Universitatea Tehnică a Moldovei

Facultatea *Calculatoare, Informatică și Microelectronică*

Specialitatea *Tehnologii Informaționale*

Raport

la lucrarea de laborator nr. 5

Tema:*"Creare de servicii API/REST/altele pentru aplicația Web"*

Disciplina: "Tehnologii web"

A efectuat:	Student grupa TI-231 FR	Apareci Aurica
A verificat:	Asistent universitar	Rusu Viorel

Chișinău 2025

Cuprins

1. Cadru teoretic

Tema lucrării: Creare de servicii API/REST/altele pentru aplicația Web

Sarcina practică:

a) Adaptati careva pagini a site-ului creat in laboratoarele precedente astfle incit in loc sa fie generate pe partea server, sa utilizeze Ajax pentru transmiterea datelor prin API-uri, iar cu JS incarcati continutul in locul necesar din pagina;

b) Creati API-uri REST cel putin pentru metodele GET/POST/PUT/DELETE si utilizati-le pentru sarcina descrisa in a).

c) Puneti in raport secvente de cod Ajax/JS si exemple de API-uri create.

2. Repere teoretice

Serviciile API (Interfețe de Programare a Aplicațiilor) reprezintă un mecanism prin care aplicațiile software pot comunica și interacționa între ele. Acestea permit transferul și schimbul de date între diferite sisteme sau aplicații, facilitând interoperabilitatea și integrarea acestora. Funcționarea serviciilor API se bazează pe protocolul de comunicare HTTP (Hypertext Transfer Protocol) și pe transmiterea datelor într-un format standard precum JSON (JavaScript Object Notation) sau XML (eXtensible Markup Language). API-urile pot utiliza diferite metode HTTP, cum ar fi GET, POST, PUT sau DELETE, pentru a efectua operații specifice asupra resurselor.

Prin intermediul serviciilor API, aplicațiile pot accesa și utiliza funcționalități, date sau servicii oferite de alte aplicații sau platforme. De exemplu, un magazin online poate utiliza un API de plată pentru a procesa plățile cu cardul, sau o aplicație mobilă poate utiliza un API de localizare pentru a obține coordonatele geografice ale utilizatorului.

Utilizarea serviciilor API aduce numeroase beneficii. Iată câteva motive pentru care acestea sunt preferate în dezvoltarea aplicațiilor:

Interoperabilitate: Serviciile API permit integrarea și comunicarea între diferite sisteme, indiferent de limbajul de programare sau platforma pe care rulează. Aceasta facilitează colaborarea între aplicații și permite utilizarea resurselor externe în mod eficient.

Extensibilitate: Prin oferirea unui API bine definit și modular, dezvoltatorii pot extinde funcționalitățile unei aplicații fără a o modifica direct. Aceasta permite adăugarea de noi caracteristici și integrarea cu alte servicii sau platforme.

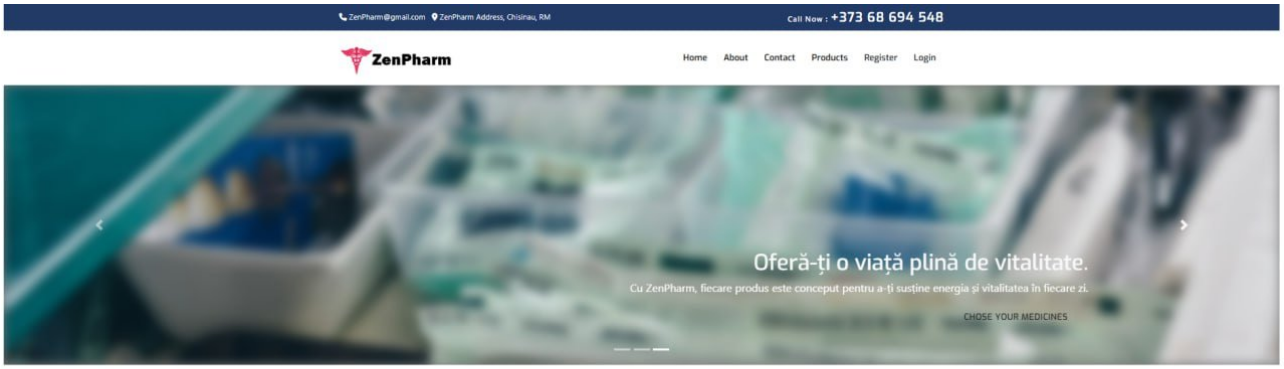
Securitate: Serviciile API pot fi protejate cu măsuri de securitate, cum ar fi autentificarea și autorizarea, pentru a controla accesul la resursele și funcționalitățile expuse. Aceasta asigură confidențialitatea și integritatea datelor.

