Lab 7: ASP NET Core

Bài 1:

Thực hành: Tạo chức năng tìm kiếm trong ASP.NET Core MVC kết hợp với thư viện jQuery AJAX

JQUERY

Cú pháp jQuery

Cú pháp cơ bản: **$(selector).action()**

Trong đó:

- $ chỉ định định nghĩa/truy cập jQuery
- selector: truy vấn hoặc tìm các phần tử HTML elements
- jQuery action() hành động thực hiện trên các phần tử.

Ví dụ:

$("p").hide() - hides all <p> elements.

$(".test").hide() - hides all elements with class="test".

$("#test").hide() - hides the element with id="test"

jQuery với HTML:

- text() - thiết lập hoặc trả về về nội dung text của các phần tử được lựa chọn.
- html() - thiết lập hoặc trả về về nội dung của cácphần tử được lựa chọn (bao gồm HTML markup)
- val() - thiết lập hoặc trả về giá trị của form fields
- attr() – phương thức dùng để thiết lập hoặc lấy giá trị của thuộc tính.
- append() - Chèn nội dung vào cuối phần tử đã chọn

Một số hàm thường dùng của Jquery:

.on()

Cú pháp:

```
.on( events [, selector ] [, data ], handler )
```

Ví dụ:

Attach a click event to the <p> element:

```
$("p").on("click", function(){
  alert("The paragraph was clicked.");
});
```

Đoạn mã trên sẽ thiết lập khi click chuột vào thẻ p sẽ kích hoạt hàm alert (hiển thị) nội dung của thẻ p.

.html()

.html(): Lấy nội dung HTML của thành phần, hoặc gán giá trị HTML cho thành phần.

Cú pháp:

Trả về nội dung:

```
$(selector).html()
```

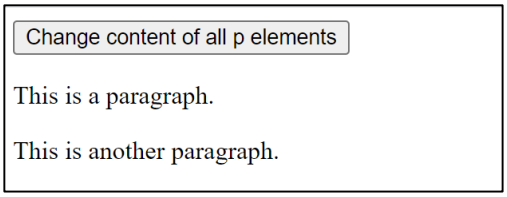
Thiết lập nội dung:

```
$(selector).html(content)
```

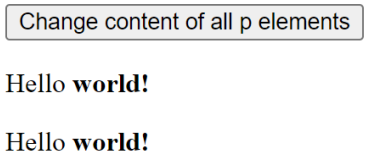
Ví dụ:

```
$("button").click(function(){
  $("p").html("Hello <b>world</b>!");
});
```

Thực hiện sự kiện click chuột vào button:



Kết quả:



each()

Cú pháp:

```
$(selector).each(function(index,element))
```

.each(): Thực hiện một hành động cho mỗi phần tử.

Ví dụ:

```
const numbers = [1, 2, 3, 4, 5];
$.each(numbers, function(index, value){
  console.log(`${index}: ${value}`);
});
```

Kết quả:

```
0:1
1:2
2:3
3:4
4:5
```

Phương thức Ajax Jquery.

ajax() method: Được sử dụng để thực hiện một AJAX (asynchronous HTTP) request.

Cú pháp: **$.ajax({*name:value, name:value,...*})**

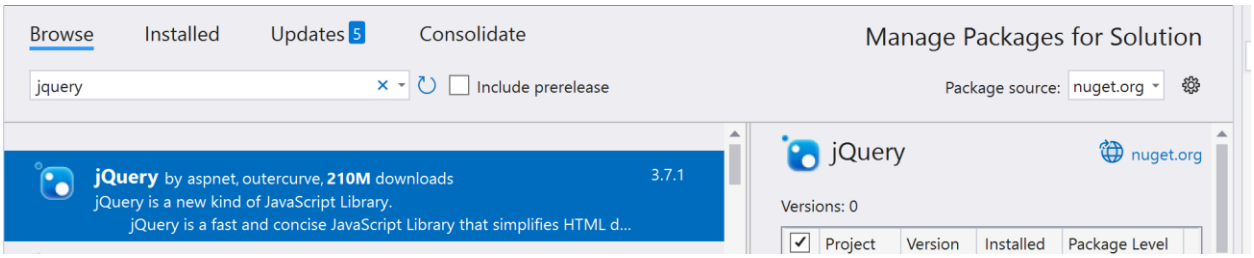
với một số cặp names/values:

- url: đường dẫn đến nơi lấy kết quả
- type: post hoặc get
- dataType: kiểu dữ liệu trả về, có thể là json, xml, script hoặc text
- data: Chỉ định dữ liệu được gửi đến Server.
- success : là hàm xử lý kết quả trả về, nó có tham số lưu trữ kết quả trả về.

