### 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

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vitae eros sit amet sem ornare varius. Duis eget felis eget risus posuere luctus. Integer odio metus, eleifend at nunc vitae, rutrum fermentum leo. Quisque rutrum vitae risus nec porta. Nunc eu orci euismod, ornare risus at, accumsan augue. Ut tincidunt pharetra convallis. Maecenas ut pretium ex. Morbi tellus dui, viverra quis augue at, tincidunt hendrerit orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis sollicitudin nunc. Sed sollicitudin purus dapibus mi fringilla, nec tincidunt nunc eleifend. Nam ut molestie erat. Integer eros dolor, viverra quis massa at, auctor.

#### Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vitae eros sit amet sem ornare varius. Duis eget felis eget risus posuere luctus. Integer odio metus, eleifend at nunc vitae, rutrum fermentum leo. Quisque rutrum vitae risus nec porta. Nunc eu orci euismod, ornare risus at, accumsan augue. Ut tincidunt pharetra convallis. Maecenas ut pretium ex. Morbi tellus dui, viverra quis augue at, tincidunt hendrerit orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis sollicitudin nunc. Sed sollicitudin purus dapibus mi fringilla, nec tincidunt nunc eleifend. Nam ut molestie erat. Integer eros dolor, viverra quis massa at, auctor

# Cuprins

1. [Introducere](#_bookmark0) 4
   1. [Motivație](#_bookmark1) 4
   2. [Dolor sit amet](#_bookmark2) 5
      1. [Quis tellus vitae](#_bookmark3) 5
2. [Preliminarii](#_bookmark4)
3. Contribuție
4. [Concluzii](#_bookmark5) 7

[Bibliografie](#_bookmark6) 8

**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 baza de date cu itemi relevanți.

**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.[1]

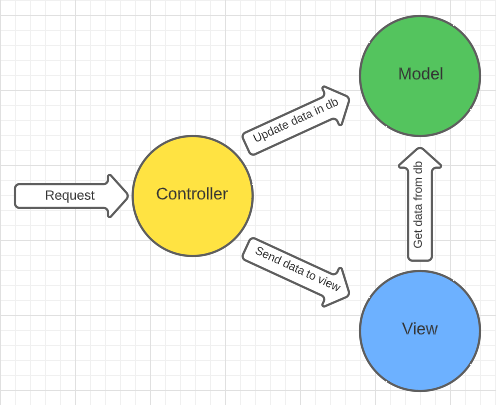


Figura 2.1 – Arhitectura MVC

**2.1.1 Model**

*Modelul* reprezintă ansamblul tuturor claselor ce depind de scopul aplicației.[2] 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 în urma *Modelului 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. [3]

**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. [3]

**2.2.3 IDE**

Termenul de *IDE* face referire la spațiul sau programul în care se realizează o aplicație, întru-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.

**2.3 NuGet, Entity Framework și Migrații**

**2.3.1 NuGet**

Microsoft a dezvoltat o platformă numită *NuGet* prin care pot fi descărcate și 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.[4] În aplicația dezvoltată am folosit versiunea EF6, mai exact 6.4.4.

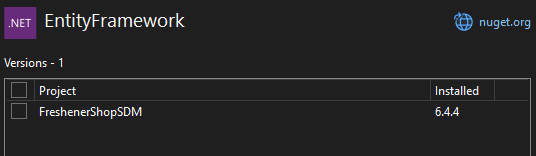
****

Figura 2.2 – Versiune Entity Framework

**2.3.3 Migrații**

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

Migrațiile de tip *Code First* 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 un *updatează* baza 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.[3]

**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()*. În continuare voi atașa o secvență de cod ce reprezintă unul din *bundle-urile­* definite în aplicație. Acest *bundle* cuprinde două fișiere de bootstrap, un fișier numit *site.css* ce conține cod CSS și un fișier pentru font-awesome. Elementele prezentate de bootstrap și font-awesome sunt descărcate automat prin platforma NuGet.

