



**Universitatea
Transilvania
din Brașov**

**FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ**

Programul de studii:
Informatică

Lucrare de licență Crearea unei aplicații moderne web: Wheels&Deals

Autor:

Tăbăcaru Andrei Cosmin

Coordonator științific:

Conf. Dr. Livia Sangeorzan

Asist. Univ. Drd. Dragoș Tohănean

Brașov, Iunie 2024

Cuprins

1.Introducere	7
1.1 Context și motivație	7
1.2. Obiectivul aplicației	7
1.3. Relevanța temei	7
1.4. Noutatea aplicației	8
1.5. Beneficiile aplicației	8
1.6. Viitorul aplicației	8
2.Metodologia cercetării.....	9
2.1. Abordarea general	9
2.2. Etapele dezvoltării	9
2.2.1. Analiza cerințelor.....	9
2.2.2. Proiectarea arhitecturii	9
2.2.3. Dezvoltarea modulară.....	9
2.2.4. Testarea și validarea	9
2.3. Instrumente utilizate	9
2.4. Metode de colectare și interpretare a datelor	10
2.4.1. Colectarea datelor	10
2.4.2. Interpretarea datelor	10
2.5. Cum este mai ușor să găsești mașina dorită online	10
2.6. Concluzii metodologice	10
3. Prezentarea succinta a aplicației.....	11
3.1. Obiectivele aplicației Wheels&Deals.....	11
3.2. Interfața prietenoasă	11
3.3. Performanța aplicației	11
3.4. Securitatea aplicației	12
3.5. Ușurința de folosire.....	12
4. Tehnologii folosite	13
4.1. Tehnologii back-end	13
4.1.1.1 Caracteristici esențiale ale C#	13
4.1.2 .NET 8.....	16
4.1.2.1 Caracteristici esențiale ale .NET 8	17
4.1.3. AutoMapper	18
4.1.4 Cloudinary	21

4.1.4.1 Exemplu de cod pentru încărcarea fotografiilor în Cloudinary:	21
4.1.5 Microsoft.AspNetCore.Authentication.JwtBearer	22
4.1.4.1 Utilizarea JWT pentru autentificare și autorizare:	22
4.1.4.2 Configurarea JWT în Program.cs:	22
4.1.4.5 Generarea token-ului JWT în AccountController:	23
4.1.6 EntityFrameworkCore	25
4.1.5.1 Configurarea Entity Framework Core în Program.cs:	25
4.1.5.2 String-ul de conexiune:	26
4.1.5.3 Definirea contextului de date (DataContext.cs):	26
4.1.5.4 Definirea entităților:	26
4.1.4.5 Utilizarea EF Core în repository:	28
4.1.5.6 Migrații de baze de date:	28
4.1.7 Swagger	30
4.1.7.1 Caracteristici esențiale ale Swagger	30
4.1.7.2 Beneficiile utilizării Swagger	33
4.1.8. Newtonsoft.Json	33
4.1.8.1 Beneficiile utilizării Newtonsoft.Json:	33
4.1.8.2 Utilizarea Newtonsoft.Json pentru serializare și deserializare:	33
4.1.9 NuGet	34
4.1.9.1 Beneficiile utilizării NuGet	34
4.2. Tehnologii front-end	34
4.2.1. Angular	35
4.2.1.1 Caracteristici esențiale ale Angular	35
4.2.1.2 Exemple de utilizare a Angular	36
4.2.1.3 Beneficiile utilizării Angular	36
4.2.2 HTML	37
4.2.2.1 Caracteristici esențiale ale HTML :	37
4.2.2.2 Exemple din cod:	37
4.2.3 CSS	40
4.2.3.1 Caracteristici esențiale ale CSS	40
4.2.4. TypeScript	41
4.2.4.1 Caracteristici esențiale ale TypeScript	41
4.2.4.2 Utilizarea TypeScript în proiect	42
4.2.5. Bootstrap	43

4.2.5.1 Caracteristici esențiale ale Bootstrap.....	43
4.2.5.2 Utilizarea Bootstrap în proiect	43
4.2.5.3. Integrarea Bootstrap în proiect	45
4.2.6. AlertifyJS	46
4.2.6.1 Caracteristici esențiale ale AlertifyJS.....	46
4.2.6.2 Utilizarea AlertifyJS în proiect	47
1. Notificare de succes sign out	47
2. Notificare de eroare la schimbarea pozei principale a anutului.....	47
5. Descriere baza de date.....	48
5.1. Introducere	48
5.2. Crearea bazei de date	48
5.2.1. Abordarea "code first"	48
5.3. Migrațiile	48
5.4. Tabele și relații	49
5.4.1. Tabelul Users	50
5.4.2. Tabelul Cars.....	50
5.4.3. Tabelul CarTypes.....	51
5.4.4. Tabelul Brands.....	51
5.4.5. Tabelul FuelTypes.....	51
5.4.6. Tabelul Photos	51
5.4.7. Tabelul UserFavorites	51
5.5. Relațiile dintre tabele	52
6. Platforme folosite in dezvoltarea aplicatiei	53
6.1 Visual Studio 2022.....	53
6.1.1 Caracteristici principale ale Visual Studio 2022	53
6.2 Visual Studio Code	54
6.2.1 Caracteristici principale ale Visual Studio Code	54
6.3 Git.....	55
6.3.1 Caracteristici esențiale ale Git.....	56
6.3.2 Beneficiile utilizării Git	57
6.4 Microsoft SQL Server Management Studio (SSMS)	57
6.4.1 Caracteristici esențiale ale SSMS.....	57
7. Ghidul aplicatiei	59
8. Concluzii	64

Lista Figuri

Figura 1: Logo C#	13
Figura 2: Logo .Net.....	16
Figura 3: Evolutie .Net.....	16
Figura 4: Logo AutoMapper	18
Figura 5: Exemplu mapare	19
Figura 6: Exemplu functie unde se foloseste maparea	20
Figura 7: Logo Cloudinary	21
Figura 8: Functie pentru incarcarea pozelor	21
Figura 9: Logo JWT	22
Figura 10: Configurare JWT in program.cs	23
Figura 11: Generarea token-ului	24
Figura 12: Folosirea token-ului	24
Figura 13: Logo Entity Framework.....	25
Figura 14: Configurare Entity Framework.....	25
Figura 15: String conexiune baza de date.....	26
Figura 16: Clasa DataContext	26
Figura 17: Entitatea Car.....	27
Figura 18: Utilizare Entity Framework in repository	28
Figura 19: Creare migratie	29
Figura 20: Aplicare Migratie	29
Figura 21: Exemplu migratie	29
Figura 22: Logo Swagger.....	30
Figura 23: Interfata Swagger.....	32
Figura 24: Logo NuGet	34
Figura 25: Logo Angular	35
Figura 26: Logo HTML.....	37
Figura 27: Cod HTML pagina login	38
Figura 28: Cod HTML adaugare masini	39
Figura 29: Logo CSS.....	40
Figura 30: Logo TypeScript.....	41
Figura 31: Exemplu cod TypeScript.....	42
Figura 32: Logo Bootstrap	43
Figura 33: Exemplu utilizare Bootstrap pentru autentificare	44
Figura 34: Exemplu utilizare Botstrap pentru listarea masinilor	45
Figura 35: Integrare Bootstrap in proiect	45
Figura 36: Logo AlertifyJS	46
Figura 37: Exemplu notificare de succes in cod	47
Figura 38: Exemplu notificare de eroare in cod.....	47
Figura 39: Diagrama bazei de date.....	49
Figura 40: Logo Visual Studio 2022	53
Figura 41: Logo visual Studio Code.....	54
Figura 42: Logo Git	55
Figura 43: Logo Microsoft SQL Server Management Studio.....	57

Figura 44: Pagina principala a aplicatiei.....	59
Figura 45: Pagina de inregistrare	59
Figura 46: Pagina de autentificare	60
Figura 47: Pagina de anunturi.....	60
Figura 48: Pagina pentru adaugarea unei masini.....	61
Figura 49: Pagina pentru detaliile unei masini	61
Figura 50: Pagina pentru editatarea unei masini	62
Figura 51: Meniul de utilizator	62
Figura 52: Pagina favorite	63
Figura 53: Profil comerciant.....	63
Figura 54: Fereastra schimbare parola	63

1.Introducere

1.1 Context și motivație

În era digitală în care trăim, internetul a devenit o platformă esențială pentru desfășurarea multor activități, inclusiv pentru tranzacțiile de bunuri și servicii. Unul dintre domeniile care au beneficiat semnificativ de pe urma dezvoltării tehnologice este cel al comerțului auto. Aplicația Wheels&Deals a fost concepută pentru a răspunde acestei nevoi crescânde de a avea un mediu online eficient și sigur pentru vânzarea și închirierea de mașini.

Evoluția tehnologică a transformat modul în care oamenii interacționează cu piața auto. În trecut, cumpărătorii și vânzătorii de mașini se bazau pe anunțuri în ziare sau pe intermedierea dealerilor auto. Astăzi, internetul oferă o platformă globală unde tranzacțiile se pot desfășura rapid și eficient, fără limite geografice. În acest context, dezvoltarea unei aplicații precum Wheels&Deals vine ca o soluție modernă și necesară pentru a facilita aceste tranzacții.

1.2. Obiectivul aplicației

Scopul principal al Wheels&Deals este de a oferi utilizatorilor o platformă centralizată prin intermediul căreia pot vinde, cumpăra și închiria mașini cu ușurință. Aplicația se adresează atât vânzătorilor care doresc să își promoveze mașinile, cât și cumpărătorilor și chiriașilor care sunt în căutarea celor mai bune oferte.

Un alt obiectiv esențial este crearea unui mediu de tranzacționare transparent și sigur. Utilizatorii trebuie să poată avea încredere că datele lor sunt protejate și că tranzacțiile se desfășoară în condiții de siguranță. De asemenea, aplicația trebuie să fie ușor de utilizat, oferind o interfață intuitivă și funcționalități clare pentru toate categoriile de utilizatori.

1.3. Relevanța temei

Aplicația este relevantă din punct de vedere științific și tehnologic deoarece implică utilizarea unor tehnologii de ultimă generație în dezvoltarea web. Aceasta include framework-uri moderne, baze de date eficiente și tehnici avansate de autentificare și autorizare, toate menite să creeze o experiență de utilizare optimă și sigură. În plus, tema este de actualitate, având în vedere tendințele pieței auto și necesitatea de a avea soluții online fiabile pentru tranzacțiile de mașini.

Din perspectivă economică, piața auto este una dintre cele mai dinamice și competitive piețe. Introducerea unei platforme online care să faciliteze vânzarea și închirierea de mașini poate avea un impact semnificativ asupra eficienței tranzacțiilor și asupra satisfacției utilizatorilor. Prin integrarea unor tehnologii avansate, aplicația noastră răspunde nu doar nevoilor actuale, ci și cerințelor viitoare ale pieței.

1.4. Noutatea aplicației

Wheels&Deals se distinge prin integrarea unor funcționalități avansate care nu sunt adesea întâlnite în aplicațiile similare. De exemplu, utilizarea autentificării prin JWT (Json Web Token) pentru securizarea tranzacțiilor și stocarea fotografiilor pe un serviciu cloud cum ar fi Cloudinary. Aceste aspecte tehnice contribuie la un plus de siguranță și la o gestionare eficientă a resurselor.

O altă inovație a aplicației este implementarea unor algoritmi de căutare și filtrare avansați care permit utilizatorilor să găsească rapid mașinile care corespund cel mai bine criteriilor lor. De asemenea, integrarea cu diverse servicii de plată online și funcționalități de evaluare și recenzii oferă utilizatorilor o experiență completă și transparentă.

1.5. Beneficiile aplicației

Aplicația oferă numeroase beneficii utilizatorilor săi, inclusiv:

- **Eficiență:** Proces simplificat de postare a anunțurilor și de căutare a mașinilor.
- **Securitate:** Măsuri avansate de securitate pentru protejarea datelor utilizatorilor și a tranzacțiilor.
- **Interfață prietenoasă și responsive:** Un design intuitiv care facilitează utilizarea aplicației pentru toate categoriile de utilizatori, indiferent de dispozitivul utilizat.
- **Transparență:** Informații detaliate și actualizate despre fiecare mașină listată, inclusiv istoricul proprietarilor, starea tehnică și recenziile utilizatorilor.

1.6. Viitorul aplicației

Pe măsură ce tehnologia avansează, Wheels&Deals își propune să se adapteze și să integreze noi funcționalități care să răspundă nevoilor în continuă schimbare ale utilizatorilor. Planurile de viitor includ dezvoltarea de aplicații mobile dedicate, utilizarea inteligenței artificiale pentru recomandări personalizate și îmbunătățirea continuă a securității datelor.

2. Metodologia cercetării

2.1. Abordarea general

Pentru dezvoltarea aplicației Wheels&Deals, am adoptat o metodologie bazată pe practici agile. Aceasta mi-a permis să iterez rapid și să mă adaptez pe parcursul procesului de dezvoltare, asigurându-mă că fiecare etapă este bine gestionată și că cerințele utilizatorilor sunt îndeplinite în mod eficient.

2.2. Etapele dezvoltării

2.2.1. Analiza cerințelor

Primul pas a fost analiza cerințelor, unde am discutat cu potențialii utilizatori pentru a înțelege nevoile și așteptările acestora. Am identificat funcționalitățile esențiale ale aplicației, cum ar fi posibilitatea de a posta anunțuri, de a căuta mașini și de a gestiona conturile utilizatorilor. Această etapă a fost crucială pentru definirea clară a specificațiilor funcționale ale aplicației.

2.2.2. Proiectarea arhitecturii

Pe baza cerințelor colectate, am proiectat arhitectura aplicației. Am optat pentru o arhitectură modulară, separând logicele de business, datele și interfața utilizator. Aceasta a permis dezvoltarea independentă a fiecărui modul și a asigurat o integrare ușoară și coerentă a tuturor componentelor aplicației.

2.2.3. Dezvoltarea modulară

Dezvoltarea aplicației a fost realizată incremental. Fiecare modul a fost implementat, testat și integrat în sistem înainte de a trece la următorul. Aceasta a permis identificarea timpurie a problemelor și rezolvarea lor eficientă, asigurând în același timp o integrare fără probleme a noilor funcționalități.

2.2.4. Testarea și validarea

Testarea a fost esențială în asigurarea calității aplicației. Am folosit teste funcționale informale pentru a verifica că fiecare componentă funcționează conform așteptărilor. De asemenea, am efectuat teste de acceptanță cu utilizatori reali pentru a mă asigura că aplicația îndeplinește cerințele inițiale și oferă o experiență de utilizare optimă.

2.3. Instrumente utilizate

Pentru a facilita procesul de dezvoltare și pentru a asigura calitatea codului, am folosit următoarele instrumente și tehnologii:

- **Git:** Pentru controlul versiunilor și gestionarea codului sursă.
- **Visual Studio 2022:** IDE-ul principal folosit pentru dezvoltarea backend-ului.
- **Angular CLI:** Instrumentul utilizat pentru dezvoltarea și gestionarea frontend-ului.
- **Postman:** Pentru testarea API-urilor.
- **SQL Server Management Studio:** Pentru gestionarea bazei de date.

2.4. Metode de colectare și interpretare a datelor

2.4.1. Colectarea datelor

Colectarea datelor s-a realizat prin mai multe metode, inclusiv logurile aplicației și feedback-ul utilizatorilor. Logurile mi-au oferit informații valoroase despre performanța aplicației și comportamentul utilizatorilor. Feedback-ul utilizatorilor a fost esențial pentru a înțelege experiența de utilizare și pentru a identifica zonele care necesită îmbunătățiri.

2.4.2. Interpretarea datelor

Interpretarea datelor s-a bazat pe analiza logurilor și a feedback-ului primit. Am folosit aceste informații pentru a face ajustări și optimizări în aplicație. De exemplu, analiza logurilor a dezvăluit probleme de performanță care au fost ulterior rezolvate prin optimizarea interogărilor la baza de date și îmbunătățirea gestionării resurselor.

2.5. Cum este mai ușor să găsești mașina dorită online

Unul dintre obiectivele cheie ale aplicației Wheels&Deals a fost să simplifice procesul de găsire a unei mașini potrivite pentru utilizatori. În mediul online, acest lucru este realizat prin:

- **Funcționalități avansate de căutare:** Aplicația permite utilizatorilor să filtreze rezultatele în funcție de criterii precum marca, modelul, prețul, tipul de combustibil și altele. Aceste filtre avansate reduc timpul necesar pentru a găsi mașina dorită.
- **Interfață de utilizare intuitivă:** Designul aplicației este gândit pentru a fi ușor de navigat, astfel încât utilizatorii să poată accesa rapid informațiile de care au nevoie.
- **Anunțuri detaliate:** Fiecare anunț include informații detaliate despre mașină, fotografii de înaltă calitate și date de contact ale vânzătorului, ceea ce face procesul de luare a deciziilor mult mai simplu.
- **Notificări și alerte:** Utilizatorii pot seta notificări pentru a fi informați când sunt postate anunțuri noi care corespund criteriilor lor de căutare.

