

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IASI

**FACULTATEA DE INFORMATICA**



LUCRARE DE LICENTA

**Dogo - Pets: Walk & Care**

propusă de

**Leonard Rumeghea**

Sesiunea: iunie, 2023

Coordonator științific

**Lect. Dr. Frasinaru Cristian**

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IASI

**FACULTATEA DE INFORMATICA**

**Dogo - Pets: Walk & Care**

**Leonard Rumeghea**

**Sesiunea: iunie, 2023**

Coordonator științific

**Lect. Dr. Frasinaru Cristian**

Avizat,

Îndrumător lucrare de licență,

Lect. Dr. Frasinaru Cristian.

Data: .....

Semnătura: .....

## **Declarație privind originalitatea conținutului lucrării de licență**

Subsemnatul **Rumeghea Leonard** domiciliat în **România, jud. Botoșani, mun. Darabani, Str. Văii, nr. 51**, născut la data de **20 februarie 2001**, identificat prin CNP **5010220071367**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2023, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Dogo - Pets: Walk & Care** elaborată sub îndrumarea domnului **Lect. Dr. Frasinaru Cristian**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consumând inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: .....

Semnătura: .....

## **Declarație de consumământ**

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Dogo - Pets: Walk & Care**, codul sursă al programelor și celealte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Leonard Rumeghea**

Data: .....

Semnătura: .....

# Cuprins

<b>Motivație</b>	<b>2</b>
<b>Introducere</b>	<b>3</b>
<b>1 Specificații Funcționale</b>	<b>5</b>
1.1 Descrierea problemei . . . . .	5
1.2 Cerințe funcționale . . . . .	6
1.2.1 Cerințe funcționale pentru „Owners” . . . . .	6
1.2.2 Cerințe funcționale pentru „Walkers” . . . . .	7
1.3 Cerințe non-funcționale . . . . .	8
1.3.1 Performanță . . . . .	9
1.3.2 Scalabilitate . . . . .	9
<b>2 Arhitectura aplicației</b>	<b>10</b>
2.1 Arhitectura serverului . . . . .	10
2.1.1 Stratul de bază . . . . .	11
2.1.2 Stratul de aplicație . . . . .	12
2.1.3 Stratul de infrastructură . . . . .	13
2.1.4 Stratul de interfață . . . . .	13
2.2 Arhitectura aplicației mobile . . . . .	14
2.2.1 Model-View-Presenter (MVP) . . . . .	14
2.2.2 Componentele arhitecturii MVP . . . . .	15
2.3 Arhitectura bazei de date . . . . .	15
2.3.1 Avantajele aduse de SQL Server . . . . .	16
2.3.2 Diagrama bazei de date . . . . .	17
<b>3 Implementare</b>	<b>18</b>
3.1 Servicii externe . . . . .	18

3.1.1	Google Maps Platform . . . . .	19
3.1.2	Geolocator . . . . .	20
3.2	Generarea unei agende . . . . .	20
<b>4</b>	<b>Prezentarea aplicatiei</b>	<b>21</b>
4.1	Meniurile Owner-ului . . . . .	22
4.1.1	Manage Your Pets . . . . .	22
4.1.2	Appointments . . . . .	23
4.1.3	History & Reports . . . . .	24
4.2	Meniurile Walker-ului . . . . .	24
4.2.1	Search Appointment . . . . .	25
4.2.2	Your Preferences . . . . .	26
4.2.3	Your Agenda . . . . .	27
<b>Concluzii</b>		<b>29</b>
<b>Bibliografie</b>		<b>30</b>

# Motivație

În societatea modernă, animalele de companie joacă un rol din ce în ce mai important în viața oamenilor de zi cu zi. Cercetările recente evidențiază multiplele beneficii aduse de acestea omului, atât de natură fizică cât și psihică[1]. Printre cele fizice se enumeră: creșterea imunității, reducerea alergiilor, îmbunătățirea sănătății cardiovasculare și activitatea fizică zilnică. Cele mai importante beneficii sunt însă de natură psihologică, iar animalele de companie sunt o modalitate excelentă de a reduce stresul cât și anxietatea provocate de rutină omului modern cât și pentru îmbunătățirea stării de spirit și creșterea a sentimentului general de bunăstare. Prin mângâierea unui câine sau a unei pisici, în organism se produc o serie de modificări fizice, prin creșterea nivelului de serotonină, numită și „hormonul fericirii”, dar și prin scăderea nivelului de cortizol, numit și „hormonul stresului”.

Cu toate acestea mulți oameni consideră că nu au sau nu pot avea destul timp la dispoziție pentru a avea grija de un animal de companie și de nevoile acestuia, cum ar fi plimbările zilnice sau vizitele periodice la salon sau la veterinar. Pe lângă acestea mai apare și o problemă în momentul în care persoane doresc să plece mai multe zilele consecutive de acasă. Aceste probleme ar putea fi rezolvate totuși folosind o aplicație specializată pe nevoile „programate” ale animalului. Această aplicație este un intermediar dintre cele 2 persoane: una fiind detinătorul animalului de companie, cel care nu are timpul necesar pentru a se ocupa de o activitate, sau are nevoie de ajutor, iar celălalt este reprezentată persoană, iubitoare de animale, care doresc să se îl ajute atât de proprietar cât și pe animal.

# Introducere

Animalele de companie joacă un rol important în viața oamenilor de zi cu zi și oferă multe beneficii mentale, fizice și sociale. Acești însotitori devin adesea parte din familie, oferind dragoste necondiționată, companie și sprijin moral. Interacțiunea cu animalele de companie reduce stresul și anxietatea, îmbunătățește starea de spirit și contribuie la bunăstarea generală.

Activitățile legate de îngrijirea animalelor de companie, cum ar fi plimbările, vizitele la salon și la veterinar, sunt esențiale pentru menținerea sănătății și bunăstării animalului dvs. de companie dar adesea sunt dificil de gestionat în rutina zilnică. De exemplu, timpul în care vă plimbați câinele depinde de mărimea și nivelul de energie, dar în general sunt recomandate între 30 și 60 de minute de plimbare în fiecare zi. Vizitele la salon pot varia în funcție de nevoile fiecarui animal de companie și complexitatea serviciu, cum ar fi tunsul, spălatul sau îngrijirea blănii. Vizitele la veterinar sunt recomandate cel puțin o dată pe an pentru a verifica starea de sănătate a animalului de companie și pentru a preveni eventualele boli.