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/bootstrap.min.css",

"~/Content/site.css",

"~/Content/font-awesome.css"));

**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.

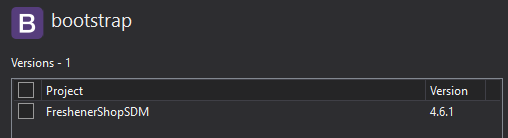


Figura 2.3 – Versiune Bootstrap

**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. Un aspect important este acela că datele trimise nu se regăsesc in *Model*.

*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 valoarea 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.

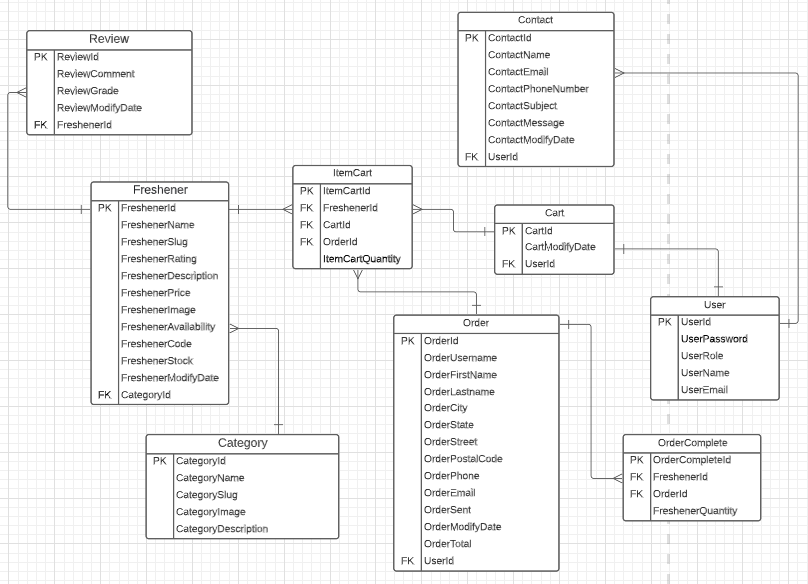


Figura 2.4 – Diagrama aplicației în Lucidchart

**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 Euro 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 mare, după cum se poate observa și în poza următoare. *Link-ul* *site-ului* este <https://freshenershopsdm20220525163306.azurewebsites.net/>.

