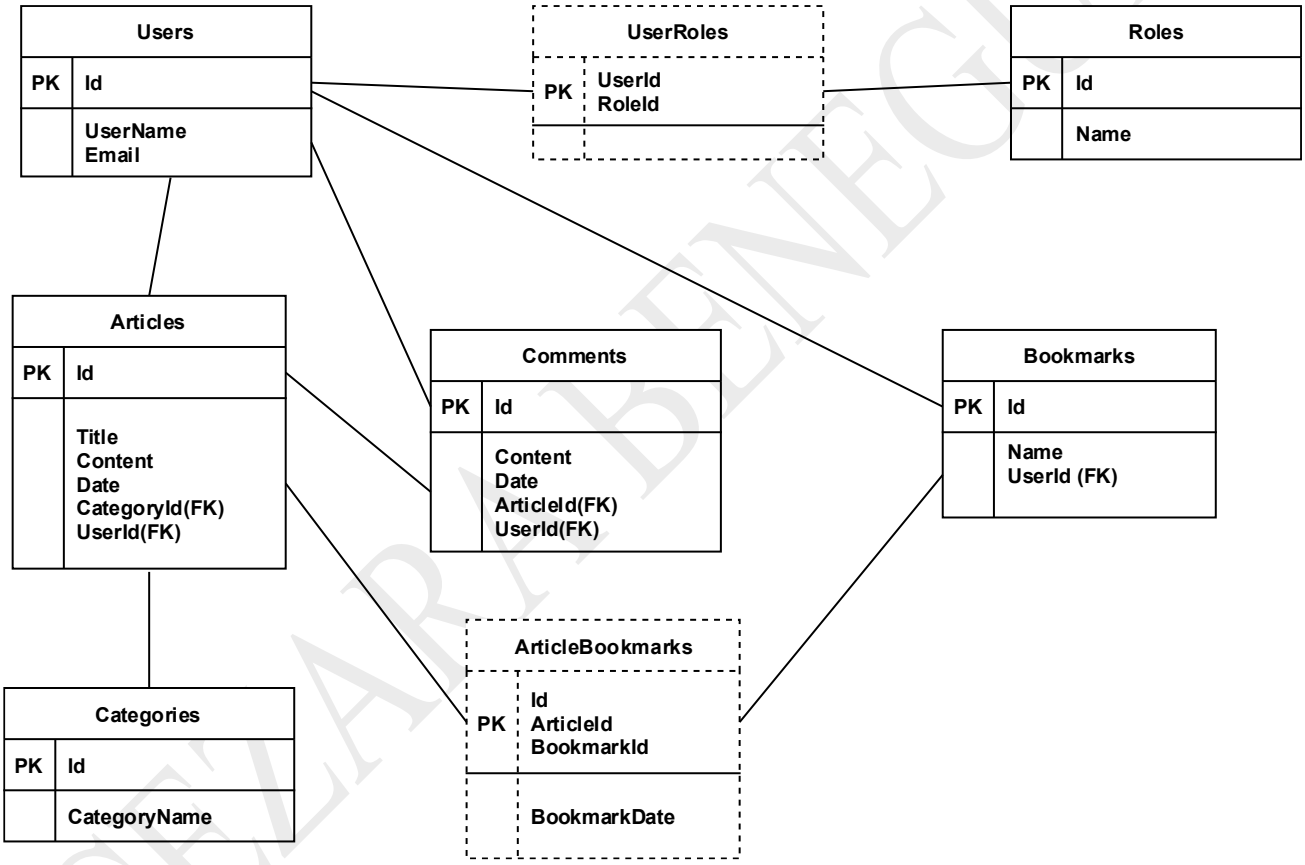
Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Curs 10

Cuprins

Implementarea relației many-to-many

Descrierea diagramei

Pentru implementarea și exemplificarea relației many-to-many, vom utiliza următoarea diagramă conceptuală. Se va extinde diagrama proiectată în cursurile anterioare, prin adăugarea unui nou tabel, **Bookmark**.



Relația many-to-many se află între ***Article*** și ***Bookmark***, având următoarele specificații:

- Un utilizator își poate crea propriile colecții (Bookmarks);
- În momentul în care un utilizator își adaugă colecții, acesta va avea acces doar la colecțiile pe care le-a creat, având posibilitatea de a le edita și șterge;
- Atât utilizatorii cu rolurile ***User*** sau ***Editor***, cât și cei cu rolul ***Admin***, pot crea propriile colecții. Utilizatorii cu rolurile User sau Editor vor avea acces doar la colecțiile create de ei, în timp ce utilizatorii cu rolul Admin vor avea acces la toate colecțiile existente în platformă;
- După crearea colecțiilor, utilizatorii își pot adăuga articole în acestea. Articolele pe care le pot adăuga trebuie să existe deja în platformă. Utilizatorii doar selectează un articol și îl adaugă într-o colecție. Utilizatorii nu pot vizualiza colecțiile altor utilizatori, având acces exclusiv la propriile colecții;
- Un articol poate face parte din mai multe colecții, iar o colecție poate conține mai multe articole;

Se consideră următoarele clase: **Bookmark**, **ArticleBookmark (tabelul asociativ),** cu următoarele proprietăți:

Bookmark:

- **Id** – int → id-ul colecției (cheie primară);
- **Name** – string → denumirea colecției, fiind o proprietate obligatorie (Required);
- **UserId** – string → cheie externă – reprezintă utilizatorul care a creat colecția;

ArticleBookmark:

- **Id**, **ArticleId**, **BookmarkId** – int → cheia primară compusă;
- **Id** – valoare unică, având auto-increment;
- **BookmarkDate** – DateTime → Data și ora la care a fost adăugat un articol în cadrul unei colecții;

Implementarea claselor

Pentru implementarea claselor, se procedează astfel:

Bookmark.cs

```
public class Bookmark

{
   [Key]
   public int Id { get; set; }

   [Required(ErrorMessage = "Numele colectiei este obligatoriu")]
   public string Name { get; set; }

   public string? UserId { get; set; }

   public virtual ApplicationUser? User { get; set; }

   public virtual ICollection<ArticleBookmark>? ArticleBookmarks
{ get; set; }

}
```

Article.cs

```
public class Article

{
   …

   public virtual ICollection<ArticleBookmark>? ArticleBookmarks
{ get; set; }

}
```

ArticleBookmark.cs

```
public class ArticleBookmark

{
   [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public int Id { get; set; }

   public int? ArticleId { get; set; }

   public int? BookmarkId { get; set; }

   public virtual Article? Article { get; set; }

   public virtual Bookmark? Bookmark { get; set; }

   public DateTime BookmarkDate { get; set; }

}
```

ApplicationUser.cs

```
public class ApplicationUser : IdentityUser

{
   public virtual ICollection<Comment>? Comments { get; set; }

   public virtual ICollection<Article>? Articles { get; set; }

   public virtual ICollection<Bookmark>? Bookmarks { get; set; }

}
```