În aceste situații a avea propria la îndemână plicătie de îngrijire a animalelor de companie are multe beneficii. Poate oferi programe pentru plimbări, saloane, vizite la veterinar și multe altele, ajutându-i pe proprietari să-și gestioneze mai bine rutina zilnică și să asigure că nevoile animalelor de companie sunt satisfăcute în mod adecvat și atunci când apar situații neprevăzute.

Este important să reținem că aplicația nu înlocuiește atenția sau interacțiunea directă cu animalul de companie, dar poate fi un instrument util pentru a asigura îngrijirea adecvată și pentru a vă menține animalul de companie sănătos și fericit.

Există o varietate de aplicații de îngrijire a animalelor de companie astăzi pe piață, care acoperă diferite aspecte, cum ar fi plimbări, programarea unei vizite la salon sau la veterinar și servicii de „sitting” a animalelor de companie. Cu toate acestea, majoritatea aplicațiilor se concentrează pe un singur aspect sau specie de animal de compa-

nie, iar utilizatorii trebuie să folosească mai multe aplicații pentru a-și gestiona sarcinile zilnice. În plus, majoritatea aplicațiilor nu oferă proprietarilor o modalitate de a găsi îngrijitori de animale de companie dacă aceștea nu au timpul necesar pentru a-și îndeplini sarcinile zilnice.

Această lucrare de licență prezintă o aplicație mobilă care oferă o soluție completă pentru îngrijirea animalelor de companie. Aplicația permite utilizatorilor să-și gestioneze rutina zilnică, să programeze plimbări, vizite la salon și la veterinar, să găsească îngrijitori pentru animalele lor de companie și să comunice cu aceștia.

Aplicația prezentată este dezvoltată folosind tehnologiile Flutter[2] și ASP.NET[3] și oferă multe beneficii și caracteristici utile pentru utilizatori. Flutter este un framework dezvoltat de Google care vă permite dezvoltarea unor aplicații mobile native pentru platformele Android și iOS cu un singur cod sursă. ASP.NET este un alt framework, dezvoltat de Microsoft, care oferă un mediu robust pentru construirea de aplicații web scalabile[5], adaptabile și sigure.

# **Capitolul 1**

## **Specificații Funcționale**

Funcția principală a acestei aplicații este de a oferi utilizatorilor o soluție completă și ușoară pentru a închiria pe cineva care să aibă grija de nevoile potențiale ale animalului lor de companie. Utilizatorii aplicației se încadrează în două categorii mari: „Owners” și „Walkers”. Un utilizator care are un animal de companie și are nevoie de ajutor este un „Owner”, iar, de partea opusă, un utilizator care răspunde acestei cereri este un „Walker”.

### **1.1 Descrierea problemei**

În prezent, există o varietate de aplicații de îngrijire a animalelor de companie pe piață, care acoperă diferite aspecte, cum ar fi plimbări, programarea unei vizite la salon sau la veterinar și servicii de „sitting” a animalelor de companie. Cu toate acestea, majoritatea aplicațiilor se concentrează pe un singur aspect sau specie de animal de companie, iar utilizatorii trebuie să folosească mai multe aplicații pentru a-și gestiona sarcinile zilnice. În plus, majoritatea aplicațiilor nu oferă proprietarilor o modalitate de a găsi îngrijitori de animale de companie dacă aceștia nu au timpul necesar pentru a-și îngriji animalele de companie.

Aplicația prezentată în această lucrare nu își propune să integreze toate aceste aplicații într-una singură, ci să ofere o soluție comodă în completarea aplicațiilor dezvoltate de diverse companii de îngrijire a animalelor care necesită deplasarea lor la un sediu fizic. În același timp, aplicația acoperă o gamă mai largă de specii de animale de companie în comparație cu alte aplicații disponibile, inclusiv nu numai animale „clasică” precum câini și pisici, ci și rozătoare, păsări, reptile, pești etc. Fiecare persoană

este liberă să își adauge propriul animal de companie în aplicație, indiferent de tipul de animal, iar „Walker” este liber să aleagă tipul de animal de care vrea să aibă grijă, astfel încât nimeni să nu fie lăsat în urmă.

Întrucât există o gama largă de specii și servicii disponibile în aplicație, „Walker” poate deveni dezorientat atunci când navighează și selectează serviciile. În același timp, unii oameni au fobii diferite, cum ar fi cinofobia<sup>1</sup> sau fobia de reptile. Acești oameni nu ar dori ca în aplicația lor să apară servicii care nu sunt relevante pentru ei. Din acest motiv, am decis să introducem un sistem de preferințe pentru fiecare „Walker” care ne permite să oferim o anumită prioritate pe specie de animal și serviciu. Prin urmare, plimbătorii pot alege să ofere doar plimbări pentru câini sau pisici și nu servicii veterinar pentru rozătoare. De asemenea, plimbătorii pot alege să ofere servicii de plimbare pentru câini, pisici și rozătoare, dar nu pot oferi servicii veterinar pentru nicio specie. În acest fel, aplicația devine mai ușor de utilizat și mai intuitivă pentru toți utilizatorii.

## 1.2 Cerințe funcționale

Principala cerință a acestei aplicații este de a oferi utilizatorilor o interfață intuitivă și ușor de utilizat, care să medieze comunicarea între „Owner” și „Walker”. În același timp, atât „Owner-ul”, cât și „Walker-ul” ar trebui să folosească aceeași aplicație, dar aceasta ar trebui să ofere capacitați diferențiate specifice fiecărui. În cele ce urmează, vom prezenta cerințele funcționale pentru fiecare tip de utilizator.