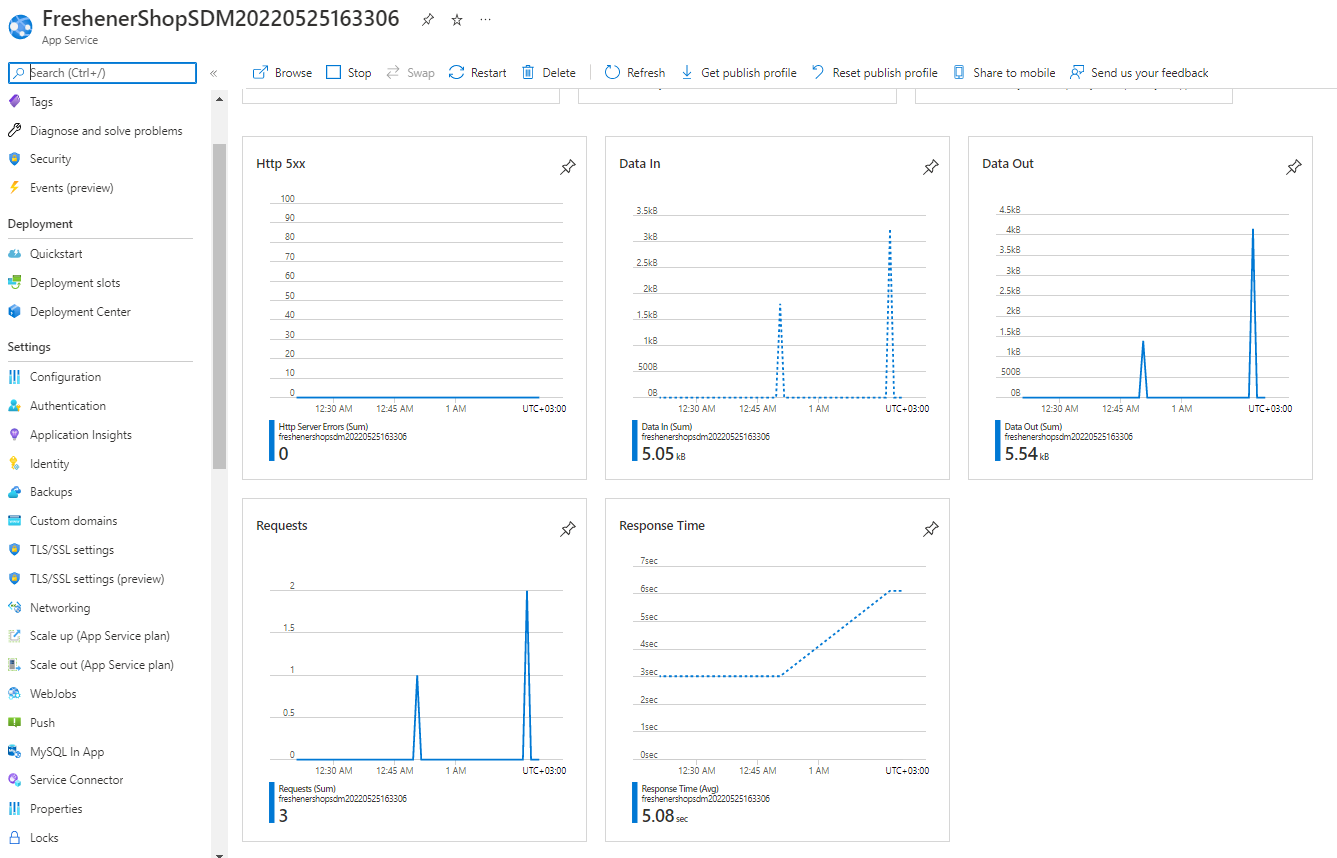


Figura 2.5 – Panoul de control Azure pentru *hosting web*

**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 pe platforma 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. [6]

**Capitolul 3**

# Contribuție

**3.1 Abordări existente**

Aplicația este realizată în ASP.NET MVC, o tehnologie ce facilitează dezvoltarea programelor complete, ușurează *debugging-ul*, 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 si 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 comercializare produse: 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 din coșul de cumpărături, *user-ul* 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 contrat apărând un text de informare.

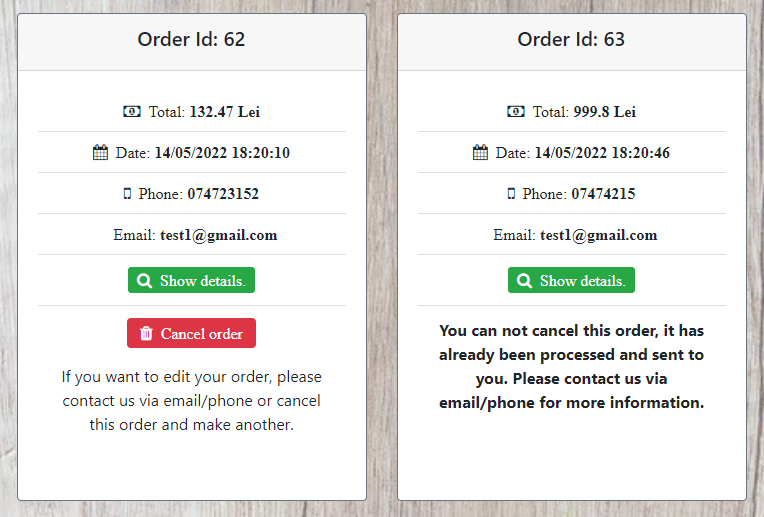


Figura 3.1 – Două comenzi plasate de un utilizator înregistrat

**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 idei create doar pentru admin. Acesta dispune și de un meniu special, de unde poate vedea lista tuturor comenzilor plasate, cu anumite detalii relevante, 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.

