



UNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE
MATEMATICĂ ȘI INFORMATICĂ**



SPECIALIZAREA INFORMATICĂ

Lucrare de licență

Aplicație web tip magazin online

Absolvent

Paraschiv

Alexandru-Andrei

Coordonator științific

Lect.dr. Sipoș Andrei-Valentin

București, iunie 2022

Rezumat

În cadrul proiectului de licență am dezvoltat o aplicație web folosind ASP.NET MVC, ce are trei moduri de folosire: utilizator neînregistrat, înregistrat și admin. Utilizatorii neînregistrați pot vedea produsele și categoriile din care fac parte, însă sunt redirecționați către pagina de autentificare atunci când doresc să efectueze o acțiune. După accesarea unui cont, aceștia pot lăsa *review-uri* produselor, le pot edita sau șterge. Produsele pot fi adăugate într-un coș de cumpărături unde există opțiunea de a le crește cantitatea, de a o scădea sau de a le șterge. Utilizatorii înregistrați mai pot plasa comenzi al căror status urmează a fi modificat de către admin. Există și abilitatea de a completa un formular de contact, în cazul în care o persoană dorește să transmită un mesaj. Pentru admin este dezvoltat un buton ce îi deschide un meniu de opțiuni special creat pentru el precum: vizualizarea tuturor comenzilor efectuate, adăugarea produselor în aplicație, afișarea fiecărui formular de contact sau a tuturor utilizatorilor pentru a le modifica anumite date. *Designul* este realizat folosind Bootstrap 4, ce oferă un aspect modern și totodată *responsive*. În lucrare am vorbit despre motivație, scop, tehnologiile folosite, contribuția personală, provocări și aspecte ce pot fi dezvoltate ulterior.

Abstract

In the project I developed a web application using ASP.NET MVC, which has three different modes of use: unregistered user, registered user and admin. Unregistered users can see the products and categories, but they are redirected to the login page when they try to make an action. After accessing an account, they can add reviews to the products, edit or delete them. Products can be added to a shopping cart where there is the option to increase the quantity, decrease it or delete them. Registered users can also place orders whose status is going to be changed by the admin. There is also the ability to fill out a contact form if a person wants to convey a thought. For admin, a button is developed to opens a menu of options specially created for him, such as: viewing all orders placed, adding products to the application, displaying each contact form or displaying all users so their data can be modified. The design is made using Bootstrap 4, which offers a modern and responsive look. In the paper, I talked about motivation, purpose, technologies used, personal contribution, challenges and aspects that can be developed later.

Cuprins

Cuprins.....	3
Introducere	5
1.1 Motivație	5
1.2 Scopul temei.....	5
1.3 Contribuție proprie.....	5
1.4 Structura lucrării	6
Preliminarii.....	7
2.1 ASP.NET MVC	7
2.1.1 Model.....	8
2.1.2 View	8
2.1.3 Controller.....	8
2.2 .NET Framework, C# și IDE	8
2.2.1 .NET Framework.....	8
2.2.2 C#	9
2.2.3 IDE	9
2.3 NuGet, Entity Framework și Migrații	9
2.3.1 NuGet	9
2.3.2 Entity Framework.....	9
2.3.3 Migrații	10
2.4 Componente și facilități ale unui View	10
2.4.1 HTML, CSS, JavaScript.....	10
2.4.2 Bootstrap	10

2.4.3 ASP.NET Razor	11
2.4.4 Helpere	11
2.5 Lucidchart	11
2.6 Microsoft Azure	12
2.6.1 Azure App Services	13
2.6.2 Azure SQL Databases	13
Contribuție.....	14
3.1 Abordări existente	14
3.2 Fluxuri de utilizare	14
3.2.1 Fluxul pentru utilizatorul neînregistrat.....	14
3.2.2 Fluxul pentru utilizatorul înregistrat.....	15
3.2.3 Fluxul pentru admin	16
3.3 Funcționalitățile aplicației.....	16
3.3.1 Modelele create	16
3.3.2 Controllerele create	20
3.3.3 View-urile create	26
3.4 Design	39
Concluzii	41
4.1 Provocări	41
4.2 Posibile dezvoltări ulterioare	43
4.3 Concluzie	43
Bibliografie	44
Lista figurilor.....	45

Capitolul 1

Introducere

1.1 Motivație

Pentru realizarea proiectului de licență am ales să dezvolt o aplicație web de tip magazin online, care va ajuta o firmă ce se ocupă de comerțul cu odorizante, de diferite tipuri. Această firmă a avut în trecut un *site* care nu avea o interfață ușor de înțeles pentru orice tip de utilizator, nu avea detalii relevante în legătură cu produsele și nici funcționalități. Consider că proiectul creat poate ajuta în proporții foarte mari firma, atât printr-un *design* mult mai plăcut pentru potențialii clienți, cât și prin organizarea mult mai ușoară a comenzilor efectuate.

1.2 Scopul temei

Tema abordată are ca scop realizarea unei aplicații web în ASP.NET MVC, folosind noțiunile învățate în cei trei ani de facultate, atât din materiile predate, cât și din lectura individuală. Prin dezvoltarea programului, am pus în practică cunoștințele acumulate în legătură cu planificarea unui proiect: am purtat discuții cu firma respectivă pentru a definitiva cerințele în funcție de importanța acestora, am ales tehnologiile, am schițat arhitectura, am implementat *backendul*, *frontendul* și am populat bazele de date cu itemi relevanți.

1.3 Contribuție proprie

În procesul de dezvoltare a aplicației am încercat să realizez idei originale. Consider că *designul* paginilor este unul profesional și simplist, fiind modelat de către mine. Sistemul de efectuare a comenzilor, prin folosirea unui tabel auxiliar pentru a salva permanent produsele comandate, este de asemenea un aspect ce intră la partea de contribuție proprie, alături de formularele de contact și *review-urile*. *Hostarea* aplicației pe Microsoft Azure a fost un aspect important adăugat pentru experiența utilizatorilor.

1.4 Structura lucrării

În capitolul întâi au fost expuse ideile de la care a pornit această aplicație, prin dorința de a ajuta o firmă și în același timp de a pune în practică abilitățile dobândite.

Capitolul al doilea, numit *Preliminarii*, cuprinde partea de tehnologii folosite, fiind explicate noțiuni teoretice alături de aplicarea lor în program. Astfel, primele informații apar despre ASP.NET MVC, urmat de limbajul cel mai popular ce caracterizează *framework-ul* .NET. Tot aici mai sunt definite anumite tehnologii des folosite, componentele din partea de *frontend*, o platformă online pentru realizarea de diagrame și platforma Microsoft Azure pentru serviciul de *hosting*.

Capitolul al treilea, *Contribuție*, este cel mai complex capitol și prezintă pentru început alte abordări existente pentru tema propusă de către mine. În continuare sunt analizate fluxurile de utilizare pentru fiecare tip de utilizator și fiecare model, controller și view dezvoltat. Partea de final a acestui capitol are în vedere *designul* aplicației.

În capitolul patru se află concluzia, provocările apărute pe durata dezvoltării aplicației alături de ideile de rezolvare abordate și posibile dezvoltări viitoare.

Ultimul capitol, *Bibliografie*, conține o listă cu sursele folosite în realizarea lucrării și una cu figurile definite.

Capitolul 2

Preliminarii

2.1 ASP.NET MVC

ASP.NET MVC este un *framework* construit pentru crearea de aplicații web, dar și a API-urilor. Arhitectura MVC se folosește de trei componente principale, bine legate între ele: modelul, view-ul și controllerul. Ca și conexiune, modelul nu depinde de nimeni, view-ul depinde de controller, iar, în final, atât controllerul cât și view-ul depind de model. Interacțiunea cu utilizatorul este realizată de către controller care preia date din model pentru a le afișa în view-ul corespunzător.^[5] În poza următoare este desenat principiul explicat anterior.

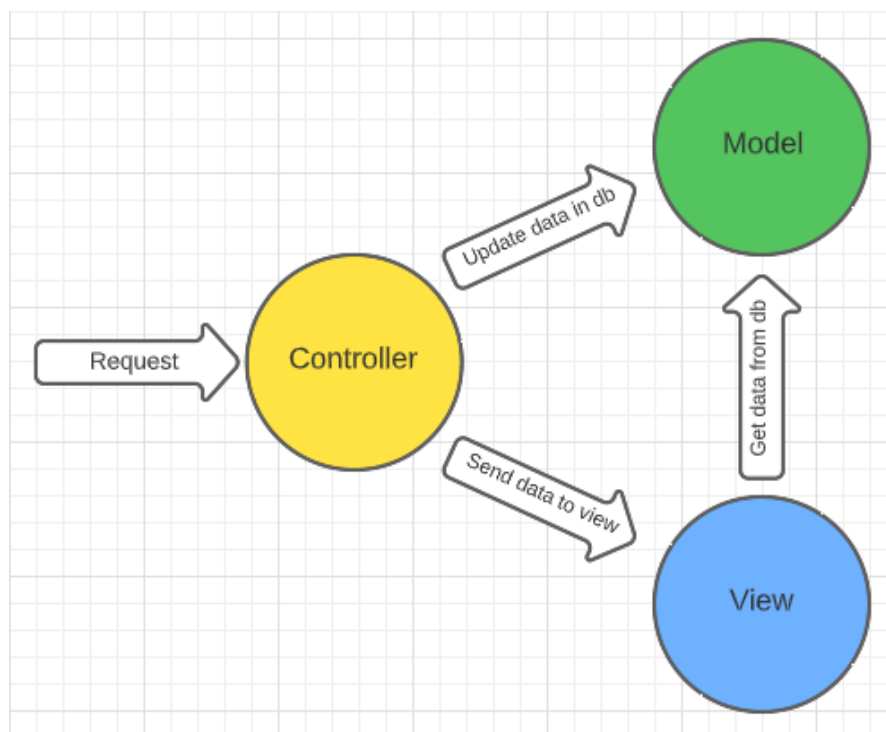


Figura 2.1 – Arhitectura MVC

2.1.1 Model

Modelul reprezintă ansamblul tuturor claselor ce depind de scopul aplicației.^[4] Datele sunt stocate într-o bază de date, gestionate și modificate în urma cererilor formulate de view-uri și controllere.

2.1.2 View

View-ul afișează datele din model, pe ecran, prin intermediul HTML. Acesta poate conține și cod JavaScript, iar stilizarea este construită prin CSS. Se mai pot alătura și diferite *framework-uri* precum Bootstrap, Angular sau librării specifice de JavaScript ca și React. Pe lângă datele din model, view-ul poate afișa orice informație prezentă în cadrul aplicației, deci acesta face legătura dintre utilizator și program. Există view-uri legate de către un anumit *request* sau acțiune dintr-un controller și view-uri de tip *Partial*. Cele de tip *Partial* definesc bucăți dintr-un view ce pot fi refolosite în una sau mai multe pagini.

2.1.3 Controller

Controllerul preia *requesturi* de tip HTTP, scrise în cod C#. Anumite instrucțiuni sunt date de către utilizator, controllerul pasează aceste date către model, iar în final este selectat view-ul corespunzător cu informațiile cerute. În ASP.NET MVC, fiecare controller are în spatele său o clasă, iar regula de numire vizează ideea *ClasăController*, fiind necesară uneori forma de plural al numelui clasei. De exemplu, pentru aplicația dezvoltată, am creat pentru modelul *Freshener*, controllerul *FreshenersController*.

2.2 .NET Framework, C# și IDE

2.2.1 .NET Framework

.NET Framework este creat pentru Windows de către Microsoft. Acesta folosește diferite limbaje de programare precum C#, F# sau Visual Basic. .NET Framework este alcătuit din două părți: *Common Language Runtime (CLR)* și *Base Class Library*. CLR execută programul, iar *Base Class Library* conține clasele din .NET.^[1]

2.2.2 C#

Limbajul C# este unul de tip compilat, codul sursă scris fiind transformat de către compilator în cod mașină. Dacă nu există erori de compilare după această transformare, din codul mașină se produce cod executabil și se rulează programul. C# este o derivare a limbajului C++, deci este și orientat pe obiect.^[1]

2.2.3 IDE

Termenul de *IDE* face referire la spațiul sau programul în care se realizează o aplicație, într-un anumit limbaj de programare. Pentru platforma creată am folosit inițial Visual Studio 2017, iar ulterior am schimbat cu Visual Studio 2022 pentru a avea o stabilitate mult mai bună și pentru a folosi funcționalitatea de publicare online.

2.3 NuGet, Entity Framework și Migrații

2.3.1 NuGet

Microsoft a dezvoltat o platformă numită *NuGet* prin care pot fi descărcate, instalate, actualizate sau șterse diferite librării, fie scrise de utilizatori, fie de companii. Cele mai populare librării din NuGet sunt: *Newtonsoft.Json*, *Microsoft.Extensions.DependencyInjection*, *Serilog*, *EntityFramework*.