### 1.2.1 Cerințe funcționale pentru „Owners”

După conectare, prima cerință și sarcină a utilizatorului este de a adauga unul sau mai multe animale de companie în aplicație. Pentru acestea există un meniu dedicat de adăugare și modificare unde puteți alege numele, specia, rasa, data nașterii, sexul și vom putea ataşa o scurtă descriere dedicată „Walker-ului”. După adăugarea unui animal de companie, acesta va apărea în lista de animale de companie a utilizatorului. Aceste informații vor fi folosite de către „Walker” pentru a decide dacă poate să se ocupe de animalul respectiv.

---

<sup>1</sup>Cinofobia (cuvânt care provine din alăturarea a două cuvinte: latinescul canis sau grecescul kýon „câine” și fobie) este o fobie (teamă patologică) de câini.

După adăugarea unui animal de companie, utilizatorii pot căuta un „Walker” care să aibă grija de animalul de companie. Pentru a face acest lucru, utilizatorul trebuie să plaseze anunțuri în aplicație. Anunțul trebuie să conțină cele mai importante informații de care ar avea nevoie un „Walker”. Acestea includ profilul animalului, dată și ora la care persoană are nevoie de asistență, tipul serviciului solicitat, durata acestuia, locația la care trebuie să ajungă animalul de companie dacă este cazul și un scurt mesaj către „Walker”. Odată ce anunțul este publicat, acesta va apărea în lista de anunțuri a utilizatorului. În acest moment, anunțul va fi vizibil pentru toți „Walker-ii”. După ce unul din acestea a acceptat anunțul, „Owner-ul” poate vedea asta în aplicație în dreptul rezervării.

Pentru a crește acuratețea și a evita confuzia, locațiile clinicilor veterinare sau a saloanelor pentru animele sunt selectate folosind o hartă. În acest fel, „Walker-ul” știe exact unde să meargă pentru a ajunge la locația dorită. Același lucru este valabil și pentru adresa utilizatorului. Acesta trebuie să selecteze pe hartă o locație de unde „Walker-ul” va ridica animalul de companie.

„Owner-ul” va avea la dispoziție și un meniu în care poate vizualiza istoricul anunțurilor publicate de acesta și satisfăcute de către „Walker-i”.

### **1.2.2 Cerințe funcționale pentru „Walkers”**

În aplicație un „Walker” are două îndatoriri principale: să caute noi anunțuri care i se potrivesc și să satisfacă anunțurile pe care le-a acceptat deja.

Pentru a căuta anunțuri noi, „Walker-ul” are o listă cu toate anunțurile postate de proprietarii, care sunt relevante pentru acel „Walker”. Pentru a face lista mai ușor de navigat, această poate fi filtrată după specie și serviciu. Pentru a modifica preferințele sale legate de animale sau servicii, „Walker-ul” are la dispoziție un meniu dedicat. Acesta poate alege pentru fiecare specie de animal de companie sau serviciu un nivel de prioritate. Aceste preferințe vor fi folosite de către aplicație pentru a afișa anunțuri relevante pentru „Walker”.

După ce a găsit un anunț care i se potrivesc, „Walker-ul” poate vedea detaliile anunțului pentru a-și da seama dacă dorește să îl acorde. Pe lângă detaliile introduse de utilizator, și enumerate mai sus, „Walker-ul” poate vizualiza și o harta care conține drumul de la locația sa curentă până la locația „Owner-ului” și după, dacă este cazul, până la locația unde trebuie să îl ducă pe animalul de companie. Pe lângă tra-

seu „Walker-ul” poate vizualiza distanțele și timpul estimat pentru a ajunge la locația „Owner-ului” de la locația sa și de la locația „Owner-ului” la locația unde trebuie să îl ducă, dacă este cazul. Aceste informații sunt oferite de Google Maps API[13] și Distance Matrix API[16]. Acestea sunt calculate pentru mersul pe jos în momentul în care „Walker-ul” vizualizează anunțul. Timpii necesari pot varia în funcție de traficul din zona la momentul respectiv.

După ce anunțul este acceptat, „Walker-ul” poate vedea asta în aplicație în meniul special pentru acestea. În momentul în care „Walker-ul” dorește să înceapă serviciul îi va fi afișată o harta care conține locațiile la care acesta trebuie să ajungă pentru a prelua animalul de companie și locația unde trebuie să îl ducă, dacă este cazul. În același timp, el poate vedea distanțele și timpul estimat pentru a ajunge la locațiile respective. Totodată pe harta va apărea și locația acestuia în timp real. Aceste informații vor fi disponibile și pentru „Owner” dacă acesta deschide anunțul în momentul în care „Walker-ul” a început activitatea. În acest fel, „Owner-ul” poate vedea când „Walker-ul” este pe drum spre el și poate să îl aștepte în locația stabilită. După preluarea animalului de companie „Owner-ul” poate vizualiza în continuare locația „Walker-ului” pentru a se asigura că totul decurge bine.

### 1.3 Cerințe non-funcționale

Pe lângă cerințele funcționale, aplicațiile trebuie să îndeplinească și multe cerințe nefuncționale. Acestea nu au nicio legătură cu funcționalitatea aplicației, dar sunt importante pentru a asigura o experiență plăcută atât utilizatorilor, cât și programatorilor care vor lucra la această aplicație în viitor. Aceste cerințe includ două lucruri cheie: performanță și scalabilitate.

### **1.3.1 Performanță**

Una dintre caracteristicile notabile ale framework-ului Flutter este reprezentată de performanța crescută a acestuia, atât pe platforma Android cât și pe iOS. Datorită arhitecturii native Flutter și compilării AOT (Ahead-of-Time)<sup>2</sup>, aplicațiile create cu aceste framework rulează eficient și fără probleme. Ca limbaj de programare Flutter folosește Dart care compilează cod nativ pe mai multe platforme, inclusiv pe cele mobile profitând de avantajele native ale dispozitivelor pe care rulează. Pentru a obține performanțe maxime, Flutter folosește un motor de randare numit Skia<sup>3</sup> care este folosit și de Google Chrome. Aceasta este un motor de randare 2D care oferă o performanță excelentă și o experiență de utilizare fluidă.

