1. Protocolul HTTP. Comunicarea client-server

HTTP (Hypertext Transfer Protocol) este un protocol text si stateless ce asigura transmiterea datelor între client și serverul web, reprezentand protolul implicit al WWW (World Wide Web). HTTP este fundamentul pe care se bazează comunicarea web, permițând navigarea pe internet și accesarea resurselor web prin URL-uri.

Fiind un **protocol text**, primite schimbul de informații într-un format ușor de citit, atât de către oameni, cât și de catre browser.

Prin **protocol stateless** înțelegem ca fiecare cerere (request) trimisă de un client (de exemplu, un browser web) către un server este independentă; serverul nu păstrează nicio informație despre cererile anterioare ale acelui client. Aceasta simplifică implementarea serverelor, dar, în același timp, introduce provocări în gestionarea stării sesiunilor utilizatorilor, deoarece serverul nu își amintește cererile anterioare ale clientului. Pentru a depăși această limitare, tehnologii precum cookie-urile și sesiunile sunt folosite pentru a "memora" informații despre interactiunile din trecut.

Comunicarea **client-server** descrie modelul fundamental de interacțiune pe internet, unde "clientul" (de obicei, browser-ul web al utilizatorului) inițiază cereri către un "server" (un program care rulează pe un computer de la distanță), care apoi procesează cererea și trimite înapoi un răspuns. Acest model de comunicare este esențial pentru funcționarea aplicațiilor web, deoarece permite distribuirea sarcinilor de procesare între clienți și servere, optimizând astfel utilizarea resurselor și scalabilitatea aplicațiilor.

În contextul dezvoltării aplicațiilor web, înțelegerea acestui model de comunicare este crucială pentru proiectarea și implementarea eficientă a aplicațiilor. Tehnologiile moderne de dezvoltare web, inclusiv ASP.NET Core și Razor Pages, despre care vom discuta în secțiunile următoare, se bazează pe acest model client-server pentru a oferi experiențe web dinamice și interactivitate utilizatorilor.

2. Utilizarea Bazelor de Date în dezvoltarea aplicațiilor web

Utilizarea bazelor de date în dezvoltarea aplicațiilor web este esentiala pentru gestionarea eficientă a datelor. Bazele de date permit stocarea, interogarea și manipularea datelor într-un mod structurat (de exemplu, stocarea informatiilor unui cont de utilizator, gestionarea tranzacțiilor și orice altă funcționalitate ce implică reținerea informațiilor pe termen lung).

În contextul ASP.NET Core și al Razor Pages, utilizarea bazelor de date este facilitata prin Entity Framework Core, un ORM (Object-Relational Mapper) care permite dezvoltatorilor să lucreze cu baze de date utilizând obiecte .NET, fără a fi necesară scrierea explicită a interogărilor SQL.

În secțiunile următoare, vom explora mai detaliat crearea și structurarea unui proiect Razor Pages, inclusiv integrarea cu baze de date și gestionarea stării aplicației.

3. Ilustrarea comunicarii intre client, server si baza de date

O diagramă simplă care ilustrează comunicarea între client, server și baza de date (BD) ar putea arăta astfel:

1. **Clientul** (de exemplu, un browser web) trimite o cerere HTTP către **server** pentru a accesa o anumită pagină sau resursă.
2. **Serverul** primește cererea și, în funcție de natura acesteia, poate necesita să interogheze **baza de date** pentru a recupera sau actualiza informații necesare pentru a procesa cererea.
3. **Baza de date** procesează solicitarea serverului și returnează datele solicitate.
4. **Serverul** prelucrează datele returnate de baza de date, generează un răspuns adecvat (de exemplu, o pagină web), și îl trimite înapoi către **client**.
5. **Clientul** afișează răspunsul primit, permițând utilizatorului să vadă informațiile sau rezultatele acțiunii sale inițiale.

Această diagramă conceptualizează fluxul de bază al informațiilor și interacțiunea dintre cele trei componente principale ale unei aplicații web, insa, in practică, pot exista multe alte interacțiuni și procesări intermediare.

4. Introducere in Razor Pages

Razor Pages este o modalitate de structurare a unui proiect ASP.NET Core, simplificand dezvoltarea aplicațiilor web prin integrarea UI (interfața cu utilizatorul) și a logicii de server în aceeași fișier. Spre deosebire de MVC, care separă codul în Models, Views și Controllers, Razor Pages încurajează o structură mai simplă, fiind ideală pentru scenarii în care se doreste ca logica unei pagini sa fie concentrată într-un singur loc.