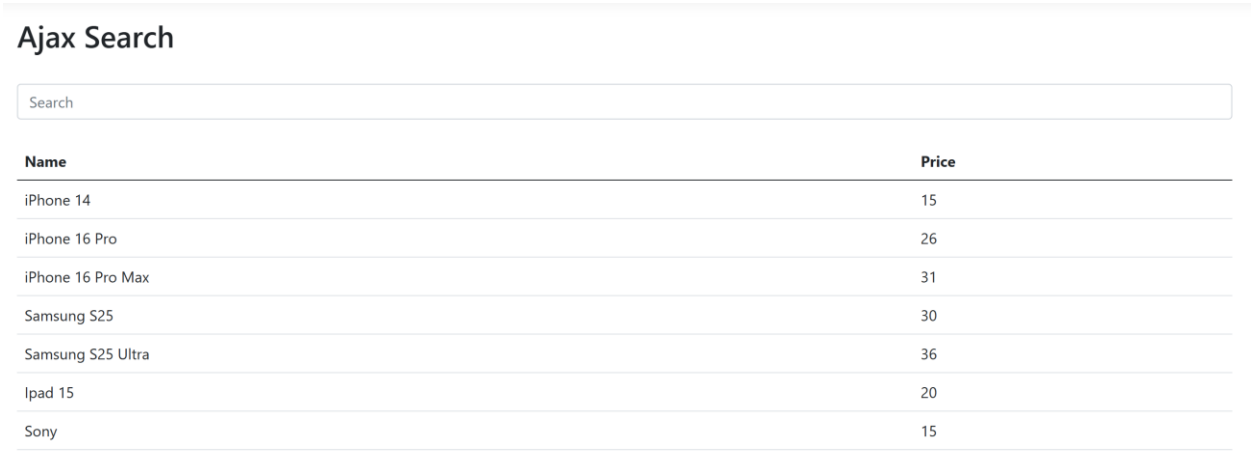
Ví dụ: Sử dụng Jquery Ajax POST

```
$(document).ready(function () {
       // Add the page method call as an onclick handler for the div.
       $("#Result").click(function () {
           $.ajax({
               type: "POST",
               url: "Home/Index",
               data: { someParameter: "some value" },
               contentType: "application/json; charset=utf-8",
               dataType: "json",
               success: function (msg) {
                   // Replace the div's content with the page method's return.
                   $("#Result").text(msg.d);
               }
           });
       });
   });
```

Cài đặt thư viện Jquery cho dự án ASP NET Core:



Yêu cầu hiện thực chức năng tìm kiếm:



Kết quả tìm kiếm:

Ajax Search

ip

Name	Price
iPhone 14	15
iPhone 16 Pro	26
iPhone 16 Pro Max	31
Ipad 15	20

Bài 2:

Quản lý Khoa và bệnh nhân trong một bệnh viện - Web API bằng .NET Core và SQL Server

Xây dựng một Web API để quản lý thông tin về các khoa và bệnh nhân trong phạm vi bệnh viện sử dụng .NET Core và SQL Server.

Yêu cầu:

Xây dựng một ứng dụng Web API bằng .NET Core.

Sử dụng SQL Server để lưu trữ thông tin về các khoa và bệnh nhân.

Thiết kế cơ sở dữ liệu gồm các bảng sau:

- Bảng "**Department**" để lưu thông tin về các khoa trong bệnh viện. Các trường gồm: ID (int), Name (string), Description (string).
- Bảng "**Patient**" để lưu thông tin về các bệnh nhân trong bệnh viện. Các trường gồm: ID (int), NamePatient(string), Age (int), Gender (string), Address (string), PhoneNumber (string), DerpartmentID (int) - khóa ngoại tham chiếu tới khoa mà bệnh nhân đang điều trị.