2.3.2 Entity Framework

Entity Framework facilitează legarea obiectelor din modele cu tabelele unei baze de date relaționale. Entity Framework permite programatorilor să folosească date ale unor obiecte din clase definite, fără să își pună problema schimbării bazei de date.^[2] În aplicația dezvoltată am folosit versiunea EF6, mai exact 6.4.4, precum se poate vedea mai jos.

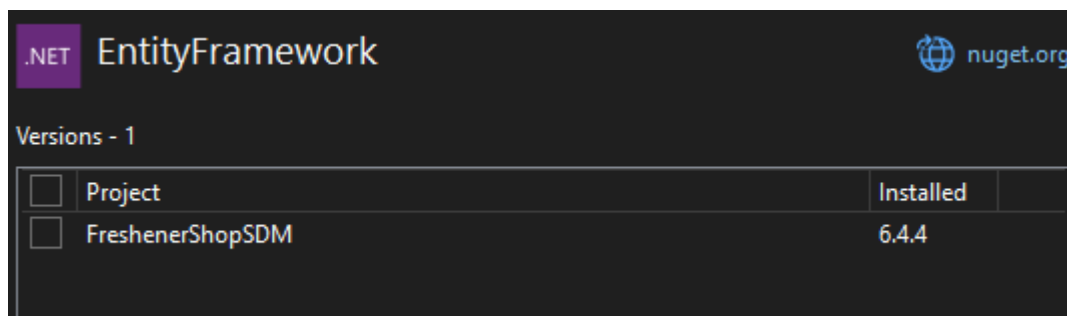


Figura 2.2 – Versiune Entity Framework

2.3.3 Migrații

Migrațiile sunt de două tipuri: *Code First Migration* (Migrații Automate) și *Code Based Migration*.

Primul tip, *Code First Migration*, dau voie Entity Framework-ului să realizeze migrații automate când apar modificări asupra bazei de date. Atunci când aplicația este rulată, Entity Framework observă acele schimbări și face o actualizare a bazei de date la ultima versiune.

Cel de-al doilea tip de migrații, *Code Based*, nu realizează acele schimbări în mod automat. Este necesar un cod adițional explicit, această variantă fiind des întâlnită atunci când se dorește un *upgrade* sau *downgrade* la o anumită versiune.^[1]

2.4 Componente și facilități ale unui View

2.4.1 HTML, CSS, JavaScript

Cele trei sunt componentele de bază pentru *frontendul* unei aplicații web. În ASP.NET MVC, fișierele de CSS și JavaScript pot fi grupate în *bundle-uri* și încărcate ușor în pagina de HTML prin metoda *Render()*.

2.4.2 Bootstrap

Bootstrap este unul dintre cele mai populare și importante *framework-uri* de CSS. Utilizarea acestuia conferă aplicației un aspect modern și totodată atinge și partea de *responsive*. În proiectul meu am folosit versiunea de Bootstrap 4.6.1, ultima versiune de Bootstrap 4, dar nu și cea mai nouă, existând Bootstrap 5, poza următoare evidențiind detaliul prezentat.

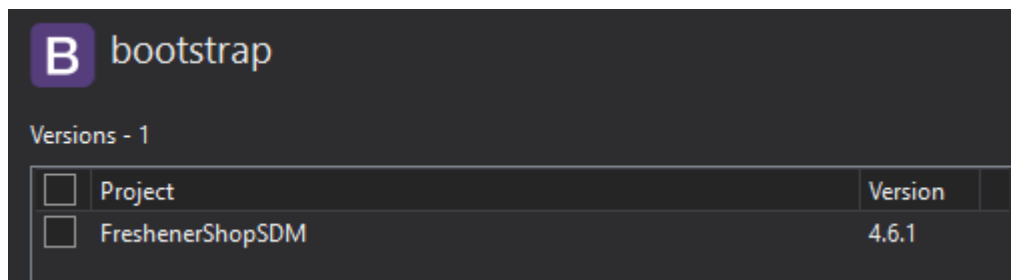


Figura 2.3 – Versiune Bootstrap

2.4.3 ASP.NET Razor

Razor oferă posibilitatea utilizatorilor de a introduce cod C# printre *tagurile* de HTML. Prin utilizarea simbolului @ se începe o secvență de cod *server-side*.

2.4.4 Helpere

ViewBag este un *helper* care, împreună cu simbolul @ specific ASP.NET Razor, ajută controllerul să trimită date către view, în special obiecte.

ViewData este un alt *helper*, asemănător cu *ViewBag*. Diferența este dată de faptul că *ViewData* este reprezentat de un dicționar.

Helperul model facilitează transmiterea datelor dintr-un anumit model la un anumit view.

TempData poate redirecționa o valoare dintr-o anumită acțiune într-alta, fiind disponibilă doar la prima accesare.

2.5 Lucidchart

Lucidchart este o platformă online care ajută utilizatorii în crearea de diagrame, în funcție de tipul dorit. Astfel, Lucidchart oferă posibilitatea de a modela diagrame entitate-relație, pentru scheme logice de Kubernetes, AWS, Cisco și multe altele. Serviciile platformei pot fi folosite atât în varianta gratuită, cât și contra cost. Pentru aplicația mea am folosit varianta gratuită, fiind suficientă pentru îndeplinirea cerințelor mele. Mai jos se poate vedea diagrama completă a aplicației dezvoltate.

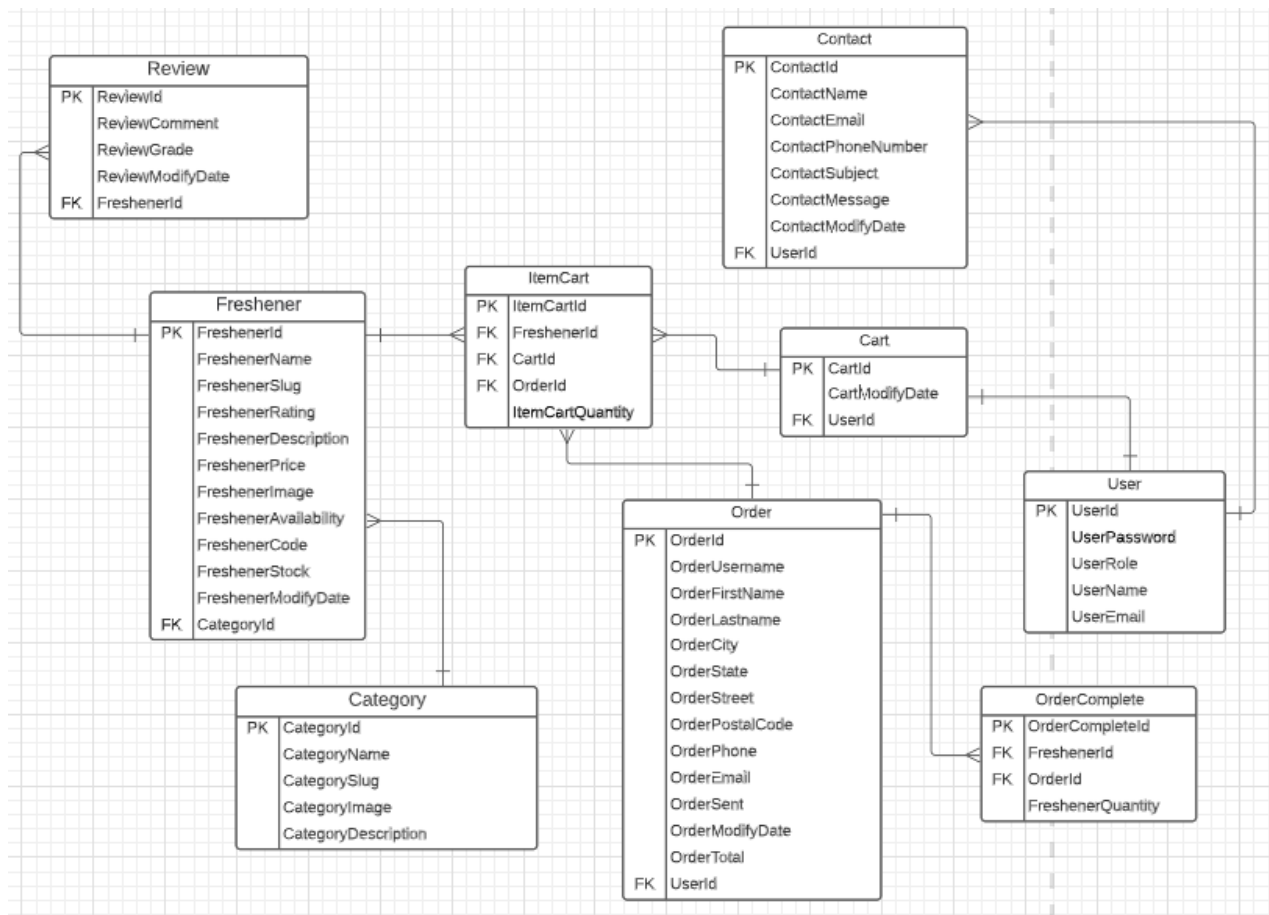


Figura 2.4 – Diagrama aplicației în Lucidchart

2.6 Microsoft Azure

Microsoft Azure este o platformă online ce are scopul de a ajuta utilizatorii în a gestiona mai ușor *site-urile*, bazele de date sau mașinile virtuale. Serviciile oferite pot fi accesate atât gratuit, cât și contra cost. Fiind necesară conectarea cu un cont Microsoft pentru a folosi platforma Azure, am ales să folosesc contul dat de către facultate și mi-au fost alocați 200 USD pentru o perioadă de 30 de zile, urmând să plătesc după această perioadă pentru a folosi în continuare serviciile create.

Pentru aplicația mea am accesat serviciul de *hosting* pentru *site-uri web*, cât și crearea și găzduirea unei baze de date. Prima funcție este în varianta gratuită, iar cea de-a doua este plătită.

2.6.1 Azure App Services

Odată cu setarea corectă a serviciului de *hosting web*, utilizatorului îi este oferit un panou de control unde poate vedea cele mai importante date în legătură cu aplicația. Este oferit numărul de erori apărute, dimensiunea datelor de intrare, cât și de ieșire, numărul de *requesturi* și timpul mediu de răspuns. Fiind varianta gratuită de server, timpul de răspuns pentru cererile de pe *site* este în general peste medie, după cum se poate observa și în poza următoare. *Link-ul site-ului* este <https://freshenershopsdm20220525163306.azurewebsites.net/>.

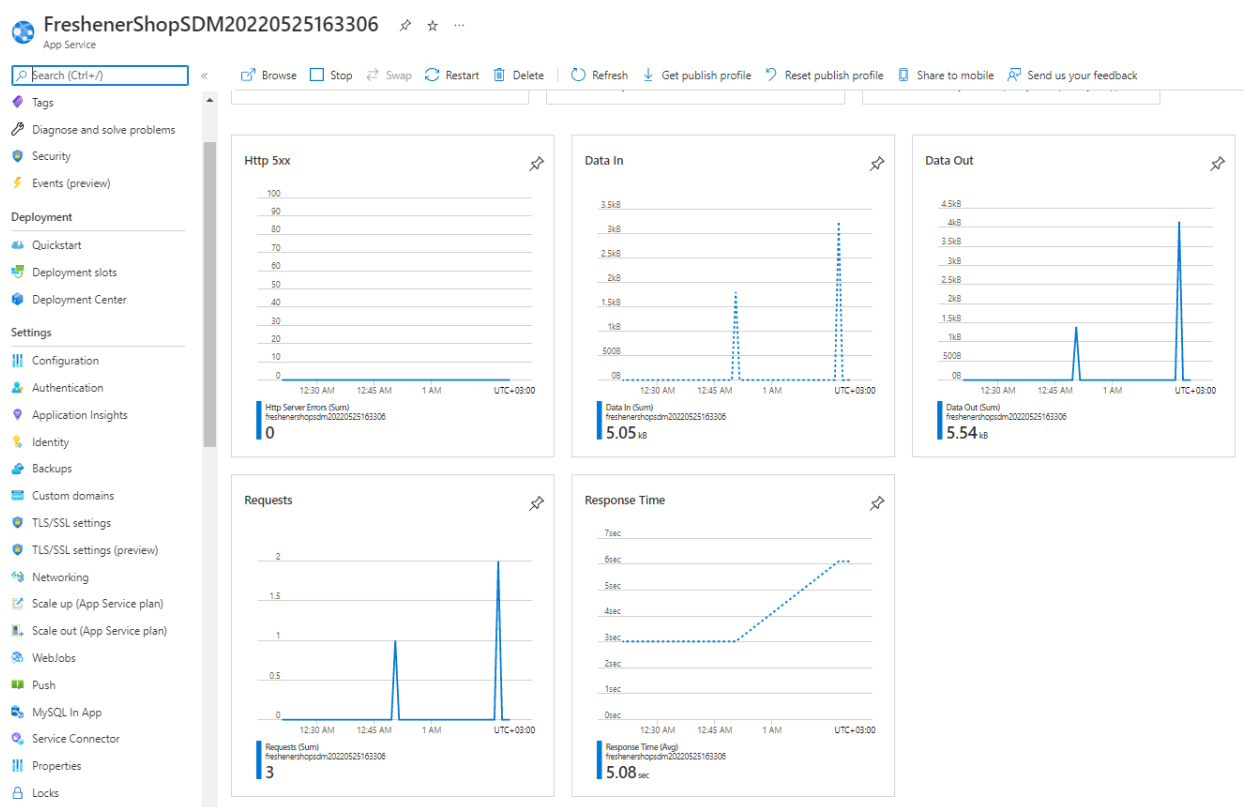


Figura 2.5 – Panoul de control Azure pentru *hosting web*

2.6.2 Azure SQL Databases

Pentru a putea folosi în totalitate funcționalitatea de *Publish* dată de către Visual Studio 2022 am configurat și o bază de date de tipul Azure SQL. Aceasta dispune de o multitudine de servicii pentru gestionare, precum *backupuri*, actualizări, modificări. Datorită faptului că baza de date Azure rulează pe SQL Server, versiunea este întotdeauna cea mai recentă. [7]