ApplicationDbContext.cs

```
public class ApplicationDbContext :
IdentityDbContext<ApplicationUser>

{
   public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
   {
   }
```

```
        public DbSet<ApplicationUser> ApplicationUsers { get; set; }

        public DbSet<Article> Articles { get; set; }

        public DbSet<Category> Categories { get; set; }

        public DbSet<Comment> Comments { get; set; }

        public DbSet<Bookmark> Bookmarks { get; set; }

        public DbSet<ArticleBookmark> ArticleBookmarks { get; set; }


        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // definire primary key compus

            modelBuilder.Entity<ArticleBookmark>()
                .HasKey(ab => new { ab.Id, ab.ArticleId, ab.BookmarkId
});

            // definire relatii cu modelele Bookmark si Article (FK)

            modelBuilder.Entity<ArticleBookmark>()
                .HasOne(ab => ab.Article)
                .WithMany (ab => ab.ArticleBookmarks)
                .HasForeignKey(ab => ab.ArticleId);

            modelBuilder.Entity<ArticleBookmark>()
                .HasOne(ab => ab.Bookmark)
                .WithMany(ab => ab.ArticleBookmarks)
                .HasForeignKey(ab => ab.BookmarkId);
        }
    }
```

Se pot adăuga și proprietăți suplimentare în clasa ***ApplicationUser***, extinzând astfel clasa. În exemplul următor, se adaugă două atribute: FirstName și LastName.

ApplicationUser.cs

```
public class ApplicationUser : IdentityUser
{
   public virtual ICollection<Comment>? Comments { get; set; }

   public virtual ICollection<Article>? Articles { get; set; }

   public virtual ICollection<Bookmark>? Bookmarks { get; set; }


   public string? FirstName { get; set; }

   public string? LastName { get; set; }

   [NotMapped]
   public IEnumerable<SelectListItem>? AllRoles { get; set; }

}
```

/!\ OBSERVAȚIE

După modificarea claselor, se execută o migrație în baza de date.

Add-Migration NumeMigratie
Update-Database

Afișarea paginată

Afișarea paginată este utilă în cazul în care trebuie să afișăm un număr mare de elemente, număr care se modifică constant. **De exemplu**, atunci când trebuie să afișăm foarte multe categorii, articole, produse etc. În acest caz, se utilizează afișarea paginată, unde dezvoltatorii implementează afișarea în funcție de o serie de reguli. Regulile trebuie stabilite luând în considerare numărul de elemente pe care dorim să le afișăm, ținând cont și de faptul că acest număr își va modifica valoarea în funcție de acțiunile utilizatorilor în cadrul aplicației (de exemplu, utilizatorii pot adăuga sau șterge elemente).

Exemplu de implementare

Exemplul următor este realizat în cadrul aplicației ***Engine de știri*** (aplicația dezvoltată în laborator).

Pentru afișarea paginată, se va utiliza componenta **Pagination** din **Bootstrap** https://getbootstrap.com/docs/5.2/components/pagination/

Se vor afișa 3 articole pe pagină, modificându-se atât metoda Index din **ArticlesController**, cât si View-ul **Index** corespunzător.

Includerea și modificarea componentei din Bootstrap:

```
<nav aria-label="Page navigation example">
    <ul class="pagination">
        <li class="page-item">
            <a class="page-link" href="#" aria-label="Previous">
                <span aria-hidden="true">&laquo;</span>
            </a>
        </li>
        <li class="page-item"><a class="page-link" href="#">1</a></li>
        <li class="page-item"><a class="page-link" href="#">2</a></li>
        <li class="page-item">
            <a class="page-link" href="#" aria-label="Next">
                <span aria-hidden="true">&raquo;</span>
            </a>
        </li>
    </ul>
</nav>
```

```
@* Afisarea paginata a articolelor *@

<div>
    <nav aria-label="Page navigation example">
        <ul class="pagination">
            <li class="page-item">
                <a class="page-link" href="@ViewBag.PaginationBaseUrl=1"
aria-label="Previous">
                    <span aria-hidden="true">&laquo;</span>
                </a>
            </li>

            @for (int i = 1; i <= ViewBag.lastPage; i++)
            {
                <li class="page-item"> <a class="page-link"
href="@ViewBag.PaginationBaseUrl=@i">@(i)</a> </li>
            }

            <li class="page-item">
                <a class="page-link"
href="@ViewBag.PaginationBaseUrl=@(ViewBag.lastPage)" aria-label="Next">
                    <span aria-hidden="true">&raquo;</span>
                </a>
            </li>
        </ul>
    </nav>
</div>
```

ArticlesController → metoda Index

```
        [Authorize(Roles = "User,Editor,Admin")]
        public IActionResult Index()
        {
            // Alegem sa afisam 3 articole pe pagina
            int _perPage = 3;

             var articles = db.Articles.Include("Category")

.Include("User").OrderBy(a => a.Date);

            if (TempData.ContainsKey("message"))
            {
                ViewBag.message =
TempData["message"].ToString();
                ViewBag.Alert = TempData["messageType"];
            }

            // Fiind un numar variabil de articole, verificam de
fiecare data utilizand
            // metoda Count()

            int totalItems = articles.Count();
```

```
            // Se preia pagina curenta din View-ul asociat
            // Numarul paginii este valoarea parametrului page
din ruta
            // /Articles/Index?page=valoare

            var currentPage =
Convert.ToInt32(HttpContext.Request.Query["page"]);

            // Pentru prima pagina offsetul o sa fie zero
            // Pentru pagina 2 o sa fie 3
            // Asadar offsetul este egal cu numarul de articole
care au fost deja afisate pe paginile anterioare
            var offset = 0;

            // Se calculeaza offsetul in functie de numarul
paginii la care suntem
            if (!currentPage.Equals(0))
            {
                offset = (currentPage - 1) * _perPage;
            }

            // Se preiau articolele corespunzatoare pentru
fiecare pagina la care ne aflam
            // in functie de offset
            var paginatedArticles =
articles.Skip(offset).Take(_perPage);


            // Preluam numarul ultimei pagini

            ViewBag.lastPage = Math.Ceiling((float)totalItems /
(float)_perPage);

            // Trimitem articolele cu ajutorul unui ViewBag
catre View-ul corespunzator
            ViewBag.Articles = paginatedArticles;

            return View();
        }
```

Includerea unui editor de text

În această secțiune, vom integra o componentă externă (componentă third-party). Din categoria componentelor third-party fac parte: servere web (Apache, Nginx, IIS etc.), framework-uri (.NET, Rails, Spring etc.) și librării.

Componenta pe care o vom integra este un editor de text open-source, bazat pe Bootstrap și jQuery, numit ***Summernote***. Aceasta este o librărie de JavaScript, utilizată pentru a implementa un editor de tip WYSIWYG (*What You See Is What You Get*). Acest tip de editor oferă utilizatorilor finali posibilitatea de a prelucra textul într-un mod similar cu scrierea textului în Word sau utilizând limbajul de markup (HTML).

Pagina oficială → https://summernote.org/

Exemplu de implementare

Exemplul următor este realizat în cadrul aplicației ***Engine de știri*** (aplicația dezvoltată în laborator).