2.6. Concluzii metodologice

Metodologia adoptată pentru dezvoltarea aplicației Wheels&Deals s-a dovedit a fi eficientă, permițându-mi să livrez un produs de calitate într-un timp rezonabil. Abordarea iterativă și utilizarea practică a testării au ajutat la asigurarea stabilității și funcționalității aplicației, răspunzând totodată nevoilor utilizatorilor.

3. Prezentarea succinta a aplicației

3.1. Obiectivele aplicației Wheels&Deals

Aplicația Wheels&Deals își propune să redefinească modul în care utilizatorii cumpără, vând și închiriază mașini online. Obiectivele principale ale aplicației includ:

1. **Crearea unui marketplace centralizat și accesibil:** Oferirea unei platforme unice unde utilizatorii pot găsi rapid și ușor mașini de vânzare sau de închiriat, eliminând astfel nevoia de a naviga prin multiple site-uri și anunțuri disparate.
2. **Simplificarea procesului de tranzacționare auto:** Facilitarea întregului proces de la postarea anunțului, căutarea mașinii dorite, negocierea prețului, până la finalizarea tranzacției, prin intermediul unei interfețe intuitive și a unor funcționalități integrate.
3. **Asigurarea unui mediu sigur și de încredere:** Implementarea de măsuri avansate de securitate pentru protejarea datelor personale și a tranzacțiilor utilizatorilor, astfel încât aceștia să se simtă în siguranță atunci când folosesc aplicația.
4. **Optimizarea experienței utilizatorului:** Oferirea unei experiențe de utilizare seamless, adaptată nevoilor și preferințelor individuale ale fiecărui utilizator, indiferent de dispozitivul folosit.

3.2. Interfața prietenoasă

Unul dintre aspectele cheie ale aplicației Wheels&Deals este interfața sa prietenoasă, care a fost concepută cu utilizatorul în minte. Design-ul aplicației este intuitiv și curat, oferind acces rapid la toate funcționalitățile importante. Utilizatorii pot naviga cu ușurință prin diverse secțiuni ale aplicației, fie că sunt interesați să posteze un anunț, să caute o mașină sau să-și gestioneze contul.

Fiecare pagină din aplicație este organizată logic, cu meniuri clare și butoane de acțiune bine definite. De asemenea, interfața este responsive, ceea ce înseamnă că se adaptează automat la dimensiunea ecranului dispozitivului utilizat, oferind o experiență de navigare optimă atât pe desktop, cât și pe mobile.

3.3. Performanța aplicației

Performanța este un factor esențial pentru succesul oricărei aplicații web. Wheels&Deals este optimizată pentru a oferi timpi de încărcare rapizi și o navigare fluidă. Prin utilizarea unor tehnologii moderne și a unor practici de codare eficiente, aplicația asigură o performanță ridicată chiar și atunci când este accesată de un număr mare de utilizatori simultan.

Backend-ul aplicației este construit pentru a gestiona eficient cererile multiple, reducând astfel latențele și oferind răspunsuri rapide la interacțiunile utilizatorilor. De asemenea, utilizarea cache-ului pentru datele frecvent accesate contribuie la reducerea încărcăturii pe server și la îmbunătățirea vitezei generale a aplicației.

3.4. Securitatea aplicației

Securitatea este o prioritate majoră pentru Wheels&Deals. Aplicația implementează măsuri de securitate avansate pentru a proteja datele utilizatorilor și tranzacțiile acestora. Printre acestea se numără:

- **Autentificarea și autorizarea securizată:** Utilizarea token-urilor JWT (Json Web Token) pentru autentificarea utilizatorilor, asigurându-se că doar persoanele autorizate pot accesa anumite resurse și funcționalități.
- **Criptarea datelor:** Toate datele sensibile, inclusiv informațiile personale și detaliile de plată, sunt criptate pentru a preveni accesul neautorizat și furtul de date.
- **Monitorizarea și detectarea amenințărilor:** Implementarea unor mecanisme de monitorizare care detectează și răspund la activități suspecte sau malițioase, asigurând astfel protecția continuă a aplicației.

3.5. Ușurința de folosire

Wheels&Deals a fost proiectată pentru a fi extrem de ușor de utilizat. De la momentul în care un utilizator accesează aplicația și până la finalizarea unei tranzacții, fiecare pas este simplu și clar definit. Funcționalitățile sunt prezentate într-o manieră accesibilă, iar ghidurile și sfaturile contextuale sunt disponibile pentru a ajuta utilizatorii în orice moment. Navigarea în aplicație este facilitată de un meniu principal intuitiv și de opțiuni de căutare avansate care permit utilizatorilor să găsească rapid ceea ce caută. De asemenea, procesele de înregistrare și autentificare sunt rapide și directe, permițând utilizatorilor să înceapă să folosească aplicația fără întârzieri inutile.

4. Tehnologii folosite

4.1. Tehnologii back-end

4.1.1. Limbajul de programare C#

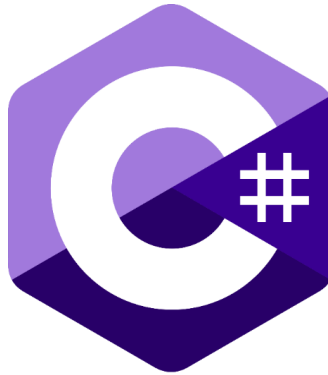


Figura 1: Logo C#

C# (pronunțat "C-sharp") este un limbaj de programare modern și robust, dezvoltat de Microsoft ca parte a inițiativei sale .NET. Acest limbaj de programare este extrem de versatil și puternic, fiind utilizat pentru o gamă largă de aplicații, inclusiv aplicații desktop, web, mobile și enterprise. În contextul proiectului Wheels&Deals, C# este utilizat pentru dezvoltarea logicii de afaceri și a interacțiunilor cu baza de date.

4.1.1.1 Caracteristici esențiale ale C#

- Sintaxă intuitivă și clară

C# are o sintaxă care este ușor de înțeles și de învățat, în special pentru cei care au experiență cu limbaje precum C++ sau Java. Acest lucru facilitează dezvoltarea rapidă și eficientă a aplicațiilor. Structura clară și coerentă a codului ajută la menținerea unei baze de cod curate și organizate, ceea ce este esențial pentru proiectele de mari dimensiuni.

- Tipizare statică și dinamică

C# este un limbaj de tipizare statică, ceea ce înseamnă că tipurile de date ale variabilelor sunt verificate la timp de compilare. Acest lucru ajută la detectarea timpurie a erorilor și îmbunătățește calitatea codului. De asemenea, C# suportă tipizarea dinamică prin utilizarea cuvântului cheie `dynamic`, permițând variabilelor să-și schimbe tipul la runtime, oferind astfel flexibilitate în anumite scenarii.

- Programare orientată pe obiecte (OOP)

C# suportă pe deplin principiile programării orientate pe obiecte, inclusiv moștenirea, încapsularea, polimorfismul și abstractizarea. Acest lucru permite

dezvoltatorilor să creeze structuri de date complexe și reutilizabile. Prin utilizarea claselor, interfețelor și moștenirii, C# facilitează organizarea codului într-un mod modular și clar, ceea ce contribuie la întreținerea și scalabilitatea aplicațiilor.

- Garbage Collection

C# gestionează automat memoria prin intermediul unui mecanism de colectare a gunoiului (garbage collection). Aceasta înseamnă că memoria ocupată de obiectele care nu mai sunt utilizate este eliberată automat, reducând riscul de scurgeri de memorie și alte probleme de gestionare a memoriei. Această caracteristică ajută la simplificarea gestionării resurselor și la îmbunătățirea stabilității aplicațiilor.

- Programare asincronă și paralelism

C# include suport nativ pentru programarea asincronă și paralelă prin intermediul cuvintelor cheie `async` și `await`. Acestea permit scrierea de cod care poate efectua operațiuni de lungă durată, cum ar fi accesul la rețea sau la bazele de date, fără a bloca firul principal de execuție. Biblioteca Task Parallel Library (TPL) oferă un set de funcții pentru execuția sarcinilor în paralel, îmbunătățind performanța aplicațiilor pe sisteme multicore.

- LINQ (Language Integrated Query)

LINQ este o caracteristică puternică a C# care permite interogarea colecțiilor de date folosind o sintaxă declarativă similară cu SQL. LINQ poate fi utilizat pentru a interoga array-uri, liste, baze de date și alte surse de date, simplificând astfel manipularea datelor. Aceasta face codul mai concis și mai ușor de citit, reducând în același timp riscul de erori.

- Suport pentru generics

Generics permit definirea de clase, metode și interfețe care funcționează cu orice tip de date, asigurând astfel reutilizarea codului și tipizarea statică. Generics îmbunătățesc siguranța și performanța codului, permițându-le dezvoltatorilor să scrie cod mai flexibil și mai robust. Aceasta contribuie la crearea unor colecții și algoritmi de date care sunt eficienți și siguri din punct de vedere al tipurilor de date.

- Interoperabilitate

C# oferă un suport excelent pentru interoperabilitate, permițând integrarea cu alte limbaje și platforme. Prin intermediul Platform Invocation Services (P/Invoke), dezvoltatorii pot apela funcții native din biblioteci scrise în limbaje precum C și C++. Aceasta permite utilizarea funcționalităților existente și a bibliotecilor de cod legacy fără a rescrie totul de la zero.

- Management avansat al memoriei

Pe lângă garbage collection, C# oferă structuri de date eficiente și alocări de memorie controlate, reducând supraîncărcarea și permițând optimizarea performanței. Structurile (structs) sunt un exemplu de tipuri de date care sunt alocate pe stivă, oferind o performanță mai bună în anumite scenarii.

- Delegates și events

Delegates sunt tipuri care reprezintă referințe la metode cu o anumită semnătură. Acestea sunt fundamentale pentru evenimente și callback-uri, permițând programarea orientată pe evenimente. Evenimentele (events) sunt utilizate pentru a semnaliza că s-a întâmplat ceva în cadrul unei aplicații, facilitând comunicarea între componente.

- Attributes și reflection

C# permite adăugarea de meta-informații la clase și metode prin intermediul atributelor (attributes). Acestea pot fi utilizate la runtime pentru a obține informații despre structura codului și pentru a controla comportamentul acestuia. Reflection oferă posibilitatea de a inspecta și manipula codul în mod dinamic, bazat pe aceste meta-informații.

- Platforma .NET și ecosistemul său

C# este strâns integrat cu platforma .NET, care oferă un set vast de biblioteci și framework-uri pentru dezvoltarea rapidă și eficientă a aplicațiilor. .NET Core, o variantă cross-platform a .NET, permite dezvoltarea de aplicații care rulează pe Windows, macOS și Linux. Acest lucru extinde considerabil aria de aplicabilitate a limbajului C# și oferă dezvoltatorilor flexibilitate în alegerea mediului de rulare.

- Suport excelent pentru testare

Limbajul C# și ecosistemul .NET includ un suport robust pentru testarea unităților (unit testing). Framework-uri precum MSTest, NUnit și xUnit permit scrierea și rularea testelor automatizate, contribuind la dezvoltarea de software de înaltă calitate. Testele automatizate sunt esențiale pentru identificarea rapidă a problemelor și pentru asigurarea faptului că noul cod nu introduce erori.

- Actualizări regulate și suport pe termen lung

Microsoft lansează actualizări regulate pentru C#, adăugând constant noi caracteristici și îmbunătățiri. Acest lucru asigură că limbajul rămâne modern și relevant în fața noilor provocări tehnologice. De asemenea, Microsoft oferă suport pe

termen lung pentru versiuni specifice ale platformei .NET, oferind stabilitate și continuitate pentru proiectele dezvoltate în C#.

- Integrare cu instrumente moderne de dezvoltare

C# beneficiază de un suport excelent în Visual Studio, unul dintre cele mai avansate și populare medii de dezvoltare integrate (IDE). Visual Studio oferă unelte puternice pentru scrierea, testarea și depanarea codului, precum și pentru gestionarea proiectelor mari și complexe. Această integrare sporește productivitatea dezvoltatorilor și facilitează gestionarea eficientă a ciclului de viață al dezvoltării software.

- Comunitate și resurse educaționale

C# are o comunitate vastă și activă de dezvoltatori care contribuie la o bogată colecție de resurse educaționale, inclusiv tutoriale, documentație, forumuri și conferințe. Acest ecosistem vibrant facilitează învățarea și dezvoltarea continuă pentru programatorii de toate nivelurile. Resursele disponibile sunt esențiale pentru rezolvarea rapidă a problemelor și pentru adoptarea celor mai bune practici în dezvoltarea software.

4.1.2 .NET 8



Figura 2: Logo .Net

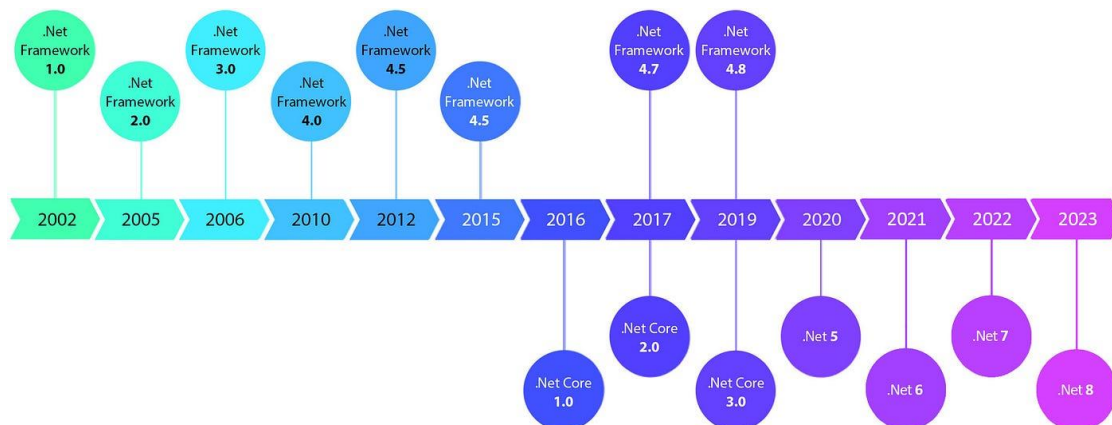


Figura 3: Evoluție .Net

.NET 8 este cea mai recentă versiune a platformei .NET dezvoltată de Microsoft. Aceasta aduce numeroase îmbunătățiri și caracteristici noi, menite să îmbunătățească performanța, securitatea și productivitatea dezvoltatorilor. Platforma .NET este un framework open-source care permite dezvoltarea de aplicații pentru diverse medii, inclusiv desktop, web, mobile și cloud.

4.1.2.1 Caracteristici esențiale ale .NET 8

- Performanță îmbunătățită

.NET 8 aduce îmbunătățiri semnificative de performanță, permițând aplicațiilor să ruleze mai rapid și mai eficient. Optimizările incluse în această versiune reduc timpul de răspuns al aplicațiilor și consumul de resurse, făcându-l ideal pentru dezvoltarea de aplicații scalabile și performante.

- Suport îmbunătățit pentru cloud

Cu .NET 8, dezvoltatorii beneficiază de un suport mai bun pentru integrarea și dezvoltarea aplicațiilor cloud-native. Aceasta include optimizări pentru Kubernetes, Docker și alte tehnologii de containerizare, facilitând implementarea și gestionarea aplicațiilor în medii cloud.

- Cross-platform

.NET 8 continuă tradiția de a fi o platformă cross-platform, permițând dezvoltatorilor să creeze aplicații care rulează pe Windows, macOS și Linux. Aceasta extinde aria de aplicabilitate a aplicațiilor dezvoltate pe .NET și oferă flexibilitate în alegerea mediului de execuție.

- Suport pentru C# 11

.NET 8 vine cu suport pentru cea mai recentă versiune a limbajului C#, C# 11, care introduce noi caracteristici și îmbunătățiri. Acestea includ optimizări de performanță, noi funcționalități pentru programarea asincronă și îmbunătățiri ale sistemului de tipuri.

- Suport îmbunătățit pentru microservicii

.NET 8 aduce îmbunătățiri pentru dezvoltarea și gestionarea arhitecturilor bazate pe microservicii. Acestea includ suport avansat pentru OpenTelemetry, care facilitează monitorizarea și diagnosticarea microserviciilor, precum și îmbunătățiri ale performanței și scalabilității acestora.

- Securitate avansată

Securitatea este o prioritate majoră în .NET 8, care include măsuri avansate pentru protejarea aplicațiilor și a datelor. Acestea includ suport îmbunătățit pentru autentificare și autorizare, criptare mai puternică și mecanisme de protecție împotriva atacurilor comune.

- Integrare cu AI și machine learning

.NET 8 oferă un suport îmbunătățit pentru integrarea cu soluții de inteligență artificială și machine learning. Acest lucru permite dezvoltatorilor să construiască aplicații inteligente care pot învăța și se pot adapta la comportamentul utilizatorilor, îmbunătățind astfel experiența generală a utilizatorului.