### **1.3.2 Scalabilitate**

Flutter este cunoscut pentru abordarea reactivă și arhitectura bazată pe widget care facilitează dezvoltarea de aplicații scalabile. Structura widgetului permite o separare clară a responsabilităților și o reutilizare sporită a codului. Aceasta înseamnă că dezvoltatorii pot crea și gestionă cu ușurință interfețe complexe și dinamice, indiferent de dimensiune sau complexitate.

.NET, pe de altă parte, are suport nativ pentru scalabilitate și este folosit pentru a dezvolta aplicații de la desktop și web până la servicii cloud și aplicații pentru întreprinderi. Platforma .NET oferă capabilități puternice de gestionare a memoriei, concurență și distribuție care permit dezvoltatorilor să construiască și să ruleze aplicații scalabile, de performanță înaltă. În plus, framework-urile și serviciile suplimentare precum ASP.NET și Azure oferă instrumente avansate pentru scalabilitatea aplicațiilor web și cloud.

Atât Flutter, cât și .NET beneficiază de comunități active de dezvoltatori și de sprijin din partea unor jucători mari precum Google în cazul Flutter și Microsoft pentru .NET. Deci dezvoltatorii au acces la resurse, documentație și actualizări continue pentru a-i ajuta să depășească provocările de scalabilitate și să implementeze soluții eficiente.

---

<sup>2</sup>În programare, compilarea Ahead-of-Time (AOT) reprezintă compilarea unui limbaj de programare de nivel înalt într-un limbaj de nivel inferior înainte de execuția programului, de obicei la momentul compilării, pentru a reduce cantitatea de lucru necesară la momentul execuției.

<sup>3</sup><https://skia.org/docs/>

# **Capitolul 2**

## **Arhitectura aplicației**

Conform cerințelor non-funcționale, aplicația constă din două componente principale: server și aplicație mobilă. Pentru server, am ales .NET care oferă o bază solidă pentru gestionarea datelor și logicii aplicației. .NET este un framework scalabil și robust, oferind suport pentru baze de date și servicii web, asigurând totodată securitate și performanță. Aplicația mobilă este dezvoltată în Flutter, un framework cross-platform, care permite dezvoltarea aplicațiilor pentru Android și iOS native folosind un limbaj de programare comun, Dart.

### **2.1 Arhitectura serverului**

Pentru realizarea acestui server a folosind o arhitectură „Clean Architecture”, fiind un sistem software modular, bine structurat, care promovează separarea clară a responsabilităților și încurajează dezvoltarea de cod ușor de întreținut și testat. O arhitectură „arhitectură curată” se bazează pe principiul inversării dependenței (Dependency Inversion Principle), împărțind o aplicație în niveluri concentrice, fiecare cu un rol bine definit. Straturi principale ale arhitecturii sunt: Stratul de bază (Core Layer), Stratul de aplicație (Application Layer) și Stratul de infrastructură (Infrastructure Layer) alături de Stratul de interfață (Interface Layer).

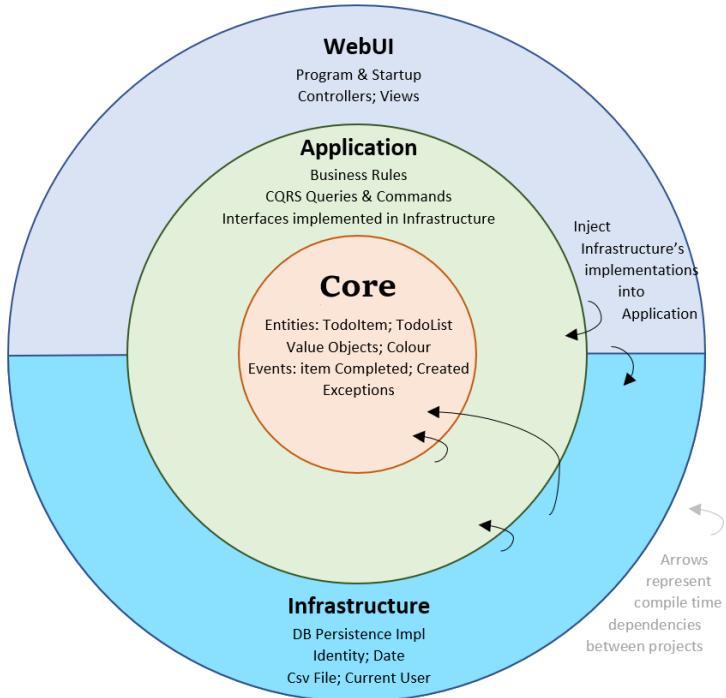


Figura 2.1: Clean Architecture[12]

### 2.1.1 Stratul de bază

Stratul de bază, sau central, reprezintă nucleul aplicației și conține regulile de business și logică aplicației. Acesta este independent față de restul straturilor, neavând dependințe către niciun dintre ele. Această independentă permite testarea și refacerea acestuia independent de restul aplicației. Acesta este împărțit în trei părți importante: Entities, Use Cases și Interfaces.

- **Entities** - reprezintă obiectele de bază ale aplicației, care conțin regulile de business și logică aplicației. În acest caz, printre entități se numără: User, Pet, Appointment, etc. și conțin doar datele și metodele necesare pentru a le manipula, fără a avea niciun fel de dependență către alte componente ale aplicației.
  - **Use Cases** - reprezintă regulile de business ale aplicației, de exemplu: adăugarea unui nou utilizator, înregistrarea unui animal de companie, realizarea unei programări, etc. Acestea sunt independente de orice altă parte a aplicației, ceea ce înseamnă că pot fi testate fără a fi nevoie de alte componente.
  - **Interfaces** - reprezintă interfețele de comunicare cu celelalte straturi ale aplicației. Aici se regăsesc interfețele pentru baza de date, serviciile web, etc.

## 2.1.2 Stratul de aplicație

Stratul de aplicație în arhitectura „Clean Architecture” reprezintă puntea între interfața utilizatorului și logica de afaceri din Stratul de paza. Gestionează fluxului de date și de interacțiunea între componente UI (User Interface) și componente din Core.