Xây dựng các endpoint sau:

- GET $/\mathrm{api}/\mathrm{departments}$: Trả về danh sách tất cả các khoa trong bệnh viện.
- GET $/\mathrm{api}/\mathrm{departments}/\{\mathrm{id}\}$: Trả về thông tin về một khoa cụ thể dựa trên ID.
- POST $/\mathrm{api}/\mathrm{departments}$: Tạo mới một khoa trong bệnh viện.
- PUT $/\mathrm{api}/\mathrm{departments}/\{\mathrm{id}\}$: Cập nhật thông tin của một khoa cụ thể dựa trên ID.
- DELETE $/\mathrm{api}/\mathrm{departments}/\{\mathrm{id}\}$: Xóa một khoa cụ thể dựa trên ID.
- GET $/\mathrm{api}/\mathrm{patients}$: Trả về danh sách tất cả các bệnh nhân trong bệnh viện.
- GET $/\mathrm{api}/\mathrm{patients}/\{\mathrm{id}\}$: Trả về thông tin về một bệnh nhân cụ thể dựa trên ID.
- POST $/\mathrm{api}/\mathrm{patients}$: Tạo mới một bệnh nhân trong bệnh viện.
- PUT $/\mathrm{api}/\mathrm{patients}/\{\mathrm{id}\}$: Cập nhật thông tin của một bệnh nhân cụ thể dựa trên ID.
- DELETE $/\mathrm{api}/\mathrm{patients}/\{\mathrm{id}\}$: Xóa một bệnh nhân cụ thể dựa trên ID.

Các API endpoint cần trả về dữ liệu dưới dạng JSON.

Sử dụng Entity Framework Core để thực hiện các thao tác CRUD với cơ sở dữ liệu.

Gợi ý:

Cấu hình chuỗi kết nối Database trong file appsettings.json:

```
"ConnectionStrings": {
    "DbConnection":"Server=YourMachineName;Database=YourDB;Trusted_Connection=True;MultipleActiveResultSets=True;TrustServerCertificate=True;"
}
```

Cài đặt các Package, sử dụng EF Core Code First tạo các bảng và mối quan hệ giữa các bảng như các bài Lab trước đã học.

Thêm class DbContext:

```
public class HospitalDbContext:DbContext
{
    public HospitalDbContext(DbContextOptions options): base(options) {

    }

    public DbSet<Patient> Patients { get; set; }
    public DbSet<Department> Department { get; set; }
}
```

File Program.cs

```
var connectionString = builder.Configuration.GetConnectionString("DbConnection");
builder.Services.AddDbContext<YourDBContext>(options => options.UseSqlServer(connectionString));
```

Lưu ý: Hai lệnh trên nằm trước lệnh $\mathrm{var\ app} = \mathrm{builder.Build()}$; trong file Program.cs

Chạy lệnh cập nhập cơ sở dữ liệu:

$\mathrm{add\text{-}migration\ InitDb}$

$\mathrm{update\text{-}database}$

Sử dụng DbContext :

```
public class MyController
{
    private readonly HospitalDbContext _hospitalDbContext;

    public MyController(HospitalDbContext context)
    {
        _hospitalDbContext = context;
    }
}
```

Hiện thực các Endpoint theo yêu cầu của đề bài:

Ví dụ:

```
[HttpPost]
 public IActionResult PostDepartment(Department department)
 {
     _hospitalDbContext.Add(department);
     _hospitalDbContext.SaveChanges();
     return Ok("Create susscess ");
 }
```

```
[HttpGet("{id}")]
public IActionResult GetDepartment(int id)
{
    var department = _hospitalDbContext.Department.Find(id);
    if(department == null)
    {
        return NotFound($"Department with id = {id} is not found");
    }
    return Ok(department);
}
```

Sử dụng Swagger (OpenAPI) được tích hợp kiểm tra các Endpoint đã hiện thực. Lưu dự án.