**UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA**

**Informatică aplicată și programare**

*Program postuniversitar de formare și dezvoltare profesională continuă*

Mockipedia

Enciclopedie online

**Constantin Alixandroaie**

Cuprins

[1.Prezentare generala : 4](#_Toc32230742)

[Mockipedia 4](#_Toc32230743)

[Ce este ASP.NET? 4](#_Toc32230744)

[Ce este Bootstrap 5](#_Toc32230745)

[Ce este MVC? 5](#_Toc32230746)

[2.Prezentarea Cerintelor Aplicatiei : 7](#_Toc32230747)

[1.Crearea bazei de date 7](#_Toc32230748)

[2.Codul ,Crearea functiilor principale indeplinite de aplicatie 7](#_Toc32230749)

[3. Crearea Interfatei de utilizator: 7](#_Toc32230750)

[4.Integrarea celor trei componente 7](#_Toc32230751)

[3. Structura Proiectului: 8](#_Toc32230752)

[4. Prezentarea Modelului – Baza de date : 9](#_Toc32230753)

[Structura: 9](#_Toc32230754)

[Crearea bazei de date : 10](#_Toc32230755)

[5. Controller-ul 15](#_Toc32230756)

[Base Site Controller 15](#_Toc32230757)

[Article Controller : 16](#_Toc32230758)

[UserController : 24](#_Toc32230759)

[6. Autentificare si Autorizare : 25](#_Toc32230760)

[7. View – Interfata cu utilizatorul 29](#_Toc32230761)

[Clasele ViewModel : 29](#_Toc32230762)

[Pagini Razor : 30](#_Toc32230763)

[Index.html : 30](#_Toc32230764)

[Article.cshtml: 32](#_Toc32230765)

[EditArticle : 35](#_Toc32230766)

[AddArticle : 36](#_Toc32230767)

[MyArticles : 38](#_Toc32230768)

[ApproveArticles : 38](#_Toc32230769)

[UserManagementPage: 39](#_Toc32230770)

[Pagini pentru managementul contului : 39](#_Toc32230771)

[8. Posibilitati de continuare si extindere : 40](#_Toc32230772)

[9. Bibliografie 41](#_Toc32230773)

[Site-uri : 41](#_Toc32230774)

[Carti 41](#_Toc32230775)

# 1.Prezentare generala :

## Mockipedia

Mockipedia este un site de tipul enciclopediilor online care permite utilizatorilor sa se documenteze si sa creeze articole pentru a ajuta alți utilizatori sa se documenteze despre subiecte multiple aparținând unui număr aproape nelimitat de subiecte.

Aplicația dezvoltata permite in principal utilizatorilor înregistrați sa scrie articole descriptive despre un anumit concept care apoi devine disponibil pentru orice fel de utilizator al aplicației, înregistrat sau nu.

Fiecare articol scris trece printr-un proces de „screening” prin care un administrator aproba validitatea articolului si îl face vizibil pentru toți utilizatorii. Atât timp cat articolul nu este aprobat acesta nu poate fi văzut de către utilizatorii aplicație ca o măsura de prevenire a spam-ului, si a articolelor menite sa arate conținut neacceptabil pentru aplicație.

Aplicația astfel conține multiple definiții pentru drepturile utilizatorilor : administrator, manager, editor, utilizator. Fiecare tip de utilizator are drepturi specifice rolului sau.

Pentru a îndeplini aceste condiții am ales sa dezvolt aplicatia in ASP.NET in paralel cu folosirea serverului de baze de date MS SQL Server si dezvoltarea interfetei grafice cu ajutorul paginilor Razor ca baza, HTML+CSS si Bootstrap.

Codul scris in C# va fi structura aplicatiei de pe server in totalitate si functiile de backend din partea clientului.

Paginile Razor sunt folosite pentru interfatarea simpla si productiva a formularelor cu codul de pe aplicatia client. Orice modifcare adusa acestor pagini este adusa folosind HTML+CSS.

## 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 Bootstrap

Bootstrap este un framework CSS gratuit și opend-source, direcționat către dezvoltarea web front-end receptivă și mobilă. Conține șabloane de design bazate pe CSS și JavaScript pentru tipografie, formulare, butoane, navigare și alte componente de interfață.

## 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.

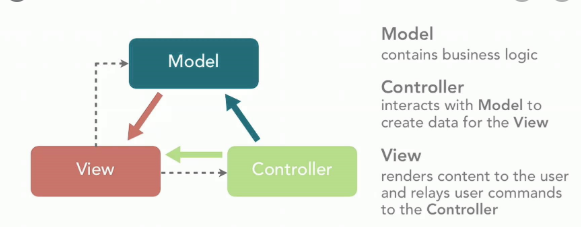
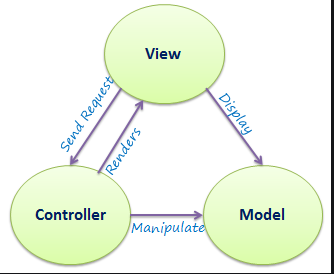
* **View**

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 Cerintelor Aplicatiei :

## 1.Crearea bazei de date

Baza de date va contine toate datele necesare aplicatiei. Este nevoie de o structura care sa permita crearea utilizatorilor, asocierea acestora cu roluri, crearea articlolelor, asocierea acestora cu autorul si editorii, inregistrarea statusului „Aprobat”.

De asemenea baza de date contine un tabel care memoreaza categoriile din care face parte un articol. Acest tabel incepe gol si este populat pe masura ce utilizatorii creeaza articole si completeaza campul asociat categoriei din care face parte articolul.

