Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC
Curs 11

Cuprins

Afisarea paginata

Afisarea paginata este utila in cazul in care trebuie sa afisam un numar mare de elemente, numar care se modifica constant. De exemplu, trebuie sa afisam foarte multe categorii, articole, produse, etc. In acest caz se utilizeaza afisarea paginata, in cadrul careia dezvoltatorii implementeaza afisarea in functie de o serie de reguli. Regulile trebuie stabilite luand in considerare numarul de elemente pe care dorim sa-l afisam, tinand cont si de faptul ca acest numar o sa isi modifice valoarea in functie de actiunile utilizatorilor in cadrul aplicatiei (de ex: utilizatorii pot adauga/sterge articole sau produse).

Exemplu de implementare

Exemplul urmator este realizat in cadrul aplicatiei Engine de stiri (aplicatia dezvoltata in laborator).

Pentru afisarea paginata o sa se utilizeze din Bootstrap – Pagination → https://getbootstrap.com/docs/5.2/components/pagination/

Se vor afisa 3 articole pe pagina, modificandu-se atat metoda Index din ArticlesController, cat si View-ul Index corespunzator.

Includerea si 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>
```

Secventa de cod anterioara o sa se modifice astfel. Modificarile sunt scrise cu negru si bold.

```
@* Afisarea paginata a articolelor *@

<div>
    <nav aria-label="Page navigation example">
        <ul class="pagination">
            <li class="page-item">
                <a class="page-link"
href="/Articles/Index?page=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="/Articles/Index?page=@i">@(i)</a> </li>
            }

            <li class="page-item">
                <a class="page-link"
href="/Articles/Index?page=@(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();
            }


            // 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

In aceasta sectiune vom integra o componenta externa (componenta 3rd party). Din categoria componentelor 3rd party fac parte: servere web (Apache, Nginx, IIS, etc), framework-uri (.NET, Rails, Spring, etc), librarii.

Componenta pe care o vom integra este un editor de text open-source, bazat pe Bootstrap si jQuery, numita **Summernote**. Este o librarie de JavaScript folosita pentru a implementa un editor de tipul **WYSIWUG - What You See Is What You Get**

Acest tip de editor ofera posibilitatea prelucrarii textului de catre utilizatorii finali, asemanator cu scrierea textului in Word sau folosind limbaj de markup (HTML).

Pagina oficiala → https://summernote.org/

Exemplu de implementare

Exemplul urmator este realizat in cadrul aplicatiei Engine de stiri (aplicatia dezvoltata in laborator).

Se doreste utilizarea editorului de text atat in momentul adaugarii unui articol, cat si in momentul editarii acestuia.

Se integreaza Summernote accesand link-ul urmator si urmand pasii:

PASUL 1:

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

PASUL 2:

In **_Layout.cshtml** se include in Head componenta de css:

```
<link href="https://cdn.jsdelivr.net/npm/summernote@0.8.18/dist/summernote-lite.mi
n.css" rel="stylesheet">
```

```
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - ArticlesApp</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.1/font/bootstrap-icons.css">

    <link rel="stylesheet" href="~/ArticlesApp.styles.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />


    <!-- se include summernote css -->
    <link href="https://cdn.jsdelivr.net/npm/summernote@0.8.18/dist/summernote-lite.min.css" rel="stylesheet">

</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Articles" asp-action="Index">ArticlesApp</a>
```

PASUL 3:

In **_Layout.cshtml** se include in Body componenta de js:

```
<script src="https://cdn.jsdelivr.net/npm/summernote@0.8.18/dist/summernote-lite.min.js"></script>
```