Pentru a crea un proiect Razor Pages de la zero, puteti urma acești pași simpli în Visual Studio:

1. Selectați "Create a new project" și alegeti template-ul "ASP.NET Core Web Application".
2. Numiti proiectul și selectați locația acestuia.
3. În fereastra "Create a new ASP.NET Core Web Application", selectați versiunea dorită a .NET Core.
4. După crearea proiectului, veți avea o structură de bază care include directorul **Pages**, unde fiecare pagină web este reprezentată de un fișier **.cshtml** (conține marcajul HTML și codul Razor) și, opțional, un fișier de model **.cshtml.cs** (conține backend-ul pentru pagina respectivă).

5. Structura unui proiect Razor Pages

Structura unui proiect Razor Pages este una intuitivă, oferind o alternativă eficientă la modelul MVC:

- **/Pages**: Directorul principal unde sunt stocate paginile Razor. Fiecare pagină este compusă dintr-un fișier **.cshtml** pentru markup și cod Razor, care conține in interiorul său (colapsat) și un fișier **.cshtml.cs** pentru codul C# care gestionează backend-ul.
- **/wwwroot**: Acest director conține resursele statice ale aplicației, precum CSS, JavaScript și imagini.
- **appsettings.json**: Fișier de configurare pentru setări specifice aplicației, inclusiv conexiuni la baze de date.
- **Program.cs**: Punctul de intrare al aplicației, unde este configurat și lansat serverul web.

6. MVC vs Razor Pages

În **MVC** (Model-View-Controller), logica este împărțită între modele (datele aplicației), vizualizări (interfața cu utilizatorul), și controlere (logica de manipulare a cererilor). Acest model încurajează separarea clară a codului, fiind ideal pentru aplicații complexe și mari.

În **Razor Pages**, pagina web și logica aferentă sunt integrate într-un singur loc, ceea ce facilitează managementul funcționalităților care sunt strâns legate de o anumită pagină. Acest model este adesea mai simplu și mai direct pentru dezvoltarea aplicațiilor mai mici sau a celor care necesită o structură mai simplă.

7. Fisierele .cshtml și .cshtml.cs

Fiecare pagină Razor este alcătuită din două fișiere principale:

1) Fișierul **.cshtml** care conține markup-ul HTML și codul Razor. Codul Razor este un mix între HTML și C#, permițând inserarea logicii de server direct în pagină. Acest lucru facilitează generarea dinamică a conținutului HTML pe baza datelor procesate de server.
2) Fișierul **.cshtml.cs**, cunoscut și ca fișierul "model de pagină" sau "code-behind", conține logica C# pentru pagina respectivă. Aici se gestionează evenimentele, se interacționează cu bazele de date și se procesează datele pentru a fi afișate în pagina **.cshtml** corespondentă.

Veți vedea ulterior cum împărțiți o aplicație astfel încât să fie modulară și extensibilă și vrem sa fie cât mai portabila (Windows/MacOS/Linux/Mobile/etc). Această separare între structura paginii și logica de backend permite o organizare clară a codului și facilitează testarea și dezvoltarea aplicației.

8. Template-ul Layout.cshtml

Fișierul **_Layout.cshtml** servește ca șablon comun pentru aplicația Razor Pages. Acesta definește structura de bază a paginilor web (de exemplu, head-ul, header-ul și footer-ul) și permite includerea de conținut specific paginii în sectiuni predefinite. Utilizarea unui layout comun asigură consistența interfeței și reduce duplicarea codului, deoarece elementele comune tuturor paginilor sunt definite într-un singur loc.

9. Directorul wwwroot și conținutul său

Directorul **wwwroot** găzduiește resursele statice ale aplicației, precum fișierele CSS, JavaScript, imaginile și fonturile. Aceste resurse sunt accesibile public și sunt trimise direct către browser fără a fi procesate de server. Organizarea resurselor statice în wwwroot facilitează gestionarea acestora și asigură că aplicația web este rapidă și **responsive**.