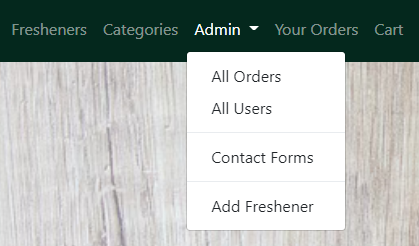


Figura 3.2 – Meniu special pentru admin

**3.3 Funcționalitățile aplicației**

**3.2.1 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.

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 și care nu este vizibil în partea de *frontend*, numele utilizatorului, o adresă de email pe care dorește să fie contactat, 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 acelui tip de produs și o referință către modelul *Order*. În tabela menționată 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 *regex-uri*[5]. Mai jos este afișat *regex-ul* 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 *regex-ul* 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 pentru 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, id-ul produsului și cantitatea setată.

În *IdentityModels* este definit fiecare model creat, un vector de tip *IEnumerable* pentru tipurile de roluri, cât și inițializarea bazei de date, fie locală, fie pe serverul Azure, la migrații.

**3.2.2 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.[7] 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 odorizant 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 produsului, 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*, *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 alcătuiește o listă cu toate *review-urile* aceluia, se șterg întai 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 precedentă.

În controllerele pentru *review-uri* și pentru *formulare* 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 definește o altă listă cu toate *review-urile* acelui produs și se șterg pe rând, întâi toate *review-urile*, iar apoi produsul.

Controllerul *ItemCarts* conține pentru început metoda *Index*, ce caută utilizatorul curent, caută coșul de cumpărături asociat acelui 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 daca 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 pordusul 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. Ca fiecare utilizator în parte să își vadă comenzile plasate de către el, am definit metoda *UserOrders*, foarte asemănatoare cu cea de *Index*, doar că fac o legătură între id-ul utilizatorului current și id-ul celui ce a plasat comanda. În funcția *New* este apelată initial funcția *CheckEmptyCart,* ce verifică dacă există vreun produs în coș, afișând un mesaj pe ecran în cazul în care nu există. Dacă există produse ses 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 controllerul pentru comenzi mai sus definite și operațiile de *delete*, *show* și *edit*, asemănătoare cu cele din controllerele precedente. Funcția *UserShowOrder* afișează detaliile unei comenzi pentru un anumit utilizator, iar *CompleteOrder* afișează detaliile unei comenzi după ce este efectuată.

Pentru UserController am implementat operațiile de bază *CRUD*, cu tratarea cazurilor special pentru abilitatea de *delete*. Atunci când adminul șterge un utilizator din baza de date, initial se șterg toate *review-urile* create de acel utilizator, i se șterge coșul de cumpărături, toate comenzile effectuate și formularele de contact, evitând în acest fel orice eroare ce putea apărea.

În HomeController este definite 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 odorizant, se ordonează descrecător după numărul de apariții și se selectează primele trei.

**3.2.3 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 prealua datele din modelul *Freshener*. Pentru câmpurile obligatorii din cele două formulare am folosit un mesaj de validare cu simbolul “\*”. Mai jos atașez codul pentru formularul de *New*, primele trei câmpuri.

@using (Html.BeginForm(actionName: "New", controllerName: "Fresheners"))

{

@Html.ValidationSummary(false, "", new { @class = "text-danger" })

<div class="form-row text-left justify-content-center">

<div class="form-group col-md-4">

@Html.Label("FreshenerName", "Freshener Name", new { @class = "text-center " })

@Html.ValidationMessageFor(m => m.FreshenerName, " \*", new { @class = "text-danger" })

@Html.TextBox("FreshenerName", null, new { @class = "form-control" })

</div>

<div class="form-group col-md-4">

@Html.Label("FreshenerSlug", "Freshener Slug", new { @class = "text-center" })

@Html.ValidationMessageFor(m => m.FreshenerSlug, "", new { @class = "text-danger " })

<br />

@Html.TextBox("FreshenerSlug", null, new { @class = "form-control" })

</div>

<div class="form-group col-md-4">

@Html.Label("FreshenerDescription", "Freshener Description", new { @class = "text-center" })

@Html.ValidationMessageFor(m => m.FreshenerDescription, " \*", new { @class = "text-danger " })

<br />

@Html.TextArea("FreshenerDescription", null, new { @class = "form-control" })

</div>

</div>

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 a produsului. Poza următoare prezintă aspectul paginii descrise anterior.