Se dorește utilizarea editorului de text atât în momentul adăugării unui articol, cât și în momentul editării acestuia.

Pentru integrarea editorului Summernote, se accesează link-ul următor și se urmează pașii indicați:

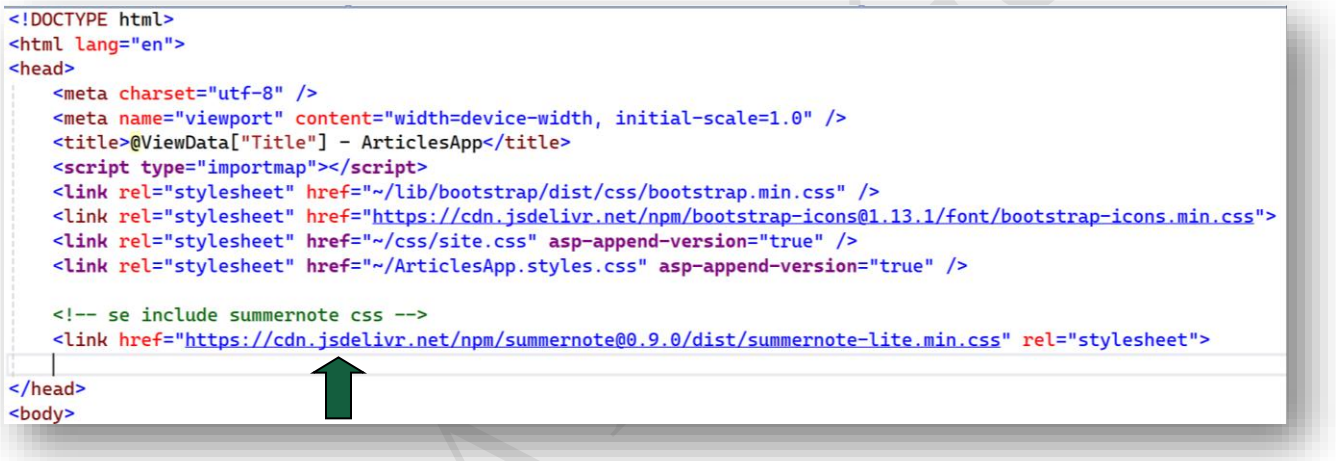
PASUL 1:
https://summernote.org/getting-started/#without-bootstrap-lite

PASUL 2:

Adăugarea fișierelor CSS și JS necesare – se includ fișierele Summernote CSS și JS în proiect (de obicei, printr-un CDN sau descărcarea fișierelor local).

În fișierul **_Layout.cshtml** se include în secțiunea **<head>** componenta CSS pentru editorul Summernote. Aceasta asigură încărcarea stilurilor necesare pentru editorul **Summernote** în toate paginile care utilizează layout-ul specificat.

```
<link href="https://cdn.jsdelivr.net/npm/summernote@0.9.0/dist/summernote-lite.min.css" rel="stylesheet">
```



PASUL 3:

În **_Layout.cshtml** se include în body componenta de JS:

```
<script src="https://cdn.jsdelivr.net/npm/summernote@0.9.0/dist/summernote-lite.min.js"></script>
```

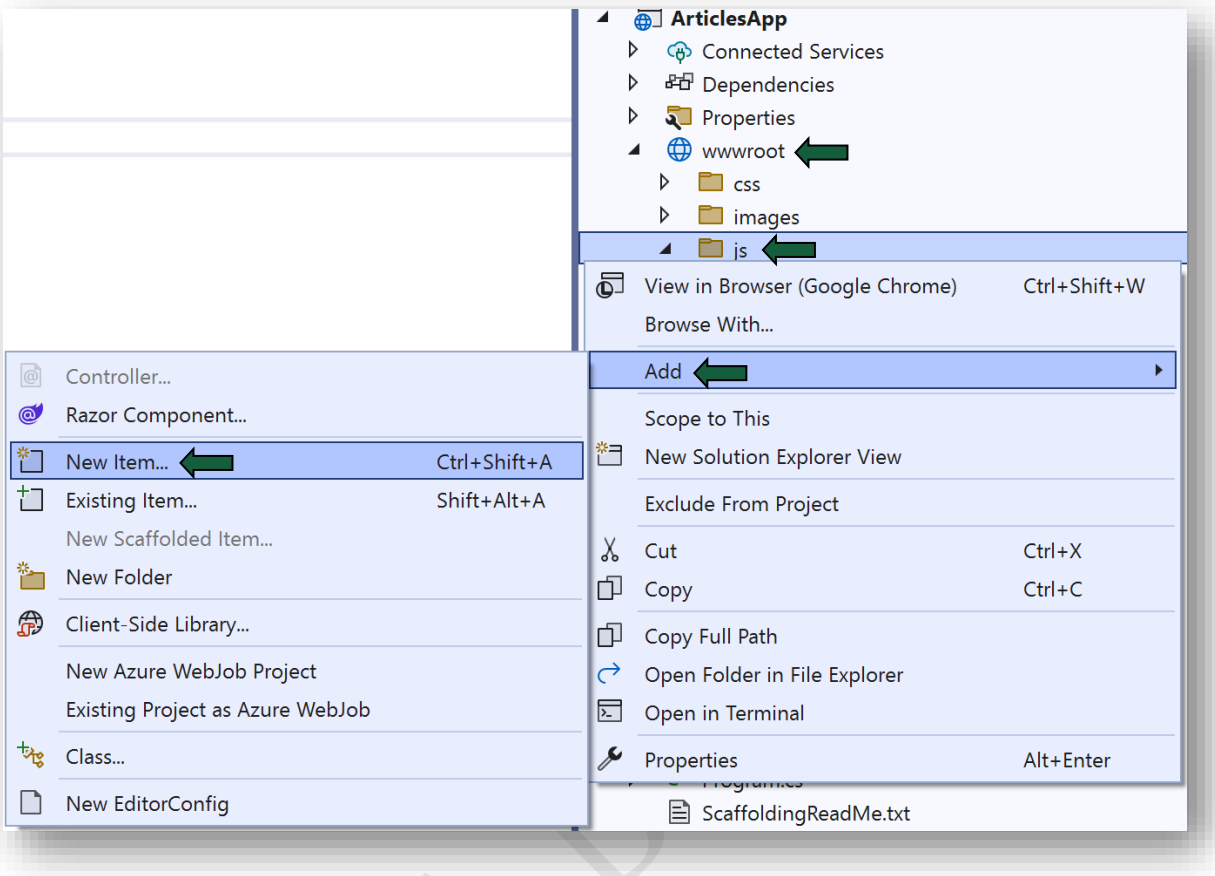
```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2025 - ArticlesApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>

    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>


    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

    <!-- se include summernote js -->
    <script src="https://cdn.jsdelivr.net/npm/summernote@0.9.0/dist/summernote-lite.min.js"></script>

    <script src="~/js/site.js" asp-append-version="true"></script>


    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

PASUL 4:

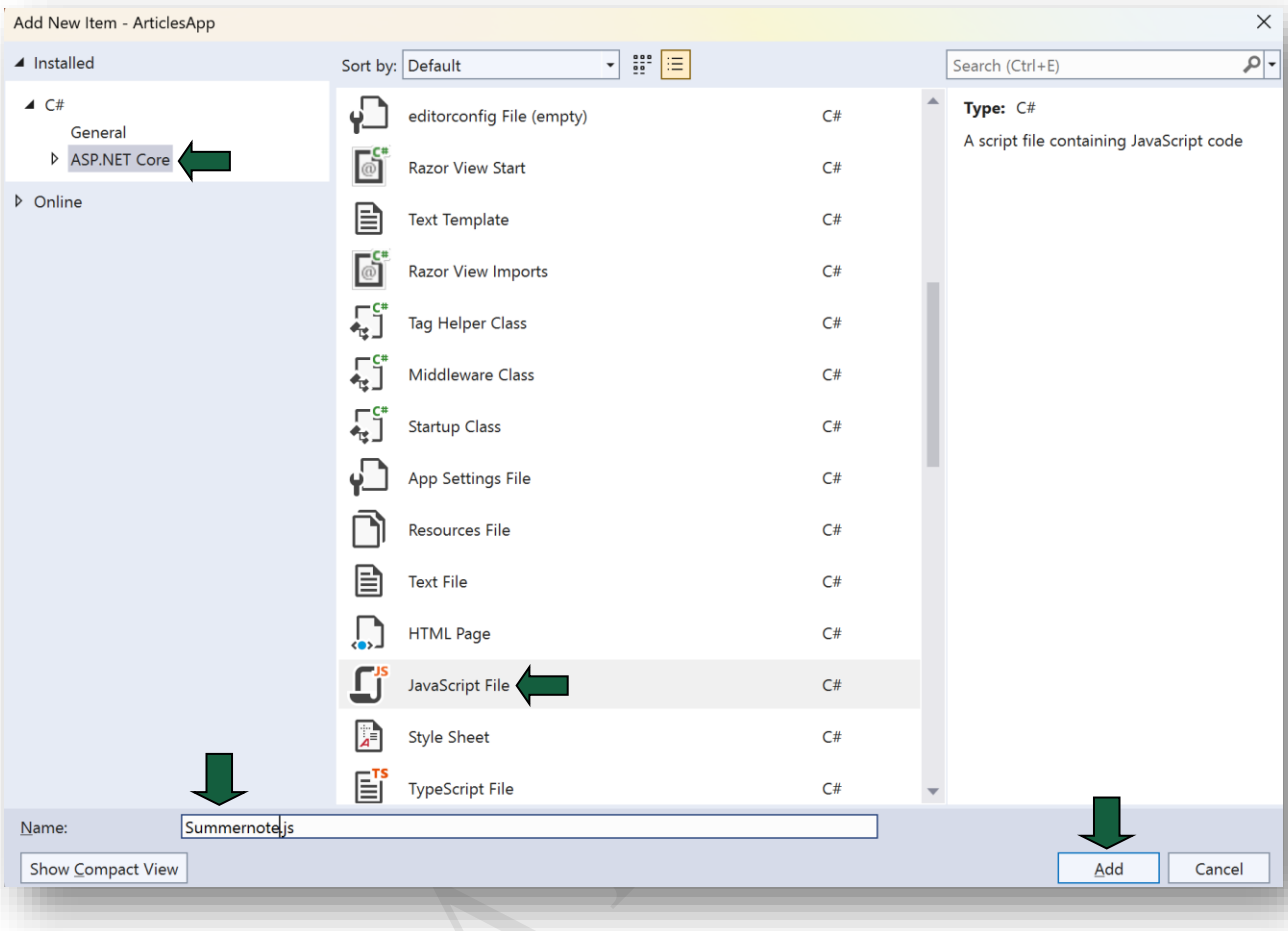
Editorul se inițializează adăugând următoarea secvență de cod într-un fișier de tip JavaScript (.js):