## 2.Codul ,Crearea functiilor principale indeplinite de aplicatie

Functiile aplicatiei vor fi de Creare, Editare, Citire si Stergere a articolelor si a categoriilor. Fiecare functie in parte va fi prezentata in detaliu in capitolele urmatoare.

Creare – va adauga un articol sau o categorie in baza de date. In functie de situatie aceeasi functie va crea si alte intrari in baza de date.

Citire – Va prelua datele din baza de date, si le va transpune in format lizibil pentru utilizator in interfata grafica prin ViewModels –clase speciale care primesc date de la baza de date dar nu pot modifica aceste date- scopul lor este doar de interfatare.

Editare – Schimbare a informatiilor din baza de date in tabelele necesare.

Stergere – Stergere a unei inregistrari din baza de date si a tuturor asocierilor acesteia din baza de date.

## 3. Crearea Interfatei de utilizator:

Interfata de utilizator trebuie sa cuprinda toate informatiilor necesare pentru ca utilizatorul sa o poata folosi fara probleme si sa poata intelege scopul aplicatiei. Pentru asta s-a ales crearea interfetei de utilizator cu pagini Razor pentru interfatarea usoara a codului de pe server cu cel de pe client, la care s-au efectuat modificari pentru a arata controalele fiecarei pagini in parte sau date din baza de date legate de articole sau de utilizator.

Interfata trebuie sa cuprinda o pagina pentru crearea articolelor, o pagina pentru citirea fiecarui articol in parte, o pagina pentru a arata toate articolele aprobate si o pagina pentru a permite utilizatorului sa caute articole.

Pentru administrator este nevoie de o pagina speciala de unde acesta poate vedea toate articolele ce au nevoie de aprobare, o pagina speciala de unde acesta poate promova utilizatori la alte roluri (editor, manager).

Fiecare pagina in parte este tratata diferit din punct de vedere al accesului astfel incat exista pagini care sunt accesibile utilizatorilor, pagini care sunt accesibile doar utilizatorilor inregistrati sau pagini care sunt accesibile doar utilizatorilor care au drepturi elevate (administrator, manager).

## 4.Integrarea celor trei componente

Cele trei componente vor fi integrate pentru a indeplini impreuna toate necesitatile aplicatiei astfel incat metodele definite in Controller vor executa corect modificarile si inregistrarile asupra Modelului(Baza de date), iar acestea vor fi usor interpretabile de catre utilizator, intuitive, si totodata descrise suficient in View(interfata) de catre butoane si input-uri pentru ca utilizatorul sa inteleaga in mare masura ce se va intampla atunci cand actioneaza asupra controalelor fara a fi nevoie de instructiuni

# 3. Structura Proiectului:

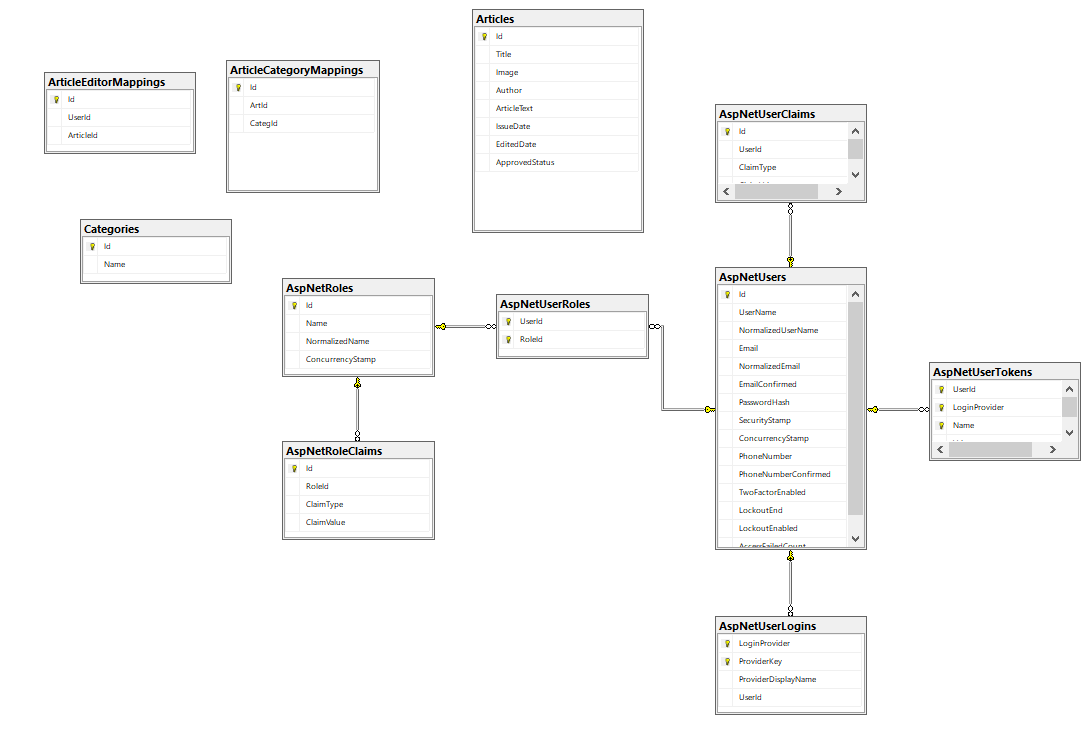
|  |  |
| --- | --- |
| Proiectul este alcatuit dintr-o serie de foldere fiecae continand o parte separata a aplicatiei.  Areas : contine o serie de pagini generate de catre Visual Studio pentru administrarea ccontului utilizatorului.  Authorization : o serie de clase create pentru a verifica drepturile utilizatorului, si o clasa ajutatoare ce contine cateva constante.  Controllers : Clasele controller folosite de catre aplicatie.  Data : acest folder contine clasele model folosite de aplicatie . |  |
| Pages : Contine toate paginile Razor folosite pentru afisarea interfatei. |  |

# 4. 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.

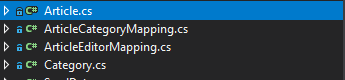
Tabelele destinate managementului articolelor contrin articolele scrise de utilizatori, categoriile, de asemenea date de utilizatori si asocierile necesare intre articole, categorii si editorii.