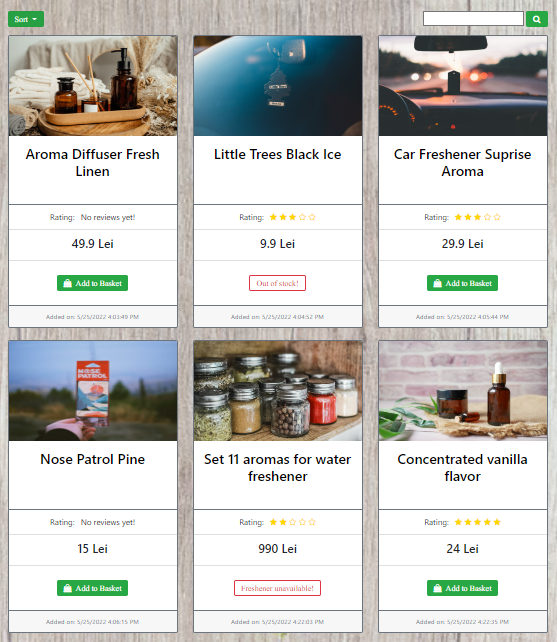


Figura 3.3 – Lista de produse din perspectiva adminului

Partea de *Show* pune în lumina 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.

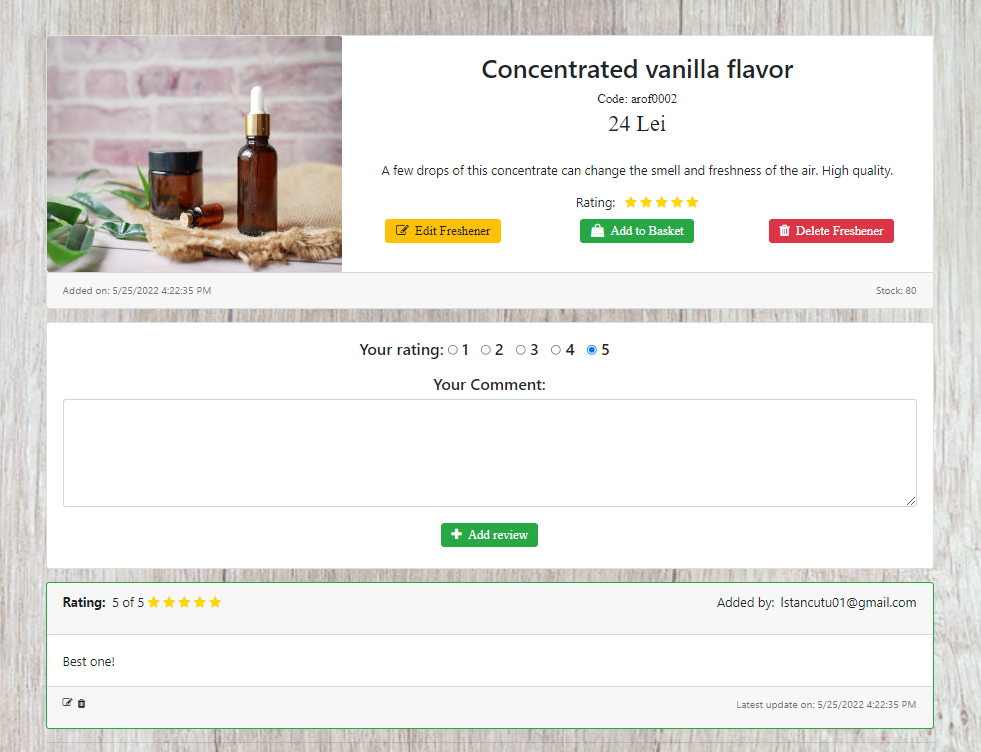


Figura 3.4 – Pagina unui produs din perspectiva adminului

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. La *Index* un buton, pentru redirecționare la lista completă de produse, dar și carduri Bootstrap, initial câte patru pe rând pentru ecranele mari, dar cu schimbarea acestui număr în funcție de *display*. Cardurile conțin câte o poză definitorie pentru fiecare categorie, numele și descrierea.Dacă utilizatorul curent are rolul de admin, mai apare un buton pentru adăugarea de categorii și încă unul pentru editarea unei categorii.