```
    $(document).ready(function () {
   $('.summernote').summernote({
       height: 300, // inaltimea editorului
       placeholder: 'Continut articol…',
       tabsize: 2,
       minHeight: 200,
       focus: true,
    });
});
```

PASUL 5:



PASUL 6:



PASUL 7:

Scriptul-ul se include în _Layout, în body:

```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2025 - ArticlesApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>

    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>


    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

    <!-- se include summernote js -->
    <script src="https://cdn.jsdelivr.net/npm/summernote@0.9.0/dist/summernote-lite.min.js"></script>

    <script src="~/js/site.js" asp-append-version="true"></script>

    <!-- se include Summernote.js -->
    <script src="~/js/Summernote.js" asp-append-version="true"></script>


    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

PASUL 8:

Pentru utilizarea editorului, se folosește clasa definită în fișierul Summernote.js

```
<textarea asp-for="Content" class="summernote"></textarea>
```

- **asp-for="Content"** – când se utilizează *asp-for="Content"* într-un element HTML, cum este <textarea>, acest atribut creează o legătură (**binding**) între câmpul de pe interfață și proprietatea **Content** din modelul asociat cu pagina Razor. Acest proces este gestionat de mecanismul de **Model Binding** al ASP.NET Core. Când se adaugă asp-for="Content" pe un element, ASP.NET Core generează automat atributul **name** pentru acel element.

Atributul name este esențial, deoarece backend-ul ASP.NET Core folosește acest nume pentru a asocia valoarea trimisă de formular cu proprietatea Content din model.

De exemplu: dacă modelul are o proprietate **Content**, atunci codul Razor o să fie transformat în cod HTML, astfel:

Razor

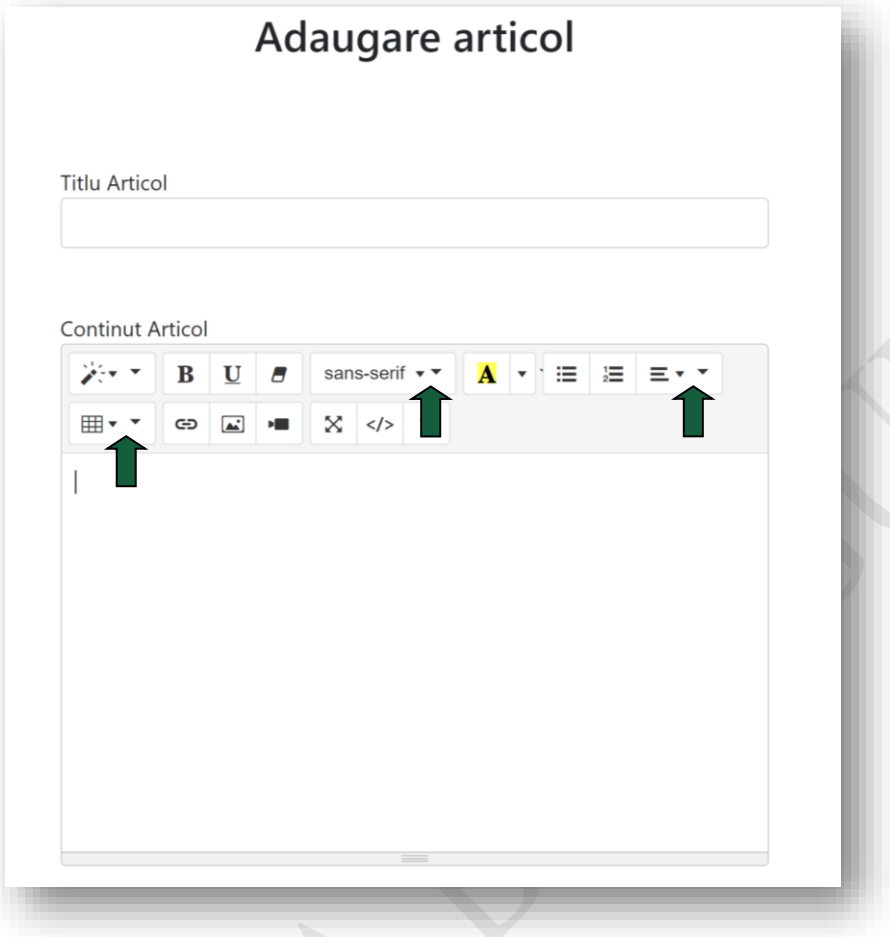
```
<textarea asp-for="Content"></textarea>
```