## 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 inclusamai 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.

# 5. 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.



Aceasta metoda a fost testata din interfata si modificata din varianta originala deoarece vrem sa afisam doar articole aprobate de catre administrator. Astfel s-au adaugat cateva conditii de verificare, prima conditie se asigura ca interfata nu cere o instanta de clasa de tip NULL astfel blocand interfata si afisand erori, a doua conditie este ca acest articol sa fie aprobat de catre administrator si nu poate fi citit pana cand nu este aprobat.

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 functie 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. Parametrul category este de asemenea folosit pentru a filtra articolele dupa categoria selectata.

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.



1. GetByAuthor(ClaimsPrincipal user)

Preia toate articolele scrise de catre user-ul primit ca paramentru. In acest proiect folosim aceasta functie doar ca metoda de a returna o lista cu toate articolele create de catre userul logat, dar functia poate returna articolele oricarui autor din baza de date.

1. GetByEditor(ClaimsPrincipal user)

La fel ca si metoda precedenta aceasta preia totalitatea articolelor editate de catre un utilizator-in cazul aplicatiei utilizatorul logat- dar parametrul poate fi oricare user din baza de date.



Pana acum am prezentat toate functiile de citire din baza de date. Fiecare functie returneaza informatii specifice pentru a fi apelate de catre client. In continuare o sa prezentam cele functia de DELETE si functia de EDIT. Aceste functii sunt mult mai simple si mai usor de implementat decat cele prezentate antrior.

DELETE :



Functia cauta din baza de date o instanta de clasa Article cu Id-ul primit din interfata, daca aceasta este NULL vom returna o exceptie, daca Articolul nu este NULL se vcontinua in a verifica daca userul care incearca sa stearga articolul are permisiunea sa faca aceasta. Metoda AuthorizeAsync returneaza un rezultat de tip Succeeded de tip boolean, daca acest rezultat este corect atunci articolul este sters, altfel se va returna alta exceptie si functia nu va fi executata.

EDIT :



Aceaste functie preia un articol cu Id-ul dat, verifica daca exista si nu este NULL ca si pana acum, apoi verifica daca user-ul are permisiuni sa editeze articolul. Daca toate conditiile sunt indeplinite se editeaza fiecare camp primit din interfata care nu este NULL, daca este NULL, acea proprietate a articolului ramane neschimbata.

Ultima functie din controllerul pentru articole este functia de adaugare. Aceasta functie a necesitat folosirea unei tranzactii.

Tranzactiile sunt un tip de actiune care uneste mai multe actiuni interdependente intr-o singura actiune. De exemplu daca dorim sa introducem in baza de date informatii in doua tabele in acelas timp care depind una de cealalta, trebuie sa introducem datele pe rand in fiecare tabel , iar in cazul in care una din actiuni esueaza baza de date va reveni la forma dinaintea inceperii tranzactiei.

In aceasta aplicatie necesitatea a rasarit din nevoia de a crea un articol in baza de date si o categorie sau mai multe care se asociaza cu articolul creat. Ordinea crearii intrarilor devine astfel : creare articol- creare categorii- creare asocieri articol-categorii. Pentru a preveni inregistrarile duble atunci cand se adauga categorii se verifica daca numele categoriei adaugate de utilizator exista, si daca da aceasta nu se mai adauga.



Functia de mai sus adauga atat articole cat si categorii in baza de date, de aceea o functie pentru adaugarea categoriilor nu mai este necesara in acest moment.

## UserController :

Acest controller este folosit pentru a promova utilizatorii la noi roluri de catre administrator.

Contine o singura metoda de promovare a utilizatorului :



# 6. Autentificare si Autorizare :

Toate functiile de mai sus au introduce in cod o verificare daca utilizatorul are drepturile necesare sa execute functiile. Aceasta metoda se numeste AuthorizeAsync() si face parte din frameworkul AspNetCore.Authorization.

Pentru a putea folosi aceste functionalitati si functii predefinite in framework este nevoie in primul rand sa configuram cateva servicii pentru aplicatie care se afla in clasa Startup.cs. Aceasta clasa contine o functie apelata in momentul in care aplicatia este pornita si permite configurarea serviciilor.

Serviciile adaugate pentru a putea folosi frameworkul de autorizare si autentificare sunt :



Odata configurate serviciile putem sa cream clase care vor verifica userii fata de rolul pe care il au si cel necesar pentru efectuarea actiunilor.

Pentru aceasta aplicatie am creat 4 clase de acest fel :

1. Administrator



1. Manager



1. Editor
2. Utilizator(autor)



Atfel fiecare clasa va fi parcursa de functia AuthorizeAsync() si va returna daca utilizatorul este autorizat sau nu.

Pentru a evita repetitia in cod am creat doua clase de ajutor in care sunt continute valori constante folosite de aceste clase numite ArticleOperations.cs si Constants.cs.