10. Fișierul appsettings.json

Fișierul **appsettings.json** este utilizat pentru configurarea aplicației ASP.NET Core, permițând definirea setărilor, cum ar fi string-urile de conexiune la baza de date, parametrii specifici mediului de execuție și alte opțiuni de configurare necesare aplicației. Aceste setări pot fi accesate din codul C# pentru a adapta comportamentul aplicației în funcție de nevoile specifice.

11. Fișierul Program.cs

Fișierul **Program.cs** este punctul de intrare al aplicației ASP.NET Core. Aici se configurează și se lansează aplicația web, inclusiv setarea middleware-urilor și serviciilor necesare. Codul din Program.cs definește cum sunt procesate cererile HTTP, configurând aspecte esențiale precum redirecționarea HTTPS, servirea fișierelor statice, rutarea și autorizarea.

12. Middleware în ASP.NET Core

Middleware-urile sunt componente software care sunt executate în cadrul pipeline-ului de procesare a cererilor HTTP în aplicațiile ASP.NET Core. Ele permit interceptarea și procesarea cererilor în diferite puncte, oferind o modalitate flexibilă de a adăuga funcționalități la aplicație. Iată câteva middleware-uri importante menționate în note:

- **app.UseHttpsRedirection()**: Acest middleware redirecționează automat cererile HTTP către HTTPS, asigurând că comunicația între client și server este securizată.
- **app.UseStaticFiles()**: Permite servirea fișierelor statice din directorul wwwroot. Este esențial pentru a include resursele precum CSS, JavaScript și imagini în aplicația web.
- **app.MapRazorPages()**: Configurează rutarea pentru paginile Razor, permițând aplicației să identifice care pagină trebuie servită în funcție de URL-ul cerut.
- **app.UseAuthorization()**: Adaugă suportul pentru autorizarea accesului la diferite părți ale aplicației. Acest middleware asigură că doar utilizatorii autorizați pot accesa resursele protejate.
- **app.Run()**: Marchează sfârșitul pipeline-ului de procesare a cererilor. Acesta pune în funcțiune aplicația, așteptând să primească și să proceseze cereri.

13. Razor Page = HTML + C#

Paginile Razor combină markup-ul HTML cu codul C# într-un eficient, permițând crearea de interfețe dinamice in aplicațiile web. Prin utilizarea sintaxei Razor, se pot injecta date din codul C# direct în HTML, facilitând generarea conținutului dinamic pe baza datelor procesate pe server.

14. Directiva @page

Directiva **@page** la începutul unui fișier **.cshtml** indică faptul că fișierul respectiv funcționează ca o pagină Razor, care poate procesa cereri HTTP. Această directivă transformă fișierul într-un endpoint (pagina concretă accesată) accesibil prin rutarea URL-urilor.

15. Directiva @model

Directiva **@model** definește tipul modelului de date care este utilizat într-o pagină Razor, permițând accesul la datele modelului direct în markup-ul HTML. Acesta facilitează afișarea și manipularea datelor în pagină.

16. Tag Helpers: asp-for, asp-page, asp-route-{numeVariabila}

Tag Helpers permit crearea unui cod HTML mai curat și mai expresiv, prin adăugarea de attribute specifice ASP.NET Core în tagurile HTML. Acestea includ:

- **asp-for**: Leagă un element de formular de o proprietate a modelului.
- **asp-page**: Generează URL-uri către alte pagini Razor.
- **asp-route-{numeVariabila}**: Adaugă parametri de rutare la URL-uri, facilitând transmiterea datelor între pagini.

17. Partials

Paginile parțiale (partials) sunt folosite pentru a refactoriza și reutiliza părți de interfață comună în mai multe pagini, îmbunătățind astfel manevrabilitatea și consistența aplicației.

18. [BindProperty]

Atributul **[BindProperty]** se aplică proprietăților din modelul de pagină pentru a le lega automat de datele trimise prin formular, facilitând colectarea și procesarea datelor de intrare de la utilizatori.

19. Atribute C# în general

Atributele C# oferă o metodă de adăugare de metadata la cod, care pot influența comportamentul în timpul execuției. În contextul paginilor Razor, sunt folosite pentru a controla aspecte precum binding-ul de date, validarea și autorizarea.

20. ModelState.IsValid

ModelState.IsValid verifică dacă datele trimise într-un formular respectă regulile de validare definite pentru model, fiind esențial pentru asigurarea integrității datelor înainte de a le procesa.