- Instrumente de dezvoltare avansate

Cu .NET 8, dezvoltatorii beneficiază de instrumente de dezvoltare mai avansate și eficiente. Visual Studio și Visual Studio Code primesc actualizări care îmbunătățesc experiența de dezvoltare, inclusiv debugging mai rapid, suport pentru noi limbaje și framework-uri, și o integrare mai bună cu platformele cloud.

- Suport pe termen lung (LTS)

.NET 8 este o versiune cu suport pe termen lung (LTS), ceea ce înseamnă că va beneficia de actualizări și suport extins din partea Microsoft pentru o perioadă mai lungă. Aceasta oferă stabilitate și încredere dezvoltatorilor care construiesc aplicații critice pentru afaceri, asigurându-se că vor primi actualizări de securitate și îmbunătățiri pe o perioadă îndelungată.

4.1.3. AutoMapper



Figura 4: Logo AutoMapper

AutoMapper este o librărie extrem de utilă în dezvoltarea aplicațiilor .NET, deoarece simplifică procesul de mapare a obiectelor. În mod specific, AutoMapper este folosit pentru a mape obiectele DTO (Data Transfer Object) la entități și invers, fără a fi nevoie de scrierea

manuală a codului de mapare. Această librărie este foarte apreciată pentru reducerea semnificativă a codului boilerplate și pentru îmbunătățirea clarității și întreținerii codului.

Unul dintre beneficiile principale ale utilizării AutoMapper este că permite dezvoltatorilor să se concentreze pe logica de business, în loc să se ocupe de detaliile implementării mapării obiectelor. În contextul aplicației Wheels&Deals, AutoMapper este folosit pentru a mapa obiectele de tip CarDto la entitățile Car, BrandDto la entitățile Brand, și multe altele.

```
CreateMap<Car, CarListDto>()  
    .ForMember(dest => dest.Brand, opt => opt.MapFrom(src => src.Brand.Name))  
    .ForMember(dest => dest.CarType, opt => opt.MapFrom(src => src.CarType.Name))  
    .ForMember(dest => dest.FuelType, opt => opt.MapFrom(src => src.FuelType.Name))  
    .ForMember(dest => dest.Km, opt => opt.MapFrom(src => src.Km))  
    .ForMember(dest => dest.Engine, opt => opt.MapFrom(src => src.Engine))  
    .ForMember(dest => dest.Power, opt => opt.MapFrom(src => src.Power))  
    .ForMember(dest => dest.Year, opt => opt.MapFrom(src => src.Year))  
    .ForMember(dest => dest.Description, opt => opt.MapFrom(src => src.Description))  
    .ForMember(dest => dest.Photo, opt => opt.MapFrom(src => src.Photos  
        .FirstOrDefault(p => p.IsPrimary).ImageUrl));
```

Figura 5: Exemplu mapare

În acest cod, configurarea profilelor AutoMapper definește regulile de mapare între diferitele tipuri de obiecte. CreateMap<TSource, TDestination> specifică tipurile sursă și destinație pentru mapare, iar AutoMapper se ocupă de restul, inclusiv de recunoașterea automată a proprietăților corespunzătoare și maparea acestora.

Pe lângă simplificarea procesului de mapare, AutoMapper poate fi personalizat pentru a gestiona scenarii mai complexe, cum ar fi mapările condiționate, mapările de colecții și multe altele. De exemplu, se pot defini conversii personalizate pentru anumite proprietăți sau se poate specifica cum să fie tratate proprietățile care nu corespund direct între obiectele sursă și destinație.

```

[HttpPost("addCarWithPhotos")]
[Authorize]
public async Task<IActionResult> AddCarWithPhotos([FromForm] CarDto carDto, [FromForm] List<IFormFile> files)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var userId = GetUserIdFromToken();
    if (userId == 0)
    {
        return Unauthorized("User must be logged in.");
    }

    var car = _mapper.Map<Car>(carDto);
    car.PostedBy = userId;
    _unitOfWork.CarRepository.AddCar(car);
    car.Photos = new List<Photo>();

    _unitOfWork.CarRepository.AddCar(car);

    if (files == null || !files.Any())
    {
        return BadRequest("Photos are required");
    }

    foreach (var file in files)
    {
        var result = await _photoService.UploadPhotoAsync(file);
        if (result.Error != null)
        {
            return BadRequest(result.Error.Message);
        }
        var photo = new Photo
        {
            ImageUrl = result.SecureUrl.AbsoluteUri,
            PublicId = result.PublicId,
            IsPrimary = car.Photos.Count == 0
        };
        car.Photos.Add(photo);
    }

    if (await _unitOfWork.SaveAsync())
    {
        return Ok(_mapper.Map<CarDto>(car));
    }

    return BadRequest("Failed to add car with photos");
}

```

Figura 6: Exemplu functie unde se foloseste maparea

Detalierea codului:

1. **Validarea modelului:** Înainte de a continua cu procesarea cererii, se verifică dacă modelul primit este valid. Dacă nu este valid, se returnează un BadRequest cu mesajul de eroare.
2. **Obținerea ID-ului utilizatorului:** Se extrage ID-ul utilizatorului din token-ul JWT. Dacă nu se poate obține un ID valid, se returnează un Unauthorized.
3. **Maparea DTO la entitate:** Utilizând AutoMapper, DTO-ul CarDto este mapat la entitatea Car. Aceasta include toate proprietățile necesare pentru a salva entitatea în baza de date.
4. **Adăugarea mașinii în baza de date:** Mașina este adăugată în repository-ul relevant folosind pattern-ul Unit of Work.
5. **Gestionarea fotografiilor:** Se verifică dacă sunt încărcate fișiere și dacă acestea sunt valide. Pentru fiecare fișier, fotografia este încărcată în Cloudinary și se creează un obiect Photo care este adăugat la entitatea Car.
6. **Salvarea modificărilor:** Dacă toate operațiunile au fost efectuate cu succes, modificările sunt salvate în baza de date și se returnează DTO-ul actualizat. În caz contrar, se returnează un mesaj de eroare.

4.1.4 Cloudinary



Figura 7: Logo Cloudinary

CloudinaryDotNet este o librărie utilizată pentru integrarea cu serviciul Cloudinary, care oferă soluții pentru gestionarea și stocarea fișierelor media în cloud. Cloudinary permite dezvoltatorilor să încarce, stocheze, manipuleze și livreze imagini și videoclipuri eficient. În aplicația Wheels&Deals, Cloudinary este utilizat pentru stocarea fotografiilor mașinilor.

4.1.4.1 Exemplu de cod pentru încărcarea fotografiilor în Cloudinary:

În acest exemplu, vedem cum Cloudinary este utilizat pentru a încărca fotografii în cloud, preluând fișierele încărcate de utilizatori și stocându-le în Cloudinary.

```
public async Task<ImageUploadResult> UploadPhotoAsync(IFormFile file)
{
    var uploadResult = new ImageUploadResult();

    if (file.Length > 0)
    {
        using var stream = file.OpenReadStream();
        var uploadParams = new ImageUploadParams
        {
            File = new FileDescription(file.FileName, stream),
            Transformation = new Transformation().Width(500).Height(500).Crop("fill").Gravity("face")
        };

        uploadResult = await _cloudinary.UploadAsync(uploadParams);
    }

    return uploadResult;
}
```

Figura 8: Funcție pentru încărcarea pozelor

Detalierea codului:

1. **Verificarea fișierului:** Se verifică dacă fișierul încărcat de utilizator are o lungime mai mare de 0. Dacă nu, metoda se oprește aici și returnează un rezultat implicit.
2. **Deschiderea fluxului fișierului:** Dacă fișierul este valid, se deschide un flux de citire a fișierului pentru a fi utilizat în încărcarea acestuia în Cloudinary.

3. **Configurarea parametrilor de încărcare:** Parametrii de încărcare sunt configurați, inclusiv fișierul propriu-zis și transformarea care va fi aplicată imaginii (de exemplu, redimensionarea la 500x500 pixeli și decuparea acesteia pentru a se potrivi).
4. **Încărcarea fișierului în Cloudinary:** Fișierul este încărcat în Cloudinary folosind metoda UploadAsync, care returnează un ImageUploadResult ce conține informații despre imaginea încărcată, cum ar fi URL-ul public și ID-ul public al imaginii.
5. **Returnarea rezultatului:** Rezultatul încărcării este returnat, permițând altor părți ale aplicației să acceseze informațiile despre imaginea încărcată.

4.1.5 Microsoft.AspNetCore.Authentication.JwtBearer



Figura 9: Logo JWT

JWT (Json Web Token) este un standard de industrie pentru autentificarea și autorizarea utilizatorilor. Microsoft.AspNetCore.Authentication.JwtBearer este o librărie utilizată în cadrul ASP.NET Core pentru a implementa autentificarea pe bază de JWT, oferind un mod sigur și scalabil de a proteja aplicațiile web.

4.1.4.1 Utilizarea JWT pentru autentificare și autorizare:

JWT este utilizat pentru a transmite informații de autentificare între client și server într-o manieră compactă și securizată. Token-urile JWT sunt compuse din trei părți: header, payload și semnătură. Acestea sunt codificate folosind Base64 și semnate pentru a asigura integritatea datelor.

4.1.4.2 Configurarea JWT în Program.cs:

Pentru a utiliza JWT în aplicația Wheels&Deals, trebuie să configurăm middleware-ul de autentificare în fișierul Startup.cs. Aceasta include setarea parametrilor de validare a token-ului și specificarea cheii de semnare.

```

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(option =>
    {
        option.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = key,
            ValidateIssuer = false,
            ValidateAudience = false
        };
    });

```

Figura 10: Configurare JWT in program.cs

Detalierea configurării:

1. **Cheia de securitate simetrică:** Se creează o cheie de securitate simetrică folosind o cheie secretă codificată în UTF-8. Aceasta va fi folosită pentru semnarea token-urilor JWT.
2. **Adăugarea schemei de autentificare:** Se adaugă schema de autentificare JWT Bearer la serviciile aplicației, specificând parametrii de validare a token-ului.
3. **Parametrii de validare a token-ului:**
 - ValidateIssuerSigningKey: Validează cheia de semnare a token-ului pentru a asigura integritatea acestuia.
 - IssuerSigningKey: Specifică cheia de semnare care va fi utilizată pentru validarea token-ului.
 - ValidateIssuer și ValidateAudience: Acestea sunt setate pe false pentru a simplifica validarea în acest context, dar în aplicații mai complexe ar trebui configurate corespunzător.

4.1.4.5 Generarea token-ului JWT în AccountController:

După ce utilizatorul este autentificat, un token JWT este generat și returnat clientului. Acest token este utilizat ulterior pentru a autoriza accesul la resursele protejate ale aplicației.

```

private string CreateJWT(User user)
{
    var secretKey = _configuration.GetSection("AppSettings:Key").Value;
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey));

    var claims = new Claim[]
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
        new Claim(ClaimTypes.Email, user.Email)
    };
    var signingCredentials = new SigningCredentials(
        key, SecurityAlgorithms.HmacSha256Signature);
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(claims),
        Expires = DateTime.UtcNow.AddDays(5),
        SigningCredentials = signingCredentials
    };
    var tokenHandler = new JwtSecurityTokenHandler();
    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
}

```

Figura 11: Generarea token-ului

Detalierea codului:

1. **Obținerea cheii secrete:** Cheia secretă este obținută din configurația aplicației. Aceasta este utilizată pentru a crea o cheie de securitate simetrică.
2. **Crearea revendicărilor (claims):** Se creează o serie de revendicări care conțin informații despre utilizator (ID și email). Aceste revendicări vor fi incluse în token-ul JWT.
3. **Credențialele de semnare:** Credențialele de semnare sunt create folosind cheia de securitate și algoritmul de semnare HmacSha256.
4. **Descriptorul token-ului:** Un SecurityTokenDescriptor este configurat cu revendicările, data de expirare și credențialele de semnare.
5. **Generarea token-ului:** JwtSecurityTokenHandler este utilizat pentru a crea și a scrie token-ul JWT.

```

[HttpPost("login")]
0 references
public async Task<IActionResult> Login(LoginRequestDto loginRequest)
{
    if (loginRequest.Email.IsNullOrEmpty() ||
        loginRequest.Password.IsNullOrEmpty())
    {
        return BadRequest("No fields can be blank");
    }
    var user = await _unitOfWork.UserRepository.Authenticate(loginRequest.Email, loginRequest.Password);

    if (user == null)
    {
        return Unauthorized("Invalid User ID or password");
    }
    var loginResponse = new LoginResponseDto();
    loginResponse.Email = loginRequest.Email;
    loginResponse.Token = CreateJWT(user);
    return Ok(loginResponse);
}

```

Figura 12: Folosirea token-ului

Detalierea codului:

1. **Validarea cererii:** Se verifică dacă câmpurile email și parolă sunt completate. Dacă oricare dintre acestea este gol, se returnează un BadRequest.
2. **Autentificarea utilizatorului:** Metoda Authenticate din UserRepository este utilizată pentru a verifica dacă email-ul și parola furnizate sunt corecte. Dacă autentificarea eșuează, se returnează un Unauthorized.
3. **Generarea token-ului JWT:** Dacă autentificarea este reușită, se generează un token JWT pentru utilizator utilizând metoda CreateJWT.
4. **Returnarea răspunsului:** Token-ul generat și email-ul utilizatorului sunt incluse în răspunsul LoginResponseDto și sunt trimise înapoi clientului.

4.1.6 EntityFrameworkCore



Figura 13: Logo Entity Framework

Entity Framework Core (EF Core) este un ORM (Object-Relational Mapper) pentru .NET, care facilitează accesul și manipularea datelor într-o bază de date. În aplicația Wheels&Deals, EF Core este utilizat împreună cu extensiile specifice pentru SQL Server și unele instrumente de dezvoltare pentru a crea, gestiona și manipula baza de date.

4.1.5.1 Configurarea Entity Framework Core în Program.cs:

Configurarea contextului de date și a serviciilor necesare este esențială pentru utilizarea EF Core. În Program.cs, configurăm contextul de date pentru a se conecta la SQL Server și adăugăm serviciile necesare pentru utilizarea EF Core.

```
builder.Services.AddDbContext<DataContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"),
        o => o.UseQuerySplittingBehavior(QuerySplittingBehavior.SplitQuery));
});
```

Figura 14: Configurare Entity Framework

Detalierea codului:

1. **AddDbContext:** Adaugă contextul de date DataContext la colecția de servicii, specificând utilizarea SQL Server ca motor de baze de date.

2. **UseSqlServer:** Configurează contextul de date pentru a utiliza SQL Server, utilizând string-ul de conexiune definit în fișierul de configurare.
3. **UseQuerySplittingBehavior:** Setează comportamentul de împărțire a interogărilor pentru a îmbunătăți performanța interogărilor complexe.

4.1.5.2 String-ul de conexiune:

String-ul de conexiune este definit în fișierul appsettings.json și conține toate informațiile necesare pentru a se conecta la baza de date SQL Server.

```
"ConnectionStrings": {  
  "DefaultConnection": "Server=DESKTOP-HC74F8S;Database=EasyWheels;Trusted_Connection=True;TrustServerCertificate=True"  
}
```

Figura 15: String conexiune baza de date

4.1.5.3 Definirea contextului de date (DataContext.cs):

Contextul de date reprezintă o unitate de lucru care permite interacțiunea cu baza de date. În EF Core, contextul este definit prin derivarea unei clase din DbContext.

```
public class DataContext: DbContext  
{  
    0 references  
    public DataContext(DbContextOptions<DataContext>options):base(options) { }  
    4 references  
    public DbSet<Brand>Brands { get; set; }  
    3 references  
    public DbSet<User> Users{ get; set; }  
    11 references  
    public DbSet<Car > Cars { get; set; }  
    1 reference  
    public DbSet<CarType> CarTypes { get; set; }  
    1 reference  
    public DbSet<FuelType> FuelTypes { get; set; }  
}
```

Figura 16: Clasa DataContext

Detalierea codului:

1. **Constructorul DataContext:** Constructorul primește opțiuni de configurare care sunt transmise clasei de bază DbContext.
2. **Definirea DbSet-urilor:** Fiecare DbSet reprezintă o colecție de entități care vor fi mapate la tabelele corespunzătoare din baza de date SQL Server.