Acest procedeu de impiedicare a utilizatorilor in a efectua operatii pe care nu au dreptul sa le efectueze este facil doar daca si in paralel interfata grafica impiedica utilizatorii in a accesa paginile, sau de a ascunde din interfata controalele care apeleaza aceste functii. Trebuie clarificat insa ca aceasta ascundere este doar pentru a ajuta utilizatorii sa inteleaga ce pot si ce nu pot face si nu prezinta nici un fel de securitate. De aceea pana acum s-a observat in functiile CRUD cum sunt conditionate de raspunsul metodei AuthorizeAsync().

Primul fel in care interfata devine mai prietenoasa este impiedicarea utilizatorilor neinregistrati in a vedea anumite pagini. Apoi cu fiecare elevatie in drepturi dispar alte restrictii – un manager poate accesa toate paginile mai putin cea de administrare roluri si aprobare articloe,dar nu poate sterge nici un articol, administratorul poate accesa toate paginile si efectua orice operatie.

Pentru ca un utilizator sa isi inregistreze un cont acesta trebuie sa intre pe pagina Register si sa completeze campurile aferente. Functia care ii va crea contul de utilizator este din nou o functie din framework si se numeste CreateAsync() – o functie continuta de clasa UserManager<IdentityUser>.



O data ce utilizatorul si-a inregistrat contul acesta va fi logat in aplicatie si va primi un email pe adresa folosita cu un link de confirmare a emailului. Ca utilizator inregistrat acesta poate crea articole, citi articole si i se va crea o lista cu toate articolele create de el.

Aplicatia nu tine cont daca emailul utilizatorului este confirmat deoarece am reusit sa trimit acest email de confirmare doar de pe localhost. Serviciul de trimitere email este SendGrid.com, iar acesta foloseste un certificat SSL care se instaleaza la prima folosire, in schimb dupa ce aplicatia este publicata acest certificat nu exista pe serverul de hosting si trebuie creat.

# 7. View – Interfata cu utilizatorul

## Clasele ViewModel :

Clasele ViewModel sunt clase special create pentru a crea o interfata intre utilizator si baza de date, fara ca acesta sa execute schimbari direct in baza de date. Clasa ViewModel poate contine aceleasi proprietati si functii ca si clasa Model cu acelasi nume, dar nu este folosita pentru a modifica datele din baza de date ci petnru a trimite datele ce trebuie modificate catre o instanta de clasa Model. Astfel se protejeaza accesul la baza de date deoarece utilizatorul poate adauga orice fel de date in aceasta clasa, insa exista metofde ce exclud datele nedorite.

Mai jos o sa prezint clasele de ViewModel folosite:

1.ArticleViewModel :



2.CategoryViewModel :



3.UserViewModel:



Dupa cum se poate observa aceste clase contin aceleasi proprietati ca si clasele de tip model, mai putin UserViewModel, deoarece aceasta clasa este necesara doar pentru transmiterea numelui utilizatorului catre server.

## Pagini Razor :

Aplicatia are 18 pagini Razor fiecare creata separat pentru diferite functii. Majoritatea acestor pagini tin de managementul utilizatorilor de catre ei insisi sau de catre administrator, restul paginilor sunt legate de administrarea articolelor scrise de utilizatori.

Pagina care contine majoritatea elementelor de front-end care sunt folosite ca template se numeste \_Layout.cshtml si este un view partial – adica nu are un model dupa care sa isi schimbe datele sau afisajul, insa preia modelul paginii care se incarca in <body>. Aceasta pagina este un document de html simplu care nu necesita sa fie prezentat, dar contine doua elemente care sunt specifice si trebuie mentionate si anume :

1. Aceasta pagina nu contine un element body, inschimb contine o functie @RenderBody() care va crea aceasta sectiune si o va popula cu codul din fiecare pagina Razor care foloseste ca referinta pagina \_Layout.cshtml.
2. Sectiunea <scripts> lispeste si este adaugata in acelasi mod ca si sectiunea body cu functia @RenderSection("Scripts", required: false)

.

### Index.html :

Aceasta pagina contine un slide show cu poze create de mine in Photoshop pentru a face o scurta introducere in aplicatie.



Sub acest slide show avem afisate in ordine cronologica cele mai noi 9 articole adaugate pe site. Codul html de mai jos este repetat in mai multe pagini pentru a crea un scurt preview al articolului.



Acest cod permite preluarea unei liste de ViewModel si afisarea pentru fiecare structura html din interiorul functiei. Am folosit aceasta metoda pentru mai multe elemente nu doar cele de tip card deoarece imi permite sa creez foarte repede o multitudine de elemente html fara a scrie codul pentru fiecare in parte.

### Article.cshtml:

Aceasta pagina este dinamica si se schimba in functie de parametrul „qid”. Daca acest parametru este nul, atunci pagina va fi o colectie ce contine toate articolele din baza de date , daca acesta are o valoare atunci pagina va arata articolul cu Id-ul egal cu acest parametru.

Pentru acest comportament am creat doua functii in modelul paginii Razor :

1. OnGet():



1. OnGetWithId(int id);



Fiecare pagina Razor are un model unde se declara variabilele ce vor fi folosite, in special se cer instante de Controller pentru a putea efectua actiuni asupra baze de date prin intermediul interfatei.