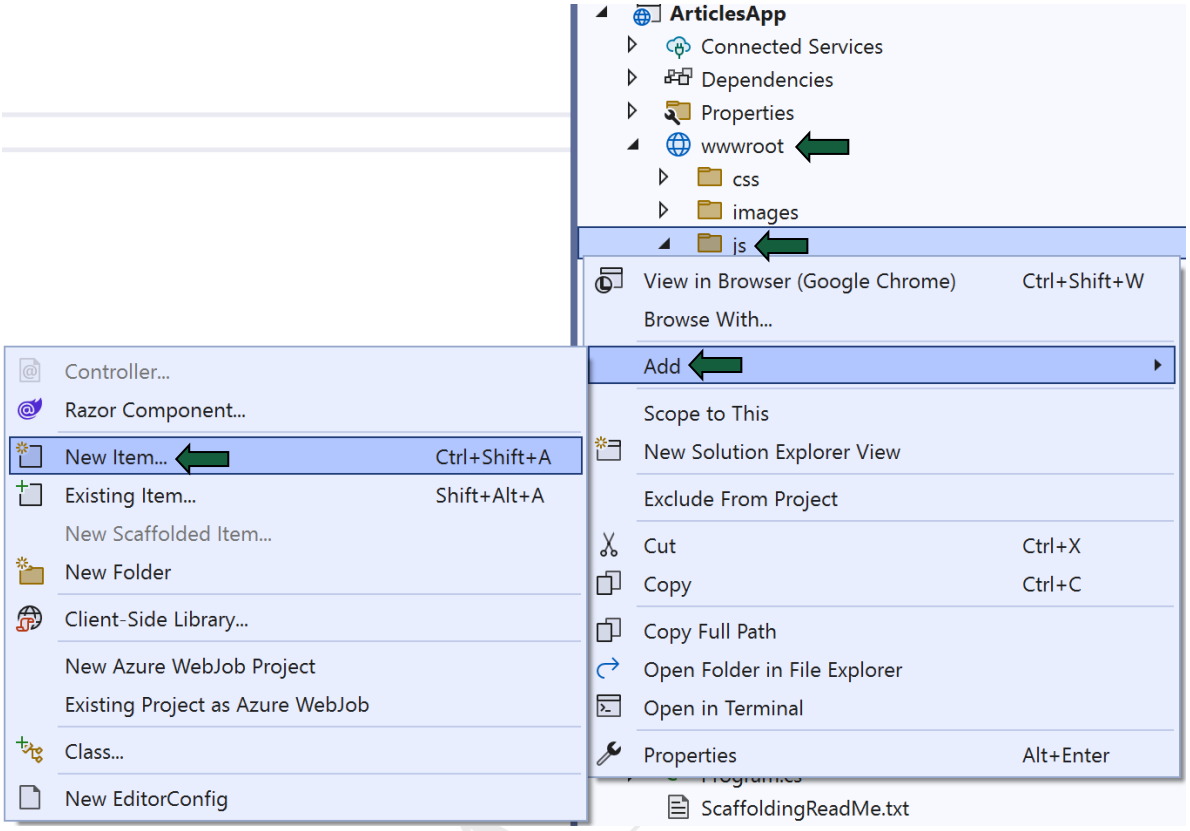
```
94      <footer class="border-top footer text-muted">
95          <div class="container">
96              &copy; 2022 - ArticlesApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
97          </div>
98      </footer>
99
100     <script src="~/lib/jquery/dist/jquery.min.js"></script>
101     <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
102
103     <script src="https://cdn.jsdelivr.net/npm/summernote@0.8.18/dist/summernote-lite.min.js"></script>
104
105     <script src="~/js/site.js" asp-append-version="true"></script>
106
107
108     @await RenderSectionAsync("Scripts", required: false)
109
110
111 </body>
112 </html>
```

PASUL 4:

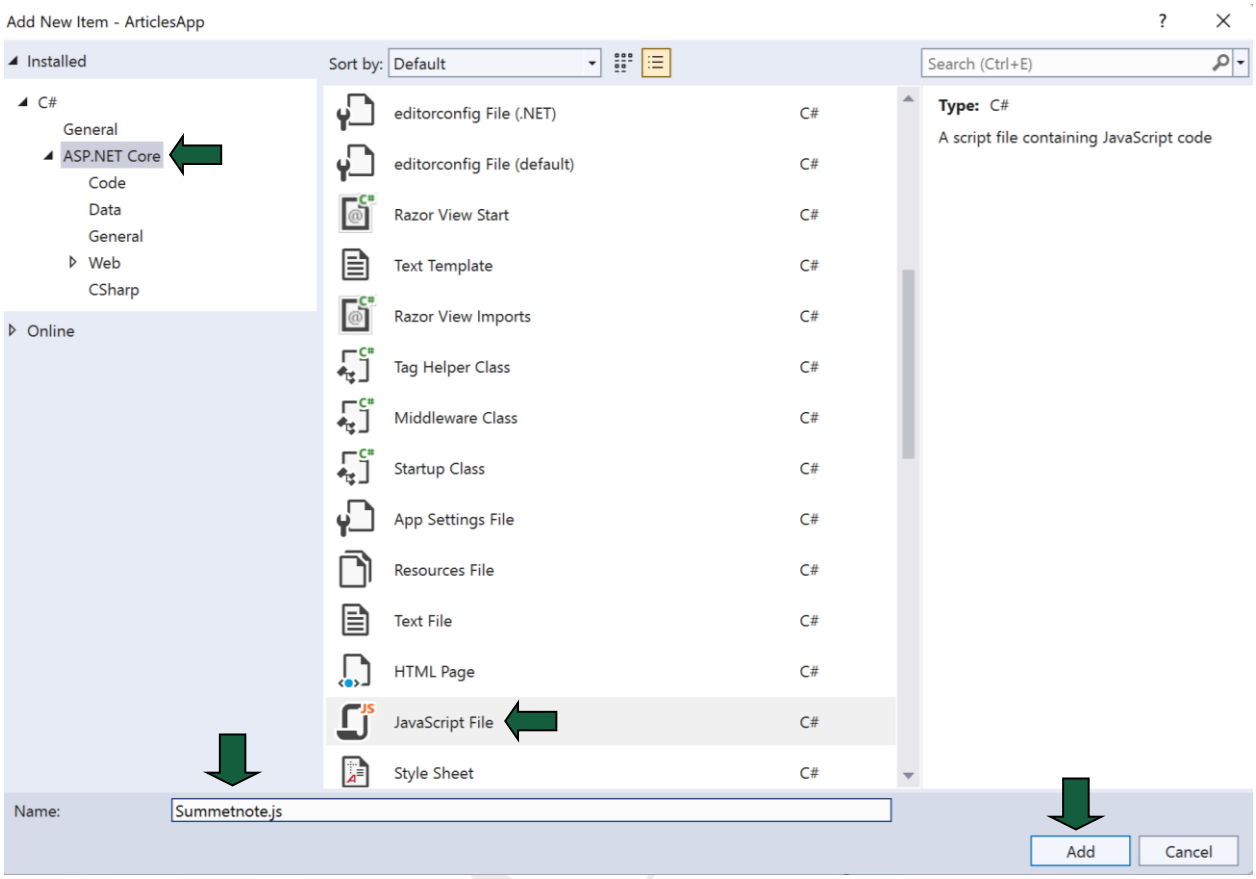
Editorul se initializeaza adaugand urmatoarea secventa de cod intr-un fiser de tip JavaScript (js).

```
    $(document).ready(function () {
    $('.summernote).summernote({
        height: 300,
        minHeight: 200,
        focus: true,
    });
});
```

PASUL 5:



PASUL 6:



PASUL 7:

Scriptul-ul se include in _Layout in Body:

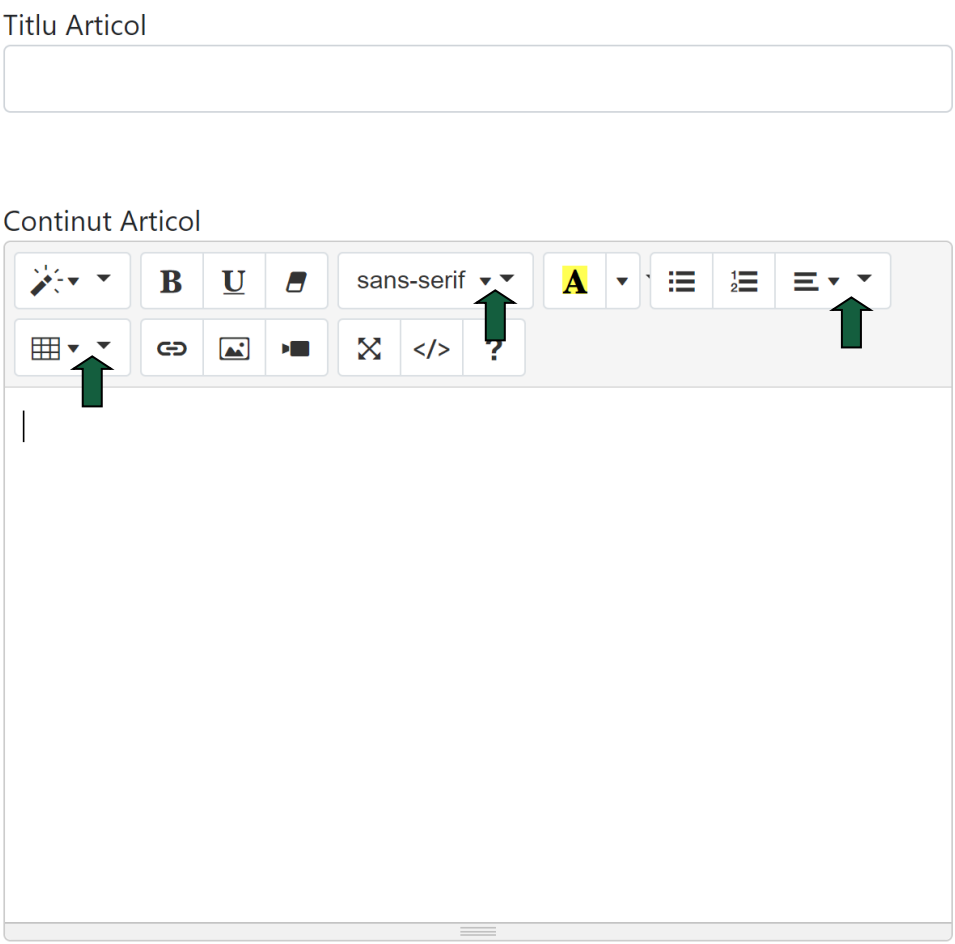
```
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2022 - ArticlesApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">P
        </div>
    </footer>

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

    <script src="https://cdn.jsdelivr.net/npm/summernote@0.8.18/dist/summernote-lite.min.js"></scri

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

    @await RenderSectionAsync("Scripts", required: false)


</body>
</html>
```