Capitolul 3

Contribuție

3.1 Abordări existente

Aplicația este realizată în ASP.NET MVC, o tehnologie ce facilitează dezvoltarea programelor complete, ușurează *debuggingul*, are o comunitate bine definită, iar numeroasele librării ce pot fi accesate de către dezvoltatori sunt însoțite de *update-uri* regulate. În prezent, domeniul actual pune la dispoziție și alte tehnologii, precum WordPress, Laravel sau Django, însă acestea au diferite minusuri importante. Spre exemplu, aplicațiile de tip WordPress au numeroase *template-uri*, dar flexibilitatea în schițarea propriului *design* lipsește. Laravel, la fel ca și alte *framework-uri* de PHP, scad în popularitate, librăriile lor nemaiprimind la fel de mult suport tehnic. Iar Django poate avea probleme în viteza limbajului, având la bază Python, un limbaj interpretat (există un interpretor care schimbă fiecare linie de cod în cod mașină și îl execută).

3.2 Fluxuri de utilizare

În prezent, aplicația dispune de trei fluxuri de utilizare, fiecare definind un mod de lucru des întâlnit pe *site-urile* de tip magazin online: utilizator neînregistrat, utilizator înregistrat și admin.

3.2.1 Fluxul pentru utilizatorul neînregistrat

Acest flux este destinat persoanelor care descoperă recent *site-ul* și doresc să vadă produsele și categoriile din care fac parte. *Review-urile* articolelor din aplicație pot fi și acestea văzute, alături de paginile de *Home*, *About* și *Contact*. Lista de produse poate fi modificată în funcție de sortarea aleasă, mai exact crescător sau descrescător după preț, *rating* sau data adăugării. De asemenea, poate fi folosită și funcționalitatea de *search* ce filtrează rezultate în funcție de nume sau descriere. Atunci când un utilizator neînregistrat dorește să lase un *review*, să

trimită un formular de contact sau să adauge în coșul de cumpărături un articol, este redirecționat către pagina de *Login*, care duce către următorul flux de utilizare.

3.2.2 Fluxul pentru utilizatorul înregistrat

Odată cu înregistrarea utilizatorului pe platforma web, acesta se poate bucura de un număr mult mai mare de funcționalități. Pot fi trimise formulare de contact și pot fi lăsate și editate *review-uri*. Utilizatorul are dreptul de a adăuga produse în coșul de cumpărături, de a le modifica numărul sau de a le șterge din acesta. După stabilirea finală a listei de produse din coșul de cumpărături, utilizatorul este redirecționat către un formular pentru completarea adresei de livrare, în final afișându-i-se un mesaj de mulțumire, urmat de numărul comenzii. Totodată, pentru acest flux de utilizare este disponibil și un meniu special care afișează lista de comenzi, alături de detaliile fiecăreia. Aceste comenzi pot fi anulate doar dacă administratorul aplicației nu a trimis deja comanda, în caz contrar apărând un text de informare, precum este exemplificat mai jos.

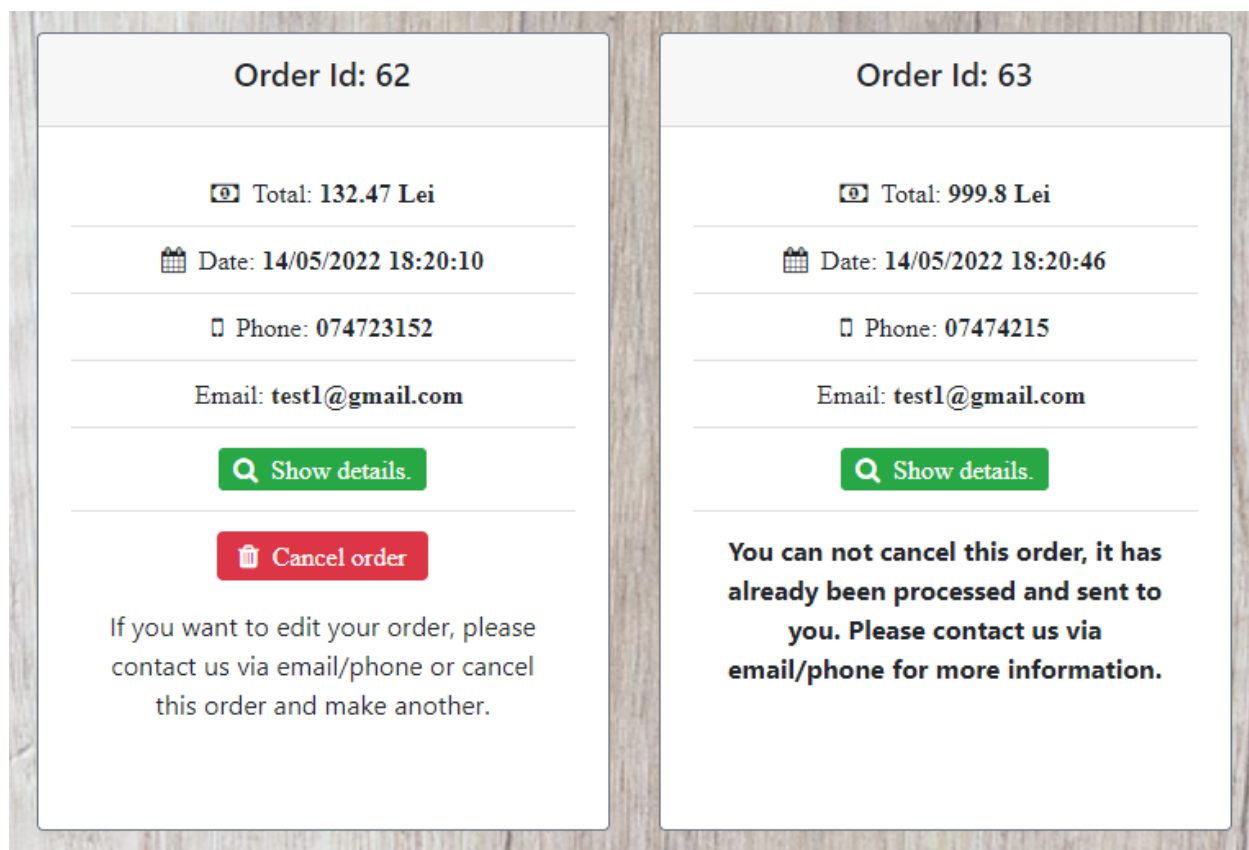


Figura 3.1 – Două comenzi plasate de un utilizator înregistrat

3.2.3 Fluxul pentru admin

Fluxul pentru admin conține toate funcționalitățile utilizatorului înregistrat, la care se alătură și altele noi. Dreptul de a adăuga, edita sau șterge atât produse cât și categorii, dreptul de a șterge sau edita *review-urile* oricărui utilizator (utilizatorul înregistrat poate edita doar *review-urile* proprii) sunt funcționalități create doar pentru admin. Acesta dispune și de anumite butoane speciale, de unde poate vedea lista tuturor comenzilor plasate, cu detaliile relevante a fiecăreia și le poate șterge, edita sau marca ca și *trimis*. Tot în meniul descris se mai găsește și o listă a tuturor utilizatorilor, adminul putând să îi șteargă, să le editeze unele date și să îi promoveze. Lista tuturor formularelor de contact este și ea prezentă și oferă posibilitatea adminului de a vedea toate detaliile trimise și de a le șterge. În final, fiind o funcționalitate des folosită, adminul dispune și de un buton pentru adăugarea unui produs nou pe platforma web, ca cel atașat în continuare.

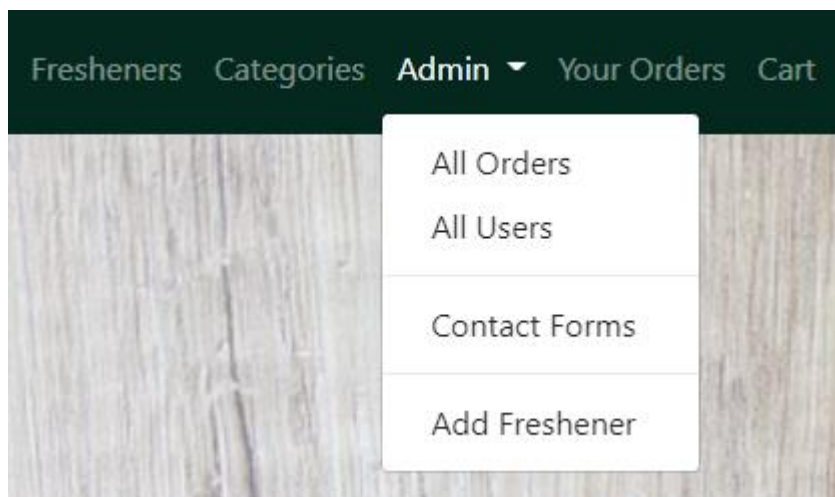


Figura 3.2 – Meniu special pentru admin

3.3 Funcționalitățile aplicației

3.3.1 Modelele create

Modelul principal al aplicației se numește *Freshener* și conține date relevante pentru fiecare produs afișat pe *site* precum: nume, *slug* (ajută partea de URL a unui *site*, de obicei este numele produsului, iar între cuvinte se află cratime), descriere, preț, *rating*, adresa către o imagine, disponibilitate, cod, stoc, data adăugării în baza de date, categoria din care face parte și o listă de *review-uri*. Datele cele mai importante ale unui produs au validări, nepermițând

necompletarea acestora atunci când sunt adăugate în baza de date. Lista de *review-uri* este definită de tip *ICollection*, care pe lângă tipul *IEnumerable* ce conține doar metodele de *get* și *loop*, adaugă funcționalități de ștergere, numărare, verificare și adăugare. Pe lângă cheia primară incrementată automat *FreshenerId*, modelul definește și o cheie externă pentru categoria din care face parte și una pentru utilizatorul ce a adăugat produsul (*CategoryId* și *UserId*). Mai sunt prezente liste de legătură cu tabelele *ItemCart* și *OrderComplete* ce vor fi prezentate ulterior. Mai jos am atașat codul pentru cele explicate anterior.

```
public class Freshener
{
    [Key]
    public int FreshenerId { get; set; }

    [Required(ErrorMessage = "The name is mandatory.")]
    [StringLength(128, ErrorMessage = "The name is maximum 128
character long.")]
    public string FreshenerName { get; set; }

    [StringLength(256, ErrorMessage = "The slug is maximum 256
character long.")]
    public string FreshenerSlug { get; set; }

    [Required(ErrorMessage = "The description is mandatory.")]
    [DataType(DataType.MultilineText)]
    public string FreshenerDescription { get; set; }

    [Range(0.1, int.MaxValue, ErrorMessage = "The price can't be
below 0.")]
    public float FreshenerPrice { get; set; }

    public int FreshenerRating { get; set; }

    [Required(ErrorMessage = "Freshener image is mandatory.")]
    public string FreshenerImage { get; set; }

    [Required(ErrorMessage = "False OR True to tell if the
freshener is on stock.")]
    public bool FreshenerAvailability { get; set; }

    public string FreshenerCode { get; set; }

    public int FreshenerStock { get; set; }
```

```

        public DateTime FreshenerModifyDate { get; set; }

        [Required(ErrorMessage = "The category is mandatory.")]
        public int CategoryId { get; set; }
        public virtual Category Category { get; set; }
        public IEnumerable<SelectListItem> CategoryList { get; set; }
    }

    public virtual ICollection<Review> Reviews { get; set; }

    public string UserId { get; set; }
    public virtual ApplicationUser User { get; set; }

    public virtual ICollection<ItemCart> ItemCarts { get; set; }
    public virtual ICollection<OrderComplete> OrderCompletes { get;
set; }
}

```

Modelul pentru categorii este format din id, nume, *slug*, adresa imaginii, descriere și o listă de produse, existând o relație *one-to-many* între categorie și produs, iar fiecare câmp descris are și validări.