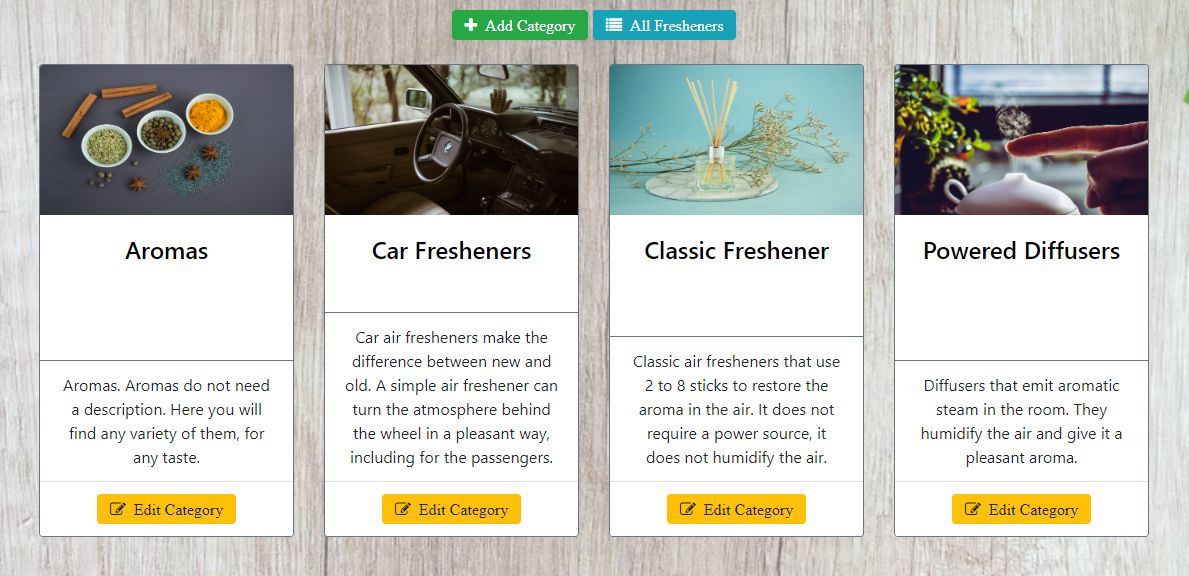


Figura 3.5 – Lista de categorii din perspectiva adminului

*Show-ul* are un design asemănător cu pagina de *Index* a produselor. Aici sunt illustrate produsele 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. Imaginea următoare ilustrează pagina pentru odorizantele de mașină.

În ASP.NET MVC există *Partial View* ce reprezintă un fișier în care se pun anumite date și urmează sa 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 antentul paginii, cu diferite butoane ce apar în funcție de rolul utilizatorului. 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 duc pe anumite pagini: *Home, About us, 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.