PASUL 8:

Pentru utilizarea editorului se foloseste clasa definita in fisierul Summernote.js

```
@Html.TextArea("Content", null, new { @class = "summernote" })
```

Dupa integrarea editorului se observa o problema de afisare:



Pentru afisarea corecta a dropdown-ului se utilizeaza (astfel incat sa nu afiseze acele sageti duplicat):

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

PASUL 9:

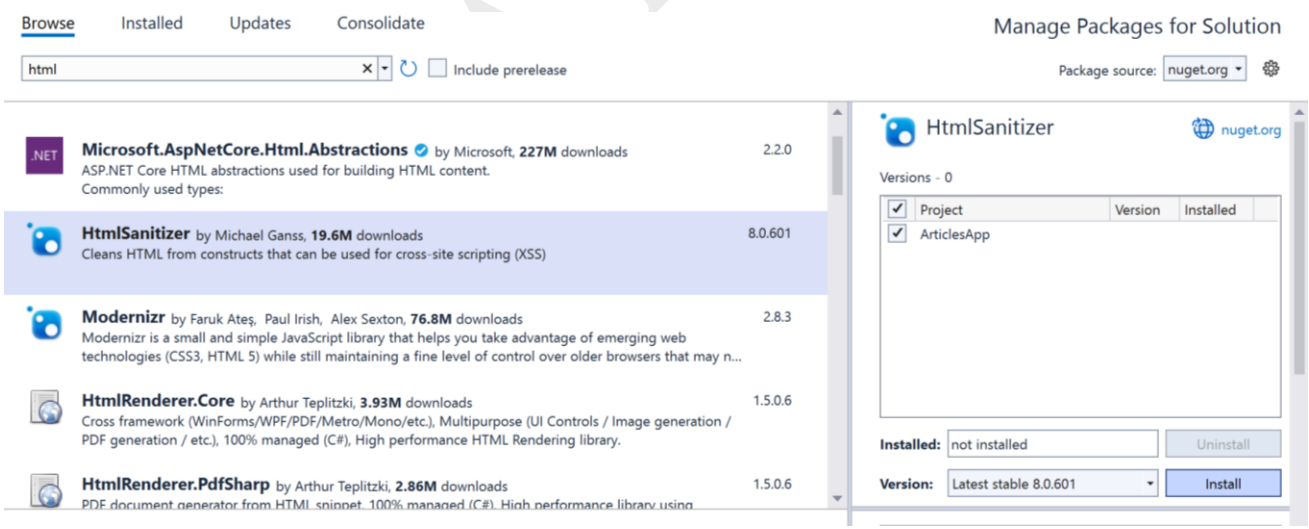
Pentru afisare vom utiliza `@Html.Raw()`

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

PASUL 10:

Prevenire XSS (Cross Site Scripting). XSS presupune posibilitatea inserarii unui cod, de catre o persoana neautorizata, intr-o pagina web. In cazul exemplului din cadrul cursului, un utilizator neautorizat poate introduce cod JavaScript, in formular, in momentul inserarii unui nou articol in platforma.