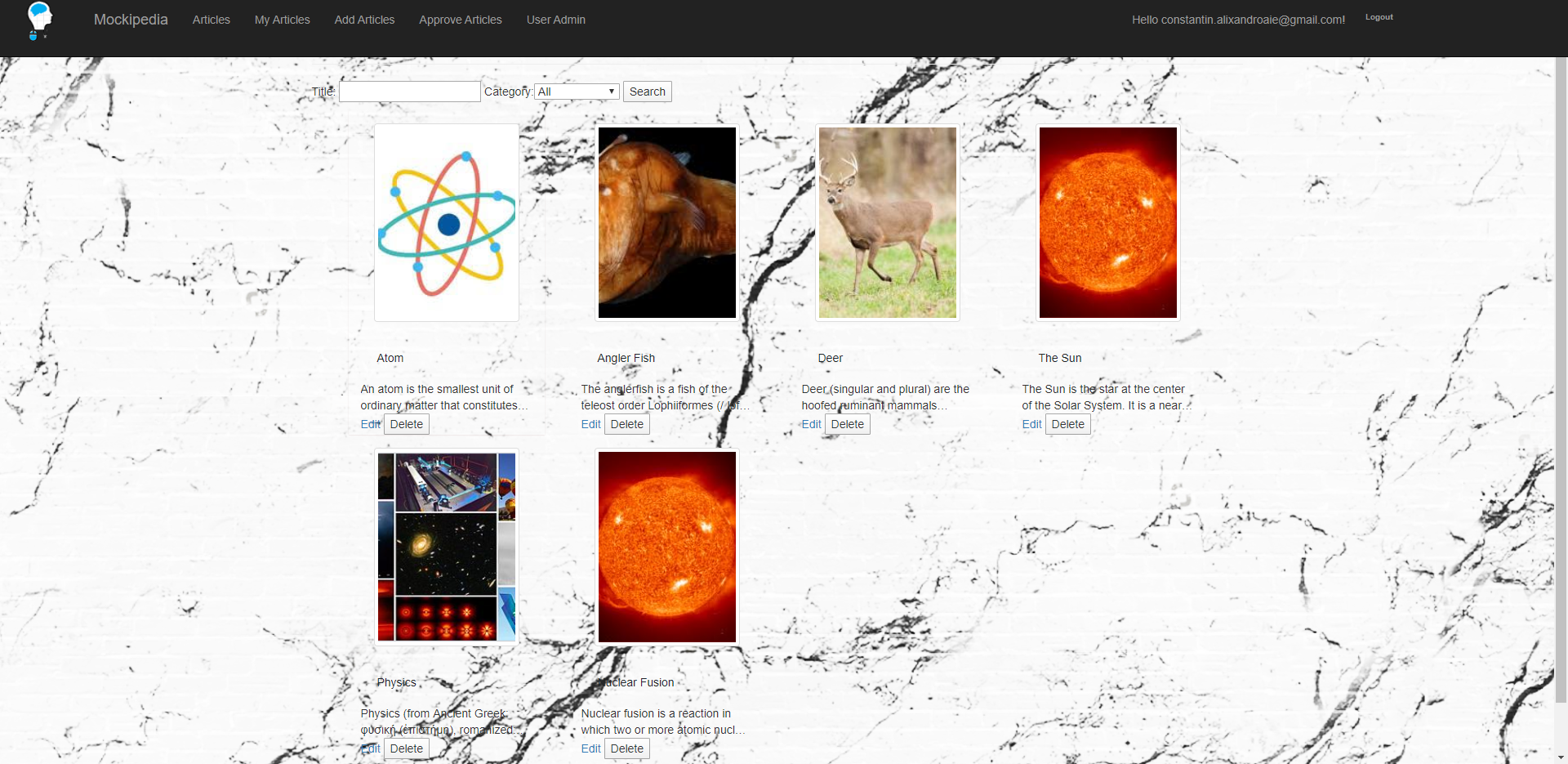


Dupa cum se poate observa mai sus atunci cand se deschide pagina se apeleaza functiile din controller catre care se transmit parametri din interfata pentru executie. Functiile in schimb returneaza un ViewModel sau mai multe, depinzand de situatie care sunt apoi afisate. Astfel datele din baza de date sunt vizibile insa nu pot fi afectate sau modificate din greseala.