Nivelul de aplicație conține următoarele componente:

- **Commands** - Comenzile sunt reprezentate de cereri sau acțiuni specifice pe care un utilizator le poate efectua în intermediul aplicației. Acestea includ acțiuni precum crearea obiectelor, actualizarea datelor sau ștergerea acestora. Comenzile sunt de obicei reprezentate de structuri de date care conțin informațiile necesare pentru a efectua acțiunea dorită.
- **Handlers** - Handler-ii, sau gestionatorii, sunt componente responsabile pentru acceptarea și gestionarea comenziilor primite, ocupându-se de execuția logicii asociate cu acea comandă. Handler-ul primește comanda, extrage datele necesare și apelează funcțiile corespunzătoare din stratul de bază pentru a efectua operația necesară. Ele pot interacționa cu alte servicii din stratul curent, cum ar fi mappers sau responses, pentru a efectua acțiunile dorite și a returna rezultatele corespunzătoare.
- **Queries** - interogările reprezintă în general cereri de informații din partea utilizatorului. Acestea implică de obicei extragerea datelor din baza de date, prelucrarea lor într-un anumit mod și returnarea acestora către utilizator. Spre deosebire de comenzi care modifică starea sistemului, interogările au doar rolul de a returna informații într-o formă cerută.
- **Mappers** - mapperele sunt componente care se ocupă de transformarea datelor în formatul necesar. Acestea sunt folosite pentru a transforma obiectele primite de la utilizator în obiecte din stratul de bază, sau pentru a transforma obiectele din stratul de bază în obiecte care pot fi trimise către utilizator, în funcție de caz.

- **Responses** - răspunsurile sunt entități asemănătoare celor din stratul de bază, fiind formate însă doar variabile. Rolul acestora este de a returna datele cerute de utilizator, după ce au fost prelucrate de către handler, dar într-un format sigur. Returnarea unui obiect din stratul de bază ar putea expune către exterior date sensibile, cum ar fi parolele, id-uri, sau ar putea expune detalii despre implementarea internă a aplicației.

### 2.1.3 Stratul de infrastructură

Stratul de infrastructură oferă implementarea concretă a interfețelor definite ulterior în stratul de bază. Aici sunt incluse componente care îndeplinesc accesul la baza de date, serviciile de comunicare externă, sistemul de fișiere și alte resurse externe. Stratul conține următoarele componente:

- **Repositories** - reprezintă implementarea detaliată a interfețelor din stratul de bază și sunt responsabile pentru accesul la baza de date. Ele sunt folosite de handleri pentru a introduce, extrage, modifica sau șterge informațiile din baza de date.
- **Services** - serviciile sunt componente care asigură comunicarea cu alte servicii externe, precum servicii web sau servicii de stocare în cloud. Acestea sunt folosite de handleri pentru a efectua operațiunile necesare.

### 2.1.4 Stratul de interfață

Stratul de interfață reprezintă este responsabil pentru interacțiunea cu utilizatorul. Aici se gasesc componentele pentru interfață cu utilizatorul (interfață web, API, etc.). Acest strat are rolul de a prelua datele de la utilizator, de a le trimite mai departe către handleri și de a returna datele cerute. Principala componentă a acestui strat este API-ul, care este folosit de aplicația mobilă pentru a comunica cu serverul.

- **Controllers** - Controller-ii din API-uri acționează ca intermediari între partea de prezentare, interfața utilizatorului, și restul aplicației. Aceștea primesc cereri HTTP de la clienti pe care le trimit către handlerii din stratul de applicatie. Aici cererea este îndeplinită și ulterior handlerii returnează datele către controller care, la rândul sau, le trimit mai departe către client.

## 2.2 Arhitectura aplicației mobile

Pentru dezvoltarea aplicației mobile, am ales să folosesc Flutter, un framework cross-platform dezvoltat de Google. Aceasta permite dezvoltarea aplicațiilor native pentru Android și iOS folosind un limbaj de programare comun, Dart. În cazul aplicației am folosit o arhitectură Model-View-Presenter (MVP), care separă logica de prezentare de logica de afaceri și de date. Această arhitectură este asemănătoare cu arhitectura Model-View-Controller (MVC), care este una dintre cele mai populare arhitecturi pentru dezvoltarea aplicațiilor web.

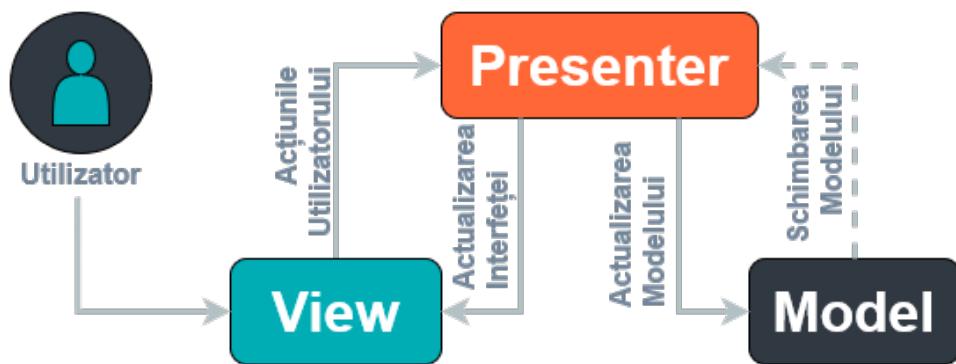


Figura 2.2: Model-View-Controller[11]

### 2.2.1 Model-View-Presenter (MVP)

Pentru a înțelege mai bine arhitectura MVP o vom compara cu arhitectura MVC. În arhitectura MVC, modelul reprezintă datele și logica aplicației, iar controller-ul este responsabil de gestionarea interacțiunii dintre model și view. View-ul reprezintă interfața cu utilizatorul, care este responsabilă de afișarea datelor și interacțiunea cu utilizatorul. În arhitectura MVP, modelul și controller-ul sunt unite într-o componentă unitară, numită presenter. Presenter-ul este responsabil pentru gestionarea logicii și a datelor, iar view-ul este responsabil pentru afișarea informațiilor și interacțiunea cu utilizatorul, ceea ce permite o separare mai mare între logica și interfața utilizatorului. Această separare facilitează testarea codului și modularitatea sa, făcându-l mai ușor de întreținut și extins. MVP oferă, de asemenea, o flexibilitate sporită în cazul schimbările de interfață, deoarece presenter-ul controlează modul în care informațiile sunt prezentate utilizatorului. Prin separarea strictă a responsabilităților, MVP asigură o mai bună organizare și înțelegere a codului, ceea ce poate duce la o dezvoltare mai rapidă și mai eficientă a aplicațiilor.