HTML

```
<textarea name="Content"></textarea>
```

- **class="summernote"** – se atribuie clasa summernote pentru a aplica funcționalitatea editorului;

După integrarea editorului, se observă o problemă de afișare:



Pentru a corecta afișarea dropdown-ului și a evita să apară săgețile duplicat, stilizarea se poate face prin CSS. Acest lucru se întâmplă frecvent când dropdown-ul este personalizat sau folosește anumite framework-uri (**de ex:** Bootstrap).

```
.note-editor .dropdown-toggle::after {
    display: none;
}
```

PASUL 9:

Pentru afișarea conținutului din baza de date, se utilizează `@Html.Raw()`

```
<div class="card-text">@Html.Raw(Model.Content)</div>
```

@Html.Raw() execută conținutul HTML exact așa cum este, fără nicio verificare. Dacă baza de date conține cod malițios (cum ar fi <script>), acesta va fi executat în browser, fiind posibile vulnerabilități de tip **Cross-Site Scripting (XSS)**.

Recomandări privind securitatea:

- **Filtrarea conținutului înainte de salvarea în baza de date**:
  - Se utilizează o librărie de filtrare a conținutului (cum ar fi ***HtmlSanitizer***) pentru a elimina tagurile HTML periculoase înainte de a salva datele în baza de date.

- **Restricționarea la nivel de rol – cine poate introduce HTML în aplicație**:
  - Se acordă permisiuni, astfel încât doar utilizatorii de încredere (cum ar fi administratorii sau editorii) să adauge conținut HTML.

PASUL 10:

Prevenire XSS (Cross-Site Scripting). XSS presupune posibilitatea inserării unui cod malițios, de obicei JavaScript, de către o persoană neautorizată, într-o pagină web. Acest cod este apoi executat în browserul utilizatorului final. Acest lucru se poate întâmpla dacă aplicația web afișează conținut nesecurizat din surse nesigure, cum ar fi formularele introduse de utilizatori. În cazul exemplului din cadrul cursului, un utilizator neautorizat poate introduce cod JavaScript în formular, în momentul inserării unui nou articol în platformă.

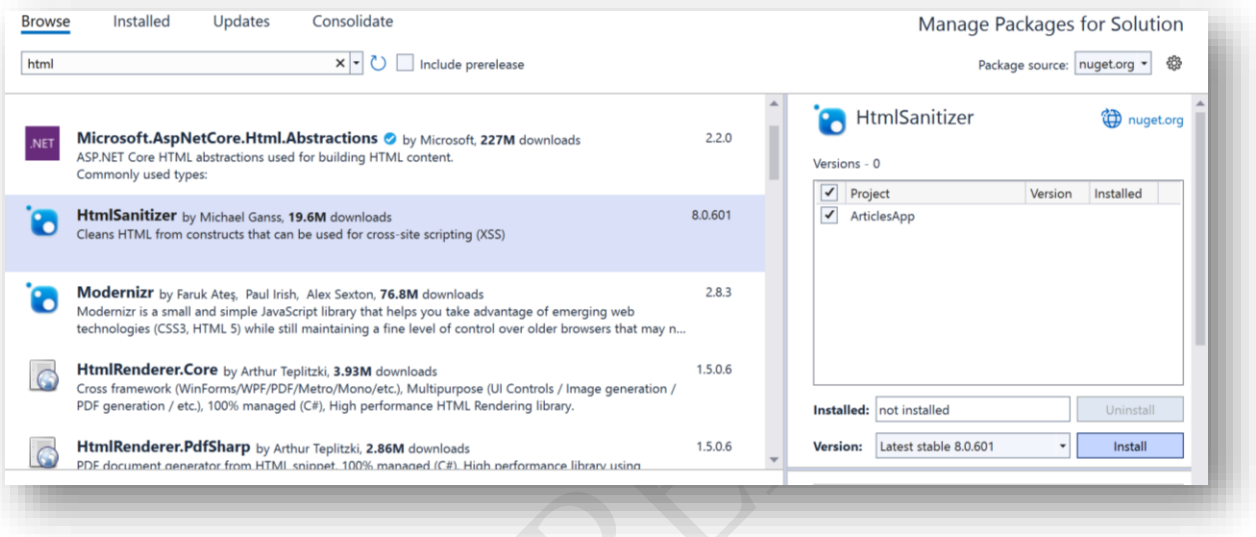
Exemplu de XSS:

Utilizatorul introduce cod malițios într-un formular (**de exemplu:** la crearea unui articol).

```
<script>alert('Atac XSS!');</script>
```

Dacă aplicația salvează acest text în baza de date și îl afișează ulterior folosind **@Html.Raw()**, fără niciun filtru sau validare, codul JavaScript este executat în browserul altor utilizatori, cauzând efecte nedorite.

