Mockipedia

Autor : Constantin Alixandroaie

Profesor Îndrumător : Ildiko Revnik

Cuprins

[1.Prezentare generala : 3](#_Toc31967904)

[Ce este ASP.NET? 3](#_Toc31967905)

[Ce este MVC? 4](#_Toc31967906)

[2. Prezentarea Modelului – Baza de date : 6](#_Toc31967907)

[Structura: 6](#_Toc31967908)

[Crearea bazei de date : 7](#_Toc31967909)

# 1.Prezentare generala :

## Ce este ASP.NET?

ASP.NET este un framework destinat mai multor platforme, open-source pentru a crea aplicații moderne bazate pe tehnologia cloud. Acesta poate fi folosit pentru a crea aplicații web direct compatibile pe mai multe dispozitive : PC, Tableta, Mobil.

ASP.NET Core MVC oferă funcții ajutătoare pentru a construi API-uri web și aplicații web:

* Modelul Model-View-Controller (MVC) vă ajută să faceți testarea API-urilor web și a aplicațiilor web.
* Paginile Razor sunt un model de programare bazat pe pagini care face construirea UI web mai ușoară și mai productivă.
* Markup-ul Razor oferă o sintaxă productivă pentru paginile Razor și pentru vizualizările MVC.
* „Tag helper” permite codului din partea serverului să participe la crearea și redarea elementelor HTML în fișiere Razor.
* Suportul integrat pentru mai multe formate de date și negocierea conținutului permite API-urilor dvs. web să ajungă la o gamă largă de clienți, inclusiv browsere și dispozitive mobile.
* Legarea modelului mapează automat datele de la cererile HTTP la parametrii metodei de acțiune.
* Validarea modelului efectuează automat validarea din partea clientului și din partea serverului.

ASP.NET Core integrează cu ușurință framework-uri pentru developmentul Client-site –adică front-end , script-uri etc.- incluzând Blazor, Angular, React si Bootstrap.

Toate motivele de mai sunt motive pentru care am optat crearea aplicației folosind ASP.NET

## Ce este MVC?

Model-view-controller (MVC) (din engleză, aproximativ: model-vizualizare-controlor) este un model arhitectural utilizat în ingineria software. Succesul modelului se datorează izolării logicii de business față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual sau/și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele.

Design:

* **Model**

Această parte a controllerului manipulează operațiunile logice și de utilizare de informație (trimisă dinainte de către rangul său superior) pentru a rezulta de o formă ușor de înțeles.

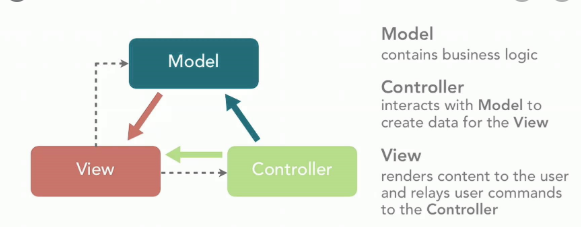
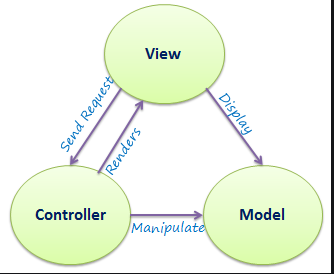
* **View**

Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controlor.

* **Controller**

Cu acest element putem controla accesul la aplicația noastră. Pot fi fișiere, scripturi (eng. scripts) sau programe, in general orice tip de informație permisă de interfață. În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp.

Mai jos am adăugat doua scheme simple pentru a ilustra modelul MVC.



In development acest „design pattern” se poate echivala astfel Model – Baza de date cu întreaga structura, View – Interfața grafica de unde utilizatorul observa datele interpretate din model într-un format inteligibil si trimite comenzi către Controller – totalitatea funcțiilor ce trebuie îndeplinite de aplicație. O parte din aceste funcții se numesc CRUD – Create , Read, Update, Delete – si stau la baza funcționării unei aplicații.