4.1.5.4 Definirea entităților:

Pentru a defini structura tabelor și relațiile dintre acestea, EF Core utilizează clase de entitate. Să vedem un exemplu de clasă de entitate pentru Car.

```

public class Car : BaseEntity
{
    0 references
    public int BrandId { get; set; }
    5 references
    public Brand Brand { get; set; }
    0 references
    public string Model { get; set; }
    2 references
    public int Year { get; set; }
    2 references
    public int Km { get; set; }
    3 references
    public int Engine { get; set; }
    2 references
    public int Power { get; set; }
    0 references
    public int FuelTypeId { get; set; }
    5 references
    public FuelType FuelType { get; set; }
    0 references
    public int CarTypeId { get; set; }
    5 references
    public CarType CarType { get; set; }
    1 reference
    public int Price { get; set; }
    1 reference
    public int State { get; set; }
    2 references
    public string Description { get; set; }
    0 references
    public bool Available { get; set; }
    0 references
    public DateTime PostedOn { get; set; } = DateTime.Now;
    14 references
    public ICollection<Photo> Photos { get; set; }
    0 references
    public string Address { get; set; }
    0 references
    public int CityId { get; set; }
    0 references
    public DateTime EstPossessionOn { get; set; }
    [ForeignKey("User")]
    5 references
    public int PostedBy { get; set; }
    0 references
    public User User { get; set; }
}

```

Figura 17: Entitatea Car

Detalierea codului:

1. **Proprietăți de bază:** Proprietățile precum BrandId, Model, Year, Km, Engine, Power, Price, State, Description, Available, PostedOn, Address, CityId, EstPossessionOn definesc coloanele corespunzătoare din tabelul Cars.
2. **Relații:** BrandId, CarTypeId și FuelTypeId sunt chei străine care definesc relațiile dintre Car și entitățile Brand, CarType și FuelType. Proprietățile Brand, CarType și FuelType reprezintă navigațiile către entitățile respective.
3. **Colecția de fotografii:** Photos este o colecție de entități Photo care reprezintă relația unul-la-mulți între Car și Photo.

4.1.4.5 Utilizarea EF Core în repository:

EF Core este utilizat pentru a efectua operațiuni CRUD (Create, Read, Update, Delete) în repository. Să vedem un exemplu de metodă din CarRepository.

```
public class CarRepository : ICarRepository
{
    private readonly DataContext _dataContext;

    1 reference
    public CarRepository(DataContext dataContext)
    {
        _dataContext = dataContext;
    }

    4 references
    public void AddCar(Car car)
    {
        _dataContext.Cars.Add(car);
    }

    1 reference
    public void DeleteCar(int id)
    {
        var car = _dataContext.Cars.Find(id);
        if (car != null)
        {
            _dataContext.Cars.Remove(car);
        }
    }

    2 references
    public async Task<IEnumerable<Car>> GetCarAsync(int state)
    {
        var cars = await _dataContext.Cars
            .Include(c => c.CarType)
            .Include(c => c.Brand)
            .Include(c => c.FuelType)
            .Include(c => c.Photos)
            .Where(c => c.State == state)
            .ToListAsync();
        return cars;
    }
}
```

Figura 18: Utilizare Entity Framework in repository

Detalierea codului:

1. **Constructorul CarRepository:** Constructorul primește o instanță a contextului de date DataContext.
2. **AddCar:** Această metodă adaugă o nouă entitate Car în contextul de date.
3. **DeleteCar:** Această metodă elimină o entitate Car din contextul de date pe baza ID-ului.
4. **GetCarAsync:** Această metodă returnează toate entitățile Car care au un anumit state, inclusiv relațiile lor cu Brand, CarType, FuelType și Photos, folosind metodele Include și ToListAsync.

4.1.5.6 Migrații de baze de date:

EF Core oferă suport pentru migrații de baze de date, permițând dezvoltatorilor să gestioneze modificările schemelor de baze de date în mod incremental. Să vedem un exemplu de creare și aplicare a unei migrații.

- Crearea unei migrații:

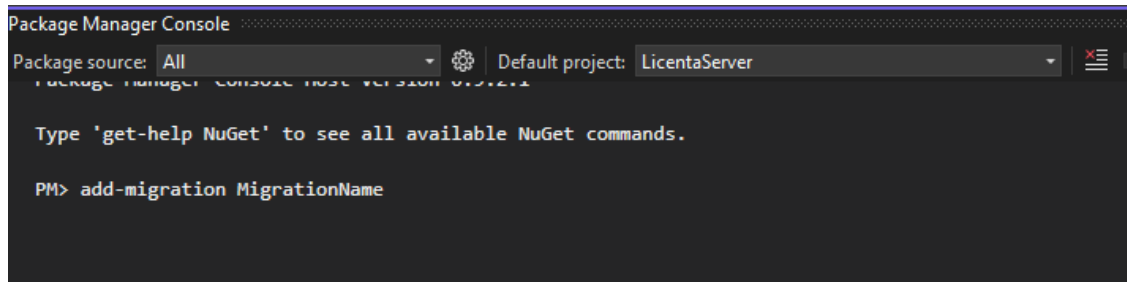


Figura 19: Creare migratie

- Aplicarea migrației:

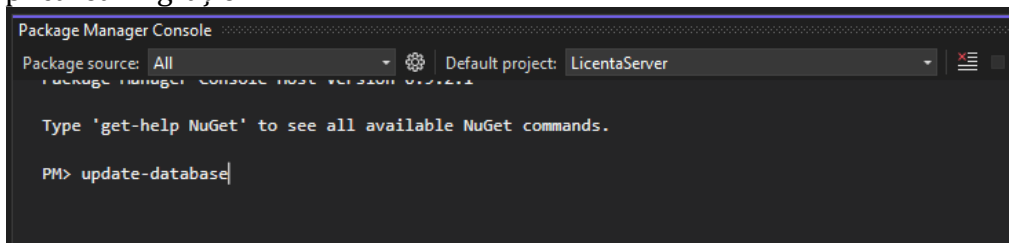


Figura 20: Aplicare Migratie

- Exemplu dintr-o migrație generată:

```
public partial class AddedNewEntities : Migration
{
    /// <inheritdoc />
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.RenameColumn(
            name: "LastUpdateOn",
            table: "Brands",
            newName: "LastUpdatedOn");

        migrationBuilder.AddColumn<int>(
            name: "LastUpdatedBy",
            table: "Users",
            type: "int",
            nullable: false,
            defaultValue: 0);

        migrationBuilder.AddColumn<DateTime>(
            name: "LastUpdatedOn",
            table: "Users",
            type: "datetime2",
            nullable: false,
            defaultValue: new DateTime(1, 1, 1, 0, 0, 0, 0, DateTimeKind.Unspecified));

        migrationBuilder.CreateTable(
            name: "CarTypes",
            columns: table => new
            {
                Id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
                LastUpdatedOn = table.Column<DateTime>(type: "datetime2", nullable: false),
                LastUpdatedBy = table.Column<int>(type: "int", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_CarTypes", x => x.Id);
            });
    }
}
```

Figura 21: Exemplu migratie

Beneficiile utilizării EF Core și a extensiilor sale:

1. **Gestionarea ușoară a migrărilor:** EF Core Tools facilitează crearea și aplicarea migrărilor, asigurând că schema bazei de date este întotdeauna sincronizată cu modelul de date.
2. **Generarea automată de cod:** Instrumentele generează automat codul necesar pentru migrații, reducând timpul și efortul necesar pentru gestionarea modificărilor în schema bazei de date.
3. **Flexibilitate și control:** Dezvoltatorii au control total asupra procesului de migrare, permițându-le să personalizeze migrațiile în funcție de nevoile aplicației.

4.1.7 Swagger



Figura 22: Logo Swagger

Swagger este un set de instrumente open-source pentru proiectarea, construirea, documentarea și consumarea serviciilor web RESTful. Inițial dezvoltat de SmartBear Software, Swagger este acum parte din ecosistemul OpenAPI Initiative. Utilizarea Swagger în dezvoltarea aplicațiilor web oferă numeroase beneficii, inclusiv documentație automată, testare ușoară și generarea de clienți și servere pentru diverse limbaje de programare.

4.1.7.1 Caracteristici esențiale ale Swagger

- Documentație automată

Swagger facilitează generarea automată a documentației pentru API-uri RESTful. Documentația este generată pe baza definițiilor API-ului scrise în format OpenAPI (anterior cunoscut sub numele de Swagger Specification). Aceasta include informații despre toate rutele, metodele HTTP, parametrii, răspunsurile și erorile posibile ale API-ului.

- Interfață interactivă

Swagger UI este o interfață grafică interactivă care permite dezvoltatorilor și utilizatorilor să exploreze și să interacționeze cu API-ul. Prin intermediul Swagger UI, utilizatorii pot trimite cereri către API și pot vedea răspunsurile în timp real, fără a scrie cod suplimentar. Aceasta facilitează testarea și depanarea API-urilor.

- Generarea de cod

Swagger Codegen este un instrument care generează automat clienți și servere pentru API-uri în diverse limbaje de programare. Aceasta simplifică procesul de creare a aplicațiilor

care consumă API-uri, reducând timpul și efortul necesar pentru a scrie cod de la zero. Printre limbajele suportate se numără C#, Java, Python, Ruby, PHP și multe altele.

- Specificații standardizate

Swagger utilizează specificațiile OpenAPI, care reprezintă un standard recunoscut pentru descrierea API-urilor RESTful. Aceasta asigură compatibilitatea și interoperabilitatea între diferite instrumente și platforme, permițând dezvoltatorilor să utilizeze diverse servicii și librării fără probleme de compatibilitate.

- Integrare ușoară

Swagger se integrează ușor cu diverse framework-uri și platforme de dezvoltare web, inclusiv ASP.NET Core, Spring Boot, Express și multe altele. Aceasta permite dezvoltatorilor să adauge documentația Swagger la proiectele lor existente fără a face modificări semnificative în codul sursă.

- Versionare și control al versiunilor

Swagger permite documentarea și gestionarea versiunilor multiple ale unui API. Aceasta este esențială pentru API-urile care evoluează în timp și care trebuie să mențină compatibilitatea cu versiunile anterioare. Prin documentarea clară a schimbărilor între versiuni, Swagger ajută la prevenirea problemelor de compatibilitate și la menținerea unei baze de cod stabilă.

- Utilizarea Swagger în proiectul Wheels&Deals

În cadrul aplicației Wheels&Deals, Swagger este utilizat pentru a documenta și testa API-urile RESTful care gestionează funcționalitățile de bază ale aplicației, cum ar fi gestionarea utilizatorilor, a mașinilor, a anunțurilor și a tranzacțiilor.

- Configurarea Swagger în ASP.NET Core

Pentru a configura Swagger în proiectul ASP.NET Core, se adaugă pachetul NuGet Swashbuckle.AspNetCore și se configurează serviciile Swagger în fișierul Program.cs:

- Documentație automată

Adăugarea și configurarea Swagger în aplicația ta permite generarea automată a documentației API. Aceasta include toate endpoint-urile disponibile, metodele HTTP utilizate, parametrii de intrare, tipurile de date și răspunsurile posibile. Acest lucru este esențial pentru dezvoltatori și utilizatori, deoarece oferă o înțelegere clară și detaliată a modului de interacțiune cu API-ul.

- Testare interactivă

Cu Swagger UI activat, poți testa interactiv API-ul direct din browser. Aceasta facilitează testarea rapidă a endpoint-urilor, trimiterea de cereri și vizualizarea răspunsurilor fără a necesita alte unelte externe. Este deosebit de util pentru depanare și pentru a asigura că toate endpoint-urile funcționează conform așteptărilor.

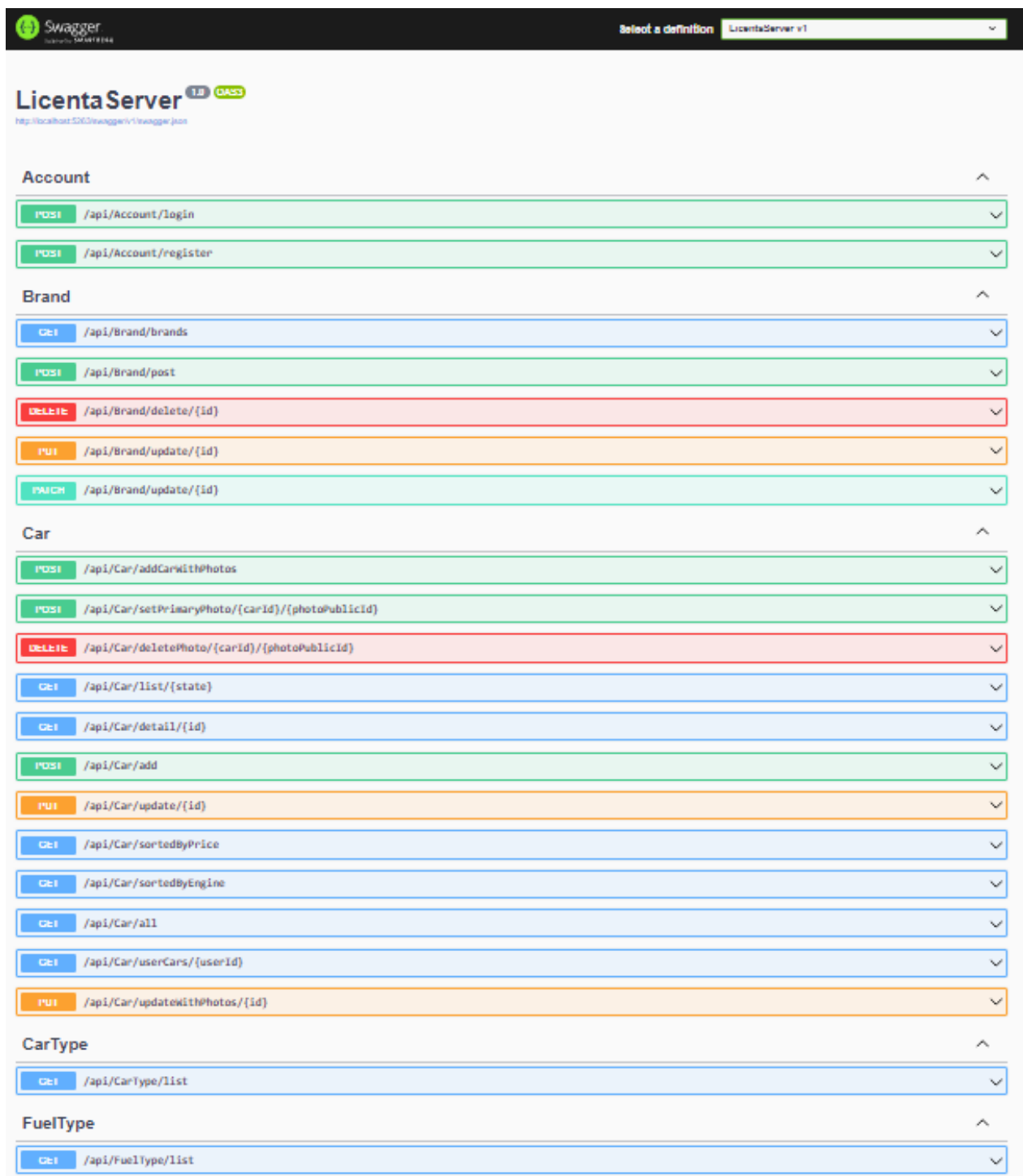


Figura 23: Interfața Swagger

4.1.7.2 Beneficiile utilizării Swagger

1. **Documentație clară și accesibilă:** Swagger generează automat o documentație clară și detaliată a API-ului, care este accesibilă printr-o interfață web. Aceasta facilitează înțelegerea și utilizarea API-ului de către dezvoltatori și utilizatori.
2. **Testare simplificată:** Prin intermediul Swagger UI, dezvoltatorii și testerii pot trimite cereri și pot vedea răspunsurile API-ului direct din browser. Aceasta simplifică testarea și depanarea, reducând timpul necesar pentru identificarea și rezolvarea problemelor.
3. **Generarea automată de clienți:** Swagger Codegen permite generarea automată de clienți pentru API în diverse limbaje de programare. Aceasta economisește timp și efort, permițând dezvoltatorilor să se concentreze pe funcționalitățile specifice ale aplicației lor.
4. **Interoperabilitate și standardizare:** Utilizarea specificațiilor OpenAPI asigură că API-ul este documentat conform unui standard recunoscut, ceea ce facilitează interoperabilitatea cu alte servicii și librării. Aceasta contribuie la adoptarea mai largă a API-ului și la integrarea sa cu alte platforme și aplicații.

4.1.8. Newtonsoft.Json

Microsoft.AspNetCore.Mvc.Newtonsoft.Json este o extensie care permite utilizarea librăriei Newtonsoft.Json pentru serializarea și deserializarea JSON în cadrul aplicațiilor ASP.NET Core. Newtonsoft.Json, cunoscut și sub numele de Json.NET, este o librărie populară și performantă pentru manipularea JSON în .NET.

4.1.8.1 Beneficiile utilizării Newtonsoft.Json:

1. **Flexibilitate:** Newtonsoft.Json oferă numeroase opțiuni de configurare, permițând personalizarea comportamentului de serializare și deserializare conform nevoilor aplicației.
2. **Performanță:** Este cunoscut pentru performanța sa excelentă în manipularea datelor JSON, făcându-l potrivit pentru aplicațiile web care necesită procese intensive de date.
3. **Compatibilitate:** Este larg utilizat în comunitatea .NET și este bine suportat, oferind compatibilitate cu o gamă largă de tipuri de date și scenarii de utilizare.