Pentru a preveni XSS este nevoie de instalarea unui nou pachet din ***NuGet package → HtmlSanitizer***



În cadrul adăugării și editării din Controller, o să avem următoarele modificări:

ArticlesController → metoda New cu POST

```
[HttpPost]

public IActionResult New(Article article)
{
     var sanitizer = new HtmlSanitizer();

     article.Date = DateTime.Now;
     article.UserId = _userManager.GetUserId(User);

     if (ModelState.IsValid)
     {
          article.Content = sanitizer.Sanitize(article.Content);

          db.Articles.Add(article);
          db.SaveChanges();
          TempData["message"] = "Articolul a fost adaugat";
          TempData["messageType"] = "alert-success";
```

```
            return RedirectToAction("Index");
        }
        else
        {
            article.Categ = GetAllCategories();
            return View(article);
        }
}
```

ArticlesController → metoda Edit cu POST

```
[HttpPost]
[Authorize(Roles = "Editor,Admin")]
public IActionResult Edit(int id, Article requestArticle)
{
     Var sanitizer = new HtmlSanitizer();

     Article article = db.Articles.Find(id);

     if (ModelState.IsValid)
     {
          if (article.UserId == _userManager.GetUserId(User) ||
User.IsInRole("Admin"))
          {
                    article.Title = requestArticle.Title;

                    requestArticle.Content =
sanitizer.Sanitize(requestArticle.Content);

                    article.CategoryId =
requestArticle.CategoryId;
                    TempData["message"] = "Articolul a fost
modificat";
                     TempData["messageType"] = "alert-success";
                    db.SaveChanges();
                    return RedirectToAction("Index");
          }
          else
          {
                    TempData["message"] = "Nu aveti dreptul sa
faceti modificari asupra unui articol care nu va apartine";
                    TempData["messageType"] = "alert-danger";
                    return RedirectToAction("Index");
            }
     }
```

```
        else
        {
            requestArticle.Categ = GetAllCategories();
            return View(requestArticle);
        }
}
```

În cazul editării, pentru preluarea corectă a conținutului articolului din baza de date, se va utiliza ***TextArea***.

```
<textarea asp-for="Content" class="form-control summernote">
</textarea>
```

Funcționalitatea de căutare

Funcționalitatea de căutare se afla integrată în orice aplicație, indiferent de natura ei (aplicație web, aplicație mobile, desktop, etc).

Google este, de departe, cel mai mare motor de căutare existent pe piață, deținând o cotă de piață globală de peste 90%. Conform datelor din ianuarie 2024, Google avea o cotă de 91,6% din piața globală a motoarelor de căutare.

Când vine vorba de toate căutările online, un raport realizat în 2025 a plasat ChatGPT la ~ 9.0 % din "digital queries" globale, iar Google la ~ 81.6 %.

Descrierea și implementarea motorului de căutare

Pentru a exemplifica integrarea motorului de căutare, o să utilizăm aplicația dezvoltată în laborator – ***Engine de știri***. În cadrul aplicației, se integrează funcționalitatea de căutare a articolelor după titlu, conținut, dar și după conținutul comentariilor postate de utilizatori în cadrul articolelor respective.

Se utilizează componenta Bootstrap → ***Input group***

https://getbootstrap.com/docs/5.0/forms/input-group/

Implementare:

View-ul Index

```
<form method="GET">

    <div class="input-group mb-3">

        <input type="text" class="form-control"
placeholder="Search topics or keywords" name="search"
               value="@ViewBag.SearchString">

        <button class="btn btn-outline-success"
type="submit">Search</button>

    </div>

</form>
```

ArticlesController – Metoda Index

```
[Authorize(Roles = "User,Editor,Admin")]
public IActionResult Index()
{
     var articles = db.Articles.Include("Category")
                      .Include("User").OrderBy(a => a.Date);

     var search = "";

     // MOTOR DE CAUTARE

     if (Convert.ToString(HttpContext.Request.Query["search"]) !=
null)
     {
          // eliminam spatiile libere
          search =
     Convert.ToString(HttpContext.Request.Query["search"]).Trim();
```

```
            // Cautare in articol (Title si Content)

            List<int> articleIds = db.Articles.Where
                                (
                                 at => at.Title.Contains(search)
                                       || at.Content.Contains(search)
                                ).Select(a => a.Id).ToList();

            // Cautare in comentarii (Content)
            List<int> articleIdsOfCommentsWithSearchString =
db.Comments.Where
                    (
                      c => c.Content.Contains(search)
                     ).Select(c => (int)c.ArticleId).ToList();

             // Se formeaza o singura lista formata din toate id-urile
selectate anterior
             List<int> mergedIds =
articleIds.Union(articleIdsOfCommentsWithSearchString).ToList();


            // Lista articolelor care contin cuvantul cautat
            // fie in articol -> Title si Content
            // fie in comentarii -> Content
             articles = db.Articles.Where(article =>
mergedIds.Contains(article.Id))
                         .Include("Category")
                         .Include("User")
                         .OrderBy(a => a.Date);

      }

             ViewBag.SearchString = search;

             // AFISARE PAGINATA

            { … implementarea se afla in sectiunea anterioara }



             if(search != "")
             {
                 ViewBag.PaginationBaseUrl = "/Articles/Index/?search="
+ search + "&page";
             } else
             {
                 ViewBag.PaginationBaseUrl = "/Articles/Index/?page";
             }


             return View();
}
```

După cum se observă în implementarea din cadrul metodei **Index**, din **ArticlesController**, trebuie să ne asigurăm că afișarea paginată din cadrul secțiunii anterioare funcționează și în momentul integrării motorului de căutare.

Astfel, în cazul în care se folosește motorul de căutare, URL-ul o să fie de forma: ***/Articles/Index/?search&page***, ceea ce înseamnă că trebuie să avem în rută atât șirul de caractere căutat, cât și numărul paginii.

Afișarea paginată din cadrul View-ului **Index** o să se modifice astfel:

```
@* Afisarea paginata a articolelor *@

<div>
    <nav aria-label="Page navigation example">
        <ul class="pagination">
            <li class="page-item">
                <a class="page-link"
href="@ViewBag.PaginationBaseUrl=1" aria-label="Previous">
                    <span aria-hidden="true">&laquo;</span>
                </a>
            </li>

            @for (int i = 1; i <= ViewBag.lastPage; i++)
            {
                <li class="page-item"> <a class="page-link"
href="@ViewBag.PaginationBaseUrl=@i">@(i)</a> </li>
            }

            <li class="page-item">
                <a class="page-link"
href="@ViewBag.PaginationBaseUrl=@(ViewBag.lastPage)" aria-
label="Next">
                    <span aria-hidden="true">&raquo;</span>
                </a>
            </li>
        </ul>
    </nav>
</div>
```

Design-ul într-o aplicație Web

Reguli de bază în design

Pentru realizarea aplicațiilor Web, în ceea ce privește design-ul acestora, se pot respecta anumite reguli generale:

- **Simplitatea** – Prea multă informație într-o pagină distrage atenția utilizatorului, deoarece acesta trebuie să citească și să parcurgă mult prea multă informație până la cea de care are nevoie. Păstrând paginile simple, aplicația va fi mult mai ușor de folosit.

- **Design-ul este esențial** – Prima impresie contează. Această regulă se aplică și în dezvoltarea unei aplicații web, deoarece este primul lucru pe care îl observă un utilizator atunci când accesează aplicația.

- **Culorile sunt foarte importante** – Atunci când alegeți culorile, folosiți o paletă consistentă. De asemenea, asigurați-vă că nu există nuanțe foarte apropiate, care nu pot fi deosebite, și mai ales că există un contrast puternic între text și fundal.

- **Navigarea ar trebui să fie intuitivă** – Un utilizator nu trebuie să caute ceea ce dorește să acceseze. Paginile trebuie să fie bine organizate, cu un design de tip top-down, utilizatorii navigând ușor printre diferitele secțiuni existente într-o pagină.