În continuare este prezentat modelul *Contact*, construit pentru a stoca anumite formulare cu întrebări sau opinii adresate de către utilizatori. Fiecare formular este alcătuit dintr-un id incrementat automat care nu este vizibil în partea de *frontend*, numele utilizatorului, o adresă de email pe care dorește să fie contactat, aceasta putând fi diferită de cea folosită la înregistrarea în aplicație. Tot aici se mai poate completa și un număr de telefon, subiectul mesajului cât și conținutul acestuia, iar data completării formularului este preluată automat. Pentru a evita *spamul*, am ales să permit trimiterea formularului de contact doar utilizatorilor înregistrați, iar prin folosirea validărilor, toți itemii prezentați trebuie completați.

Pentru a strânge în baza de date *review-urile* unui produs, am dezvoltat un model ce reține comentariul, nota într-un interval de la unu la cinci și data adăugării pe *site*. Fiecare *review* este legat la un singur obiect de tip *Freshener* și un utilizator, iar un produs poate avea mai multe *review-uri*.

Funcționalitatea de plasare a comenzilor este inițiată în modelul *Cart*, unde fiecărui utilizator îi este atribuit câte un coș de cumpărături și o listă de produse. Această listă este ulterior definită în modelul *ItemCart*, fiind alcătuită din id, produs, cantitatea celui tip de produs și o referință către modelul *Order*. În tabela *Order* se găsesc datele pentru livrarea comenzii:

username-ul, numele, prenumele, orașul, județul, strada, codul poștal, numărul de telefon, emailul, totalul comenzii și statusul de comandă (dacă a fost sau nu trimisă). Pentru validarea numărului de telefon, cât și a emailului, am folosit *regexuri*^[3]. Mai jos este afișat *regexul* pentru numărul de telefon și explicația lui, sub formă de tabel.

```
[RegularExpression(@"^(\\+4|)?[0-9]{6,}", ErrorMessage = "This phone
number is not valid.")]
public string OrderPhone { get; set; }
```

Element din regex	Explicație
^	anunță <i>regexul</i> să pornească de la începutul liniei
(\\+4/)	potrivește combinația +4 la începutul unui număr de telefon
?	stabilește ca partea de +4 să apară cel mult o dată
[0-9]	ia un singur caracter din intervalul dat, adică un număr între zero și nouă
{6,}	repetă alegerea de caracter anterioară de cel puțin șase ori

Pentru a putea stoca permanent atât adresele de livrare, cât și produsele achiziționate, am implementat un nou model numit *OrderComplete* ce reține pentru fiecare *OrderId*, pe câte un rând separat în baza de date, id-ul produsului și cantitatea setată. Am atașat codul pentru acest model mai jos.

```
public class OrderComplete
{
    [Key]
    public int OrderCompleteId {get; set;}
    public int FreshenerId { get; set; }
    public int OrderId { get; set; }
    public int FreshenerQuantity { get; set; }

    public virtual Freshener Freshener { get; set; }
```

```

    public virtual Order Order { get; set; }
}

```

În *IdentityModels* este definit fiecare model creat, un vector de tip *IEnumerable* pentru rolurile prezente, cât și inițializarea migrațiilor bazei de date, fie locală, fie pe serverul Azure.

3.3.2 Controllerele create

Toate controllerele au implementate operațiile *CRUD*, adică funcționalitățile de creare, citire, actualizare și ștergere, iar unele dintre acestea dețin și alte metode utile. Fiecare are definită o conexiune către baza de date curentă.

În controllerul de odorizante este definită la început metoda *Index* ce cuprinde funcționalitatea de a căuta produse după nume sau descriere, de a le sorta crescător sau descrescător după preț, *rating* sau dată, toate acestea fiind în plus față de capacitatea de a prelua o listă cu toate produsele.^[6] Metoda primește ca parametri un string pentru sortare, unul pentru căutare și unul pentru căutarea curentă. Se verifică dacă nu există nicio căutare și în acest caz filtrul devine căutarea curentă. Apoi, pentru fiecare produs se apelează funcția *RatingChecker*, ce primește un articol ca și parametru și îi calculează *ratingul*, salvându-l ulterior în baza de date. Dacă există un string de căutare dat de utilizator, programul îl caută în numele sau descrierea produselor, iar dacă există un string de ordonare, programul utilizează un *switch* pentru a determina soluția. În continuare este metoda *Show* care primește un id ca și parametru, caută produsul ce are acel id și afișează toate detaliile despre el. Pentru cele trei metode rămase de *CRUD*, adică *New*, *Edit*, *Delete*, am filtrat accesul doar utilizatorilor cu rolul de admin. În *New* se adaugă produsul direct în baza de date, în *Edit* se caută produsul, și se editează, iar în *Delete* se caută produsul, se alcătuieste o listă cu toate *review-urile* aceluia, se șterg întâi toate *review-urile*, iar pe urmă produsul. Controllerul mai conține și metoda *GetCategories*, ce crează o listă goală, iterează prin fiecare categorie și o adaugă la lista aceea.

În controllerele pentru *review-uri* și pentru formularele de contact se află doar operațiile de bază *CRUD*, iar în *CategoriesController* am tratat cazuri speciale atunci când se șterge o categorie pentru a nu apărea *buguri*. La eliminarea unei categorii se definește o listă cu toate produsele din acea categorie. Pentru fiecare produs, se alcătuieste o altă listă cu toate *review-urile* acelui produs și se șterg pe rând, întâi toate *review-urile*, iar apoi produsele.

Controllerul *ItemCarts* conține pentru început metoda *Index*, ce caută utilizatorul curent, caută coșul de cumpărături asociat celui utilizator și preia o listă cu tot ce se află în acel coș. Dacă lista nu este goală se calculează totalul până în acel punct, cu precizarea că pentru comenzile sub 300 Lei, se adaugă o taxă de transport în valoare de 20 Lei. Metoda *New* este principală în funcționalitatea coșului de cumpărături și caută, în același mod ca la *Index*, utilizatorul și coșul. La adăugarea unui produs se verifică dacă acesta există deja în coș. În caz afirmativ, se crește cantitatea cu unu, iar în caz contrar se adaugă în coș. Funcțiile *IncrementItem* și *DecreaseItem* cresc și scad cantitatea unui produs, ștergându-l din coș dacă se ajunge la o cantitate egală cu zero. Metoda *Delete* caută produsul și îl șterge din coș.

OrdersController returnează o listă cu toate comenzile plasate vreodată și neșterse, prin metoda *Index*, listă accesibilă doar pentru utilizatorul cu rolul de admin. Codul pentru metoda *Index* este mai jos.

```
[Authorize(Roles = "Admin")]
public ActionResult Index()
{
    if (TempData.ContainsKey("message"))
    {
        ViewBag.message = TempData["message"].ToString();
    }

    var orders = from order in db.Orders
                 orderby order.OrderId
                 select order;

    ViewBag.Orders = orders.ToList();
    return View();
}
```

Pentru ca fiecare utilizator în parte să își poată vedea comenzile proprii, am definit metoda *UserOrders*, foarte asemănătoare cu cea de *Index*, doar că fac o legătură între id-ul utilizatorului curent și id-ul celui ce a plasat comanda. Pentru a înțelege mai bine această legătură am atașat liniile de cod diferite față de *Index*.

```
var currentUser = User.Identity.GetUserId();

var orders = from order in db.Orders
              where order.UserId == currentUser
              orderby order.OrderId
              select order;
```

În funcția *New* este apelată initial metoda *CheckEmptyCart*, ce verifică dacă există vreun produs în coș, afișând un mesaj pe ecran în caz negativ. Dacă există produse, se iau toate detaliile din coșul de cumpărături și se recalculează totalul, cu două zecimale. Tot aici se adaugă detaliile comenzii în baza de date și se apelează *AddOrderToItemCart* și *CompleteOrderDeleteItemsFromItemCart*. *AddOrderToItemCart* iterează prin fiecare produs din coș și îl adaugă într-o listă. Fiecare produs de acest tip este ulterior inserat în tabelul pentru comenzi efectuate pentru a rămâne permanent salvate. Cea de-a doua metodă șterge produsele din coșul utilizatorului. Tot în controllerul pentru comenzi mai sunt definite și operațiile de *delete*, *show* și *edit*, asemănătoare cu cele din controllerele precedente. Am adăugat mai jos codul pentru funcția *New*, *AddOrderToItemCart* și pentru *CompleteOrderDeleteItemsFromItemCart*.

```
[Authorize(Roles = "Admin, User")]
public ActionResult New()
{
    Order order = new Order();

    order.UserId = User.Identity.GetUserId();

    if (CheckEmptyCart() == false)
    {
        TempData["message"] = "Your cart is empty!";
        return Redirect("/ItemCarts/Index/");
    }

    return View(order);
}

[Authorize(Roles = "Admin, User")]
[HttpPost]
public ActionResult New(Order order)
{
    order.OrderModifyDate = DateTime.Now;
    order.UserId = User.Identity.GetUserId();

    var currentUser = User.Identity.GetUserId();
    Cart cart = db.Carts.Where(a => a.UserId ==
currentUser).First();

    var itemsInCart = from ic in db.ItemCarts where ic.CartId
== cart.CartId select ic;
```

```

        if (itemsInCart != null)
        {
            ViewBag.ItemCarts = itemsInCart;
            ViewBag.currentUser =
db.Users.Find(User.Identity.GetUserId());

            float totalSum = 0;

            foreach (var ic in cart.ItemCarts)
            {
                totalSum = totalSum + ic.ItemCartQuantity *
ic.Freshener.FreshenerPrice;
            }

            if (totalSum < 300)
            {
                totalSum += 20;
            }

            float TS = (float)Math.Round(totalSum * 100f) / 100f;

            order.OrderTotal = TS;
        }

        try
        {
            if (ModelState.IsValid)
            {
                db.Orders.Add(order);
                AddOrderIdToItemCart(order.OrderId);

                db.SaveChanges();

                TempData["message"] = "The order has been added!";

                ViewBag.OrderId = order.OrderId;

                var currentUserEmail = db.Users.FirstOrDefault(u =>
u.Id == currentUser);
                order.OrderUsername = currentUserEmail.Email;

                CompleteOrderDeleteItemsFromItemCart();

                return RedirectToAction("CompleteOrder", new { id =
order.OrderId });
            }
        }
    }
}

```

```

    }
    else
    {
        Console.WriteLine("Error on modelstate.isvalid for
adding a new order.");
        TempData["message"] = "Order not added!";
        return View(order);
    }
}
catch (Exception)
{
    Console.WriteLine("Error on try catch for adding a new
order.");
    TempData["message"] = "Order not added!";
    return View(order);
}
}

public void AddOrderIdToItemCart(int id)
{
    var currentUser = User.Identity.GetUserId();
    var cart = db.Carts.Where(c => c.UserId ==
currentUser).FirstOrDefault();
    var items = db.ItemCarts.Where(i => i.CartId ==
cart.CartId);

    foreach (var item in items)
    {
        item.OrderId = id;
    }

    var completedItems = items.ToList().Select(i => new
OrderComplete
    {
        FreshenerId = i.FreshenerId,
        OrderId = id,
        FreshenerQuantity = i.ItemCartQuantity
    });

    foreach (var completedItem in completedItems)
    {
        db.OrderCompletes.Add(completedItem);
    }
}

public void CompleteOrderDeleteItemsFromItemCart()

```



```

{
    var currentUser = User.Identity.GetUserId();
    var cart = db.Carts.Where(c => c.UserId ==
currentUser).FirstOrDefault();
    var items = db.ItemCarts.Where(i => i.CartId ==
cart.CartId);

    foreach (var item in items)
    {
        db.ItemCarts.Remove(item);
    }

    db.SaveChanges();
}

```

Metoda *UserShowOrder* afișează detaliile unei comenzi pentru un anumit utilizator, iar *CompleteOrder* ilustrează detaliile unei comenzi după ce este efectuată.