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



In cadrul adaugarii si editarii din Controller o sa avem urmatoarele modificari:

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);

          article.Content = (article.Content);

          db.Articles.Add(article);
          db.SaveChanges();
          TempData["message"] = "Articolul a fost adaugat";
          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.Content = requestArticle.Content;

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

In cazul editarii, pentru preluarea corecta a continutului articolului din baza de date, trebuie sa utilizam TextArea. Astfel, se inlocuieste helperul @Html.Editor cu helperul @Html.TextArea.

```
@* @Html.Editor("Content", new { htmlAttributes = new { @class =
"form-control summernote" } }) *@

@Html.TextArea("Content", Model.Content, new { @class = "form-
control summernote" })
```

Functionalitatea de cautare

Functionalitatea de cautare se afla integrata in orice aplicatie, indiferent de natura ei (aplicatie web, aplicatie mobile, desktop, etc).

Cel mai mare motor de cautare este **GOOGLE**, urmat de Bing si Baidu. Acestea ocupa in momentul de fata primele 3 locuri.

Descrierea si implementarea motorului de cautare

Pentru a exemplifica integrarea motorului de cautare, o sa utilizam aplicatia dezvoltata in laborator – Engine de stiri. In cadrul aplicatiei se integreaza functionalitatea de cautare a articolelor dupa **titlu**, **continut**, dar si dupa **continutul comentariilor** postate de utilizatori in cadrul articolelor respective.

