chọn **Entity Framework 6.x**. Bước này có thể là tùy chọn nếu đang sử dụng phiên bản cao hơn của studio.

## ASP.NET Web API sử dụng SQL Server - 3

- **Bước 3:** Thêm Trình điều khiển API Web (Web API Controller)
  - Nhấp chuột phải vào thư mục Controllers trong dự án EmployeeService và chọn Add – Controller
  - Sau đó, chọn **Web API 2 Controller – Empty** và sau đó nhấp vào nút Add.
  - Trên bộ màn hình tiếp theo, Controller Name là **EmployeesController** và bấm vào nút Add.

# ASP.NET Web API sử dụng SQL Server - 4

```
public class EmployeesController : ApiController {
    public IEnumerable Get() {
        using (EmployeeDbContext db = new EmployeeDbContext())
        {
            return db.Employees.ToList();
        }
    }
    public Employee Get(int id) {
        using (EmployeeDbContext db = new EmployeeDbContext())
        {
            return db.Employees.FirstOrDefault(e => e.ID == id);
        }
    }
}
```

# Authentication và Authorize trong Web API

- **Bước 1:** Tạo Project Web Application với tên Writeup\_WebAPI. Chọn **Web API**. (Sử dụng như bài 1)
- **Bước 2:** Hạn chế truy cập với Authorize Attribute. Vào web.config

```
<authentication mode="Forms">  
<forms loginUrl="~/Account/Login" timeout="2880" />  
</authentication>
```

- Với form authentication được kích hoạt, nếu người dùng cố gắng truy cập vào một trang cần được xác thực, họ sẽ được chuyển hướng đến trang login (Account/Login) nhập tên người dùng và mật khẩu

## Authentication và Authorize trong Web API - 2

- **Bước 3:** Thực hiện chức năng lưu lại trạng thái đăng nhập. Tạo 3 class gồm **User.cs** để lưu thông tin người dùng, **LoginModel.cs** để lưu thông tin đăng nhập, **UserRepository.cs** để thực hiện truy vấn và thao tác với cơ sở dữ liệu

```
public class User
{
    public int Id { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
    public string Fullname { get; set; }
    public bool IsActive { get; set; }
}
```

```
public class LoginModel
{
    public string Username { get; set; }
    public string Password { get; set; }
    public bool RememberMe { get; set; }
}
```



# Authentication và Authorize trong Web API - 3

```
public class UserRepository
{
    private static List<User> _userList = new List<User>
    {
        new User{ Id = 1, Username = "admin", Password = "123"},
        new User{ Id = 2, Username = "quangle", Password = "123"}
    };

    public bool Validate(string username, string password)
    {
        var user = _userList.FirstOrDefault(t => t.Username == username);
        if (user == null)
            return false;
        if (user.Password == password) return true;
        return false;
    }
}
```

# Authentication và Authorize trong Web API - 4

- **Bước 4:** Thực hiện việc Login trên AccountController

```
private UserRepository _userRepository;  
  
public AccountController()  
{  
    _userRepository = new UserRepository();  
}  
  
[HttpGet]  
public ActionResult Login()  
{  
    return View();  
}
```

# Authentication và Authorize trong Web API - 5

```
[HttpPost]
[AllowAnonymous]
public ActionResult Login(LoginModel model, string returnUrl)
{
    if (!_userRepository.Validate(model.UserName, model.Password))
    {
        FormsAuthentication.SetAuthCookie(model.UserName, model.RememberMe);
        return RedirectToLocal(returnUrl);
    }
    // If we got this far, something failed, redisplay form
    ModelState.AddModelError("", "The user name or password provided is
incorrect.");
    return View(model);
}
```

# Authentication và Authorize trong Web API - 6

```
[HttpPost]
public ActionResult LogOff()
{
    FormsAuthentication.SignOut();
    return RedirectToAction("Index", "Home");
}

private ActionResult RedirectToLocal(string returnUrl)
{
    if (Url.IsLocalUrl(returnUrl))
    {
        return Redirect(returnUrl);
    }
    else
    {
        return RedirectToAction("Index", "Home");
    }
}
```

## Authentication và Authorize trong Web API - 7

- **Bước 5:** Để phân quyền thì thêm **[Authorize]** vào phương thức hoặc cả Controller