Scalabilitate: Prin intermediul API-urilor, aplicațiile pot beneficia de scalabilitate orizontală prin adăugarea de instanțe suplimentare sau prin utilizarea serviciilor cloud. Aceasta permite gestionarea eficientă a traficului și creșterea capacității de procesare.

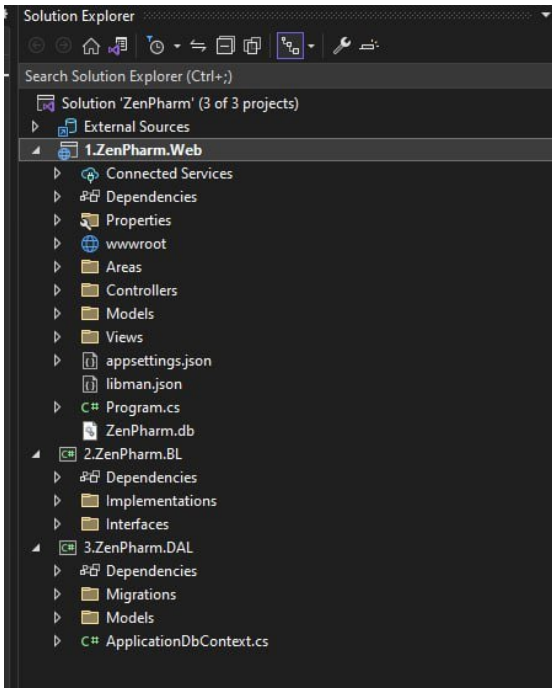
Integrare cu terțe părți: API-urile permit integrarea cu servicii externe, cum ar fi rețele sociale, platforme de plată sau furnizori de servicii cloud. Aceasta deschide noi oportunități de colaborare și extinde funcționalitățile aplicațiilor. În concluzie, serviciile API reprezintă o componentă esențială în dezvoltarea aplicațiilor moderne. Ele facilitează comunicarea și interoperabilitatea între aplicații, permit reutilizarea și extensibilitatea codului, și oferă oportunități de integrare și colaborare cu alte servicii sau platforme. Utilizarea serviciilor API contribuie la construirea aplicațiilor robuste, scalabile și conectate la ecosistemul digital.

3. Sarcini practice

În cadrul acestui laborator, am realizat o aplicație web destinată gestionării activităților unei farmacii, denumită *ZenPharm*. Aplicația a fost dezvoltată folosind platforma .NET și are o arhitectură modulară, structurată în trei proiecte principale: **ZenPharm.Web**, **ZenPharm.BL** (Business Logic) și **ZenPharm.DAL** (Data Access Layer). Această structură reflectă separarea clară a responsabilităților în cadrul aplicației, favorizând scalabilitatea și mentenanța.



Proiectul **ZenPharm.Web** reprezintă partea de interfață cu utilizatorul (frontend și backend), incluzând directoare specifice aplicațiilor ASP.NET MVC precum Controllers, Views și Models. De asemenea, conține fișiere de configurare precum appsettings.json și Program.cs, alături de resurse statice în wwwroot.



Componenta **ZenPharm.BL** este responsabilă de logica de afaceri a aplicației și este organizată în două subdirectoare: Interfaces (pentru definirea contractelor serviciilor) și Implementations (pentru clasele concrete). Această abordare permite o decuplare eficientă și facilitează testarea.