4.1.8.2 Utilizarea Newtonsoft.Json pentru serializare și deserializare:

În contextul API-urilor, Newtonsoft.Json este folosit pentru a serializa obiectele în JSON și pentru a deserializa JSON-ul în obiecte. Aceasta facilitează comunicarea între client și server.

4.1.9 NuGet



Figura 24: Logo NuGet

NuGet este managerul de pachete oficial pentru platforma .NET, dezvoltat de Microsoft. Acesta permite dezvoltatorilor să adauge, să actualizeze și să gestioneze librării de cod extern în proiectele lor .NET într-un mod simplu și eficient. NuGet facilitează partajarea codului între proiecte și dezvoltatori, contribuind la crearea unui ecosistem bogat de componente reutilizabile.

4.1.9.1 Beneficiile utilizării NuGet

1. **Eficiență:** Adăugarea și gestionarea pachetelor se face rapid și eficient, economisind timp prețios pentru dezvoltatori.
2. **Reutilizare:** Dezvoltatorii pot reutiliza codul existent, reducând efortul necesar pentru a scrie și testa noi funcționalități de la zero.
3. **Actualizări și securitate:** Pachetele sunt ușor de actualizat, asigurându-se că proiectele beneficiază de cele mai recente îmbunătățiri și corecții de securitate.
4. **Compatibilitate:** Gestionarea dependențelor asigură că toate pachetele funcționează bine împreună, minimizând problemele de compatibilitate.

4.2. Tehnologii front-end

În dezvoltarea aplicației Wheels&Deals, front-end-ul joacă un rol crucial în asigurarea unei interacțiuni plăcute și eficiente cu utilizatorii. Tehnologiile front-end utilizate sunt selectate pentru a oferi o experiență de utilizare optimă, incluzând framework-uri și librării moderne care facilitează dezvoltarea rapidă și întreținerea ușoară a aplicației.

4.2.1. Angular



Figura 25: Logo Angular

Angular este un framework front-end dezvoltat de Google, utilizat pentru crearea de aplicații web dinamice și complexe. Angular oferă o structură solidă și un set complet de instrumente pentru dezvoltarea de interfețe de utilizator robuste.

4.2.1.1 Caracteristici esențiale ale Angular

- Arhitectură bazată pe componente

Folosește o arhitectură bazată pe componente, unde fiecare parte a interfeței este separată într-o componentă proprie. Aceasta permite dezvoltatorilor să construiască interfețe modulare și reutilizabile, facilitând întreținerea și scalabilitatea aplicației.

- Two-way data binding

Suportă two-way data binding, ceea ce înseamnă că modificările din modelul de date sunt reflectate automat în interfața utilizatorului și viceversa. Aceasta simplifică sincronizarea datelor între model și interfață, reducând necesitatea de cod suplimentar pentru actualizarea manuală a interfeței.

- Dependency Injection

Include un sistem de dependency injection care gestionează automat dependențele dintre componente și servicii. Aceasta îmbunătățește modularitatea și testabilitatea codului, permițând dezvoltatorilor să creeze aplicații mai ușor de întreținut și extins.

- CLI puternic

CLI (Command Line Interface) este un instrument puternic care facilitează crearea, dezvoltarea și testarea aplicațiilor Angular. CLI-ul oferă comenzi pentru generarea de componente, servicii, module și alte entități, simplificând procesul de dezvoltare și asigurând respectarea celor mai bune practici.

- Suport pentru Reactive Programming

Integrează RxJS (Reactive Extensions for JavaScript), care permite dezvoltarea de aplicații reactive. Prin utilizarea RxJS, dezvoltatorii pot gestiona fluxuri de date asincrone și evenimente în mod declarativ, îmbunătățind performanța și ușurința de gestionare a codului.

- Utilizarea Angular în proiectul Wheels&Deals

În cadrul aplicației Wheels&Deals, Angular este utilizat pentru a construi interfața de utilizator, incluzând paginile de listare și detalii ale mașinilor, formularele de autentificare și înregistrare, și multe altele. Angular facilitează crearea unei interfețe de utilizator responsive și interactive, asigurând o experiență de utilizare fluidă și intuitivă.

4.2.1.2 Exemple de utilizare a Angular

- Structura componentelor

Fiecare funcționalitate majoră din aplicație este implementată ca o componentă Angular. De exemplu, componentele pentru adăugarea de anunțuri noi, listarea mașinilor disponibile, vizualizarea detaliilor unei mașini și editarea profilului utilizatorului sunt toate implementate ca module separate.

- Detalii despre module și servicii

Permite organizarea codului în module și servicii, ceea ce facilitează gestionarea dependențelor și partajarea codului între diferite părți ale aplicației. De exemplu, un serviciu pentru gestionarea autentificării utilizatorilor poate fi utilizat de multiple componente pentru a verifica starea autentificării și a obține informațiile despre utilizator

4.2.1.3 Beneficiile utilizării Angular

1. **Modularitate și reutilizare:** Arhitectura bazată pe componente și sistemul de module permit crearea de interfețe modulare și reutilizabile, facilitând întreținerea și scalabilitatea aplicației.
2. **Performanță și reactivitate:** Utilizarea RxJS și a programării reactive îmbunătățește performanța și facilitează gestionarea fluxurilor de date asincrone.
3. **Productivitate ridicată:** CLI-ul Angular și suportul pentru dependency injection îmbunătățesc productivitatea dezvoltatorilor, simplificând procesul de dezvoltare și testare.

4.2.2 HTML



Figura 26: Logo HTML

HTML (HyperText Markup Language) este limbajul standard pentru crearea și structura paginilor web. În cadrul proiectului Wheels&Deals, HTML este utilizat pentru a structura diferitele pagini și componente ale aplicației, oferind un fundament solid pe care se aplică stilurile CSS și se adaugă funcționalitatea JavaScript.

4.2.2.1 Caracteristici esențiale ale HTML :

1. **Structurarea conținutului:** HTML permite organizarea conținutului paginilor web într-un mod logic și semantic. Utilizarea corectă a elementelor HTML ajută la crearea unei structuri clare și ușor de înțeles, atât pentru utilizatori cât și pentru motoarele de căutare.
2. **Formulare și inputuri:** HTML oferă elemente pentru crearea de formulare și colectarea de date de la utilizatori, cum ar fi `<form>`, `<input>`, `<textarea>`, și `<button>`.
3. **Multimedia:** HTML5 introduce elemente multimedia precum `<photo>`, care permite includerea directă de fișiere media pe paginile web, fără a necesita plugin-uri externe.

4.2.2.2 Exemple din cod:

- Pagina de login, una dintre componentele esențiale ale aplicației.

```

<div class="row">
  <div class="col-md-6 col-lg-4 m-auto">
    <div class="card shadow-sm bg-white rounded-3 p-3">
      <div class="card-header text-center bg-dark-green text-white rounded-top">
        Login
      </div>
      <div class="card-body p-3">
        <form #loginForm='ngForm' (ngSubmit)="onLogin(loginForm)">
          <div class="form-group mb-2">
            <label for="email" class="form-label text-dark">E-mail</label>
            <input class="form-control form-control-sm" required ngModel name="email">
          </div>
          <div class="form-group mb-2">
            <label for="password" class="form-label text-dark">Password</label>
            <input class="form-control form-control-sm" required type="password" ngModel name="password">
          </div>
          <div class="form-group d-flex justify-content-between mb-2">
            <button [disabled]="!loginForm.valid" type="submit" class="btn btn-dark-green btn-sm">Login</button>
            <button type="button" class="btn btn-outline-secondary btn-sm">Cancel</button>
          </div>
        </form>
        <div class="social-login mt-2">
          <button class="btn btn-google btn-sm w-100 mb-1"><i class="fab fa-google"></i> Login with Google</button>
        </div>
        <div class="register-link mt-2 text-center">
          <p>Don't have an account? <a routerLink="/register" class="text-dark-green">Create one</a></p>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figura 27: Cod HTML pagina login

Explicație

- **Structura generală:** Folosim un container de tip row pentru a centra formularul în pagina. Elementele col-md-6 și col-lg-4 controlează lățimea formularului pe diferite dimensiuni de ecran.
 - **Formularul de autentificare:** Conține două câmpuri de input pentru email și parolă, fiecare cu etichete (label) și stilizate folosind clase CSS.
 - **Butonul de login:** Activ doar dacă formularul este valid (toate câmpurile obligatorii sunt completate).
 - **Opțiuni de social login:** Un buton pentru autentificarea cu Google.
 - **Link-ul de înregistrare:** Redirecționează utilizatorul către pagina de înregistrare dacă nu are un cont.
- Portiune din pagina pentru adaugarea masinilor

```

<tabset class="member-tabset" #formTabs>
  <tab heading="Basic Info" [formGroupName]='BasicInfo'>
    <div class="mb-3">
      <label for="Brand">Brand</label>
      <select class="form-select" formControlName="brand" (change)="updateBrand($event)">
        <option value="">-- Select brand --</option>
        <option *ngFor="let brand of brandList" [value]="brand.id">{{ brand.name }}</option>
      </select>
    </div>
    <div class="mb-3">
      <label for="Model">Model</label>
      <input type="text" class="form-control" formControlName="model" (input)="updateModel($event)">
    </div>
    <div class="mb-3">
      <label for="Type">Car Type</label>
      <select class="form-select" formControlName="type" (change)="updateCarType($event)">
        <option *ngFor="let type of carTypes" [value]="type.id">{{ type.name }}</option>
      </select>
    </div>
    <div class="mb-3">
      <label for="Year">Year</label>
      <input type="number" class="form-control" formControlName="year" (input)="updateYear($event)">
    </div>
    <div class="mb-3">
      <label for="Price">Price ($)</label>
      <input type="number" class="form-control" formControlName="price" (input)="updatePrice($event)">
    </div>
    <button type="button" class="btn btn-dark-green w-100 mt-4" (click)="selectTab(1, addCarForm.get('BasicInfo')?.valid)">Next</button>
  </tab>
</tabset>

```

Figura 28: Cod HTML adaugare masini

Explicație

- **Structura generală:** Utilizăm un tabset pentru a organiza formularul în secțiuni logice. Prima secțiune este dedicată informațiilor de bază ale mașinii.
- **Câmpurile formularului:** Câmpurile includ selectarea brandului, introducerea modelului, tipului de mașină, anul și prețul. Fiecare câmp este legat de un formControlName specific, permițând Angular să gestioneze starea și validarea acestora.
- **Butonul de Next:** Butonul pentru a trece la următoarea secțiune a formularului, activ doar dacă toate câmpurile din secțiunea curentă sunt valide.

HTML joacă un rol crucial în structura și organizarea interfeței de utilizator în aplicația Wheels&Deals. Utilizând elemente HTML bine definite, aplicația oferă o experiență de utilizare clară și intuitivă, facilitând interacțiunea utilizatorilor cu funcționalitățile platformei.

4.2.3 CSS



Figura 29: Logo CSS

CSS (Cascading Style Sheets) este limbajul folosit pentru a descrie aspectul și formatul unui document HTML. CSS permite dezvoltatorilor web să controleze stilizarea și aspectul paginilor web, separând conținutul de prezentare. În proiectul Wheels&Deals, CSS este esențial pentru a crea un design plăcut și a asigura o experiență de utilizare consistentă.

4.2.3.1 Caracteristici esențiale ale CSS

- Stilizare și layout

CSS permite aplicarea de stiluri la elementele HTML pentru a defini cum arată acestea pe pagină. Aceasta include setarea culorilor, fonturilor, marginilor, spațierilor și multe altele.

- Responsivitate

CSS Media Queries permit dezvoltarea de design-uri care se adaptează la diferite dimensiuni de ecran și dispozitive. Acest lucru este crucial pentru asigurarea unei experiențe de utilizare optimă pe desktop-uri, tablete și telefoane mobile.

- Flexbox și Grid

CSS Flexbox și Grid sunt module de layout avansate care permit aranjarea elementelor pe pagină într-un mod flexibil și aliniat. Acestea sunt foarte utile pentru crearea de layout-uri complexe și responsive.

4.2.4. TypeScript



Figura 30: Logo TypeScript

TypeScript este un limbaj de programare dezvoltat de Microsoft care extinde JavaScript prin adăugarea de tipuri statice. Acesta este utilizat pe scară largă în dezvoltarea aplicațiilor Angular pentru a îmbunătăți calitatea codului și a facilita detectarea erorilor la compilare.

4.2.4.1 Caracteristici esențiale ale TypeScript

- Tipuri statice

TypeScript adaugă tipuri statice la JavaScript, permițând dezvoltatorilor să definească tipurile variabilelor și funcțiilor. Aceasta ajută la detectarea erorilor la timp de compilare și îmbunătățește lizibilitatea și întreținerea codului.

- Suport pentru ES6 și ES7

TypeScript suportă caracteristici moderne ale JavaScript, cum ar fi clase, module și funcții arrow, permițând dezvoltatorilor să utilizeze cele mai recente inovații din limbajul JavaScript.

- Integrare strânsă cu Angular

TypeScript este limbajul de bază pentru dezvoltarea Angular, oferind o integrare strânsă și optimizări specifice pentru acest framework. Tipurile statice și modulele ajută la structurarea clară a aplicației și la menținerea consistenței codului.

- Interfețe și generice

TypeScript permite definirea interfețelor și utilizarea generics pentru a asigura reutilizarea codului și tipizarea statică. Interfețele definesc contracte stricte pentru obiecte, în timp ce generice permit crearea de clase și funcții care funcționează cu orice tip de date.

4.2.4.2 Utilizarea TypeScript în proiect

În proiectul Wheels&Deals, TypeScript este folosit pentru a scrie toate componentele Angular, serviciile și alte module de front-end. Acesta asigură un cod mai sigur și mai ușor de întreținut prin utilizarea tipurilor statice și a altor caracteristici avansate.

Exemplu de TypeScript

```
passwordMatchingValidator(control: AbstractControl): Validators | null {
  const password = control.get('password')?.value;
  const confirmPassword = control.get('confirmPassword')?.value;
  return password === confirmPassword ? null : { notMatched: true };
}

//#region Getters ...

onSubmit() {
  console.log(this.registrationForm);
  this.userSubmitted = true;

  if(this.registrationForm.valid) {
    this.authService.registerUser(this.userData()).subscribe(() => {
      this.registrationForm.reset();
      this.userSubmitted = false;
      this.alertify.success('Account created successfully, please log in');
      this.router.navigate(['/login']);
    });
  }
}

userData(): UserForRegister{
  return this.user = {
    firstName: this.firstName.value,
    lastName: this.lastName.value,
    email: this.email.value,
    password: this.password.value,
    mobileNumber: this.mobile.value
  }
}
```

Figura 31: Exemplu cod TypeScript

- TypeScript permite crearea de validatori personalizați pentru formulare, facilitând astfel gestionarea logicii de validare. În exemplu avem un validator personalizat pentru a verifica dacă parola și confirmarea parolei coincid.
- **Metoda onSubmit:** Aceasta metodă este apelată atunci când utilizatorul trimite formularul de înregistrare. Verifică dacă formularul este valid și, dacă este, trimite datele utilizatorului către serviciul de autentificare.
- **Resetarea formularului:** După trimiterea cu succes a datelor, formularul este resetat, iar utilizatorul este redirecționat către pagina de login.
- **Crearea obiectului userData:** Metoda userData colectează datele introduse de utilizator în formular și le returnează sub formă de obiect UserForRegister.

4.2.5. Bootstrap



Figura 32: Logo Bootstrap

Bootstrap este un framework front-end open-source creat de Twitter pentru dezvoltarea rapidă și facilă a interfețelor web responsive și mobile-first. Bootstrap oferă o colecție de componente HTML, CSS și JavaScript, care facilitează stilizarea și structura aplicațiilor web. În proiectul Wheels&Deals, Bootstrap este folosit pentru a asigura un design consistent și atractiv, care funcționează bine pe o varietate de dispozitive și dimensiuni de ecran.

4.2.5.1 Caracteristici esențiale ale Bootstrap

- Sistem de grilă

Una dintre cele mai puternice caracteristici ale Bootstrap este sistemul de grilă (grid system), care permite crearea layout-urilor responsive folosind o serie de coloane și rânduri. Acesta facilitează alinierea și organizarea elementelor pe pagină.

- Componente UI predefinite

Bootstrap oferă o gamă largă de componente predefinite, cum ar fi butoane, carduri, formulare, navigații și multe altele, care pot fi folosite pentru a construi rapid interfețe de utilizator atractive și funcționale.

- Formular și validare

Bootstrap oferă stiluri și componente pentru crearea de formulare elegante, cu suport pentru validare și feedback vizual.

4.2.5.2 Utilizarea Bootstrap în proiect

În cadrul aplicației Wheels&Deals, Bootstrap este utilizat pentru a stiliza diverse componente și pagini, asigurând o experiență de utilizare consistentă și plăcută. Iată câteva exemple specifice din proiect:

Formular de autentificare stilizat

Formularul de autentificare este stilizat folosind componente Bootstrap pentru a crea un aspect profesional și ușor de utilizat.