## 2.2.2 Componentele arhitecturii MVP

Arhitectura MVP este împărtită în trei componente importante: Model, View și Presenter. Acestea sunt independente și comunică între ele folosind interfețe. O astfel de separare permite testarea și refacerea componentelor independent, fără a afecta restul aplicației.

- **Model** - reprezintă datele și logica aplicației. Acesta se ocupă de manipularea datelor și de efectuarea operațiilor necesare. Tot el este responsabil și pentru comunicarea cu serverul.
- **View** - reprezintă interfața cu utilizatorul. Acesta se ocupă cu afișarea interfeței și de recepționarea acțiunilor utilizatorului. View-ul este pasiv, fiind lipsit de orice logica. El afișează utilizatorului informațiile primite de la presenter.
- **Presenter** - reprezintă componenta principală a acestei arhitecturi și acționează ca intermediar între model și view. Presenter-ul primește acțiunile utilizatorului captate de view și executa logica necesara în Model. După care actualizarea atât modelul cat și view-ul cu datele necesare.

## 2.3 Arhitectura bazei de date

Pentru baza de date, am ales să folosesc SQL Server, un sistem de management al bazelor de date relaționale. Dezvoltat de Microsoft, SQL Server oferă o bază solidă pentru stocarea datelor într-un mod structurat și eficient. Fiind un sistem relațional, acesta permite definirea și gestionarea tabelelor și relațiilor dintre ele, facilitând organizarea datelor într-un format coerent.

SQL Server acceptă o varietate de tipuri de date, cum ar fi siruri, numere, date și ore care vă permit să gestionați și manipulați aceste informații folosind funcții și operațiuni specifice. De asemenea, oferă suport pentru funcții și proceduri stocate, permitându-vă să creați o logică personalizată reutilizabilă în baza de date.

Un aspect cheie al SQL Server este capacitatea sa de a lucra cu date spațiale. Aceasta înseamnă că puteți stoca și manipula informații geospațiale, cum ar fi coordonatele geografice și poligoane. Această caracteristică este utilă în aplicații precum cea prezentată care necesită manipularea datelor spațiale pentru a oferi funcționalități precum navigarea sau afișarea unei locații pe hartă.

### 2.3.1 Avantajele aduse de SQL Server

- **Performanță [10]** - SQL Server este recunoscut pentru capacitatea sa de gestiona cantități mari de date și performanța sa ridicată. Oferă suport pentru indexarea datelor, care permite acces rapidă la date, și optimizarea interogărilor. De asemenea, oferă suport pentru funcții și proceduri stocate, care permit crearea de funcționalități complexe care pot fi apoi reutilizate în cadrul aplicației.
- **Securitatea** - acesta pune la dispozitie o mulțime de funcționalități de securitate robuste pentru protejarea datele stocate în baza de date. Aici sunt incluse autorizarea, criptarea și auditarea, care asigură integritatea și confidențialitate informațiilor.
- **Integrare cu mediul .NET** - ambele produse fiind dezvoltate de Microsoft, SQL Server și .NET sunt compatibile și se integrează perfect. Acest lucru facilitează interacțiunea dintre serverul ASP.NET și baza de date, permitându-ne să accesăm și să manipulăm date într-un mod simplu și eficient.
- **Măsuri de siguranță** - SQL Server oferă o serie de mecanisme utile pentru a asigura integritatea datelor în cazul unor evenimente neprevăzute. Funcții precum tranzacții, jurnale de tranzacții și backup-uri periodice ajută la protejarea datelor și la recuperarea lor în cazul unor erori de sistem sau erori umane.