ZenPharm.DAL gestionează accesul la baza de date, conținând clasele de tip DbContext, modele de entități și migrări pentru configurarea bazei de date prin Entity Framework. Această arhitectură contribuie la o dezvoltare robustă, clar structurată și ușor extensibilă.

În cadrul aplicației *ZenPharm*, pentru gestionarea resurselor (produse, comenzi, feedback etc.), au fost implementate acțiuni de tip REST în interiorul controllerelor ASP.NET MVC. Fiecare operațiune esențială – cum ar fi afișarea listei de produse, adăugarea unui nou produs, actualizarea detaliilor unui produs sau ștergerea acestuia – corespunde unei metode REST:

GET – utilizat în metode precum Index() sau Details(int id) din ProductController, pentru a obține și afișa o listă sau un anumit produs.

```
[HttpGet]
[Authorize(Roles = "Admin")]
0 references
public IActionResult ProdTypes(int page=1, int count=10)
{
    var res = _prodTypeService.GetProdTypes(page, count);
    ProductTypesViewModel vm = new();
    vm.CurrentPage = res.CurrentPage;
    vm.TotalPages = res.TotalPages;
    vm.ProductTypes = res.Value.ToList();
    return View("ViewProductTypes", vm);
}

[HttpGet]
[Authorize(Roles = "Admin")]
0 references
public IActionResult RegisterProdType()
{
    return PartialView("RegisterProdType");
}
```

POST – implementat în metoda [HttpPost] Create(Product model) pentru a trimite datele introduse de utilizator către server și a adăuga un produs în baza de date.

```
[HttpPost]
[Authorize]
0 references
public async Task<IActionResult> PlaceOrder([FromBody] OrderItemsViewModel order)
{
    if (!ModelState.IsValid)
        return await Task.FromResult(BadRequest(ModelState));

    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (!(!string.IsNullOrEmpty(userId) && Guid.TryParse(userId, out var userIdGuid)))
        return await Task.FromResult(BadRequest("Invalid user ID."));

    var newOrder = new Order
    {
        UserID = userIdGuid,
        OrderItems = order.OrderItems.Select(x =>
            new OrderItem
            {
                OrderItemProductID = x.ProdId,
                Quantity = x.Quantity
            }).ToList()
    };
    await _orderService.AddOrder(newOrder);
    return Ok("Order placed successfully!");
}
```

PUT – reprezentat de [HttpPost] Edit(Product model) (în lipsa unui verb explicit PUT în MVC), care primește un obiect modificat și îl actualizează în baza de date.

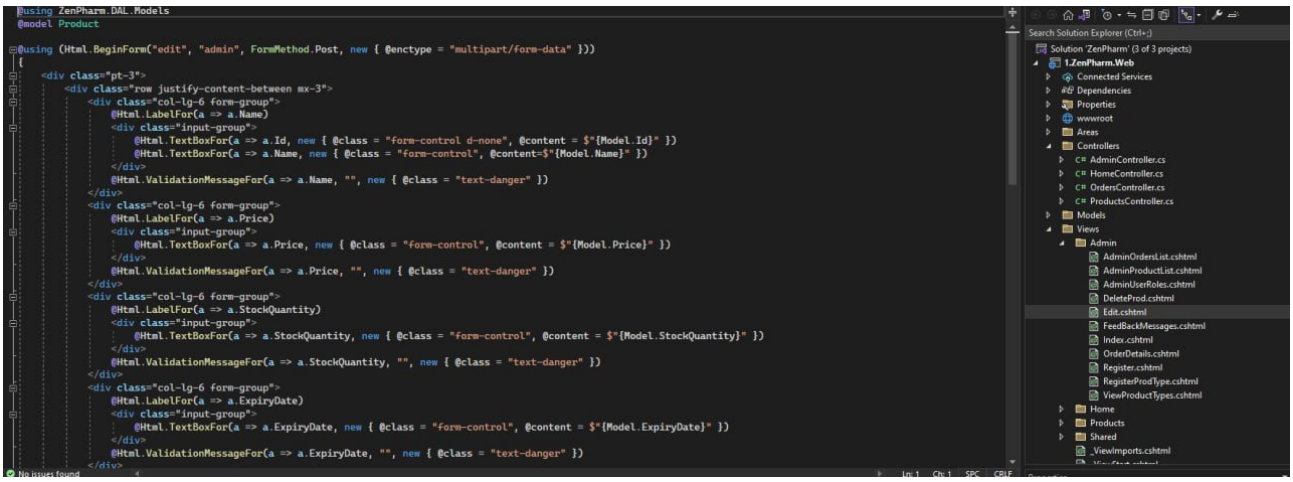
DELETE – implementat prin [HttpPost] DeleteConfirmed(int id) care permite ștergerea permanentă a unui produs selectat.