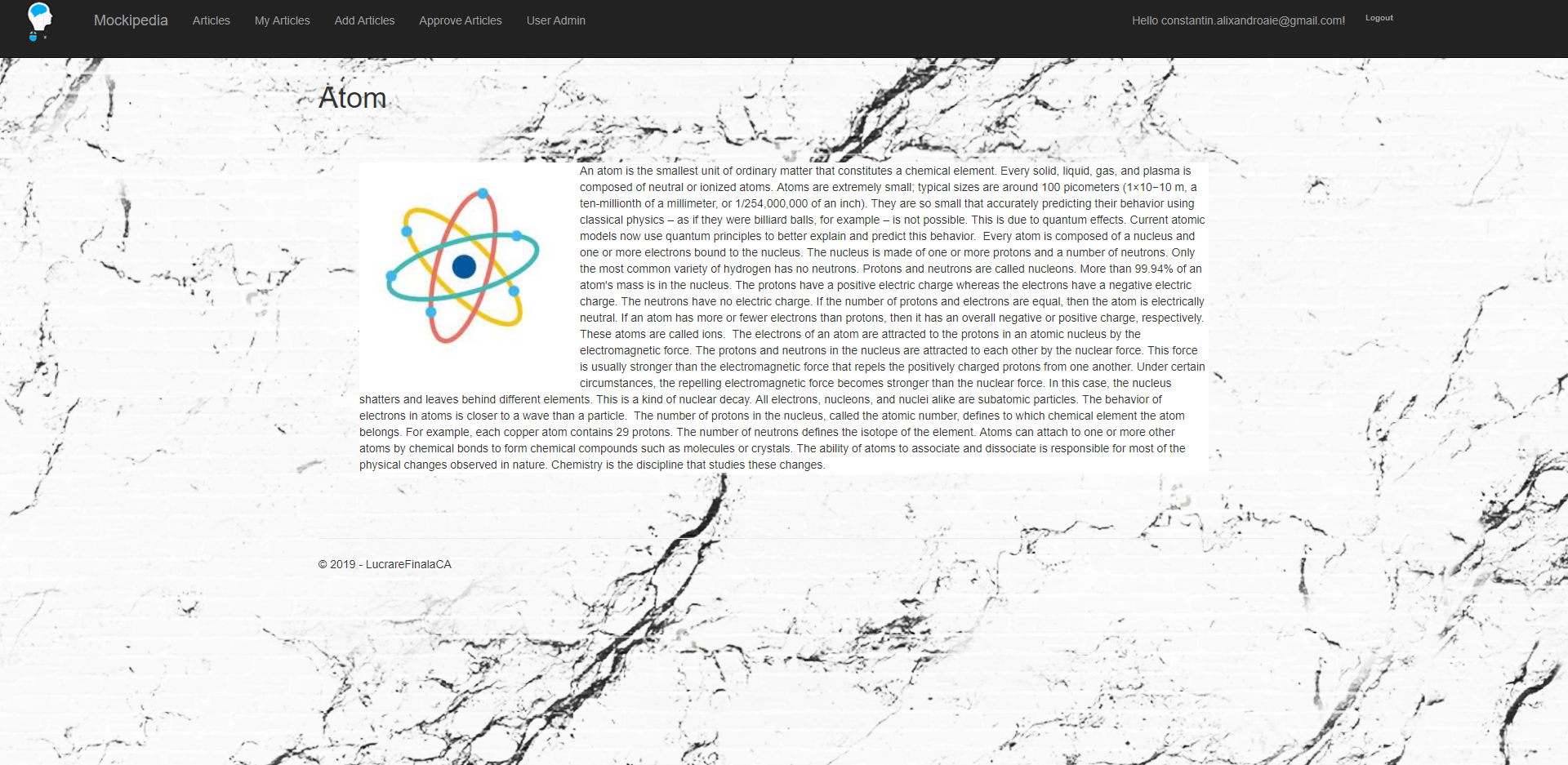
Deoarece am creat acest paramentru ni se permite crearea a doua displayuri diferite in aceeasi pagina web.

Cele doua aspecte ale paginii Article se pot observa in urmatoarele capturi de ecran:

Fara Id :



Cu Id :



Codul din pagina Razor care permite aceasta schimbare este simplu si urmareste ce status are variabila booleana „IsById”.

Daca pagina este ceruta fara Id, atunci in pagina vor fi doua formulare, unu pentru cautare si unul pentru stergere. Aici se va observa inca un comportament care impiedica utilizatorii neautorizati sa faca operatii interzise – in cazul acesta stergerea.

Formularul folosit pentru cautare este folosit pentru a cauta articole dupa cuvinte cheie atat in titlu cat si in continut si dupa categoria din care fac parte :



Acest formular va apela functia GetByLastAsync(searchString,Category), daca acesti parametric sunt null atunci toate articolele se vor afisa, daca in schimb unul dintre ei este introdus atunci se vor afisa doar articolele ce corespund parametrilor de cautare.

Cel de-al doilea form contine controalele pentru stergerea fiecarui articol, dar acesta este vizibil pentru administrator si ascuns pentru oricine altcineva :



Chiar daca butonul nu ar fi ascuns functia este conditionata de userul care o cere si se executa doar daca acesta este autorizat, este mai sigur sa conditionam functia in loc de interfata deoarece daca un utilizator poate folosi linkul poate cere functia de delete fara sa vada butonul.

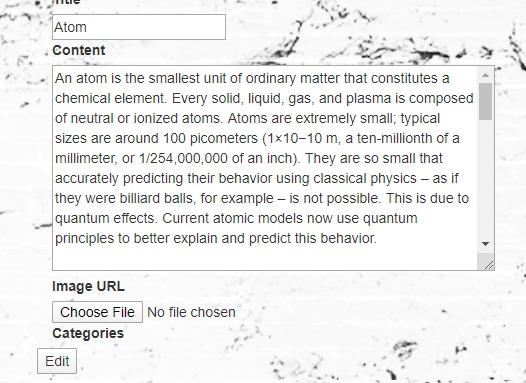
### EditArticle :

In aceasta pagina se poate edita un articol. Pentru fiecare articol se poate modifica titlul, textul si imagina uploadata. Daca oricare din cele 3 este modificat se va adauga in baza de date o asociere intre articol si utilizatorul care a facut modificarea,iar valorile modificate se vor salva in baza de date.

Modelul acestei pagini preia articolul dupa ID din baza de date :



Dupa ce articolul a fost preluat acesta este incarcat in interfata si pregatit pentru editare :



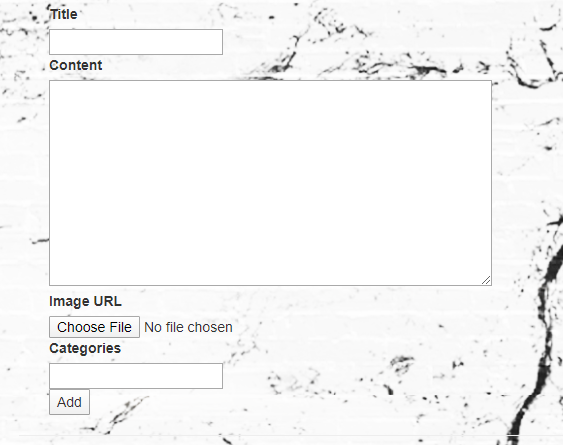
Dupa ce este editat articolul se va apasa butonul edit si se va apela functia din model :



Aceasta functie va apela functia de edit din controller care va aplica modificarile facute tinand cont prima oara de drepturile utilizatorului.