## 2.3.2 Diagrama bazei de date

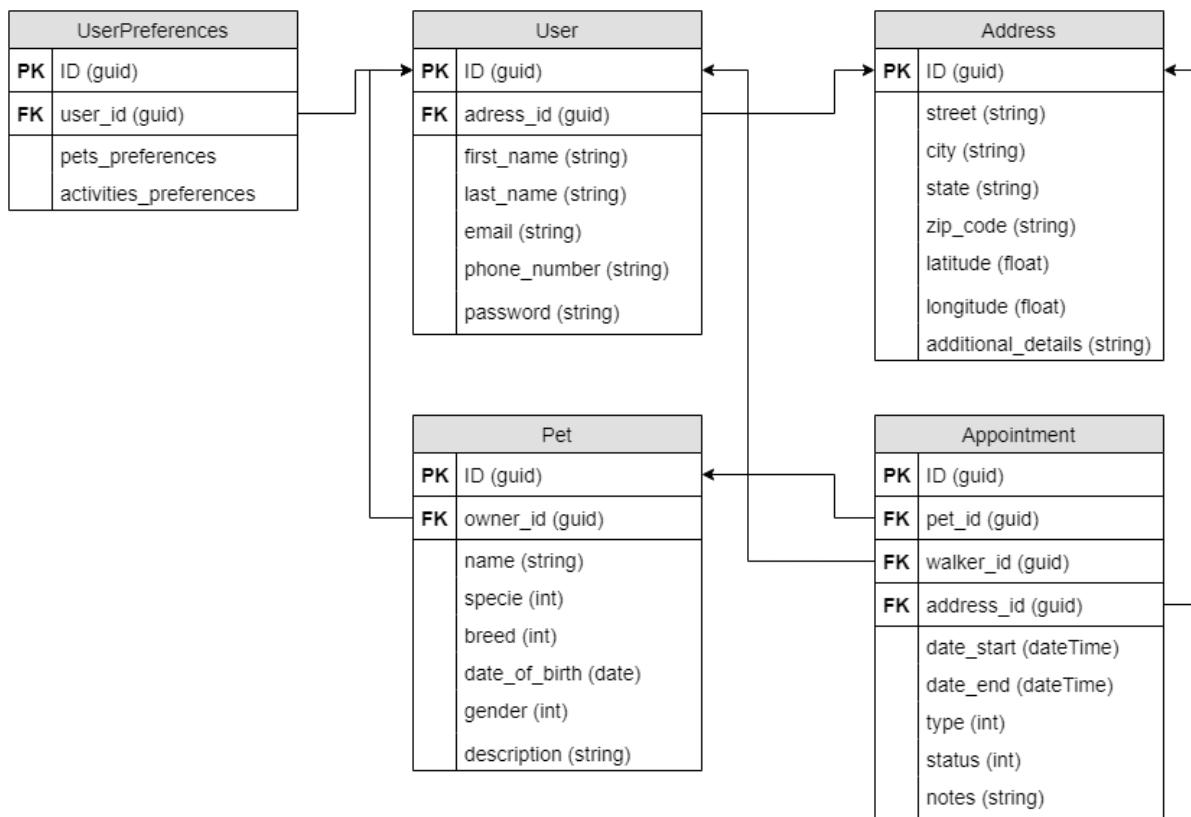


Figura 2.3: Diagrama bazei de date

# **Capitolul 3**

## **Implementare**

Implementarea aplicației a avut loc în două etape. În prima etapă a fost dezvoltat serverul, care oferea serviciile necesare pentru funcționarea aplicației. În a două etapă a fost dezvoltată aplicația mobilă, care folosește serviciile oferite de server pentru a oferi funcționalitatea necesară utilizatorului final. O dată cu dezvoltarea aplicației, au fost modificate și adăugate noi funcționalități serverului, pentru a oferi noi funcționalități care nu se aflau inițial în cerințele proiectului. Pe parcursul implementării, au fost folosite diferite tehnologii și servicii externe, actuale și moderne, care au permis dezvoltarea unei aplicații robuste și scalabile. În continuare, sunt prezentate tehnologiile folosite, precum și modul în care au fost folosite.

### **3.1 Sericii externe**

Pentru dezvoltarea aplicației, au fost folosite diferite servicii externe, în special cele oferite de Google pe partea de navigare. Am preferat folosirea unor servicii externe, deoarece acestea oferă funcționalități complexe, care ar fi necesitată o perioadă mai lungă de timp pentru a fi implementate și nu a fost scopul proiectului de a dezvolta aceste funcționalități. De asemenea, aceste servicii sunt oferite de companii mari, care oferă suport și actualizări constante, ceea ce asigură o aplicație robustă și sigură.

### 3.1.1 Google Maps Platform

Pentru a putea oferi funcționalitatea de navigare, am folosit serviciul Google Maps. Acesta oferă funcționalitatea de a afișa harta bine cunoscută din cadrul aplicație „Google Maps” și de a calcula și afișa rute între două puncte de pe harta. Pentru a putea folosi serviciul, a fost necesar crearea unui cont de dezvoltator și generarea unei chei de autentificare. Această cheie este folosită de aplicație pentru a se autentifica în cadrul serviciului Google Maps și pentru a putea folosi serviciile oferite de acesta. Din cadrul serviciului Google Maps, au fost folosite următoarele servicii:

- **Maps SDK** - oferă funcționalitatea de a afișa harta pe dispozitivul mobil. Acest serviciu este folosit pentru a afișa harta în cadrul aplicatiei. Harta conține pe lângă străzi și clădiri, și punctele de interes, precum parcuri, restaurante, etc.
- **Directions API** - oferă funcționalitatea de a calcula rute între două puncte. Acest serviciu este folosit pentru a calcula rutele dintre locația curentă a utilizatorului și locația unde se află animalul de companie, sau pentru a calcula rutele dintre locația animalului de companie și locația unde se află salonul sau cabinetul veterinar.
- **Distance Matrix API** - oferă funcționalitatea de a determina distanțele și timpii de parcursere a drumului dintre două puncte. Acest serviciu este folosit pentru a ajuta utilizatorul să estimeze timpul necesar pentru a ajunge la locația unde se află animalul de companie, sau pentru a ajunge la locația unde se află salonul sau cabinetul veterinar pentru o mai bună planificare a timpului.

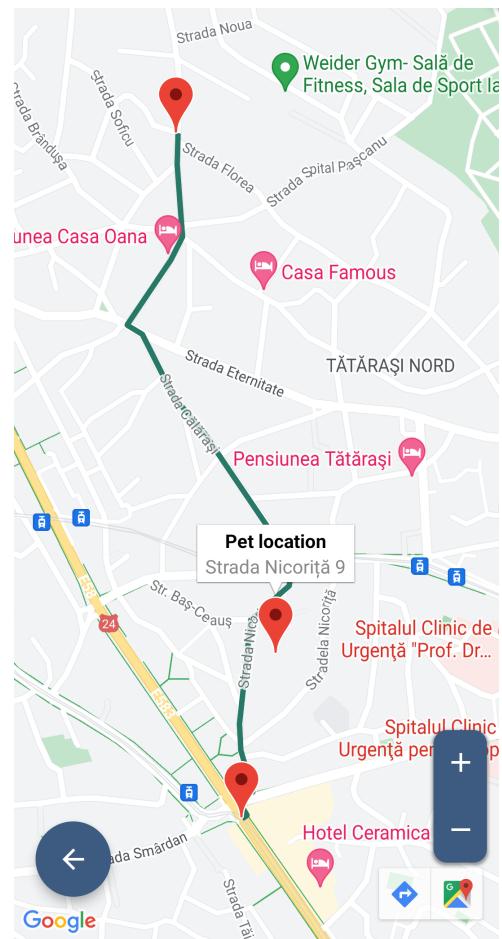


Figura 3.1: Vizualizarea hărții în cadrul aplicatiei

### **3.1.2 Geolocator**

Am utilizat acest serviciu pentru a determina locația curentă a utilizatorului și ne permite să urmărim locația acestuia pe hartă în momentul deplasării.

Geolocator oferă posibilitatea de a determina locația curentă a utilizatorului folosindu-se de GPS, rețele mobile și conexiuni WiFi. Acesta poate urmări locația utilizatorului în timp real, oferind informații despre locația curentă, viteza de deplasare, altitudine, etc. Acest serviciu este util atât „Walker” pentru vizualizarea în timp real a drumului pe care îl mai are de parcurs până la destinația curentă cât și „Owner” pentru a spori gradul de siguranță al animalului de companie.