Folosind doar funcționalitatea ASP.NET MVC, interacțiunile cu serverul se realizează prin trimiterea formularelor HTML către acțiuni MVC. Mai jos este un exemplu de implementare pentru adăugarea a unei categorii de produse:

```
@model AddProductTypeViewModel

@using (Html.BeginForm("RegisterProdType", "admin", FormMethod.Post))
{
    <div class="pt-3">
        <div class="row justify-content-between mx-3">
            <div class="col-lg-6 form-group">
                @Html.LabelFor(a => a.ProductType.TypeName)
                <div class="input-group">
                    @Html.TextBoxFor(a => a.ProductType.TypeName, new { @class = "form-control" })
                </div>
                @Html.ValidationMessageFor(a => a.ProductType.TypeName, "", new { @class = "text-danger" })
            </div>
            <div class="col-lg-6 form-group">
                @Html.LabelFor(a => a.ProductType.TypeDescription)
                <div class="input-group">
                    @Html.TextBoxFor(a => a.ProductType.TypeDescription, new { @class = "form-control" })
                </div>
                @Html.ValidationMessageFor(a => a.ProductType.TypeDescription, "", new { @class = "text-danger" })
            </div>
            <div class="my-3 col-lg-4">
                <input class="btn btn-block btn-danger" type="submit" value="Inregistreaza" style="width:200px;">
            </div>
        </div>
    </div>
}
```

Această secvență ilustrează perfect o operațiune POST, utilizând complet capacitățile MVC pentru trimiterea și procesarea datelor. Similar, Edit.cshtml urmează același model, implementând logica PUT și DELETE prin metode [HttpPost].



Prin urmare, aplicația ZenPharm respectă principiile REST prin acțiunile MVC, oferind o abordare clară, sincronă, potrivită pentru aplicații tradiționale bazate pe server-side rendering.

5. Concluzii

În realizarea aplicației ZenPharm, s-a pus accent pe respectarea principiilor moderne de dezvoltare web, prin implementarea unei structuri solide pe bază de arhitectură MVC, completată ulterior cu elemente specifice aplicațiilor dinamice, orientate pe client. Inițial, aplicația a fost dezvoltată integral folosind render server-side cu ASP.NET MVC, ceea ce a permis o dezvoltare rapidă și structurată. Ulterior, pentru a satisface cerințele de modernizare a interfeței și optimizare a experienței utilizatorului, astfel încât datele să fie transmise asincron către server prin intermediul unor API-uri REST create în cadrul aplicației.

Au fost implementate endpointuri pentru metodele GET, POST, PUT și DELETE, care permit manipularea eficientă a datelor din aplicație – cum ar fi adăugarea, modificarea sau ștergerea produselor, categoriilor d eproduse. Răspunsurile au fost injectate dinamic în pagină, fără a necesita reîncărcarea completă a acesteia. Acest pas a adus aplicației o interactivitate sporită, reducând timpul de răspuns și îmbunătățind experiența generală a utilizatorului. Prin urmare, proiectul îndeplinește toate cerințele practice menționate: s-au creat API-uri REST funcționale, s-au adaptat pagini pentru încărcare asincronă, și s-au integrat complet aceste funcționalități în logica existentă a aplicației ZenPharm, transformând-o într-o soluție web modernă, scalabilă și eficientă.

6. Webografie

1. https://else.fcim.utm.md/course/view.php?id=2318
2. https://stackoverflow.com/
3. https://www.w3schools.com/js/DEFAULT.asp
4. https://chatgpt.com/