```
<div class="row">
  <div class="col-md-6 col-lg-4 m-auto">
    <div class="card shadow-sm bg-white rounded-3">
      <div class="card-header text-center bg-dark-green text-white rounded-top">
        Register
      </div>
      <div class="card-body p-2">
        <form [formGroup]="registrationForm" (ngSubmit)="onSubmit()">
          <div class="form-group mb-1">
            <label for="firstName" class="form-label text-dark">First name</label>
            <input type="text" class="form-control form-control-sm" formControlName="firstName">
            <span *ngIf="!firstName.valid && (firstName.touched || userSubmitted)" class="error-block">
              First name is required
            </span>
          </div>

          <div class="form-group mb-1">
            <label for="lastName" class="form-label text-dark">Last name</label>
            <input type="text" class="form-control form-control-sm" formControlName="lastName">
            <span *ngIf="!lastName.valid && (lastName.touched || userSubmitted)" class="error-block">
              Last name is required
            </span>
          </div>

          <div class="form-group mb-1">
            <label for="email" class="form-label text-dark">Email</label>
            <input type="text" class="form-control form-control-sm" formControlName="email">
            <span *ngIf="!email.valid && (email.touched || userSubmitted)" class="error-block">
              <span *ngIf="email.hasError('required')">Email is required</span>
              <span *ngIf="email.hasError('email')">Email is invalid</span>
            </span>
          </div>

          <div class="form-group mb-1">
            <label for="password" class="form-label text-dark">Password</label>
            <input type="password" class="form-control form-control-sm" formControlName="password">
            <span *ngIf="!password.valid && (password.touched || userSubmitted)" class="error-block">
              <span *ngIf="password.errors?.['required']">Password is required</span>
              <span *ngIf="password.errors?.['minlength']">Password must be at least 8 characters</span>
            </span>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

Figura 33: Exemplu utilizare Bootstrap pentru autentificare

Explicație

- **Container formular:** Clasa card shadow-sm bg-white rounded-3 creează un card pentru formularul de înregistrare, cu umbră și colțuri rotunjite.
- **Etichete și input-uri:** Clasele form-label și form-control form-control-sm sunt folosite pentru a stiliza etichetele și câmpurile de input.
- **Mesaje de eroare:** Span-ul pentru mesaje de eroare folosește *ngIf pentru a afișa erorile doar atunci când câmpurile sunt invalide și utilizatorul a interacționat cu ele.

Carduri pentru listarea mașinilor

Listarea mașinilor pe pagina principală este realizată folosind carduri Bootstrap, care asigură un aspect organizat și estetic.

```

<div class="card car-card">
  <div class="img-wrapper">
    <img *ngIf="car.photo" class="card-img-top" [src]="car.photo">
    
  </div>
  <div class="icons-wrapper text-center">
    <button class="btn btn-primary" routerLink="/car-detail/{{car.id}}" data-toggle="tooltip" data-placement="top" title="Vezi Detalii">
      <i class="far fa-eye"></i>
    </button>
    <button class="btn btn-primary" routerLink="/update-car/{{car.id}}" data-toggle="tooltip" data-placement="top" title="Modifica">
      <i class="fas fa-edit"></i>
    </button>
  </div>
  <div class="card-body p-1">
    <h1>{{car.brand}} {{car.model}}</h1>
    <h2>{{car.carType}}</h2>
    <h3>Price: <span class="font-highlight">{{car.price}} $</span></h3>
    <p>
      <strong>Year:</strong> {{car.year}} <strong>| KM:</strong> {{car.km}}<br>
      <strong>Engine:</strong> {{car.engine}} <strong>| Power:</strong> {{car.power}} <strong>| Fuel:</strong> {{car.fuelType}}<br>
      <strong>Is available:</strong> {{car.available}}
    </p>
    <h5>{{car.description}}</h5>
  </div>
</div>

```

Figura 34: Exemplu utilizare Bootstrap pentru listarea masinilor

Explicație

- **Container card:** Clasa card car-card stilizează cardul vehiculului.
- **Image card:** Clasa card-img-top este folosită pentru a stiliza imaginea vehiculului.
- **Buton:** Clasa btn btn-primary stilizează butoanele de acțiune pentru a vedea detalii sau a modifica informațiile vehiculului.
- **Body card:** Clasa card-body stilizează conținutul cardului, incluzând detalii despre vehicul.

4.2.5.3. Integrarea Bootstrap în proiect

Pentru a utiliza Bootstrap în proiect, fișierele CSS și JavaScript ale Bootstrap sunt incluse în proiectul Angular. Iată cum arată configurarea în fișierul angular.json:

```

"sourceRoot": "src",
"prefix": "app",
"architect": {
  "build": {
    "builder": "@angular-devkit/build-angular:application",
    "options": {
      "outputPath": "dist/licenta-client",
      "index": "src/index.html",
      "browser": "src/main.ts",
      "polyfills": [
        "zone.js"
      ],
      "tsConfig": "tsconfig.app.json",
      "assets": [
        "src/favicon.ico",
        "src/assets",
        "src/data"
      ],
      "styles": [
        "src/styles.css",
        "node_modules/bootstrap/dist/css/bootstrap.min.css"
      ],
      "scripts": [
        "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
      ]
    }
  }
}

```

Figura 35: Integrare Bootstrap in proiect

Explicație

- **Includerea CSS:** Calea către fișierul CSS Bootstrap este adăugată în secțiunea styles a fișierului angular.json, asigurând astfel că stilurile Bootstrap sunt aplicate la nivelul întregii aplicații.
- **Includerea JavaScript:** Calea către fișierul JavaScript Bootstrap este adăugată în secțiunea scripts, permițând utilizarea componentelor Bootstrap care necesită JavaScript, cum ar fi modalurile, tooltip-urile și popover-ele.

4.2.6. AlertifyJS



Figura 36: Logo AlertifyJS

AlertifyJS este o bibliotecă JavaScript pentru notificări și alerte ușor de utilizat, care ajută dezvoltatorii să creeze rapid și eficient mesaje de tip toast, alerte, confirmări și alte tipuri de notificări în aplicațiile lor. În proiectul Wheels&Deals, AlertifyJS este folosit pentru a afișa notificări de tip toast, care sunt mesaje temporare ce apar și dispar automat după un anumit interval de timp, oferind utilizatorilor feedback instantaneu despre acțiunile lor.

4.2.6.1 Caracteristici esențiale ale AlertifyJS

- **Notificări de tip toast**

Notificările de tip toast sunt mesaje discrete care apar de obicei în colțul ecranului și dispar după câteva secunde. Ele sunt ideale pentru a informa utilizatorii despre acțiuni reușite, erori sau alte evenimente care nu necesită o interacțiune imediată din partea utilizatorului.

- **Alerte și confirmări**

Pe lângă notificările de tip toast, AlertifyJS poate afișa și alerte modale sau dialoguri de confirmare, care necesită interacțiunea utilizatorului. Acestea sunt utile pentru a solicita utilizatorilor să confirme acțiuni importante sau pentru a le atrage atenția asupra unor informații critice.

4.2.6.2 Utilizarea AlertifyJS în proiect

AlertifyJS este folosit în proiectul Wheels&Deals pentru a oferi utilizatorilor feedback instantaneu și clar despre acțiunile lor în aplicație. Iată câteva exemple de utilizare:

1. Notificare de succes sign out

Când un utilizator dă cu succes sign out, apare o notificare de tip toast care confirmă acest lucru:

```
onLogout(){
  localStorage.removeItem('token');
  localStorage.removeItem('email');
  localStorage.removeItem('userId');
  this.alertify.success('You are logged out!');
}
```

Figura 37: Exemplu notificare de succes in cod

2. Notificare de eroare la schimbarea pozei principale a anutului

Dacă apare o eroare în timpul procesului de salvare, utilizatorul primește o notificare de eroare:

```
setPrimaryPhoto(carId: number, photo: Photo): void {
  this.carsService.setPrimaryPhoto(carId, photo.publicId.toString())
  this.car.photos?.forEach(p => {
    if (p.isPrimary) p.isPrimary = false;
    if (p.publicId === photo.publicId) p.isPrimary = true;
  });
  this.mainPhotoChangedEvent.emit(photo.imageUrl);
  this.alertify.success('Main photo set successfully');
}, error => {
  this.alertify.error('Failed to set main photo');
});
}
```

Figura 38: Exemplu notificare de eroare in cod

5. Descriere baza de date

5.1. Introducere

Baza de date pentru aplicația Wheels&Deals este esențială pentru gestionarea și stocarea datelor referitoare la utilizatori, mașini, tipuri de mașini, branduri, tipuri de combustibil și fotografii. Aceasta a fost realizată folosind Microsoft SQL Server Management Studio (SSMS) și a fost creată utilizând abordarea "code first" din Entity Framework, ceea ce înseamnă că structura bazei de date a fost generată automat din codul C# al aplicației. Această metodă oferă un control total asupra modelelor de date, facilitând astfel întreținerea și actualizarea bazei de date pe măsură ce cerințele aplicației se schimbă.

5.2. Crearea bazei de date

5.2.1. Abordarea "code first"

Abordarea "code first" din Entity Framework permite dezvoltatorilor să definească modelele de date și relațiile dintre ele folosind clase C#. Odată ce modelele sunt definite, Entity Framework generează automat schema bazei de date și creează tabelele corespunzătoare. Această abordare este foarte utilă deoarece permite dezvoltarea rapidă și flexibilă, iar modificările în modele pot fi gestionate ușor prin migrații.

De exemplu, clasele C# pentru entități precum Utilizatori, Mașini, Branduri și Fotografii sunt definite în cod. Relațiile dintre aceste entități, cum ar fi relația dintre Utilizatori și Mașini (unde un utilizator poate posta mai multe mașini), sunt de asemenea definite în cod. Acest lucru asigură coerența și integritatea datelor în cadrul aplicației.

5.3. Migrațiile

Migrațiile sunt folosite pentru a aplica modificările structurale ale bazei de date fără a pierde datele existente. În Wheels&Deals, migrațiile sunt generate și aplicate pentru a crea tabelele necesare și a modifica schema bazei de date pe măsură ce aplicația evoluează.

Entity Framework oferă comenzi care facilitează crearea și aplicarea migrațiilor. De exemplu, comanda Add-Migration este folosită pentru a genera o nouă migrație care conține modificările aduse modelelor de date. Comanda Update-Database aplică aceste modificări bazei de date. Aceste comenzi asigură că schema bazei de date este mereu aliniată cu modelele de date din codul aplicației.

5.4. Tabele și relații

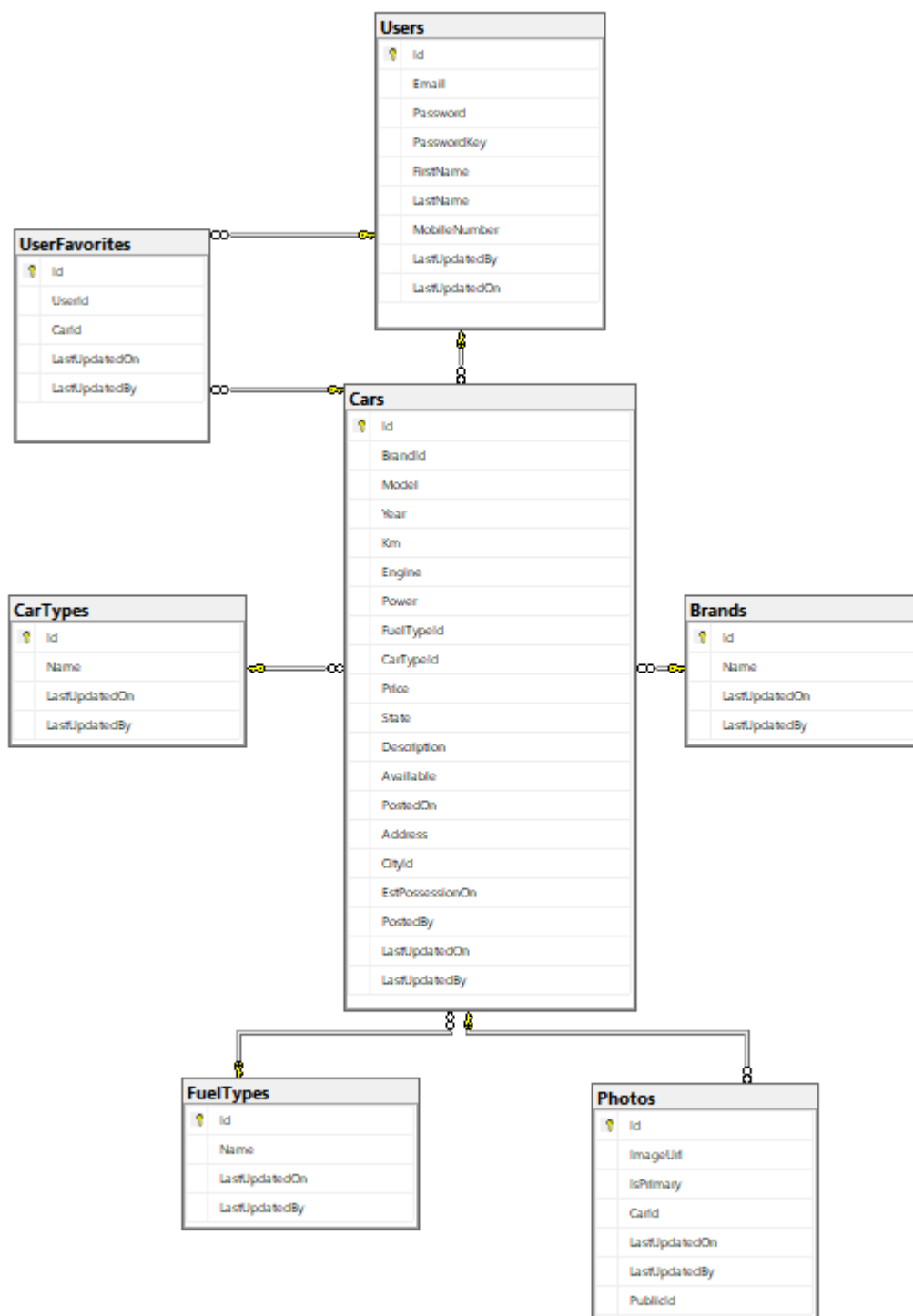


Figura 39: Diagrama bazei de date

Baza de date conține următoarele tabele:

5.4.1. Tabelul Users

Acest tabel stochează informații despre utilizatori, cum ar fi adresa de email, parola, numele și numărul de telefon.

- **Id:** Cheie primară
- **Email:** Adresa de email a utilizatorului
- **Password:** Parola utilizatorului
- **PasswordKey:** Cheie pentru securizarea parolei
- **FirstName:** Prenumele utilizatorului
- **LastName:** Numele de familie al utilizatorului
- **MobileNumber:** Numărul de telefon mobil
- **LastUpdatedBy:** Ultima persoană care a actualizat înregistrarea
- **LastUpdatedOn:** Data ultimei actualizări

5.4.2. Tabelul Cars

Acest tabel stochează informații despre mașini, inclusiv modelul, anul, kilometrajul, motorul, puterea, prețul și descrierea.

- **Id:** Cheie primară
- **BrandId:** Cheie străină către tabelul Brands
- **Model:** Modelul mașinii
- **Year:** Anul fabricării
- **Km:** Kilometrajul
- **Engine:** Tipul de motor
- **Power:** Puterea motorului
- **FuelTypeId:** Cheie străină către tabelul FuelTypes
- **CarTypeId:** Cheie străină către tabelul CarTypes
- **Price:** Prețul mașinii
- **State:** Starea mașinii
- **Description:** Descrierea mașinii
- **Available:** Disponibilitatea mașinii
- **PostedOn:** Data postării anunțului
- **Address:** Adresa unde se află mașina
- **CityId:** Cheie străină către tabelul Cities
- **EstPossessionOn:** Data estimată de posesie
- **PostedBy:** Cheie străină către tabelul Users
- **LastUpdatedBy:** Ultima persoană care a actualizat înregistrarea
- **LastUpdatedOn:** Data ultimei actualizări

5.4.3. Tabelul CarTypes

Acest tabel stochează diferitele tipuri de mașini, cum ar fi sedan, SUV, hatchback etc.

- **Id:** Cheie primară
- **Name:** Numele tipului de mașină
- **LastUpdatedBy:** Ultima persoană care a actualizat înregistrarea
- **LastUpdatedOn:** Data ultimei actualizări

5.4.4. Tabelul Brands

Acest tabel stochează brandurile mașinilor, cum ar fi BMW, Audi, Mercedes etc.

- **Id:** Cheie primară
- **Name:** Numele brandului
- **LastUpdatedBy:** Ultima persoană care a actualizat înregistrarea
- **LastUpdatedOn:** Data ultimei actualizări

5.4.5. Tabelul FuelTypes

Acest tabel stochează tipurile de combustibil, cum ar fi benzină, motorină, electric etc.

- **Id:** Cheie primară
- **Name:** Numele tipului de combustibil
- **LastUpdatedBy:** Ultima persoană care a actualizat înregistrarea
- **LastUpdatedOn:** Data ultimei actualizări

5.4.6. Tabelul Photos

Acest tabel stochează informații despre fotografiile mașinilor.