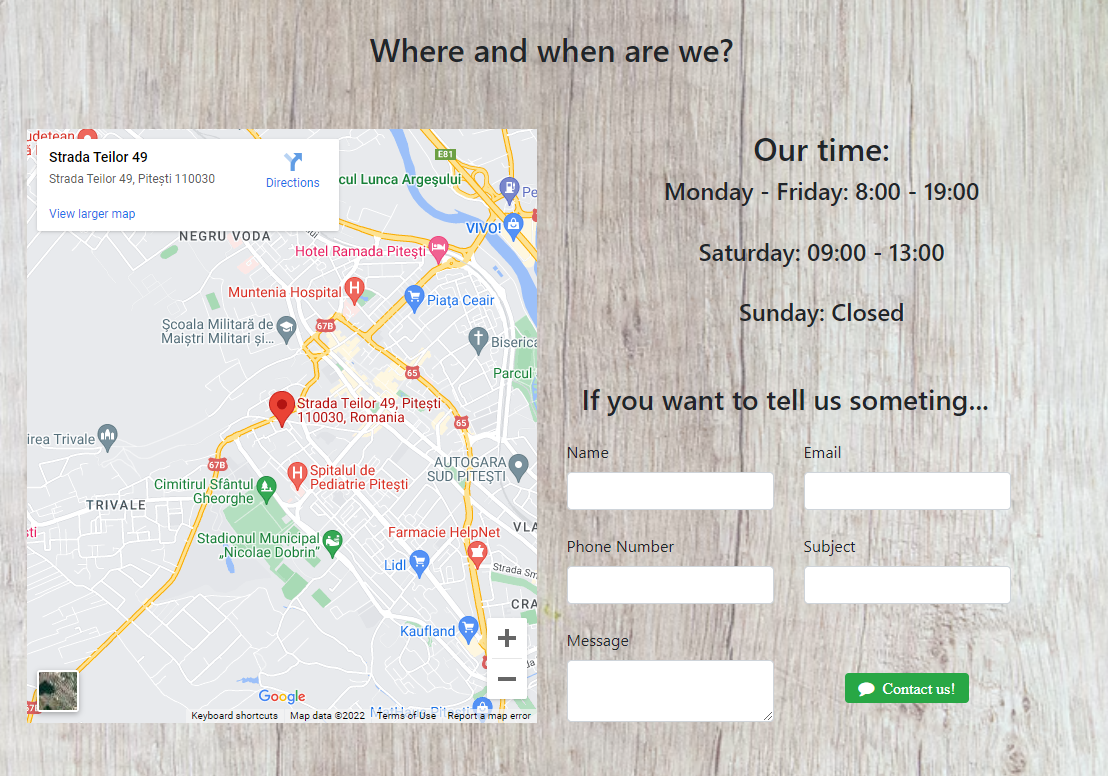


Figura 3.6 – Pagina de contact

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.

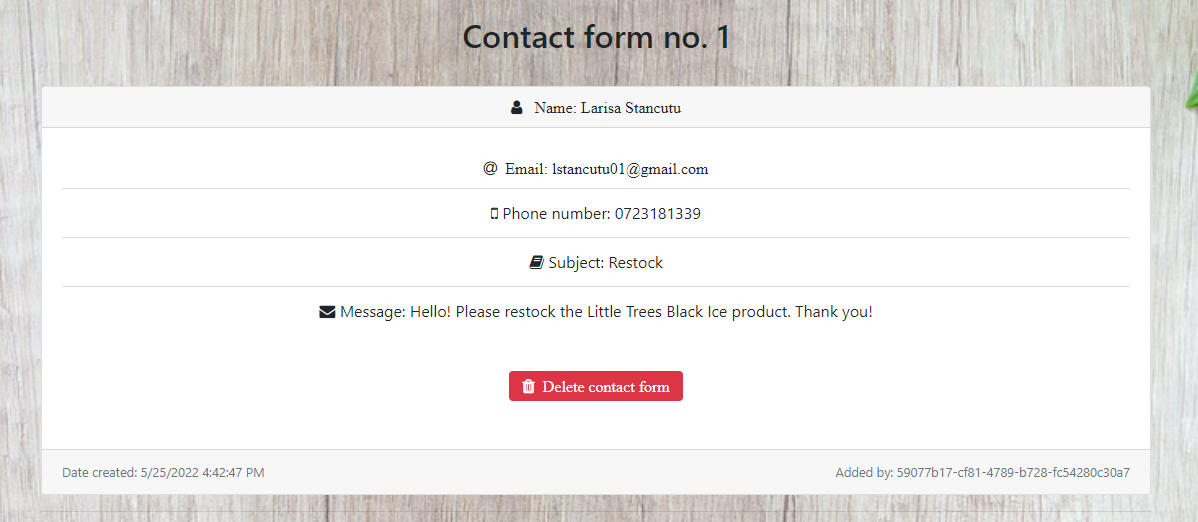


Figura 3.7 – Pagina de detalii a unui formular de contact

**Capitolul 4**

# Concluzii

**Capitolul 5**

**Bibliografie**

[1] 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.

[2] Sipoș

[3] Cezara

[4] *Entity Framework 6*, <https://docs.microsoft.com/en-us/ef/ef6/>, accesat pe 15 mai 2022.

[5] 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.

[6] *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.

[7] *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.