Se utilizeaza 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();
}
```

Dupa cum se observa in implementarea din cadrul metodei Index din ArticlesController, trebuie sa ne asiguram ca afisarea paginata din cadrul sectiunii anterioare functioneaza si in momentul integrarii motorului de cautare.

Astfel, in cazul in care se foloseste motorul de cautare, URL-ul o sa fie de forma: `/Articles/Index/?search&page` ceea ce inseamna ca trebuie sa avem in ruta atat stringul cautat, cat si numarul paginii.

Afisarea paginata din cadrul View-ului Index o sa 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 intr-o aplicatie Web

Reguli de baza in design

Pentru realizarea aplicatiilor Web, in ceea ce priveste design-ul acestora, se pot respecta anumite reguli:

- **Simplitatea** – prea multa informatie intr-o pagina distrage atentia utilizatorului deoarece acesta trebuie sa citeasca si sa parcurga mult prea multa informatie pana la informatia de care are nevoie. Pastrand paginile simple, aplicatia va fi mult mai usor de folosit.

- **Design-ul este esential** – prima impresie conteaza. Aceasta regula se aplica si in dezvoltarea unei aplicatii web deoarece este primul lucru pe care il observa un utilizator atunci cand acceseaza aplicatia.

- **Culorile sunt foarte importante** – atunci cand alegeti culorile folositi o paleta consistenta de culori. De asemenea, sa va asigurati ca nu exista nuante foarte apropiate care nu pot fi deosebite si mai ales ca exista un contrast puternic intre text si fundal.

- **Navigarea ar trebui sa fie intuitiva** – un utilizator nu trebuie sa caute ceea ce doreste sa acceseze. Paginile trebuie sa fie bine organizate cu un design de tipul top-down, utilizatorii navigand usor printre diferitele sectiuni existente intr-o pagina.

- **Consistenta este extrem de importanta** – utilizatorii nu ar trebui sa aiba sentimentul ca viziteaza un alt site sau o alta aplicatie web de fiecare data cand acceseaza o alta pagina a aplicatiei. Consistenta face navigarea in aplicatie mult mai simpla.

- **Aplicatia trebuie sa fie responsive** – utilizatorii acceseaza aplicatia utilizand o varietate de device-uri, de la smartphone-uri la calculatoare personale, de aceea este esential ca aplicatia sa se incadreze oricarei

rezolutii. CSS media queries sunt ideale pentru realizarea rapida a unei aplicatii web responsive.

- **Utilizarea unui continut real in momentul dezvoltarii design-ului**. Orice aplicatie este bazata pe continut si se dezvolta in jurul continutului. De cele mai multe ori atunci cand se dezvolta design-ul nu se acorda importanta continutului, folosindu-se Lorem Ipsum in locul textului real si placeholder in locul imaginilor. Chiar daca totul arata bine in timpul dezvoltarii design-ului, nu o sa mai fie la fel atunci cand aplicatia va contine datele reale. Scopul in dezvoltarea unei aplicatii web este de a fi cat mai aproape de experienta reala a utilizatorului

- **Fontul** – in general fonturile Sans Serif, cum ar fi Arial si Verdana, sunt mai usor de citit intr-o aplicatie web, fiind fonturi fara finisaje decorative. Dimensiunea fontului pentru citirea cu usurinta este de 16px