- **Id:** Cheie primară
- **ImageUrl:** URL-ul imaginii
- **IsPrimary:** Indică dacă fotografia este principală
- **CarId:** Cheie străină către tabelul Cars
- **LastUpdatedBy:** Ultima persoană care a actualizat înregistrarea
- **LastUpdatedOn:** Data ultimei actualizări
- **PublicId:** ID-ul public al fotografiei stocate în Cloudinary

5.4.7. Tabelul UserFavorites

Acest tabel stochează informații despre mașinile favorite ale utilizatorilor, creând o relație many-to-many între utilizatori și mașini.

- **Id:** Cheie primară

- **UserId:** Cheie străină către tabelul Users
- **CarId:** Cheie străină către tabelul Cars
- **LastUpdatedBy:** Ultima persoană care a actualizat înregistrarea
- **LastUpdatedOn:** Data ultimei actualizări

5.5. Relațiile dintre tabele

Relațiile dintre tabele sunt esențiale pentru integritatea datelor și pentru a asigura că informațiile sunt stocate eficient și coerent. În baza de date Wheels&Deals, există relații de tip one-to-many și many-to-one între tabele, după cum urmează:

- **Users și Cars:** Un utilizator poate posta mai multe mașini (one-to-many).
- **Cars și CarTypes:** O mașină are un singur tip, dar un tip de mașină poate fi asociat cu mai multe mașini (many-to-one).
- **Cars și Brands:** O mașină aparține unui singur brand, dar un brand poate avea mai multe mașini (many-to-one).
- **Cars și FuelTypes:** O mașină folosește un singur tip de combustibil, dar un tip de combustibil poate fi folosit de mai multe mașini (many-to-one).
- **Cars și Photos:** O mașină poate avea mai multe fotografii (one-to-many).
- **Users și UserFavorites:** Un utilizator poate avea mai multe mașini favorite, iar o mașină poate fi favorită pentru mai mulți utilizatori (many-to-many).

6. Platforme folosite in dezvoltarea aplicatiei

6.1 Visual Studio 2022



Figura 40: Logo Visual Studio 2022

Visual Studio 2022 este un mediu de dezvoltare integrat (IDE) dezvoltat de Microsoft, care oferă dezvoltatorilor un set cuprinzător de instrumente pentru a crea aplicații pentru diverse platforme, inclusiv web, desktop și mobile. Este cunoscut pentru capacitățile sale avansate și interfața prietenoasă, care fac dezvoltarea software mai eficientă și mai plăcută.

6.1.1 Caracteristici principale ale Visual Studio 2022

- **Performanță îmbunătățită**

Visual Studio 2022 este optimizat pentru a rula mai rapid și mai eficient, ceea ce ajută la reducerea timpului de așteptare pentru compilarea și rularea aplicațiilor. Acest lucru este esențial pentru dezvoltatori, deoarece permite un flux de lucru mai lin și mai productiv.

- **Interfață prietenoasă**

Interfața utilizator este intuitivă și bine organizată, facilitând accesul rapid la diverse instrumente și funcționalități. Editorul de cod este dotat cu funcții avansate de completare automată, evidențierea sintaxei și refactorizare, toate menite să ușureze munca dezvoltatorilor.

- **Support pentru multiple limbaje de programare**

Visual Studio 2022 suportă o gamă largă de limbaje de programare, inclusiv C#, Visual Basic, C++, Python, JavaScript și multe altele. Aceasta face din Visual Studio un instrument versatil, potrivit pentru diverse tipuri de proiecte.

- **Instrumente de colaborare**

Integrarea cu Git și GitHub permite gestionarea ușoară a codului sursă și colaborarea eficientă în echipe. Visual Studio oferă funcționalități avansate pentru controlul versiunilor, precum și instrumente pentru revizuirea codului și gestionarea cererilor de tragere (pull requests).

- **Debugging și diagnosticare avansată**

Visual Studio 2022 vine cu un set robust de instrumente pentru debugging și diagnosticare, care ajută la identificarea și rezolvarea rapidă a erorilor din cod. Funcții precum puncte de întrerupere (breakpoints), inspecția variabilelor și urmărirea execuției codului sunt esențiale pentru dezvoltatori.

- **Support pentru dezvoltarea de aplicații web și mobile**

IDE-ul oferă suport extins pentru dezvoltarea de aplicații web cu ajutorul ASP.NET, precum și pentru aplicații mobile utilizând Xamarin. Acest lucru permite dezvoltatorilor să creeze aplicații pentru diverse platforme dintr-un singur mediu de dezvoltare.

- **Extensibilitate**

Visual Studio 2022 este foarte extensibil, permițând adăugarea de plugin-uri și extensii care să îmbunătățească funcționalitățile existente sau să adauge altele noi. Magazinul de extensii Visual Studio oferă o varietate de plugin-uri pentru diferite nevoi și preferințe ale dezvoltatorilor.

6.2 Visual Studio Code



Figura 41: Logo visual Studio Code

Visual Studio Code (VS Code) a fost folosit ca editor principal pentru dezvoltarea front-end a aplicației Wheels&Deals. Este un editor de cod sursă deschis și gratuit, creat de Microsoft, care oferă o gamă largă de caracteristici ce îmbunătățesc productivitatea dezvoltatorilor.

6.2.1 Caracteristici principale ale Visual Studio Code

- **Editare de cod ușoară și eficientă**

Autocompletare inteligentă: VS Code oferă funcționalități de completare automată a codului, care sugerează posibile completări în timp ce scrii, economisind astfel timp și reducând erorile.

Sintaxă evidențiată: Codul este mai ușor de citit și de înțeles datorită evidențierii sintaxei, care colorează diferit variabilele, funcțiile, clasele etc.

- **Extensii și pluginuri**

VS Code suportă o gamă largă de extensii care pot fi instalate pentru a adăuga funcționalități suplimentare. În proiectul Wheels&Deals, extensiile pentru Angular, TypeScript și CSS au fost deosebit de utile.

Angular Language Service: Extensia aceasta ajută la scrierea codului Angular, oferind completare automată și informații despre erori.

Prettier - Code formatter: Aceasta ajută la menținerea unui stil consistent al codului prin formatarea automată a acestuia conform unor reguli predefinite.

- **Debugging integrat**

VS Code oferă instrumente de debugging care permit rularea codului și inspectarea variabilelor direct din editor. Punctele de întrerupere și monitorizarea variabilelor facilitează depistarea și rezolvarea rapidă a erorilor.

- **Controlul versiunilor**

VS Code are integrare nativă cu Git, permițând gestionarea eficientă a versiunilor codului. Aceasta include funcționalități de commit, push, pull și vizualizare a diferențelor dintre versiunile codului.

- **Terminal integrat**

Editorul include un terminal integrat, astfel încât nu trebuie să părăsești editorul pentru a rula comenzi de linie de comandă. Acest lucru este util pentru instalarea pachetelor, rularea serverelor de dezvoltare și executarea altor comenzi necesare.

- **Live Server**

Extensia Live Server permite rularea unui server local care încarcă automat pagina web în browser de fiecare dată când salvezi modificările în cod. Acest lucru este esențial pentru dezvoltarea și testarea rapidă a aplicației.

- **Personalizare**

VS Code permite personalizarea completă a interfeței și a funcționalităților. Poți alege teme, configura scurtături de tastatură și seta preferințele editorului pentru a se potrivi stilului tău de lucru.

6.3 Git



Figura 42: Logo Git

Git este un sistem de control al versiunilor distribuit, folosit pentru a urmări modificările în codul sursă pe parcursul dezvoltării software. Este esențial pentru colaborarea eficientă între dezvoltatori și pentru menținerea unui istoric detaliat al tuturor

modificărilor efectuate în proiect. În cadrul dezvoltării aplicației Wheels&Deals, Git a fost utilizat pentru a gestiona întregul proces de dezvoltare, de la scrierea codului inițial până la implementarea și întreținerea ulterioară.

6.3.1 Caracteristici esențiale ale Git

- **Versionare distribuie**

Git permite fiecărui dezvoltator să aibă o copie completă a repository-ului, inclusiv istoricul complet al modificărilor. Aceasta înseamnă că dezvoltatorii pot lucra offline și pot sincroniza modificările cu repository-ul principal atunci când sunt online.

- **Commit-uri și istoric al modificărilor**

Commit-urile sunt puncte de salvare a modificărilor în proiect. Fiecare commit are un identificator unic și păstrează un mesaj descriptiv despre modificările efectuate. Aceasta ajută la menținerea unui istoric detaliat și clar al dezvoltării proiectului.

- **Branching și merging**

Branch-urile permit dezvoltatorilor să lucreze pe diferite funcționalități sau corecții de bug-uri în mod izolat. După finalizarea lucrului pe o branchă, aceasta poate fi îmbinată (merged) în branch-ul principal, integrând modificările într-un mod controlat și organizat.

- **Managementul colaborării**

Git facilitează colaborarea între dezvoltatori prin intermediul repository-urilor remote, cum ar fi cele găzduite pe platforme precum GitHub, GitLab sau Bitbucket. Aceste platforme oferă funcționalități suplimentare, cum ar fi pull requests, code reviews și integrarea continuă.

- **Resolvarea conflictelor**

În cazul în care doi dezvoltatori modifică aceleași fișiere, Git poate detecta conflictele și oferă instrumente pentru a le rezolva. Aceasta asigură integrarea corectă a modificărilor și evitarea suprascrierii accidentale a codului.

- **Revert și reset**

Git permite revenirea la stări anterioare ale codului în cazul în care apar probleme. Funcționalitățile de revert și reset sunt utile pentru a anula modificările care cauzează erori sau pentru a restaura codul la o versiune stabilă.

6.3.2 Beneficiile utilizării Git

- **Organizare și claritate:** Commit-urile frecvente și mesajele descriptive au asigurat un istoric clar și detaliat al dezvoltării.
- **Flexibilitate în dezvoltare:** Branch-urile au permis dezvoltarea paralelă a mai multor funcționalități, reducând riscul de conflicte și facilitând testarea independentă.
- **Colaborare eficientă:** Integrarea cu GitHub a facilitat colaborarea și revizuirea codului, asigurând calitatea și consistența proiectului.
- **Siguranță și recuperare:** Backup-ul automat al repository-ului remote a oferit o siguranță suplimentară în caz de pierdere a datelor locale și a permis revenirea rapidă la versiuni stabile ale codului.

6.4 Microsoft SQL Server Management Studio (SSMS)



Figura 43: Logo Microsoft SQL Server Management Studio

Microsoft SQL Server Management Studio (SSMS) este un mediu de dezvoltare integrat (IDE) utilizat pentru gestionarea și administrarea instanțelor de SQL Server. Este un instrument esențial pentru administratori de baze de date (DBA) și dezvoltatori, oferind un set complet de funcționalități pentru a crea, configura, gestiona și administra baze de date SQL Server.

6.4.1 Caracteristici esențiale ale SSMS

1. Interfață grafică prietenoasă

SSMS oferă o interfață grafică intuitivă care facilitează interacțiunea cu bazele de date. Aceasta include ferestre multiple pentru explorarea obiectelor din bazele de date, editori de scripturi SQL și panouri pentru vizualizarea rezultatelor interogărilor.

2. Editor de scripturi SQL

SSMS vine cu un editor de scripturi SQL puternic care permite scrierea, modificarea și executarea interogărilor SQL. Editorul oferă caracteristici precum colorarea sintaxei, completarea automată a codului și sugestii de sintaxă, ceea ce îmbunătățește productivitatea și acuratețea.

3. Managementul obiectelor de bază de date

Cu SSMS, utilizatorii pot crea, modifica și șterge obiecte de bază de date, cum ar fi tabele, proceduri stocate, vizualizări, funcții și declanșatoare (triggers). Toate aceste operațiuni se pot face fie prin interfața grafică, fie prin scripturi SQL.

4. Instrumente de administrare

SSMS include instrumente avansate pentru administrarea serverelor SQL. Acestea includ monitorizarea performanței serverului, configurarea securității, backup și restaurare, și managementul utilizatorilor și permisiunilor.

5. Interogări și analiză de date

Utilizatorii pot executa interogări complexe și pot analiza datele direct din SSMS. Panoul de rezultate permite vizualizarea și exportarea datelor în diferite formate, facilitând analiza și raportarea.

6. Managementul conexiunilor

SSMS permite gestionarea ușoară a conexiunilor la multiple instanțe de SQL Server. Utilizatorii pot salva și organiza conexiunile pentru a accesa rapid serverele de care au nevoie.

7. Ghidul aplicatiei

- Pagina principala.

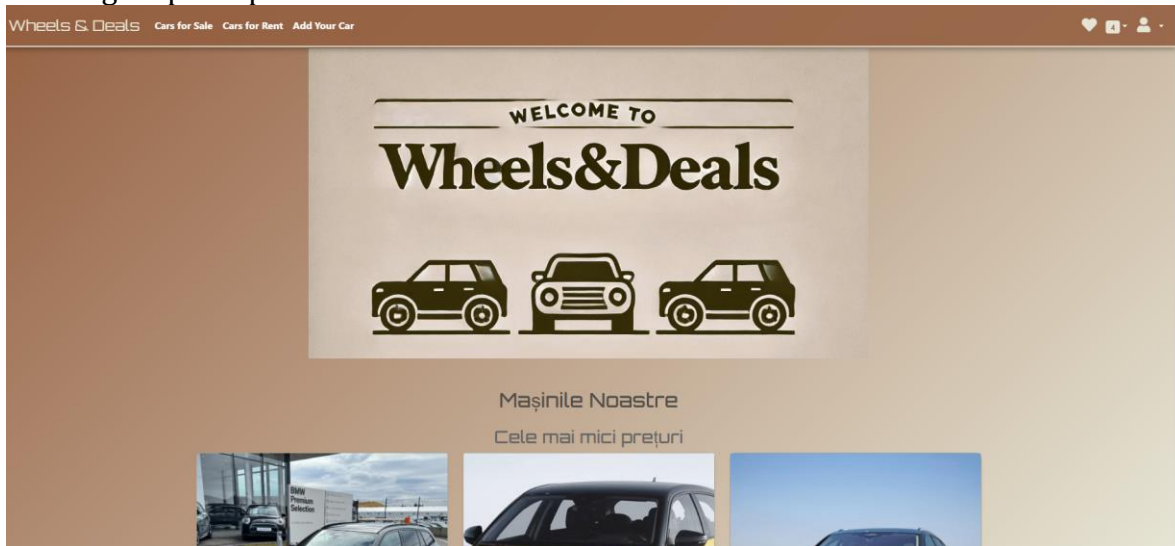


Figura 44: Pagina principala a aplicatiei

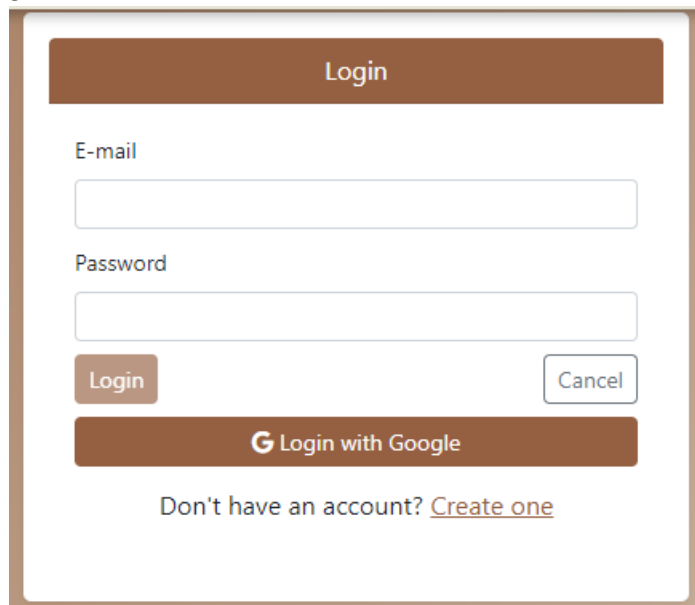
Utilizatorul este intampinat cu un mesaj de bun venit, daca gliseaza in jos vede cateva dintre cele mai ieftine masini sau masinile cu cele mai puternice motoare. In partea de sus pe bara de navigare poate alege sa cumpere, inchirieze, sa isi posteze propria masina pe site, sa vizioneze masinile adaugate la favorite sau sa deschida meniul de utilizator.

- Inregistrare

Figura 45: Pagina de inregistrare

Pentru a utiliza functionalitatile aplicatiei, utilizatorul trebuie sa se inregistreze. Se solicita informatii de baza precum numele, adresa de email, parola si numarul de telefon. Mesajul de sub fiecare camp apare daca acesta nu a fost completat.