Pentru *UserController* am implementat operațiile de bază *CRUD*, cu tratarea cazurilor speciale pentru funcția de *Delete*. Atunci când adminul șterge un utilizator din baza de date, inițial se șterg toate *review-urile* create de acel utilizator, se șterge coșul de cumpărături, toate comenzile efectuate și formularele de contact, evitând în acest fel orice eroare ce putea apărea. Mai jos am adăugat codul pentru metoda descrisă.

```

[HttpDelete]
public ActionResult Delete(string id)
{
    ApplicationDbContext context = new
ApplicationDbContext();
    var UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(context));
    var user = UserManager.Users.FirstOrDefault(u => u.Id
== id);

    var reviews = db.Reviews.Where(rev => rev.UserId ==
id);

    foreach (var review in reviews)
    {
        db.Reviews.Remove(review);
    }

    var carts = db.Carts.Where(crt => crt.UserId == id);
    foreach (var cart in carts)
    {
        db.Carts.Remove(cart);
    }
}

```

```

    }

    var orders = db.Orders.Where(ord => ord.UserId == id);
    foreach (var order in orders)
    {
        db.Orders.Remove(order);
    }

    var contactforms = db.Contacts.Where(cnt => cnt.UserId
== id);
    foreach (var contact in contactforms)
    {
        db.Contacts.Remove(contact);
    }

    db.SaveChanges();
    UserManager.Delete(user);
    return RedirectToAction("Index");
}

```

În HomeController este definită o listă de cele mai vândute trei produse, detaliu important pentru fiecare magazine online. Se accesează tabelul OrderComplete, datele fiind grupate după id-ul fiecărui articol, se ordonează descrescător după numărul de apariții și se selectează primele trei.

3.3.3 View-urile create

Fiecare model are automat un folder cu numele său. În folder se află fișiere de tip HTML cu numele fiecărei metode din controllerul aferent ce trimite un răspuns către partea de view. Este folosit Bootstrap în toate fișierele, fiecare pagină fiind *responsive*.

Pachetul de view pentru produse conține patru fișiere: *Edit*, *Index*, *New*, *Show*. În *Edit*, cât și în *Show* se află un formular pentru a completa datele fiecărui produs. Este folosită tehnologia Razor pentru a prelua datele din modelul *Freshener*. Pentru câmpurile obligatorii din cele două formulare am folosit un mesaj de validare cu simbolul “*”. Mai jos este poza cu formularul menționat.

Add Freshener

Freshener Name *

Freshener Slug

Freshener Description *

Freshener Price *

Freshener Image *

Freshener Availability * ☒

Freshener Code

Freshener Stock

Select category: *

[+ Add Freshener](#)

[HOME](#) [ABOUT US](#) [CONTACT](#)

© 2022 - SDM Office Group SRL

Figura 3.3 – Formular creare produs nou

View-ul pentru metoda *Index* conține un meniu *dropdown* cu cele șase opțiuni de sortare, o bară de căutare, un buton pentru a începe căutarea și carduri de tip Bootstrap, cu informațiile cele mai relevante pentru fiecare produs: poză, titlu, *rating*, preț, buton de adăugat în coș, de produs indisponibil sau stoc epuizat, după caz. Dacă utilizatorul ce accesează lista de produse este admin, acesta vede în *footerul* fiecărui card, data adăugării produsului. Poza următoare prezintă aspectul paginii descrise anterior.

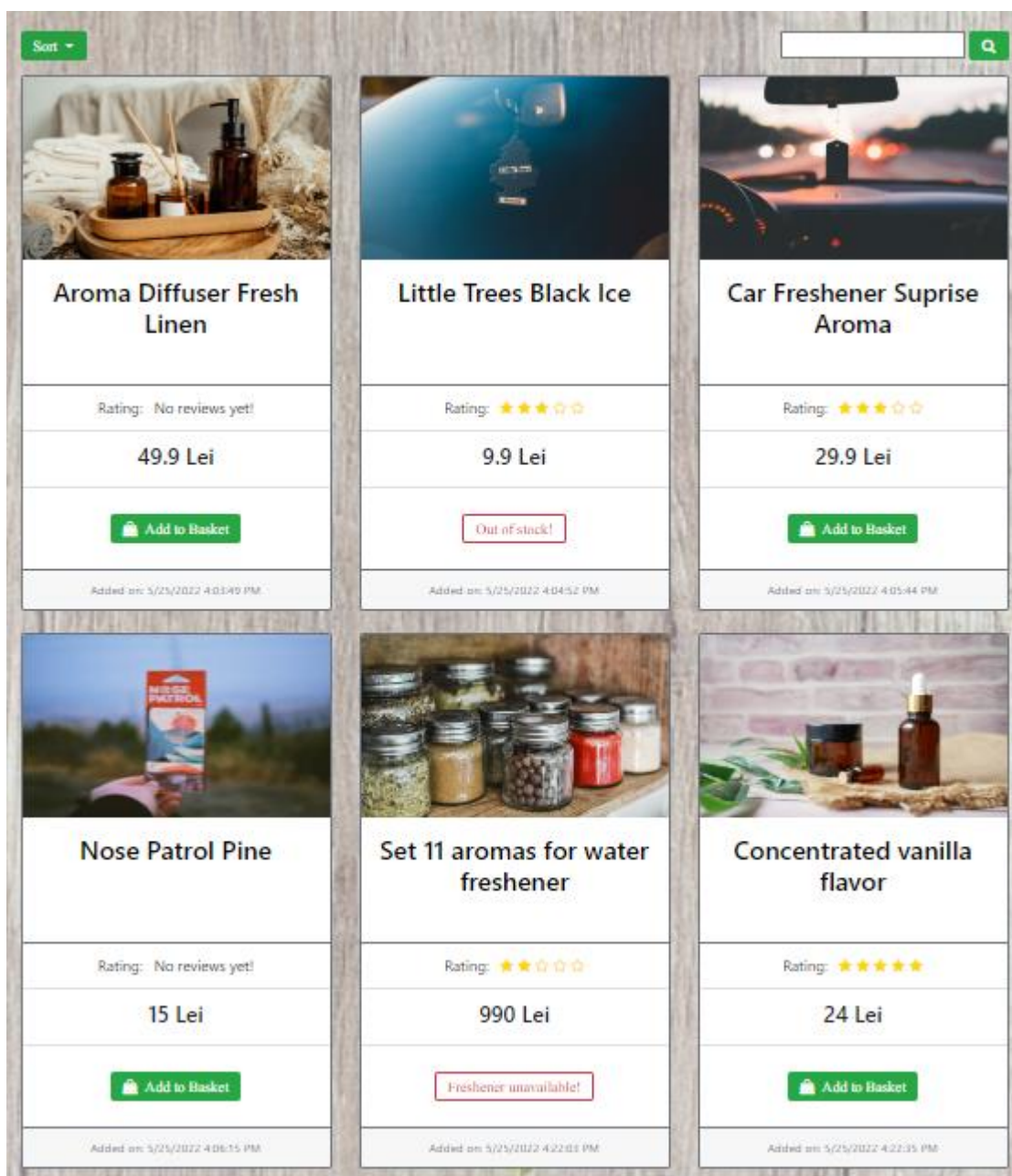


Figura 3.4 – Lista de produse din perspectiva adminului

Partea de *Show* pune în lumină un anumit produs și folosește tot carduri Bootstrap pentru a păstra aspectul plăcut al paginii, dar și partea de *responsive*. Aici sunt arătate toate datele unui produs, într-o manieră ce evidențiază lucrurile importante, precum titlul sau prețul. Pentru utilizatorul admin apar două butoane noi, de editare și ștergere. În această pagină se găsește și formularul de trimitere al unui *review*, împreună cu lista tuturor *review-urilor* trimise anterior, precum se poate vedea în poza următoare.

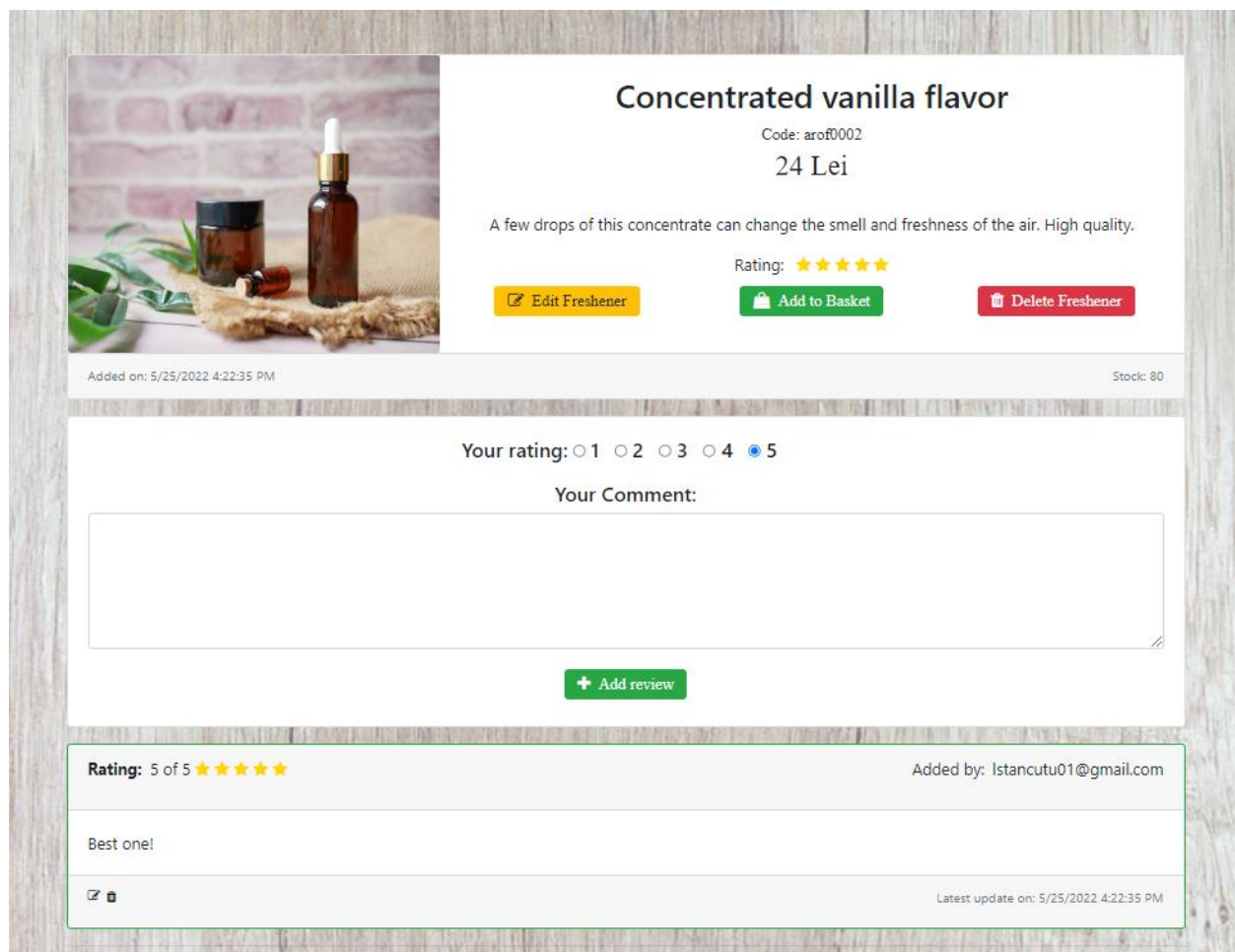


Figura 3.5 – Pagina unui produs din perspectiva adminului

În view-ul categoriilor se găsesc aceleași nume de fișiere, iar cele de *Edit* și *New* au același scop ca la produse, dar pentru câmpurile unei categorii. În *Index* există un buton pentru redirectionare la lista completă de produse, dar și carduri Bootstrap, inițial câte patru pe rând pentru ecranele mari, dar cu schimbarea acestui număr în funcție de rezoluția *display-ului*. Cardurile conțin câte o poză definitorie pentru fiecare categorie, nume și descriere. Dacă utilizatorul curent are rolul de admin, mai apare un buton pentru adăugarea de categorii noi și încă unul pentru editarea unei categorii. Am adăugat o poză pe pagina următoare pentru a ilustra *designul* discutat.