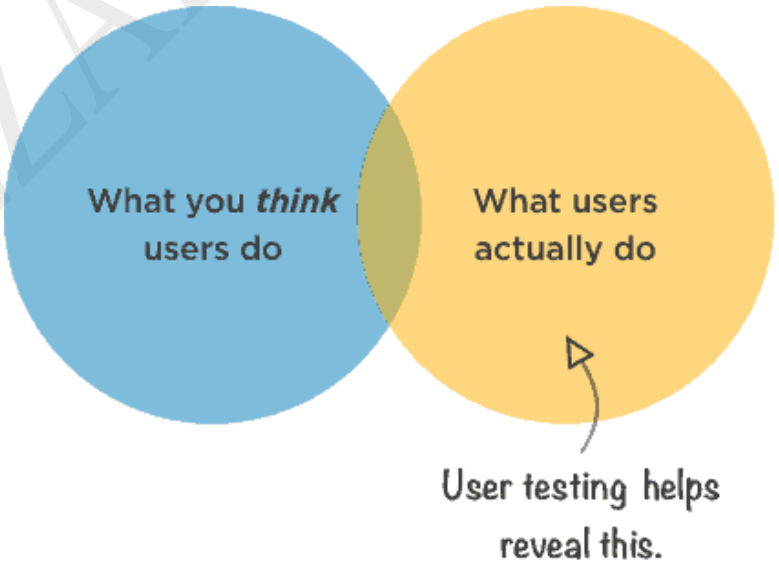
- **Imaginile** – ajuta la o buna interactiune cu utilizatorul
- **“F” Pattern Design** – informatiile sa fie prezentate, in ordinea relevantei, de la stanga la dreapta si de sus in jos, un studiu aratand ca partea de sus-stanga a ecranului este mult mai vizualizata decat cea de jos-dreapta

User Experience (UX)

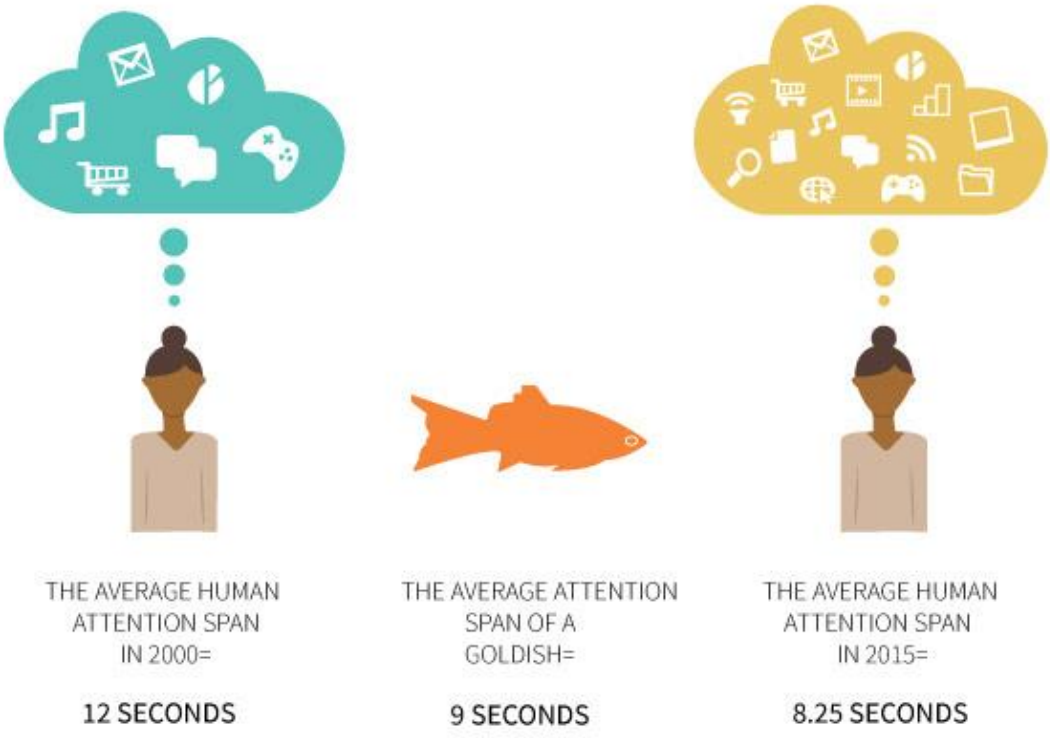
In primul rand, interfata cu utilizatorul (User Interface) este o parte din UX, existand o diferenta semnificativa intre cele doua. Interfata cu utilizatorul este spatiul unde utilizatorul interactioneaza cu aplicatia, cu diferite componente ale aplicatiei, iar User Experince este rezultatul final pe care il produce interactiune cu aplicatia.

- **Scopul aplicatiei** – cel mai important lucru este scopul aplicatiei; Cine o sa fie utilizatorii finali? Care sunt nevoile lor? Ce isi doresc ei? De aceea, dezvoltarea unei aplicatii poate incepe cu un “user research”, fiind un proces esential pentru UX.

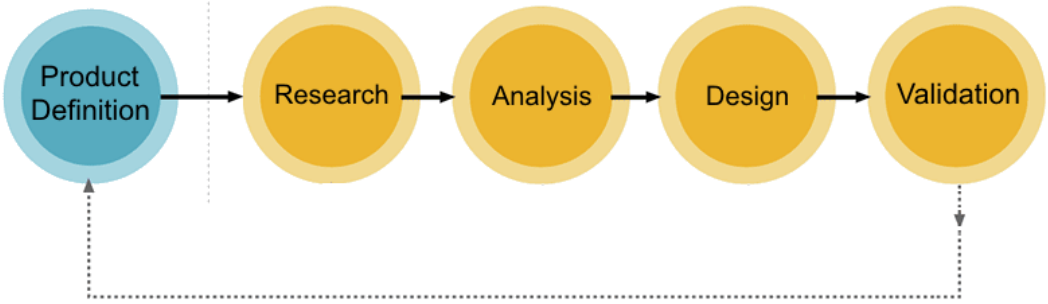
- **Tu nu esti utilizatorul final** – este un lucru esential in dezvoltarea unei aplicatii. Dezvoltatorul nu este utilizatorul final, iar cel mai important lucru este testarea de catre un utilizator real. Un dezvoltator nu se va comporta niciodata ca un utilizator real deoarece are experienta si va utiliza aplicatia exact cum a fost dezvoltata. Utilizatorii care vor folosi aplicatia au moduri diferite de gandire, un background diferit, experienta diferita sau chiar scopuri diferite, ceea ce face testarea cu useri reali cea mai buna forma de testare pentru o aplicatie web (usability testing)



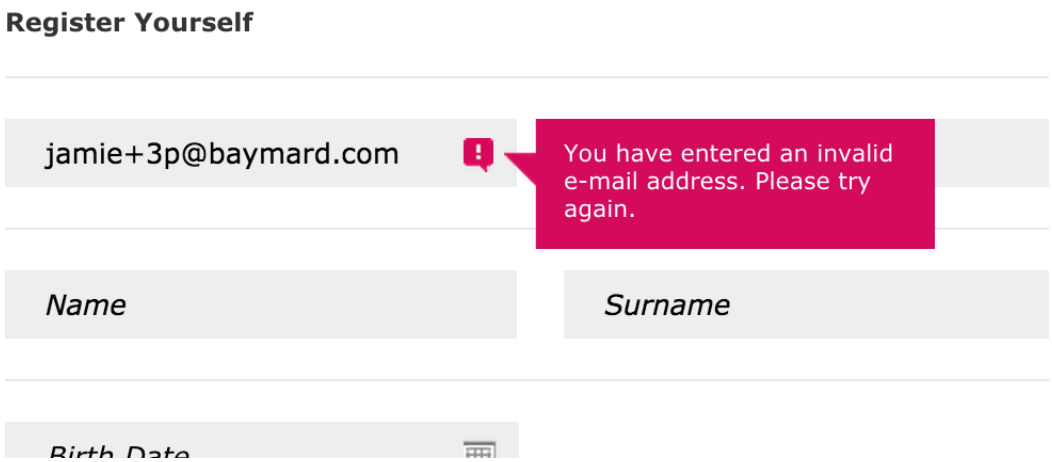
- **Adaptarea design-ului pentru atentia de scurta durata** – nu incarcati utilizatorii cu foarte multa informatie. O perioada de atentie inseamna acea perioada de timp in care cineva se concentreaza asupra unei sarcini fara sa fie distras. Un studiu realizat de Microsoft in 2015 a aratat faptul ca media atentiei umane a scazut de la 12 secunde la 8 secunde, ceea ce insemna ca avem o perioada de atentie mai scurta decat cea a unui pestisor auriu. Acest lucru inseamna ca este necesar ca designerii sa afiseze oamenilor informatiile necesara cat mai repede posibil. De aceea, lucrurile care nu sunt necesare pot fi eliminate. Acest lucru nu inseamna ca trebuie sa limitam experientele utilizatorilor intr-o aplicatie web, ci doar ca informatiile trebuie sa fie relevante.



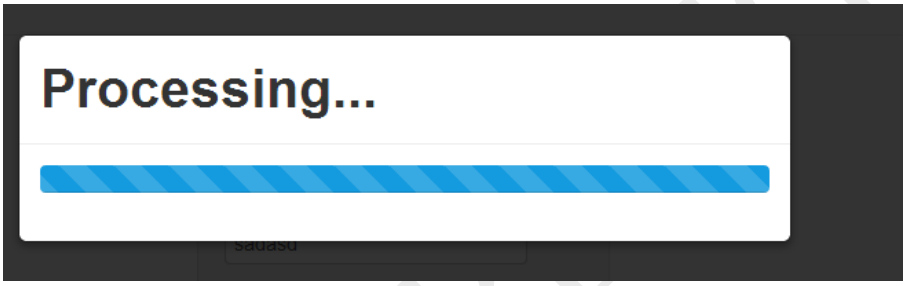
- **UX depinde de fiecare proiect in parte** – nu exista un UX general care poate fi aplicat oricarui proiect. Fiecare proiect este unic, are cerinte unice, ceea ce duce la un UX diferit fiecarei aplicatii. De exemplu, atunci cand dezvoltati un nou proiect este necesara alocarea unui timp suplimentar pentru a cerceta ce tipuri de utilizatori vor accesa aplicatia, dar si care sunt specificatiile aplicatiei.