### AddArticle :

In aceasta pagina utilizatorul va vedea o interfata unde trebuie sa populeze o serie de input-uri insotite de etichete pentru a descrie obiectivul fiecarui input, pe care acesta le va completa pentru a adauga un articol.



Aceste campuri sunt toate obligatorii pentru crearea unui articol nou de aceea daca utilizatorul apasa butonul de ADD fara sa completeze unul dintre campuri pagina nu va executa adaugarea ci va afisa un mesaj informativ pentru campurile necompletate.



Daca campurile sunt completate corect atunci aplicatia va redirectiona utilizatorul catre pagina cu articole unde poate vedea articolele deja adaugate pe site, articolul proaspat adaugat nu va fi afisat pana cand acesta nu este aprobat de catre un administrator.

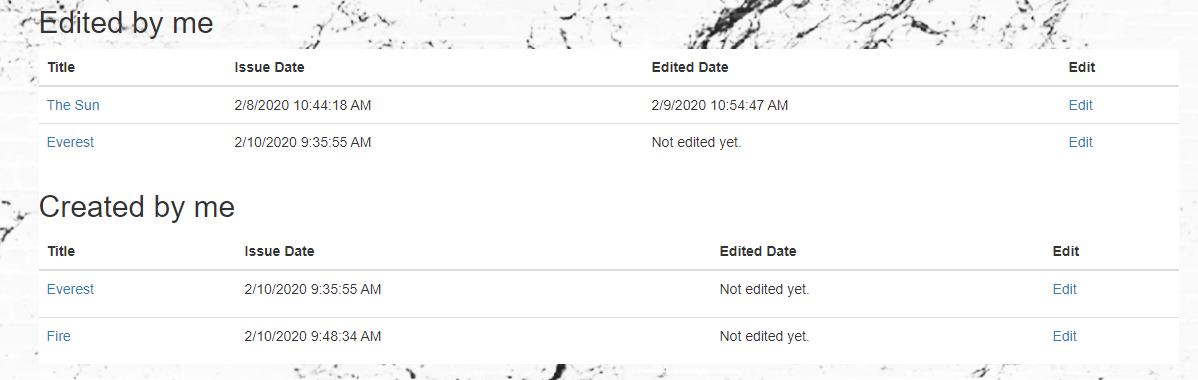
In aceasta pagina pe langa articol se adauga si o serie de categorii depinzand de cate sunt scrise de catre utilizator. Acestea pot fi scrie separate de spatiu sau virgula iar aplicatia le va prelua si le va transpune in intrari separate in baza de date in tabelul categorii. Acest lucru se face in codul urmator :



Categoriile vor fi apoi adaugate de functia Add din cotnroller prezentata mai sus.  
 Imaginea adaugata este verificata sa aiba extensia unei imagini, orice imagine cu alta extensie nu va fi acceptata si va arunca o exceptie.

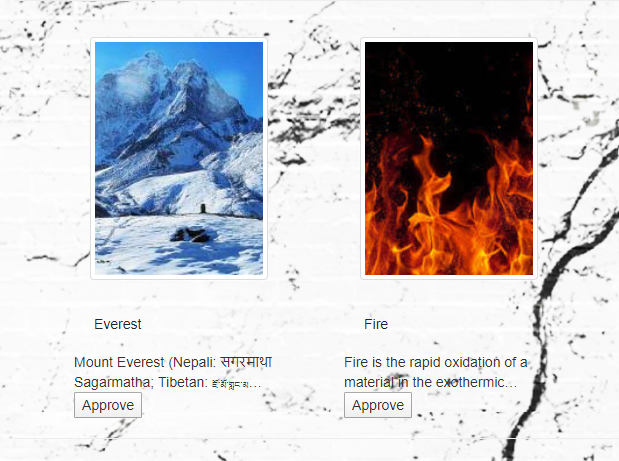
### MyArticles :

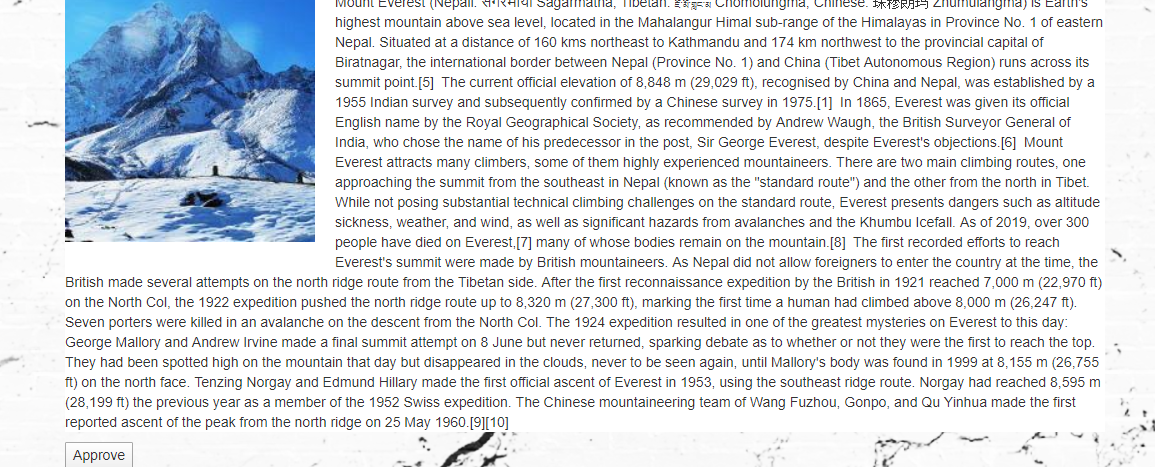
Aceasta pagina contine doua tabele care se populeaza cu articole create sau editate de catre utilizatorul logat. De aici fiecare utilizator poate vedea articolele proprii chiar daca acestea nu sunt aprobate de catre administrator, deoarece sunt vizibile doar pentru el. Tot aici utilizatorul poate vedea un link catre pagina de editare unde poate corecta orice greseli considera.