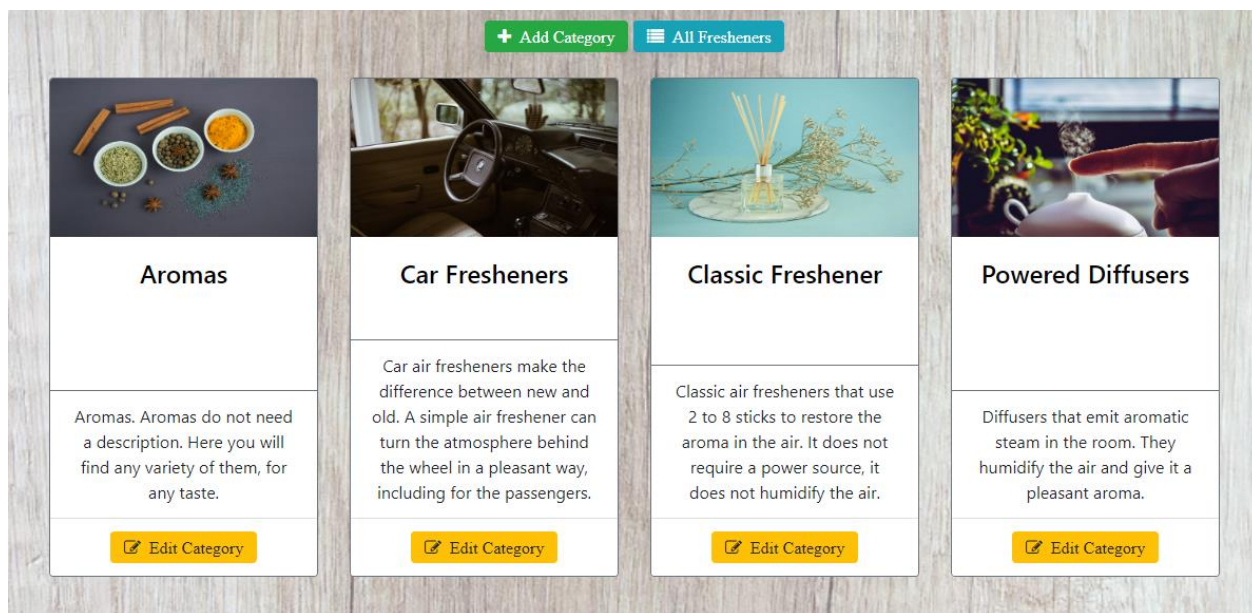


Figura 3.6 – Lista de categorii din perspectiva adminului

Show-ul are un design asemănător cu pagina de *Index* a produselor. Aici sunt ilustrate articolele ce aparțin de o anumită categorie, iar dacă pagina este accesată de către un cont de administrator, devine vizibil și buton de ștergere a categoriei.

În ASP.NET MVC există *Partial* view ce reprezintă un fișier în care se pun anumite date și urmează să fie introduse în oricare alt view mult mai ușor. Aplicația dezvoltată dispune de un *Layout* ce este aplicat pe toate paginile. Acesta conține bara de navigare din antetul paginii, cu diferite butoane ce apar în funcție de rolul utilizatorului. Dacă rezoluția paginii este până într-un anumit interval, butoanele sunt mutate într-un meniu *dropdown*. În continuare este încărcat conținutul unei pagini, în funcție de ruta accesată, iar în final apare bara de *footer*, care are trei butoane importante ce redirecționează utilizator spre *Home*, *About* sau *Contact*.

Modelul pentru formularele de contact conține în view-ul său trei fișiere importante. Unul dintre acestea este *New*, iar în el se găsesc date referitoare la orarul firmei, o imagine dinamică de *Google Maps* cu locația actuală a sediului firmei și formularul de contact ce poate fi completat și trimis. Poza următoare reprezintă pagina *Contact*.

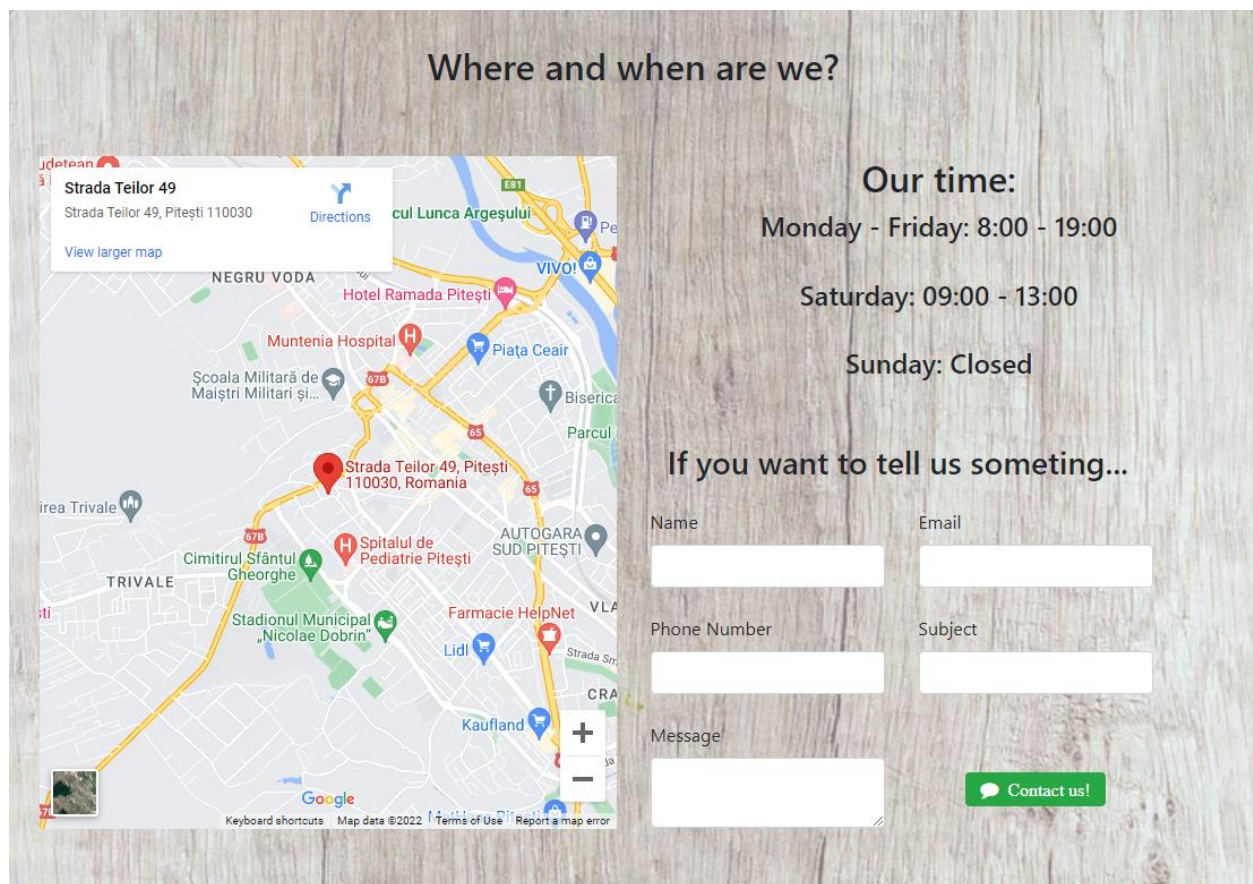


Figura 3.7 – Pagina de contact

Celelalte două fișiere sunt *Index* și *Show*. Primul dintre acestea conține lista tuturor formularelor trimise și neșterse, aranjate sub formă de carduri Bootstrap, informațiile afișate fiind *username-ul* celui ce a trimis formularul, subiectul, numărul de telefon și data trimiterii. Sunt disponibile două butoane, unul pentru a redirecționa utilizatorul pe o pagină ce oferă mai multe detalii pentru un anumit formular și unul pentru a-l șterge. Atunci când este accesată pagina de detalii se încarcă datele din *Show*, iar ca titlu este pus id-ul formularului. În continuare apar numele utilizatorului, emailul, numărul de telefon, subiectul, mesajul, un buton pentru a șterge formularul, iar în *footer* sunt scrise data trimiterii și id-ul utilizatorului. Am atașat poza primului formular adăugat pe baza de date Azure.

Contact form no. 1

Name: Larisa Stancutu

Email: lstancutu01@gmail.com

Phone number: 0723181339

Subject: Restock

Message: Hello! Please restock the Little Trees Black Ice product. Thank you!

Delete contact form

Date created: 5/25/2022 4:42:47 PM

Added by: 59077b17-cf81-4789-b728-fc54280c30a7

Figura 3.8 – Pagina de detalii a unui formular de contact

Secțiunea de view pentru modelul *Review* dispune doar de un fișier special pentru *Edit*, părțile de *Show* și *New* fiind incluse în pagina de detalii a unui produs. În momentul în care un utilizator dorește să își editeze un *review*, el este redirecționat pe o pagină care conține un card Bootstrap cu posibilitatea de a modifica *ratingul* dat, dar și comentariul. Cardul discutat este ilustrat mai jos.

Edit Review:

Your rating: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☒ 5

Review Comment:

Edit review

Figura 3.9 – Pagina de editare a unui *review*

Coșul de cumpărături poate fi găsit ca și view, sub denumirea *ItemCart*. Acesta conține un buton special pentru a ajuta utilizatorul să se întoarcă la lista de produse. Pe o porțiune de 25% din lungimea ecranului se află un chenar cu sumarul de plată, iar pe restul

de 75% se află produsele, cantitatea selectată pentru fiecare și prețul. În prima porțiune menționată avem totalul produselor, taxa de transport dacă este cazul și totalul însumat. Tot aici mai poate fi găsit și un mic text, cu explicația taxei de transport. În cea de-a doua parte, se mai află și un buton de incrementat cantitatea unui produs, unul pentru a o scădea și încă unul pentru a șterge definitiv produsul din coșul de cumpărături. Folosirea componentelor de Bootstrap asigură partea de *responsive*, fiind una dintre cele mai importante pagini a aplicației. Am atașat o poză cu aspectul paginii, atunci când se află produse în coș.

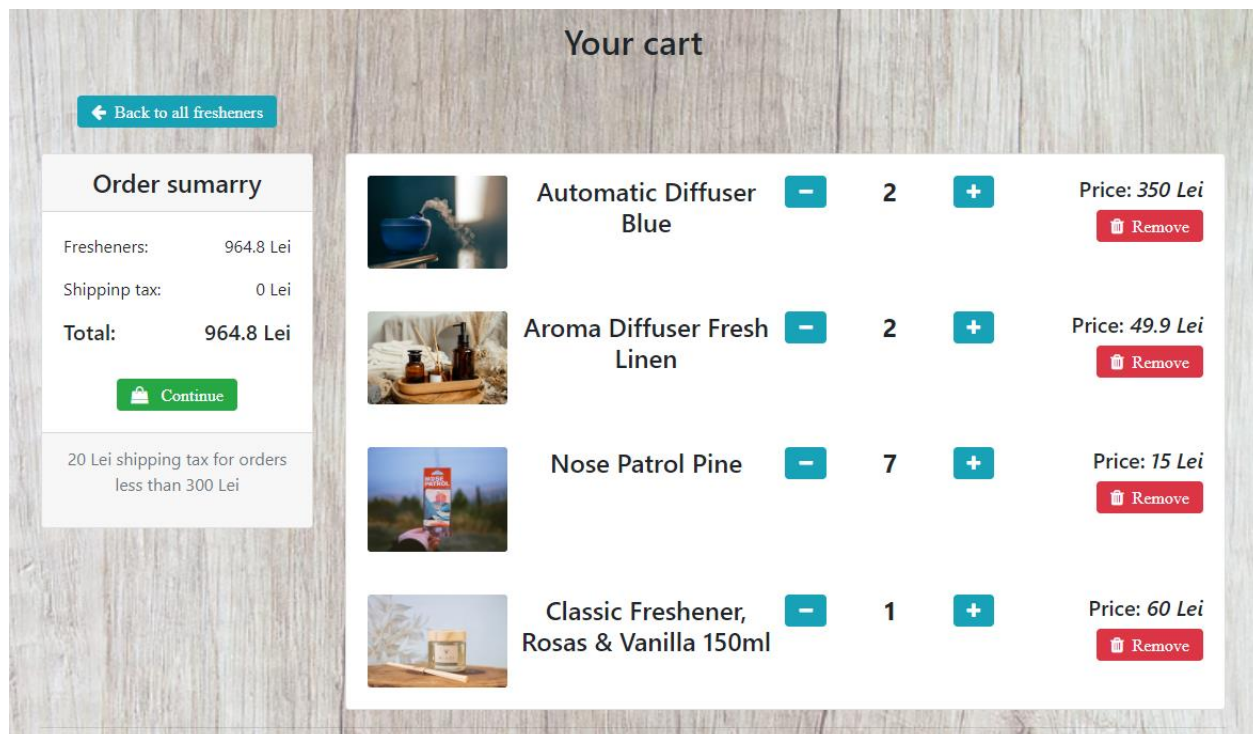


Figura 3.10 – Coșul de cumpărături

După ce utilizatorul apasă pe buton de *Continue* din pagina coșului de cumpărături, el este redirecționat către o nouă pagină unde trebuie să introducă datele adresei de livrare, precum: numele, prenumele, orașul, județul, strada, codul poștal, numărul de telefon și emailul. Când utilizatorul a completat formularul, acesta este din nou redirecționat către o nouă pagină, unde este anunțat că datele au fost trimise cu succes, îi este comunicat id-ul comenzii și numărul aproximativ de zile până va primi comanda. Imaginea următoare reprezintă ultima pagină descrisă.