- **Prevenirea erorilor este mai usoara decat rezolvarea lor** – multe erori apar din cauza utilizatorilor (email incorect, parole care nu respecta anumite reguli, etc.). In acest caz aplicatia poate fi dezvoltata oferind sugestii de completare si notificand utilizatorii de fiecare data.



- **Afisarea feedback-ului** – utilizatorii trebuie sa fie mereu informati de tot ceea ce se intampla in cadrul unei aplicatii: mesaje dupa trimiterea formularelor, mesaje in momentul editarii/stergerii datelor, feedback vizual. Acesta din urma este foarte important fiind un mijloc bun de informare.



- **Designul trebuie sa fie simplu si intuitiv**, utilizatorii folosind de fiecare data intuitia. Steve Krug spunea ca principalul motiv pentru care utilizatorii folosesc intuitia atunci cand aceseaza o aplicatie este ca nu le pasa. "*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 ca utilizatorii nu citesc informatia, ci doar o scaneaza, cautand cateva puncte de referinta care sa ii ghideze prin continutul paginii.

- **Spatiile albe sunt importante** – informatia este perceputa mult mai repede deoarece un utilizator, dupa cum am mentionat anterior, scaneaza informatia si o imparte in sectiuni usor de interpretat. Structurile complexe sunt greu de citit, scanat, analizat si lucrat cu ele.

Alegerea culorilor potrivite

Alegera culorilor in dezvoltarea unei aplicatii web, de cele mai multe ori, nu este atat de simpla. In continuare o sa urmarim cele mai importante reguli in acest sens: (https://flatuicolors.com/)

- **Tehnica 60-30-10** – aceasta tehnica se aplica si in decorarea caselor si este una simpla. Pentru a fi in balanta, culorile trebuie combinate in proportie de 60%-30%-10%. Cea mai mare parte reprezinta culoarea dominanta, urmata de culoarea secundara, iar procentul de 10% il reprezinta culoarea care ajuta la realizarea accentelor in aplicatie

- **Contrastul** – prin folosirea acestuia se aduce individualitate fiecarui element de interfata, facandu-le pe fiecare in parte vizibile si accesibile. Contrastul este folosit pentru punerea in evidenta a anumitor sectiuni sau pentru butoane de tip call-to-action

- **Psihologia culorilor** – fiecare culoare influenteaza utilizatorul intr-un anumit fel si livreaza catre acesta un anumit mesaj. De exemplu:

    - **rosu** simbolizeaza atat un sentiment pozitiv (incredere) cat si un sentiment negativ (erori, folosire incorecta a unui element)
    - **verde** simbolizeaza succes, calmitate, natura (un task executat cu succes, folosirea corecta a aplicatiei, etc)
    - **alb** reprezinta puritate si claritate (ofera sentimentul de simplitate in cadrul unei aplicatii. Cum a spus Steve Jobs, inspirat de Leonardo da Vinci: “simplitatea este sofisticarea absoluta”)

Culorile se pot alege folosind tool-uri existente. De exemplu, **https://paletton.com/** este o astfel de aplicatie care asista utilizatorul sa genereze paleta de culori corecte plecand de la o culoare de baza. Plecand de la culoarea de baza **albastru** observam ca aplicatia ne alege culorile complementare **rosu** si **galben** si genereaza si paleta de culori aferenta acestor culori.