- **Autentificare**

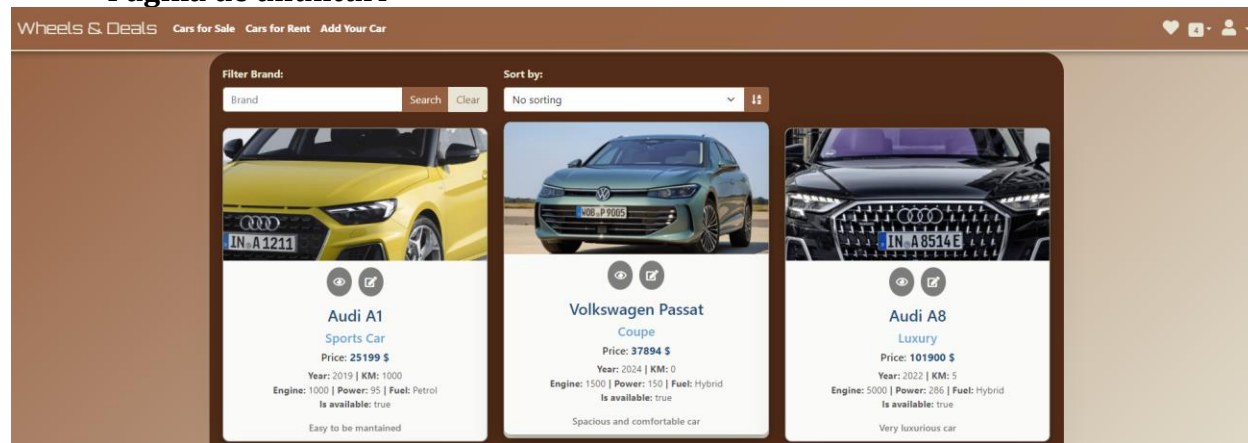


The login form is titled "Login" and is contained within a light brown border. It features two input fields: "E-mail" and "Password". Below the "Password" field is a "Login" button. To the right of the "Login" button is a "Cancel" button. Below these buttons is a large brown button with the Google logo and the text "Login with Google". At the bottom of the form, there is a link that says "Don't have an account? [Create one](#)".

Figura 46: Pagina de autentificare

Dupa inregistrare, utilizatorul se poate autentifica pentru a-si accesa contul. Optiunea de autentificare prin google este de asemenea disponibila.

- **Pagina de anunturi**



The car listings page has a header with the text "Wheels & Deals" and navigation links: "Cars for Sale", "Cars for Rent", and "Add Your Car". On the right side of the header are icons for a heart, a camera, and a user profile. Below the header is a filter section with "Filter Brand:" and a search bar containing "Brand", "Search", and "Clear" buttons. To the right of the search bar is a "Sort by:" dropdown menu showing "No sorting". The main content area displays three car listings, each with a car image, a title, a price, and detailed specifications.

Brand	Model	Price	Year	Engine	Power	Fuel	Availability	Notes
Audi	A1 Sports Car	25199 \$	2019	1000	95	Petrol	true	Easy to be maintained
Volkswagen	Passat Coupe	37894 \$	2024	1500	150	Hybrid	true	Spacious and comfortable car
Audi	A8 Luxury	101900 \$	2022	5000	286	Hybrid	true	Very luxurious car

Figura 47: Pagina de anunturi

Utilizatorul poate filtra masinile dupa brand sau sorta dupa pret, motor, cai putere sau alte criterii. Fiecare masina afisata are informatii despre pret, model, tip de combustibil, putere si disponibilitate.

- **Adaugarea unei masini**

Wheels & Deals Cars for Sale Cars for Rent Add Your Car

Basic Info Detailed Info Other Details Photos

Brand
Dacia

Model
Logan

Car Type
Hybrid

Year
2024

Price (\$ total for sale / per day for rent)
40

Next

Preview

Logan

Price: 40 \$ per day

Year: 2024 | KM: 10

Engine: 1000 | Power: 75 | Fuel: Is available: true

Good car

Figura 48: Pagina pentru adaugarea unei masini

Utilizatorul poate adauga o masina pentru vanzare sau inchiriere completand un formular cu informatii necesare inclusiv brand, model, tipul masinii, an de fabricatie, pretul, puterea, kilometrii, poze etc.

- **Detalii masina**

Wheels & Deals Cars for Sale Cars for Rent Add Your Car Register Login

Overview Address Photos Contact

Car brand: Audi
Car model: A8
Car type: Luxury
Year: 2022
Description: Very luxurious car

Price: \$101900
Km: 5
Engine: 5000
Power: 286
Fuel: Hybrid
Status: Available and ready to go

Audi A8
Year: 2022
Price: \$101900
Car Type: Luxury
Fuel Type: Hybrid
State: For sale

Add to favourites

Figura 49: Pagina pentru detaliile unei masini

Pagina pentru detaliile unei masini afiseaza informatii detaliate despre vehicul, inclusiv descrierea starea masinii pretul si tipul de combustibil. Utilizatorul poate adauga masina la favorite pentru a o urmari ulterior.

- **Modificarea unei masini**

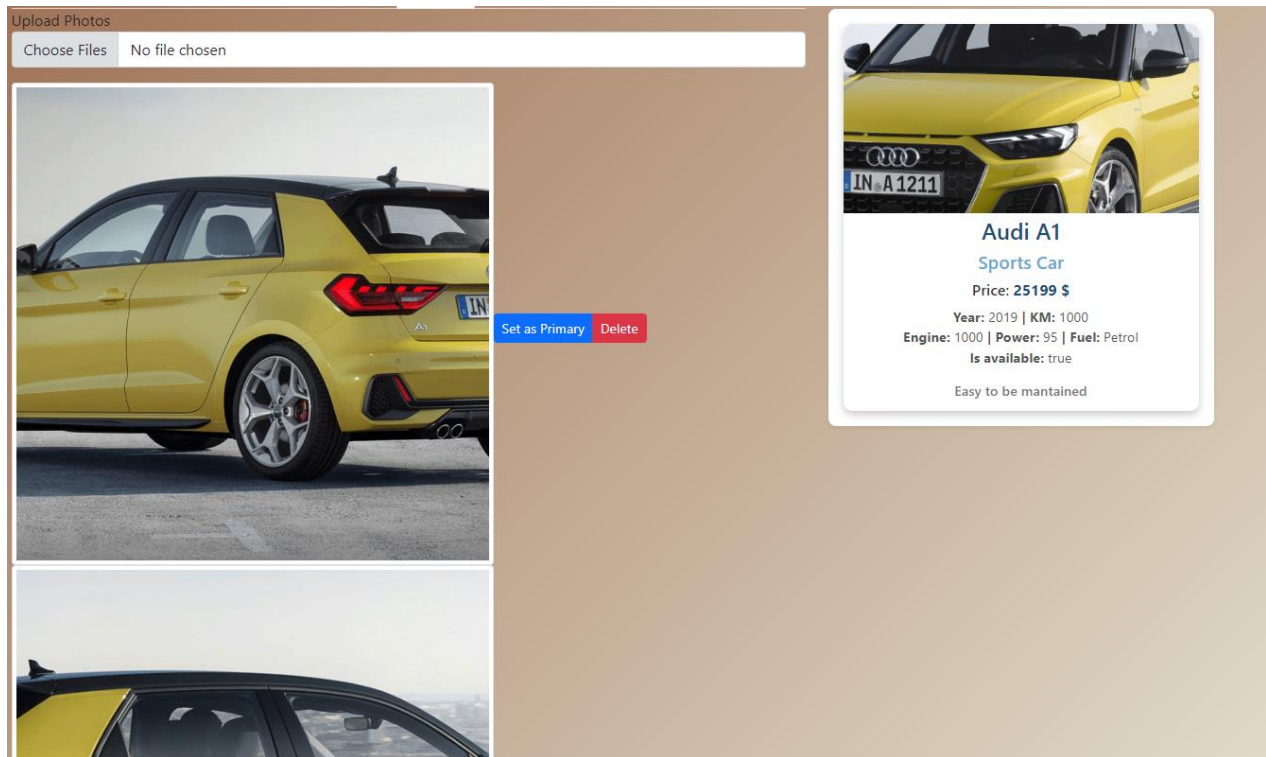


Figura 50: Pagina pentru editarea unei masini

Daca un utilizator doreste sa modifice informatiile despre o masina pe care a postat-o, poate face acest lucru prin formularul de editare. Aici poate actualiza detaliile si adauga sau sterge fotografii.

- **Meniul de utilizator**

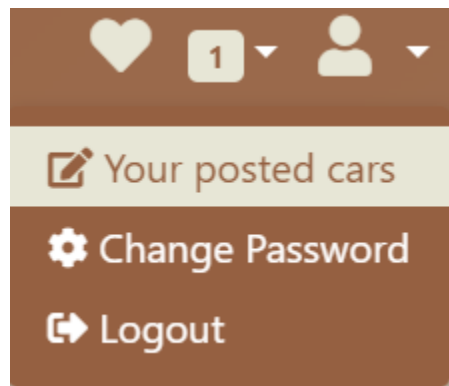


Figura 51: Meniul de utilizator

Din meniul, utilizatorul poate vedea masinile postate de acesta, poate sa-si schimbe parola contului sau se poate deloga.

- **Favorite**

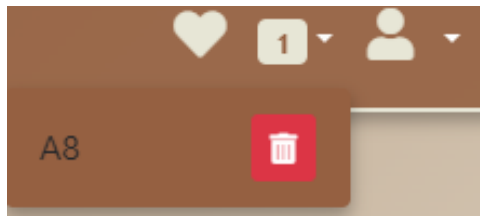


Figura 52: Pagina favorite

Utilizatorul poate vedea ce si cate masini a adaugat la favorite, apasand pe modelul masinii este redirectionat catre pagina de detalii a postarii respective

- **Profil de comerciant**

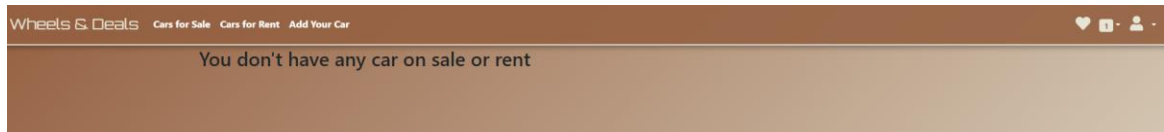


Figura 53: Profil comerciant

Accesand "Your posted cars" user-ul isi poate vedea toate anunturile si intra pe acestea sa le editeze, daca nu are anunturi postate este instiintat printr-un mesaj

- **Schimbarea parolei**

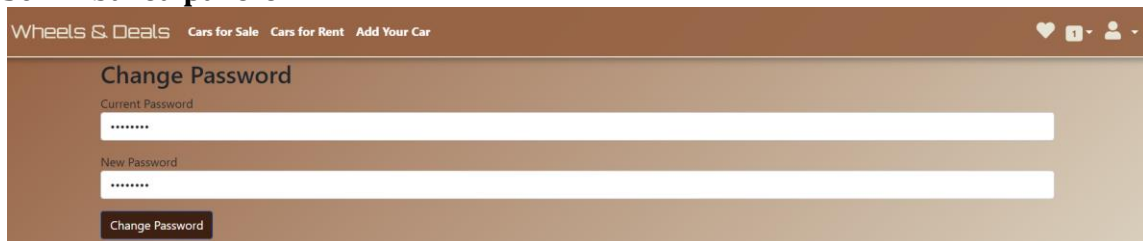


Figura 54: Fereastra schimbare parola

Accesand "Change Password" utilizatorul este redirectionat spre formularul de schimbare a parolei

8. Concluzii

Wheels&Deals este o aplicație modernă care transformă modul în care tranzacționăm mașini online. De la vânzare și cumpărare, până la închirierea de mașini, această platformă oferă utilizatorilor o soluție eficientă și sigură. Construită cu tehnologii de ultimă generație și bazată pe framework-uri robuste, Wheels&Deals se asigură că fiecare tranzacție este fluidă și fără complicații.

În ceea ce privește dezvoltarea aplicației, am adoptat o abordare agilă pentru a răspunde rapid cerințelor și nevoilor utilizatorilor. Am început cu analiza detaliată a cerințelor, urmată de proiectarea unei arhitecturi modulare care permite o dezvoltare și întreținere facilă. Fiecare modul al aplicației a fost implementat și testat riguros pentru a garanta calitatea și funcționalitatea sa.

Din punct de vedere tehnologic, Wheels&Deals utilizează C# și .NET 8 pentru a crea un backend puternic și eficient. AutoMapper simplifică maparea obiectelor, iar Cloudinary asigură stocarea fiabilă a fotografiilor. Pentru securitate, aplicația folosește JWT pentru autentificare și autorizare, garantând protecția datelor utilizatorilor. Entity Framework Core facilitează gestionarea bazelor de date, asigurând o interacțiune optimă cu Microsoft SQL Server Management Studio. În plus, Swagger este integrat pentru a documenta și testa API-urile, oferind o experiență de dezvoltare îmbunătățită.

Frontend-ul aplicației este construit cu Angular și TypeScript, oferind o interfață de utilizare dinamică și interactivă. HTML și CSS, împreună cu Bootstrap, asigură un design responsive și estetic. Biblioteci precum Ng2-file-upload facilitează încărcarea fișierelor, în timp ce AlertifyJS oferă notificări ușor de folosit pentru o experiență de utilizare îmbunătățită.

Pentru gestionarea codului și colaborare, am utilizat Visual Studio 2022 și Visual Studio Code. Git asigură controlul versiunilor, permițându-ne să urmărim și să gestionăm modificările codului eficient. Microsoft SQL Server Management Studio (SSMS) a fost instrumentul principal pentru gestionarea bazelor de date, facilitând migrațiile și administrarea datelor.

Aplicația Wheels&Deals se distinge printr-o interfață prietenoasă și intuitivă, care face navigarea și utilizarea simplă pentru orice utilizator, indiferent de nivelul său de experiență tehnică. Fie că este vorba de înregistrare, autentificare, postarea de anunțuri sau căutarea de mașini, toate funcționalitățile sunt accesibile și ușor de folosit.

În concluzie, Wheels&Deals reprezintă o soluție completă și inovatoare pentru tranzacțiile auto online. Prin combinarea tehnologiilor moderne cu o dezvoltare atentă și orientată către utilizator, aplicația reușește să răspundă eficient cerințelor pieței auto. Fiecare aspect al aplicației, de la backend până la frontend, a fost construit pentru a oferi o experiență optimă, sigură și plăcută. Wheels&Deals nu doar facilitează tranzacțiile auto, ci le transformă într-un proces simplu și lipsit de stres, redefinind astfel comerțul auto online.

În viitor, cu o echipă dedicată și tehnologie de vârf, Wheels&Deals este gata să devină liderul platformelor de tranzacții auto online, urmând să ofere utilizatorilor săi posibilitatea de a plăti online și să li se livreze produsul la domiciliu, posibilitatea de a-și promova anunțurile pentru a ieși în evidență, posibilitatea de calendar de închiriere a mașinilor, programând și plătind în avans pentru când vor avea nevoie de mașină.

Bibliografie

1. Wikipedia. "C Sharp (programming language)". [online] Available at: [https://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))
2. AutoMapper, "Getting started". [online] Available at: <https://docs.automapper.org/en/stable/Getting-started.html#what-is-automapper>
3. Wikipedia. "Cloudinary". [online] Available at: <https://en.wikipedia.org/wiki/Cloudinary>
4. JWT, "Introduction to Json Web Tokens". [online] Available at: <https://jwt.io/introduction/>
5. Apidog, "What is Swagger". [online] Available at: <https://apidog.com/articles/what-is-swagger/>
6. David's Blog. "A Comprehensive Guide to Newtonsoft.Json". [online] Available at: https://friendlyuser.github.io/posts/tech/2023/A_Comprehensive_Guide_to_Newtonsoft.Json/
7. JetBrains Rider. "Consume NuGet packages". [online] Available at: https://www.jetbrains.com/help/rider/Using_NuGet.html
8. Mdn web docs. "HTML:HyperText Markup Language". [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/HTML>
9. Mdn web docs. "CSS:Cascading Style Sheets". [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/CSS>
10. FreeCodeCamp. "Learn TypeScript". [online] Available at: <https://www.freecodecamp.org/news/learn-typescript-beginners-guide/>
11. FreeCodeCamp. "The Bootstrap Hndbook". [online] Available at: <https://www.freecodecamp.org/news/bootstrap-4-everything-you-need-to-know-c750991f6784/>
12. Microsoft. "What is Visual Studio? ". [online] Available at: <https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022>
13. Visual Studio Code. "Getting Started". [online] Available at: <https://code.visualstudio.com/Docs/editor/whyvscode>
14. Git. "-distributed-is-the-new-centralized". [online] Available at: <https://git-scm.com/about>
15. Wikipedia. "Microsoft SQL Server". [online] Available at: https://en.wikipedia.org/wiki/Microsoft_SQL_Server
16. Angular. "What is Angular? ". [online] Available at: <https://angular.dev/overview>
17. Microsoft. "Entity Framework documentation hub". [online] Available at: <https://learn.microsoft.com/en-us/ef/>
18. Microsoft. ".NET documentation". [online] Available at: <https://learn.microsoft.com/en-us/dotnet/>