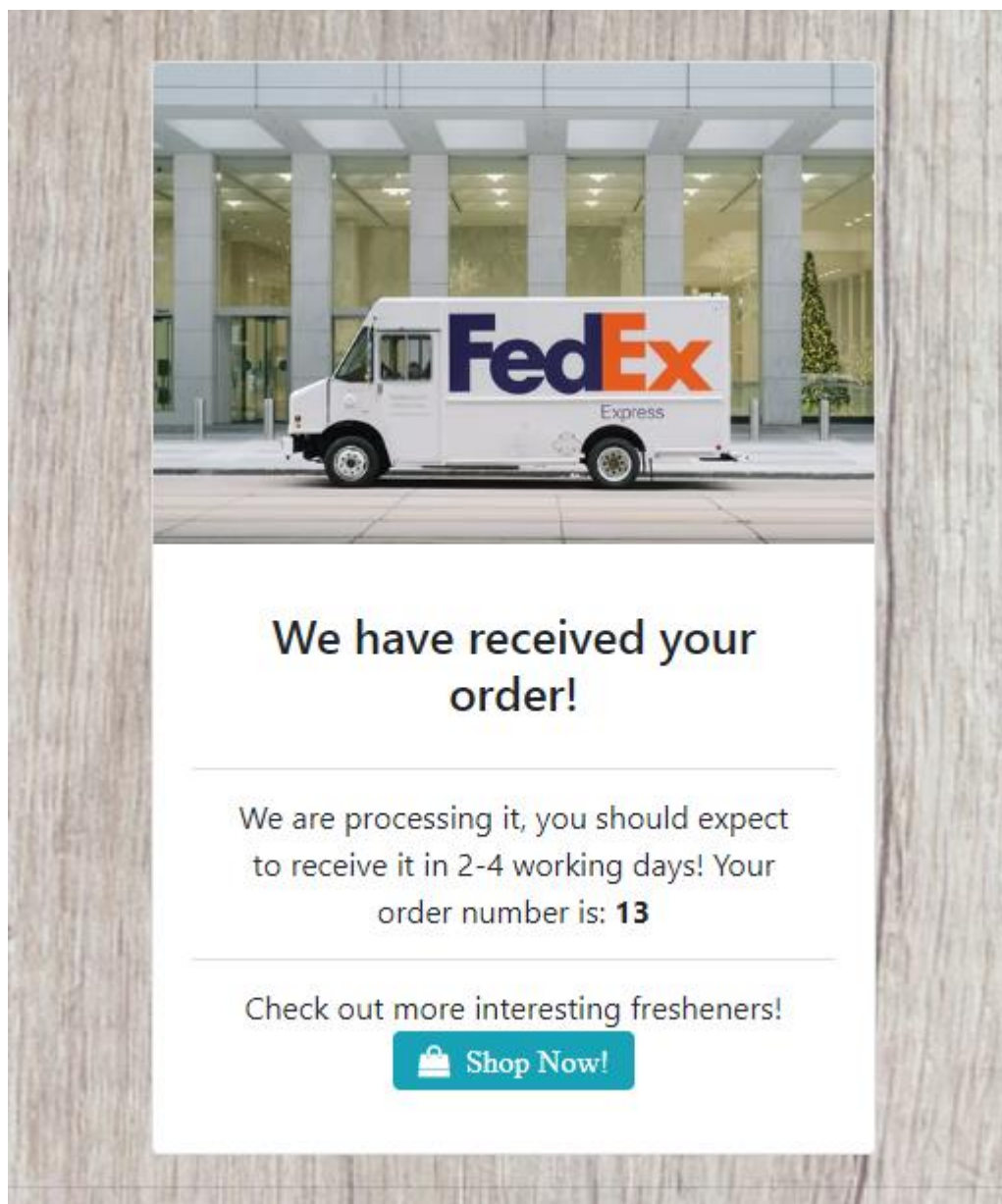


Figura 3.11 – Pagina de mulțumiri după trimiterea unei comenzi

În meniul special al adminului poate fi găsită lista tuturor comenzilor efectuate și neșterse, mai exact ca *Index* în view-ul pentru modelul *Order*. Fiecare comanda oferă în această pagină detalii precum: id-ul comenzii, statusul comenzii (dacă a fost sau nu trimisă), numele utilizatorului, totalul de plată, data, numărul de telefon, emailul și id-ul utilizatorului. Sunt disponibile și două butoane, unul cu scopul de a trimite adminul către o pagină unde poate vedea toate detaliile unei comenzi, inclusiv produsele comandate, iar altul cu scopul de a șterge comanda. Imaginea de mai jos reprezintă aspectele discutate anterior.

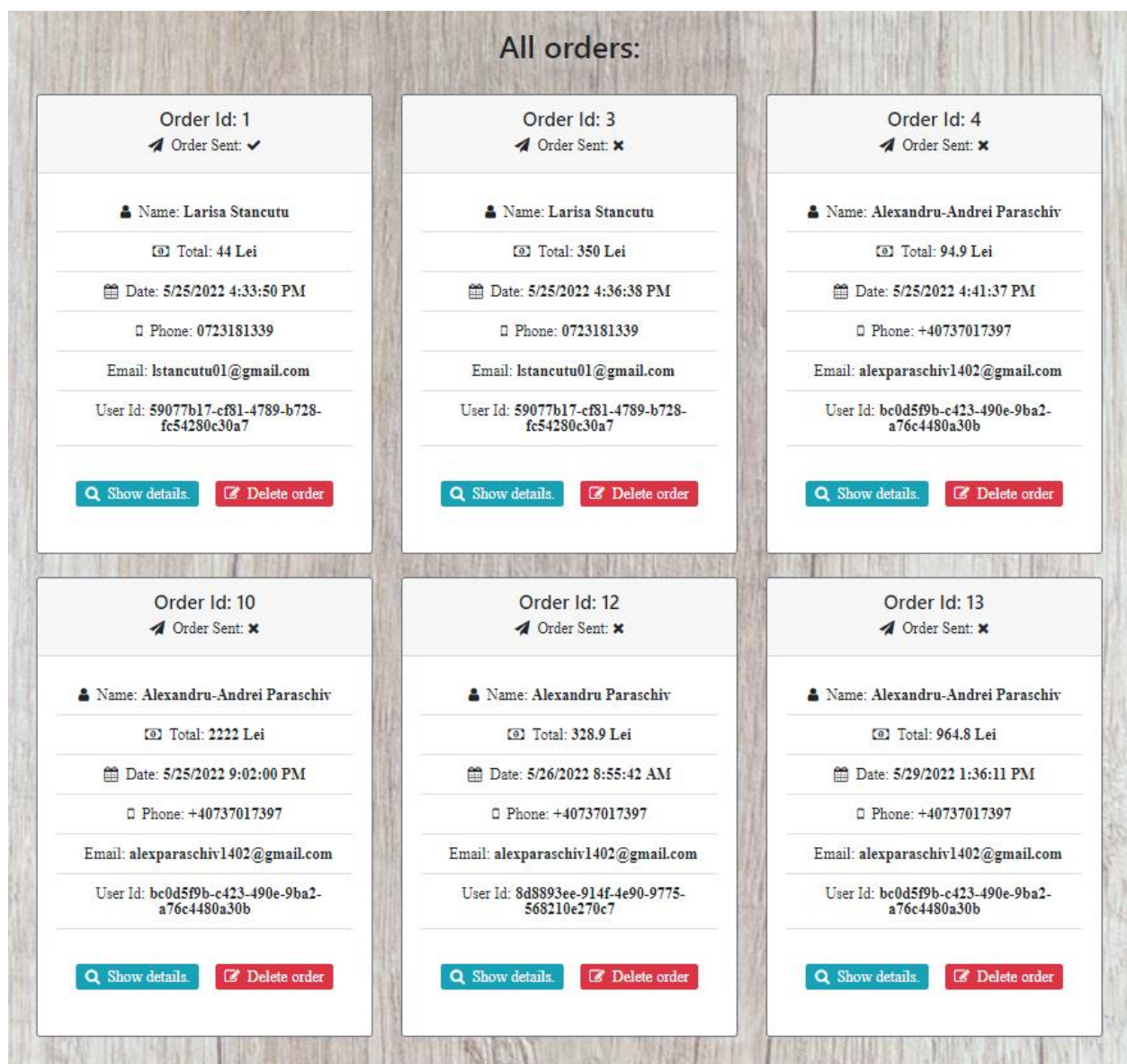


Figura 3.12 – Lista tuturor comenzilor

După accesarea paginii de detalii a unei comenzi, adminul poate vedea în plus față de pagina de *Index* toate datele de livrare, fiecare produs comandat alături de numărul de bucăți și totalul de plată explicat. Tot aici poate apărea un buton pentru editarea comenzii dacă aceasta nu a fost deja trimisă, în caz contrar apărând un text potrivit. Pentru paginile de comenzi proprii, *designul* este unul asemănător, acesta putând fi văzut în figura 3.1. În figura 3.13 se află pagina cu detaliile explicite ale unei comenzi, pentru situația în care acea comandă a fost deja trimisă.

Order no. 1

Total: 44 Lei

Date: 5/25/2022 4:33:50 PM

Order Sent: ✓

Name: Larisa Stancutu

Phone: 0723181339

Email: lstancutu01@gmail.com


City: Bucuresti

State: Bucuresti

Street name: Tineretului

Postal Code:

Userid: 59077b17-cf81-4789-b728-fc54280c30a7



Concentrated vanilla flavor

24 Lei
1 piece(s)

Total fresheners:24 Lei

Shipping Tax:20 Lei

Total:44 Lei

You can not edit this order, it has already been processed and sent. Please delete it and make another one.

Delete order

Figura 3.13 – Pagina cu detaliile unei comenzi

Adminul dispune în meniul său și de o listă a tuturor utilizatorilor, fiind un aspect ce aparține de view-ul *Index*, modelul *User*. Lista conține, pentru fiecare utilizator în parte, date precum numele de utilizator, emailul și numărul de telefon. Dacă adminul accesează pagina specială cu detalii a unui utilizator, acesta poate vedea în plus id-ul și rolul. *Indexul* prezentat este arătat în pagina următoare.

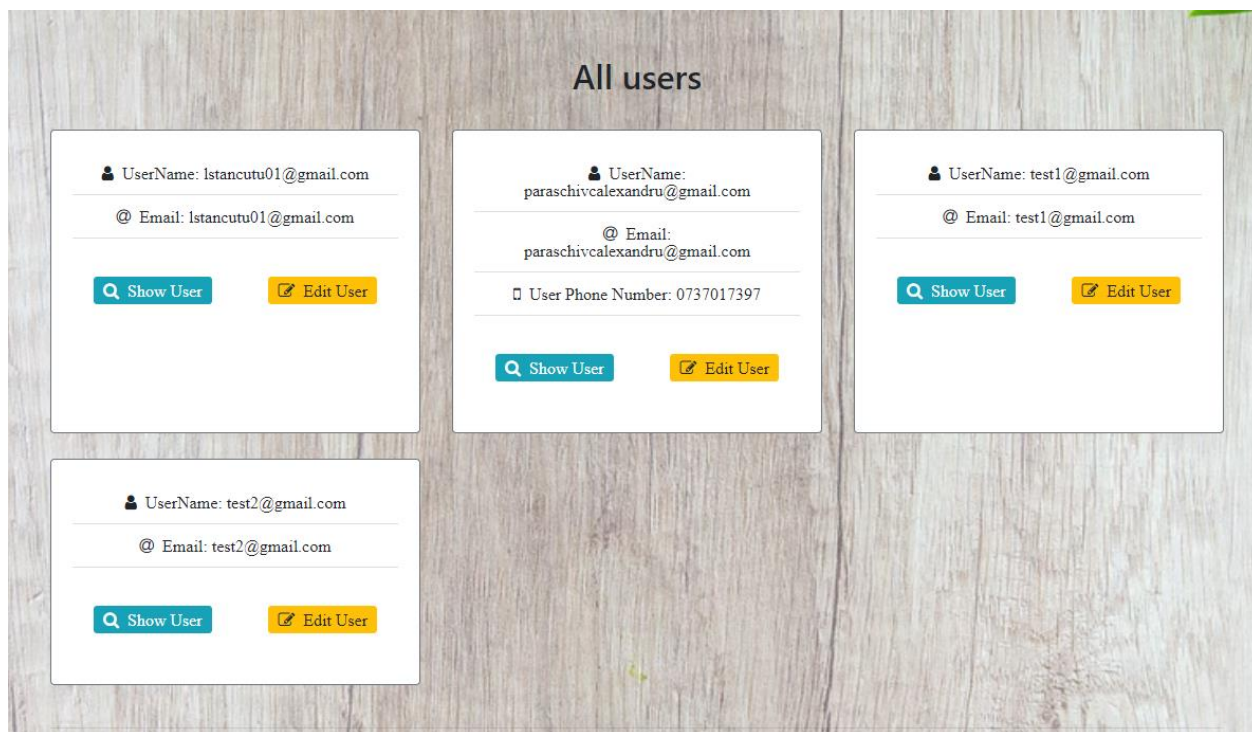


Figura 3.14 – Pagina cu utilizatorii aplicației

În final, în *folderul Home*, apar view-urile reprezentative pentru pagina de *Index* și *About*. Prima dintre cele două este cea care apare în cele mai multe cazuri atunci când un utilizator deschide *site-ul* pentru prima dată. În prima parte a paginii sunt prezente două paragrafe de text reprezentativ pentru firmă și două imagini. După aceea este ilustrată o secțiune cu cele mai populare trei produse din aplicație, produse reprezentate în mod dinamic. În final se află trei bucăți de text, aranjate sub formă de *carousel* specific Bootstrap. A doua pagină are ca scop informarea utilizatorului cu detaliile importante despre firmă și cu două opțiuni în caz că acesta dorește să se alăture firmei. Pagina de *Index* din *folderul Home* este arătată mai jos.