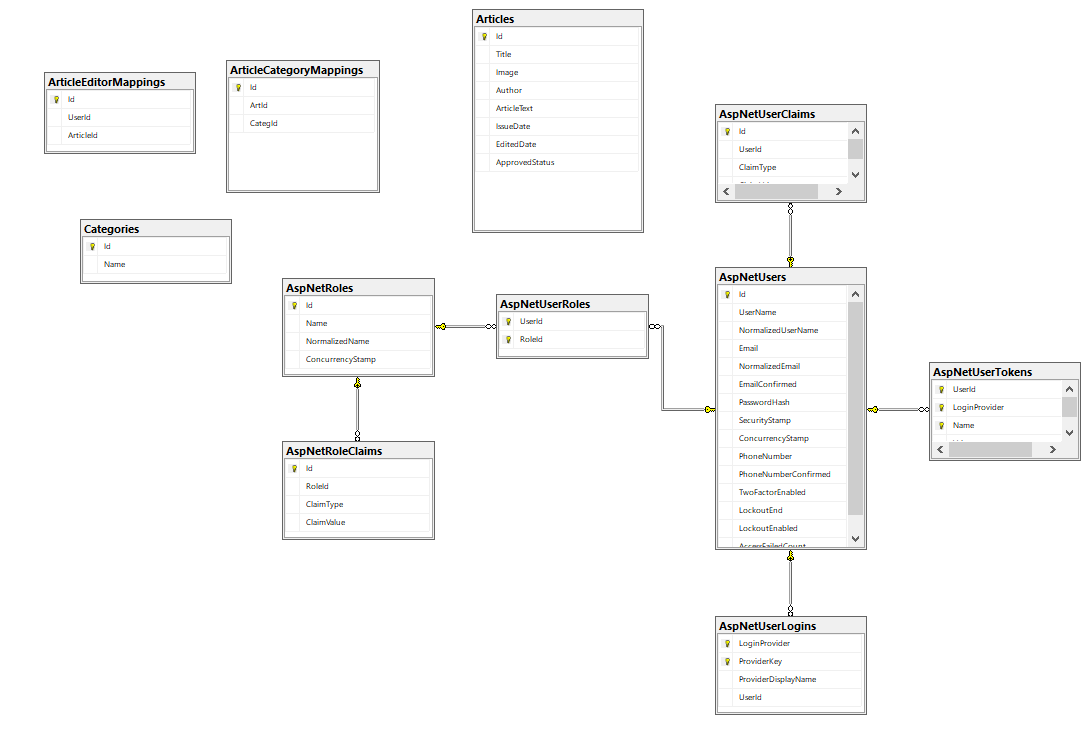
Toate aplicatiile folosesc aceste functii, dar majoritatea le compun si le adapteaza la modelul fiecarui bussines. De exemplu o functie cu scopul de a crea o factura contine operatii CREATE multiple : creare factura, creare client, creare asociere intre client si factura etc.

# 2. Prezentarea Modelului – Baza de date :

## Structura:

Baza de date este alcatuita din 11 tabele :

1. Tabele destinate managementului de utilizatori :
2. AspNetUsers;
3. AspNetUserClaims;
4. AspNetUserRoles;
5. AspNetUserTokens;
6. AspNetUserLogins;
7. AspNetRoles;
8. AspNetRoleClaims;



1. Tabele destinate managementului articolelor :
2. Articles;
3. Categories;
4. ArticleCategoryMappings;
5. ArticleEditorMappings;

Tabelele destinate managementului utilizatorilor sunt generate automat prin crearea aplicației si configurarea acesteia pentru conturi individuale de utilizatori. Aceste tabele acoperă toate funcționalitățile acoperite de framework, care sunt mai numeroase decât necesare in cazul nostru.

Aceste tabele vor conține datele utilizatorilor care își vor crea un cont pe site. Pentru fiecare utilizator daca este necesar se vor crea si asocieri intre id-ul utilizatorului si id-ul rolului-voi descrie mai târziu funcția care îndeplinește acest rol.

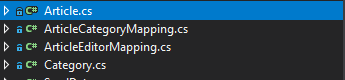
Tabelele destinate managementului articolelor contrin articolele scrise de utilizatori, categoriile, de asemenea date de utilizatori si asocierile necesare intre articole, categorii si editorii.

## Crearea bazei de date :

Baza de date a fost create „code first” : o metoda prin care se definesc entitatile ce vor fi depozitate in baza de date si relatiile dintre ele in cod iar apoi extrapolarea acestor in strunctura baze de date.

Aceasta metoda este preferata in frameworkul ASP.NET Core deoarece in acest fel programatorul este liber in crearea claselor si operatiilor pentru fiecare clasa fara a tine cont de restrictiile bazei de date, iar unde acestea exista, se vor programa si in clase astfel excluzand din cod orice eroare poate aparea atunci cand se incalca restrictiile din SQL.

Primul pas in crearea bazei de date este crearea claselor :



Clasele create sunt denumite identic cu tabelele de mai sus.

Article :



ArticleCategoryMapping



ArticleEditorMapping



Category



Pentru clase trebuie creat un obiect de tip IdentyDbContext-clasa de baza pentru baza de date- care va contine, pentru fiecare clasa un obiect de tip DbSet.

Un Microsoft.EntityFrameworkCore.DbSet poate fi utilizat pentru interogarea și salvarea instanțelor de TEntity. Interogările LINQ asupra unui Microsoft.EntityFrameworkCore.DbSet vor fi traduse în interogări împotriva bazei de date.