- **Consistența este extrem de importantă** – Utilizatorii nu ar trebui să aibă sentimentul că vizitează un alt site sau o altă aplicație web de fiecare dată când accesează o altă pagină a aplicației. Consistența face navigarea în aplicație mult mai simplă.

- **Aplicația trebuie să fie responsive** – Utilizatorii accesează aplicația utilizând o varietate de dispozitive, de la smartphone-uri la calculatoare personale. De aceea, este esențial ca aplicația să se încadreze oricărei rezoluții. CSS media queries sunt ideale pentru realizarea rapidă a unei aplicații web responsive.

- **Utilizarea unui conținut real în momentul dezvoltării design-ului** – Orice aplicație este bazată pe conținut și se dezvoltă în jurul acestuia. De cele mai multe ori, atunci când se dezvoltă design-ul, nu se acordă importanță conținutului, folosindu-se Lorem Ipsum în locul textului real și placeholder în locul imaginilor. Chiar dacă totul arată bine în timpul dezvoltării design-ului, nu va mai fi la fel atunci când aplicația va conține datele reale. Scopul în dezvoltarea unei aplicații web este de a fi cât mai aproape de experiența reală a utilizatorului.

- **Fontul** – În general, fonturile Sans Serif, cum ar fi Arial și Verdana, sunt mai ușor de citit într-o aplicație web, fiind fonturi fără finisaje decorative. Dimensiunea fontului pentru o citire ușoară este de 16px.

- **Imaginile** – ajută la o bună interacțiune cu utilizatorul

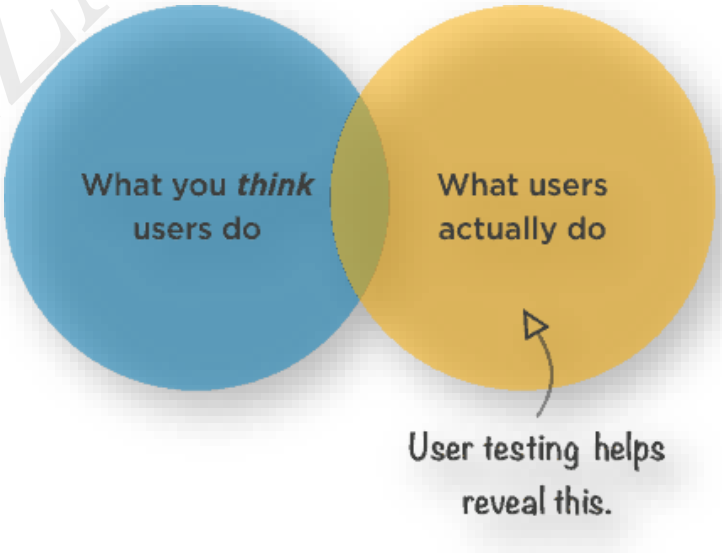
- **“F” Pattern Design** – informațiile să fie prezentate, în ordinea relevanței, de la stânga la dreapta și de sus în jos, un studiu arătând că partea de sus-stânga a ecranului este mult mai vizualizată decât cea de jos-dreapta.

User Experience (UX)

În primul rând, interfața cu utilizatorul (***User Interface***) este o parte din UX, existând o diferență semnificativă între cele două. Interfața cu utilizatorul este spațiul unde utilizatorul interacționează cu aplicația, cu diferite componente ale acesteia, iar ***User Experience*** este rezultatul final pe care îl produce interacțiunea cu aplicația.

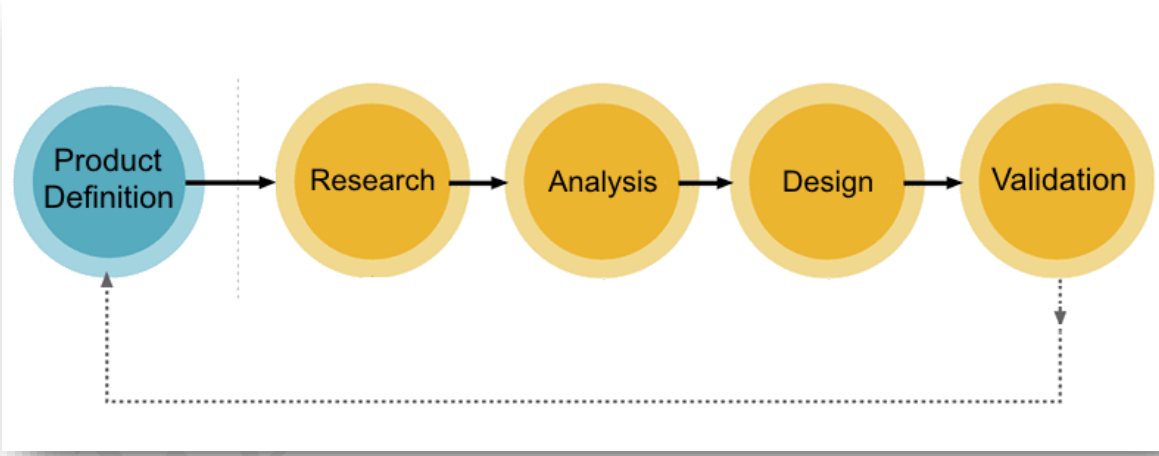
- **Scopul aplicației** – Cel mai important lucru este scopul aplicației. Cine vor fi utilizatorii finali? Care sunt nevoile lor? Ce își doresc ei? De aceea, dezvoltarea unei aplicații poate începe cu un ***user research***, fiind un proces esențial pentru UX.

- **Tu nu ești utilizatorul final** – Este un aspect esențial în dezvoltarea unei aplicații. Dezvoltatorul nu este utilizatorul final, iar cel mai important lucru este testarea de către un utilizator real. Un dezvoltator nu se va comporta niciodată ca un utilizator real, deoarece are experiență și va utiliza aplicația exact cum a fost dezvoltată. Utilizatorii care vor folosi aplicația au moduri diferite de gândire, un background diferit, experiență diferită sau chiar scopuri diferite. Acest lucru face ca testarea cu utilizatori reali să fie cea mai bună formă de testare pentru o aplicație web (*usability testing*).