### ApproveArticles :

Pagina de aprobare a articolelor este o pagina vizibila doar pentru administrator si are o serie de articole care nu sunt aprobate inca. Aici se poate aproba articolul fara a citi articolul sau se poate citi articolul si aproba din pagina articolului.



Butonul Approve apare pe pagina articolului pentru administrator daca articolul este neaprobat.

### UserManagementPage:

O pagina din nou vizibila doar pentru administrator de unde acesta poate asocia roluri la utilizatori noi.

Aici se preia o lista cu toti utilizatorii si o lista cu roluri, butonul Assign Role va crea o asociere intre utilizatorul selectat si rolul selectat.



## Pagini pentru managementul contului :

|  |  |
| --- | --- |
| Aceste pagini sunt generate automat si sunt folosite de utilizator pentru diferite functii de administrare a contului. Nu voi include codul acestor pagini deoarece nu am scrie eu acest cod, este generat automat si se foloseste in intregime de frameworkul ASP.NET, ceea ce inseamna pentru aceasta sectiune ca prezentarea unei functii se rezuma la aratarea numelui functiei si a unei descrieri scurte oferita de Visual Studio, insa codul functiei este ascuns. |  |

# Posibilitati de continuare si extindere :

In acest stagiu de dezvoltarea aplicatia permite cateva functii de baza de creare a unui cont de utilizator, administrare a acestia, stergere a contului , schimbare a parolei, confirmare e-mail etc. Functia de schimbare a parolei sau confirmare email sunt configurate si functioneaza corect doar pe aplicatia locala, deoarece este nevoie de configurari suplimentare pe aplicatia publicata si serverul unde aceasta aplicatie ruleaza.

Un prim pas in extindere este crearea acestor configuratii pe serverul de hosting pentru a putea avea aceasta functionalitate existenta pentru toti utilizatorii.

Articolele adaugate in baza de date la acest moment sunt texte simple insotite de o poza descriptiva, insa aceste caracteristici se pot schimba astfel incat articolul sa contina pargrafe multiple, poze multiple si titluri pentru fiecare paragraf. Astfel pagina in care articolul se citeste poate deveni mai ampla si mai usor de urmarit.

Metodele de verificare a drepturilor utilizatorilor sunt in acest moment create pentr 4 tipuri de utilizatori, cu diferente minore, dar pe masura ce exista o nevoie ridicata de drepturi noi acestea pot fi adaugate claselor. Se pot crea utilizatori fara drepturi de creare articole, dar cu drepturi de editare si multe alte exemple care nu tin neaparat de opereatii asupra articolelor, se poate crea o clasa care sa fie doar administrator de conturi de utilizator si o clasa administrator de articole, cu drepturi complet separate. Pentru moment acestea nu sunt necesare, dar pe masura ce aplicatia creste in dimensiune acestea devin esentiale pentru un management corect si complet al utilizatorilor, orice lipsa va face utilizatorii sa evite folosirea aplicatiei.

Desi aceasta aplicatie incearca sa imite comportamentul site-ului Wikipedia, este posibil, desi nu indicat din punctul meu de vedere, sa fie adaugata o metoda de notare a articolului cu o nota de la 1 la 10. Aceasta metoda este o metoda specifica site-urilor sociale, deci pentru a evita notarile excesiv pozitive sau negative asupra articolelor din considerente diferite de continut (autor, relatii cu autorul etc.) se recomanda crearea uinei clase de utilizatori noua care le permite acestora sa dea aceste note. Acest rol de utilizator poate fi adaugat si scos dupa necesitate de administrator.

Important pentru extinerea aplicatiei este folosirea unei baze de date mai extinsa decat cea curenta care poate contine mult mai multe informatii decat la momentul actual. Pentru a extinde baza de date este nevoie de o cercetare si descriere a noi nevoi de stocare de informatii, nu doar articole, dar si asocieri intre utilizatori si note oferite articolelor de exemplu. Pentru a efectua aceste modificari este necesar ca baza de date sa fie recreata ceea ce poate duce la pierderea datelor, de aceea se impune dezvoltarea unei proceduri de expostare si importare a datelor precedente in noua baza de date creata.

### Bibliografie

## Site-uri :

<https://docs.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-2.1>

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-2.1>

<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-2.1>

<https://docs.microsoft.com/en-us/dotnet/api/?view=aspnetcore-2.1>

<https://stackoverflow.com/>

Carti :

ASP.NET MVC with Entity Framework and CSS — Lee Naylor

Pro ASP.NET MVC 2 Framework — Steven Sanderson