  Rezultatele unei interogări LINQ împotriva unui Microsoft.EntityFrameworkCore.DbSet vor conține rezultatele returnate din baza de date și pot să nu reflecte modificările făcute în contextul care nu a fost persistat până la baza de date. De exemplu, rezultatele nu vor conține entități nou adăugate și pot conține în continuare entități care sunt marcate pentru ștergere.

În funcție de baza de date utilizată, unele părți ale unei interogări LINQ sunt aplicate unui Microsoft.EntityFrameworkCore.DbSet poate fi evaluat în memorie în loc să fie tradus într-o interogare a bazei de date.

         Obiectele Microsoft.EntityFrameworkCore.DbSet sunt obținute de obicei de la o proprietate Microsoft.EntityFrameworkCore.DbSet pe un derivat Microsoft.EntityFrameworkCore. DbContext sau de la o metoda Microsoft.EntityFrameworkCore.DbContext.Set

      Clasa derivata din IdentityContext in cazul aplicatiei mele este urmatoarea :



Aceste clase sunt necesare si suficiente pentru crearea bazei de date. Baza de date se va crea prin folosirea unor comenzi de consola in Visual Studio.

Primul pas este crearea unei clase numita „migration” cu comanda :

dotnet ef migrations add <NumeMigratie>.

Aceasta comanda va crea clasa intr-un folder si o va genera automat din clasele descrise mai sus. Clasa generata pentru acest proiect este partial inclusa mai jos : 

Se pot observa metodele care creaza si modifica tabele pentru ca acestea sa se conformeze cu modelul descris de clasele precedente. Totodata aceasca clasa contine si functiile pentru a sterge baza de date daca este nevoie unde se executa o serie de comenzi „drop <tablename>”.

Aceasta clasa deocamdata nu poate fi folosita pentru stocare de date si toate comenzile din ea trebuie executate si asta se face cu comanda :

dotnet ef database update

Aceasta comanda va executa comenzile si va crea o baza de date locala pe serverul MSSQL care este instalat odata cu VisualStudio. Baza de date poate apoi fi examinata daca se doreste cu MsSql ServerManagement Studio, sau prin folosirea tabului server explorer din Visual Studio.

Daca baza de date locala nu este tinta noastra si dorim ca aceasi structura sa fie folosita pe alt server, putem folosi o alta comanda pentru a genera un script sql care poate fi rulat in orice server de baze de date SQL :

dotnet ef migrations script.

La acest pas avem baza de date generata tot ce trebuie sa facem este sa ne conectam la aceasta folosind un connection string. Proiectul acesta contine doua connection strings pentru ca la un moment dat va fi si publicat pe alt server cu o alta baza de date :



Pentru testare se pot face cateva interogari simple pentru a fi sigur ca baza de date creata se poate accesa din applicatie pentru orice operatie necesara.

# 3. Controller-ul

## Base Site Controller

Aceasta clasa contine toate functiile si proprietatile necesare unui controller. Astfel orice controller care va fi creat ulterior poate fi bazat pe structura acestuia si in acest fel nu va fi nevoie sa specificam din nou proprietatile si sa cream instante noi pentru aceeasi proprietate de mai multe ori.



## Article Controller :

Acest controller va contine codul pentru fiecare functie din controllerul de baza, plus cateva functii specifice care depind de clasa Article si ce se vrea sa fie afisat, sau creat.

Prima metoda de READ : GetAsync()

Preia toate articolele din baza de date si le salveaza intr-o lista de ArticleViewModel- o clasa separata folosita pentru afisarea datelor in VIEW. Voi prezenta mai in detaliu aceste clase si copul lor atunci cand voi prezenta VIEW.



A doua metoda READ : GetByIdAsync(int id)

Preia articolul cu Id-ul dat ca parametru si il stocheaza intr-un ARTICLEVIEWMODEL care apoi va fi afisat in VIEW.



In aceasta metoda inainte de a afisa verificam doua lucruri, primul este ca articolul sa existe, si astfel sa evitam trimiterea catre client o instanta de clasa NULL care poate sa blocheze interfata, si al doilea verificam daca articolul este aprobat de catre administrator.

Daca oricare din cele doua conditii nu este indeplinita, functia va returna o exceptie cu mesajul respectiv.

Functiile urmatoare sunt toate functii de citire specifice create pentru a trimite catre interfata doar anumite articole :

1. GetByLastAsync(string searchString)

Aceasta funtie preia ultimele 9 articole din baza de date in ordine cronologica pentru a fi afisate pe pagina principal ca cele mai noi articole.



Parametrul searchString este folosit pentru a cauta in baza de date, cand acesta este folosit numarul articolelor afisate nu mai este limitat la 9, ci la toate articolele care contrin acest parametru.

1. GetUnapprovedAsync()

Preia toate articolele din baza de date care nu sunt inca aprobate de catre administrator pentru afisarea lor si permiterea aprobarii de catre administrator din interfata.