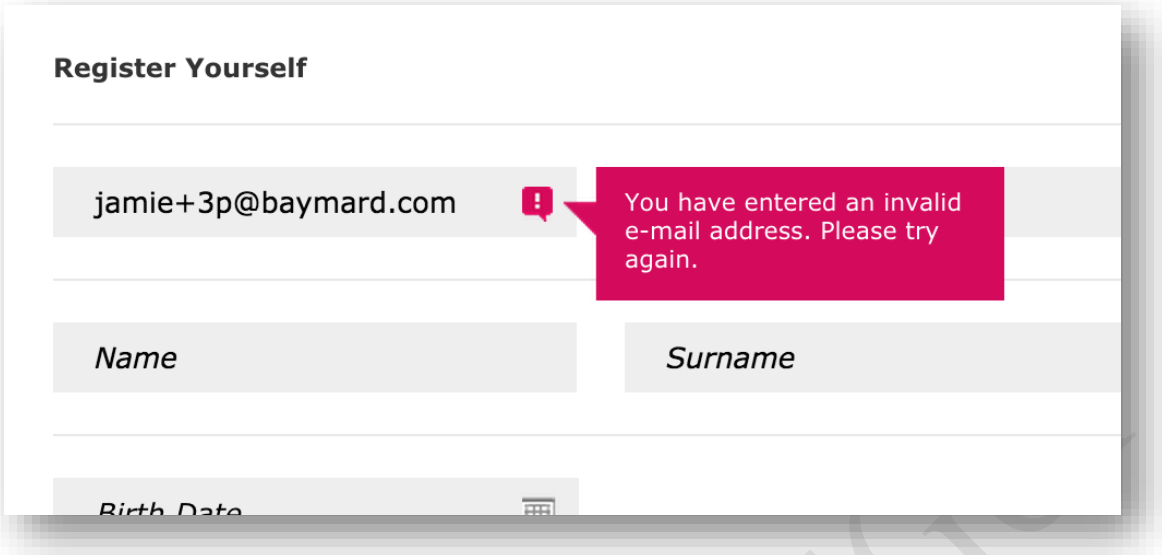


- **Adaptarea design-ului pentru atenția de scurtă durată** – Nu încărcați utilizatorii cu foarte multă informație. Perioada de atenție reprezintă timpul în care cineva se concentrează asupra unei sarcini, fără să fie distras. Este necesar ca designerii să afișeze oamenilor informațiile necesare, cât mai repede posibil. De aceea, lucrurile care nu sunt esențiale pot fi eliminate. Totuși, acest lucru nu înseamnă că trebuie să limităm experiențele utilizatorilor într-o aplicație web, ci doar că informațiile trebuie să fie relevante.

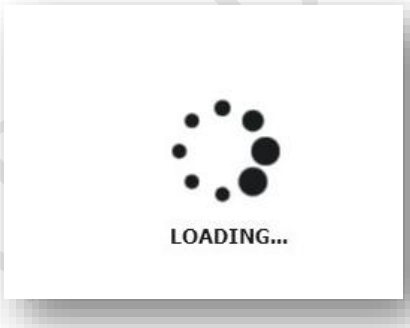
- **UX depinde e fiecare proiect în parte** – Nu există un UX general care poate fi aplicat oricărui proiect. Fiecare proiect este unic, având cerințe unice, ceea ce duce la un UX diferit pentru fiecare aplicație. De exemplu, atunci când dezvoltați un nou proiect, este necesară alocarea unui timp suplimentar pentru a cerceta ce tipuri de utilizatori vor accesa aplicația, dar și care sunt specificațiile acesteia.

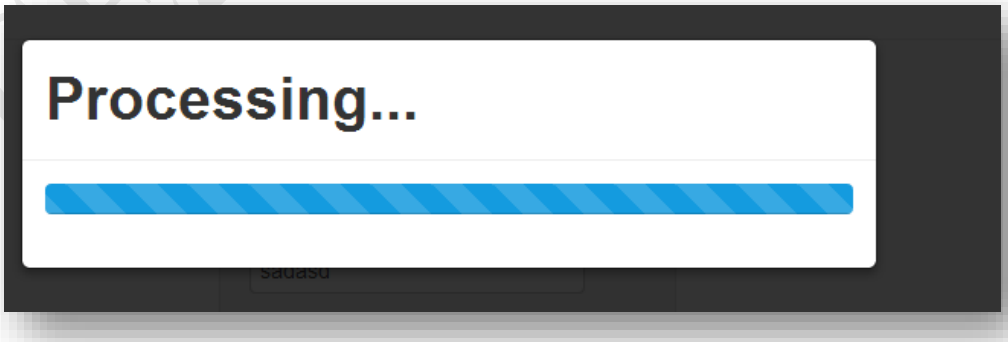


- **Prevenirea erorilor este mai ușoară decât rezolvarea lor** – Multe erori apar din cauza utilizatorilor (de exemplu: e-mail incorect, parole care nu respectă anumite reguli, etc.). În acest caz, aplicația poate fi dezvoltată astfel încât să ofere sugestii de completare și să notifice utilizatorii de fiecare dată când apare o problemă.



- **Afișarea feedback-ului** – Utilizatorii trebuie să fie mereu informați despre tot ceea ce se întâmplă în cadrul unei aplicații: mesaje după trimiterea formularelor, mesaje în momentul editării/ștergerii datelor, feedback vizual. Acesta din urmă este foarte important, fiind un mijloc eficient de informare.





- **Designul trebuie să fie simplu și intuitiv**, utilizatorii bazându-se, de fiecare dată, pe intuiție. Steve Krug spunea că principalul motiv pentru care utilizatorii folosesc intuiția atunci când accesează o aplicație este că "nu le pasă". *"If we find something that works, we stick to it. It doesn't matter to us if we understand how things work, as long as we can use them."* S-a constatat că utilizatorii nu citesc informația, ci doar o scanează, căutând câteva puncte de referință care să îi ghideze prin conținutul paginii.

- **Spațiile albe sunt importante** – Informația este percepută mult mai repede, deoarece un utilizator, așa cum am menționat anterior, scanează informația și o împarte în secțiuni ușor de interpretat. Structurile complexe sunt greu de citit, scanat, analizat și utilizat.

Alegerea culorilor potrivite

Alegerea culorilor în dezvoltarea unei aplicații web, de cele mai multe ori, nu este atât de simplă. În continuare, vom urmări cele mai importante reguli în acest sens: (https://flatuicolors.com/)

- **Tehnica 60-30-10** – Această tehnică, utilizată și în decorarea caselor, este una simplă. Pentru a menține echilibrul, culorile trebuie combinate în proporție de 60%-30%-10%. Cea mai mare parte reprezintă culoarea dominantă, urmată de culoarea secundară, iar procentul de 10% este atribuit culorii care ajută la realizarea accentelor în aplicație.

- **Contrastul** – Prin folosirea contrastului, se oferă individualitate fiecărui element de interfață, făcându-le vizibile și accesibile. Contrastul este utilizat pentru evidențierea anumitor secțiuni sau pentru butoanele de tip ***call-to-action***.

- **Psihologia culorilor** – Fiecare culoare influențează utilizatorul într-un anumit fel și transmite un anumit mesaj. **De exemplu:**

- **Roșu** simbolizează atât un sentiment pozitiv (încredere), cât și unul negativ (erori, utilizarea incorectă a unui element).

- **Verde** simbolizează succes, calm, natură (de exemplu: un task executat cu succes, utilizarea corectă a aplicației, etc.).

- **Alb** reprezintă puritate și claritate (oferă un sentiment de simplitate în cadrul unei aplicații. Așa cum a spus Steve Jobs, inspirat de Leonardo da Vinci: *"Simplitatea este sofisticarea absolută"*).

Culorile se pot alege folosind instrumente specializate. De exemplu, https://paletton.com/ este o astfel de aplicație care asistă utilizatorul în generarea unei palete de culori corecte, pornind de la o culoare de bază. Dacă selectăm ca bază culoarea albastru, aplicația va sugera culorile complementare roșu și galben și va genera și paleta de culori aferentă acestora.