SDM OFFICE GROUP

We care about you

During more than 16 years we have been delivering the best air fresheners that could be found on Earth. We are working with some of the biggest providers from Spain and Italy, and our networks are expanding in Japan.



The best freshener

Our store has a wide range of air fresheners. There can be found from car-fresheners, to room-fresheners or even automatic-fresheners. The aromas we have are unique, unprecedented so far and we are still growing.

Most popular fresheners



Nose Patrol Pine

15 Lei



Concentrated vanilla
flavor

24 Lei



Automatic Diffuser Blue

350 Lei

"5 out of 5. Great website with great products!"

Ed Sam, Entrepreneur, Constanta



HOME

ABOUT US

CONTACT

© 2022 - SDM Office Group SRL

Figura 3.15 – Pagina de *Home/Index*

3.4 Design

Simplitatea *designului* într-o aplicație web este esențială pentru a avea o conexiune cât mai bună cu utilizatorul. Din punctul meu de vedere, un *site* ce are un scop bine definit, dar fiecare pagină având informații text peste tot posibil nu este la fel de atractiv ca și unul simplist, care arată profesional. Am încercat prin folosirea elementelor de Bootstrap, să păstrez peste tot un aspect plăcut, la obiect și bine structurat.

Culorile alese pentru diferite situații, precum butoanele sau textele de validare, descriu în avans tipul de mesaj ce se vrea a fi transmis. Spre exemplu, culoarea roșie semnifică eroarea sau faptul că urmează să se întâmple o acțiune ce poate schimba definitiv un obiect. Verde simbolizează succesul, iar culoarea galbenă dorește să sporească atenția utilizatorului prin faptul că urmează ceva important. Albastrul pune în evidență informațiile. Mai jos este expus modul de folosire a primelor trei culori menționate, roșul pentru ștergerea unui produs, verdele pentru adăugarea acestuia în coșul de cumpărături și galbenul pentru butonul de editare.

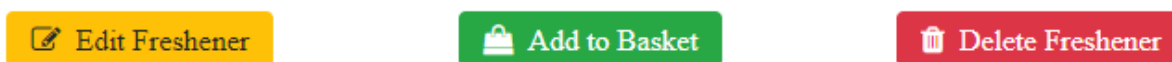


Figura 3.16 – Folosirea culorilor importante

Un alt aspect important pentru *designul* unei aplicații este păstrarea aceleiași teme pe toate paginile. Nu ar trebui ca pe o pagină să predomină cu un set de culori și un font, iar pe o alta să fie diferite.

Pentru dezvoltarea aplicației am ales să folosesc text real și poze reale. Nu am optat pentru text generat automat pentru a crea o credibilitate în rândul utilizatorilor, dar și pentru a vedea cum se structurează informația.

Pentru a stabili o conexiune cât mai bună între utilizatorul obișnuit și *site*, am rugat câteva persoane să acceseze varianta *hostată* a aplicației și să folosească funcționalitățile disponibile. Astfel, am aflat care este punctul de vedere al lor și am aplicat schimbările necesare.

Atunci când un utilizator completează un formular și anumite câmpuri sunt incomplete sau greșite, aplicația oferă o listă de sugestii și mesaje informative pentru a-l ajuta.

Bara de navigare face ca experiența utilizatorilor să fie una completă, oferindu-le accesul la resursele dorite mult mai rapid, alături de abilitatea de a căuta produsele și de a le sorta după preferințe.

Capitolul 4

Concluzii

4.1 Provocări

Pe parcursul dezvoltării aplicației m-am confruntat cu diferite provocări atât de logică, cât și de sintaxă.

Pentru început, alegerea datelor din modele a trebuit discutată în amănunt cu firma, întrucât să se plieze pe nevoile lor, dar să fie relevante și pentru lucrarea mea.

După ce am modelat produsul, categoria și *review-ul*, logica tabelelor pentru funcționalitatea de plasare a comenzilor a fost un aspect analizat pe mai multe variante de rezolvare. Inițial a fost implementat un coș de cumpărături ce permitea adăugarea produselor în el, indiferent dacă utilizator era autentificat pe platformă sau nu, produsele fiind introduse într-un tabel auxiliar. Problema majoră la această abordare avea în vedere faptul că atunci când orice utilizator se autentifica pe *site* și existau date în tabelă, acelea erau transferate în coșul de cumpărături ale persoanei, practic nefiind adăugate de către el. Rezolvarea pentru problema prezentată a constat în crearea unui coș de cumpărături unic, atunci când un utilizator se înregistrează în aplicație.

Ulterior am definit tabela *Order*, aceasta stocând datele de livrare și totalul comenzii. Am dorit să adaug funcționalitatea de a vedea comenzile efectuate cu fiecare produs cumpărat, cantitatea, totalul, dar și adresa. Datele pentru partea din urmă au putut fi luate ușor, însă provocarea a venit în a vedea produsele comandate. Pentru a păstra același coș de cumpărături al unui utilizator este nevoie să se șteargă produsele din el după plasarea comenzii. Această logică nu îmi permitea în mod direct să salvez lista de obiecte comandate pentru a o vedea ulterior. Ideea de rezolvare a fost în a crea un model nou, *OrderComplete*, ce avea aproape aceeași structură ca și *ItemCart*. Astfel, după plasarea unei comenzi, produsele sunt mutate în tabela

definită cu acest scop, alături de cantitatea lor, iar legătura către datele de livrare se face printr-o cheie externă numită *OrderId*.

Frontendul a fost construit de la zero, nefiind familiarizat cu elementele de Bootstrap, exceptând butoanele. Partea ușor dificilă a reprezentat alegerea versiunii acestui *framework*. Visual Studio mi-a preinstalat versiunea 3 ce nu avea aproape niciun element încă valabil cu cele din documentația oficială. Am instalat versiunea 5, însă și aceasta s-a dovedit a avea probleme de compatibilitate, iar după instalarea ultimei versiuni de Bootstrap 4, totul a mers cum trebuia.

Pentru a arăta că stăpânesc fiecare etapă de dezvoltare a unei aplicații web, dar și pentru că a fost o cerință a clientului, am *hostat* aplicația prin Microsoft Azure, procedeu ce nu s-a dovedit a fi foarte simplu. Proiectul a fost creat inițial în Visual Studio 2017, iar funcționalitatea de *Publish* a IDE-ului nu mai era compatibilă cu standardele curente. A fost necesară actualizarea la Visual Studio 2022. După setarea contului de Microsoft Azure cu credențialele contului instituțional, am primit pentru 30 de zile suma de 200 USD, sold ce urma a fi folosit pentru *hosting*. Partea grea a venit în momentul setării bazei de date și a identității, neștiind inițial pașii ce trebuiau a fi urmați. Pentru a face *site-ul* online funcțional, a trebuit să fie creată o bază de date Azure SQL, adăugat *connectionStringul* în fișierul *Web.config*, schimbat contextul bazei de date și modificarea fișierelor de migrații. Această modificare este necesară pentru că la schimbarea de pe o bază de date locală pe una *hostată*, aplicația rula migrațiile de fiecare dată considerând că nu au fost aplicate, astfel apărând diferite erori de creare a tabelelor. Prin rularea programului pe baza de date Azure, aplicația folosea fișiere noi de JavaScript și salvarea cu metoda *SaveChanges* nu avea întotdeauna succes. Au fost făcute modificări pentru a face totul funcțional.

O ultimă provocare a constat în găsirea tuturor situațiilor de utilizare excepționale, ce puteau duce la erori în baza de date, precum ștergerea unei categorii, fără a șterge inițial produsele din ea. Au fost implementate soluții și consider că am acoperit toate aceste situații.

4.2 Posibile dezvoltări ulterioare

În aplicație există elemente ce pot fi îmbunătățite sau chiar adăugate. Un prim aspect are în vedere funcționalitatea de a avea un coș de cumpărături pentru utilizatorul de tip neînregistrat, care să și rămână dacă se autentifică sau înregistrează pe platformă.

Mai poate fi implementat sistemul de plată online cu cardul bancar sau alte servicii de plăți online, precum PayPal. Această funcționalitate se regăsește din ce în ce mai des pe *site-urile* de acest tip.

Un alt pas de dezvoltare implică stocarea imaginilor într-o bază de date. Momentan, imaginile sunt obținute prin printr-o adresă către o sursă accesibilă oricui.

Funcționalitățile de a adăuga un produs ca fiind favorit și de a le compara ar prezenta un plus al aplicației, oferindu-i utilizatorului o experiență mult mai plăcută în alegerea produsului potrivit.

Momentan *site-ul* este în engleză, dar posibilitatea de a alege limba conținutului se încadrează ca și dezvoltare ulterioară.

Pentru a mări capacitatea de vânzare, se poate adăuga încă un rol de utilizare, ce ar permite furnizorilor să adauge produsele lor în aplicație, cu acceptul adminului.

În ultimul rând, aplicația este momentan *hostată* pe un server gratis, având doar baza de date online contra cost. Consider că mutarea platformei pe un server mai performant devine o necesitate atunci când se atinge un număr destul de mare de utilizatori în același timp pe *site*.

4.3 Concluzie

Prin dezvoltarea aplicației am pus în practică elementele parcurse în timpul celor trei ani de studii, am învățat noțiuni noi și consider că am putut ajuta o firmă în mutarea online a activității sale.

Din punctul meu de vedere, *site-ul* este unul modern ce își poate face loc printre competitorii săi de pe piață.

Capitolul 5

Bibliografie

- [1] Benegui Cezara, *Curs Dezvoltarea aplicațiilor Web*, Facultatea de Matematică și Informatică, domeniul Informatică, anul II, semestrul I, 2020-2021.
- [2] *Entity Framework 6*, <https://docs.microsoft.com/en-us/ef/ef6/>, accesat pe 15 mai 2022.
- [3] Jon Galloway, *Part 9: Registration and Checkout*, <https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-9>, accesat pe 24 mai 2022.
- [4] Sipoș Andrei-Valentin, *Curs Dezvoltarea aplicațiilor Web*, <https://cs.unibuc.ro/~asipos/daw/curs-daw-print.pdf>, accesat pe 15 mai 2022.
- [5] Steve Smith, *Overview of ASP.NET Core MVC*, https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0, accesat pe 15 mai 2022.
- [6] *Tutorial: Add sorting, filtering, and paging with the Entity Framework in an ASP.NET MVC application*, <https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/sorting-filtering-and-paging-with-the-entity-framework-in-an-asp-net-mvc-application>, accesat pe 28 mai 2022.
- [7] *What is Azure SQL Database?*, <https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql>, accesat pe 28 mai 2022.

Lista figurilor

Capitolul 1	Nu există figuri în acest capitol.
Capitolul 2	Figura 2.1 – Arhitectura MVC Figura 2.2 – Versiune Entity Framework Figura 2.3 – Versiune Bootstrap Figura 2.4 – Diagrama aplicației în Lucidchart Figura 2.5 – Panoul de control Azure pentru hosting web
Capitolul 3	Figura 3.1 – Două comenzi plasate de un utilizator înregistrat Figura 3.2 – Meniu special pentru admin Figura 3.3 – Formular creare produs nou Figura 3.4 – Lista de produse din perspectiva adminului Figura 3.5 – Pagina unui produs din perspectiva adminului Figura 3.6 – Lista de categorii din perspectiva adminului Figura 3.7 – Pagina de contact Figura 3.8 – Pagina de detalii a unui formular de contact Figura 3.9 – Pagina de editare a unui <i>review</i> Figura 3.10 – Coșul de cumpărături Figura 3.11 – Pagina de mulțumiri după trimiterea unei comenzi Figura 3.12 – Lista tuturor comenzilor Figura 3.13 – Pagina cu detaliile unei comenzi Figura 3.14 – Pagina cu utilizatorii aplicației Figura 3.15 – Pagina de <i>Home/Index</i> Figura 3.16 – Folosirea culorilor importante
Capitolul 4	Nu există figuri în acest capitol.
Capitolul 5	Nu există figuri în acest capitol.