## **3.2 Generarea unei agende**

Pentru a ușura muncă „Walker-ului” și a elimina timpul necesar de alegere a unor anunțuri pe placul său, am implementat un algoritm care generează o agenda personalizată pentru „Walker” în funcție de preferințele acestuia. Algoritmul este implementat în cadrul serverului și este apelat de către aplicația mobilă în momentul în care „Walker-ul” dorește să genereze o astfel de agenda. Algoritmul are ca input o listă de anunțuri și o listă de preferințe ale „Walker-ului” iar ca output o listă de anunțuri care respectă preferințele acestuia și care sunt disponibile în ziua respectivă, această fiind furnizată la rândul ei în momentul în care va începe generarea. Astfel el poate să-și gestioneze timpul pentru mai multe zile. Pe lângă preferințele sale, și constrângerile impuse de timpii serviciilor, generarea mai ține cont și de distanțele pe care acesta trebuie să le parcurgă. Astfel dacă un serviciu este finalizat la o ora 14:00, algoritmul nu va lua în considerare un anunț care începe la 14:20 dar care se află la o distanță de 30 de minute de locația unde s-a finalizat serviciul anterior. Pentru generarea fiecărei agende „Walker-ul” va fi nevoie să specifică modul în care se va deplasa între programări, cu mașina sau pe jos, și ziua pentru care se dorește generarea. Acest lucru este util deoarece timpii necesari sunt foarte diferiți și astfel se poate evita situația în care „Walker-ul” nu poate ajunge la timp la o programare sau are timp liber nedorit între două programări.

# Capitolul 4

## Prezentarea aplicației

In continuare voi prezenta pe scurt cele mai importante aspecte si funcționalități ale aplicației dezvolte. Pentru a putea folosi aplicația, utilizatorul trebuie să se înregistreze și să se autentifice în cadrul aplicației. După autentificare, utilizatorul va fi redirectionat către pagina principală a aplicației, unde va avea la dispozitie principalele funcționalități pentru fiecare tip de utilizator, „Walker” sau „Owner”. În continuare voi prezenta funcționalitățile oferite de aplicație pentru fiecare tip de utilizator.

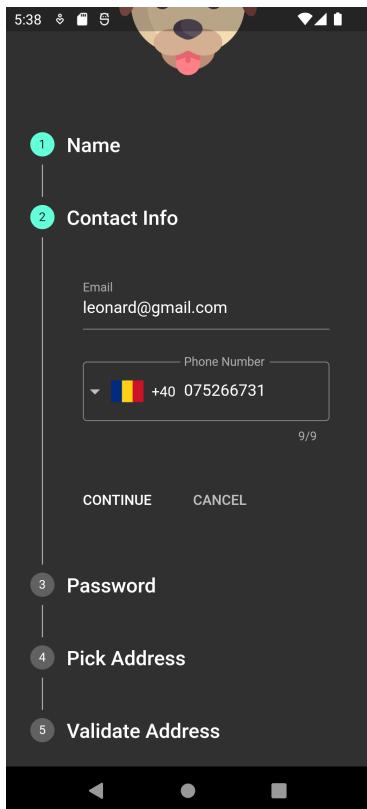


Figura 4.1: Inregistrarea utilizatorului

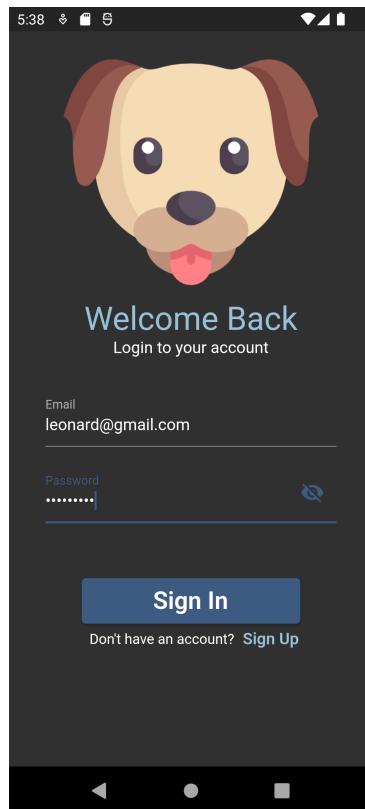


Figura 4.2: Autentificarea utilizatorului

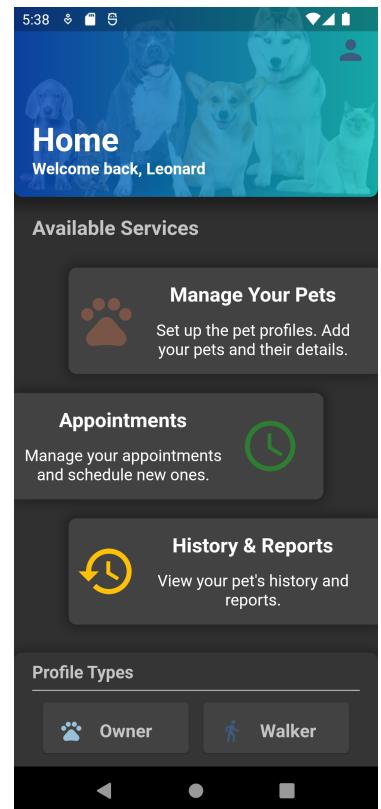


Figura 4.3: Ecranul principal

## 4.1 Meniurile Owner-ului

După ce s-a autentificat în aplicație acesta va avea la dispoziție 3 meniuuri cu informații și funcționalități diferite prezente pe ecranul principal. Acestea sunt: „Manage Your Pets”, „Appointments” și „History & Reports”. În continuare voi prezenta fiecare meniu în parte.

### 4.1.1 Manage Your Pets

Acest meniu oferă utilizatorului posibilitatea de a vizualiza animale de companie înregistrate de acesta în aplicație, de a adăuga un nou animal de companie, de a vizualiza informațiile despre acestea și de a edita aceste informații.

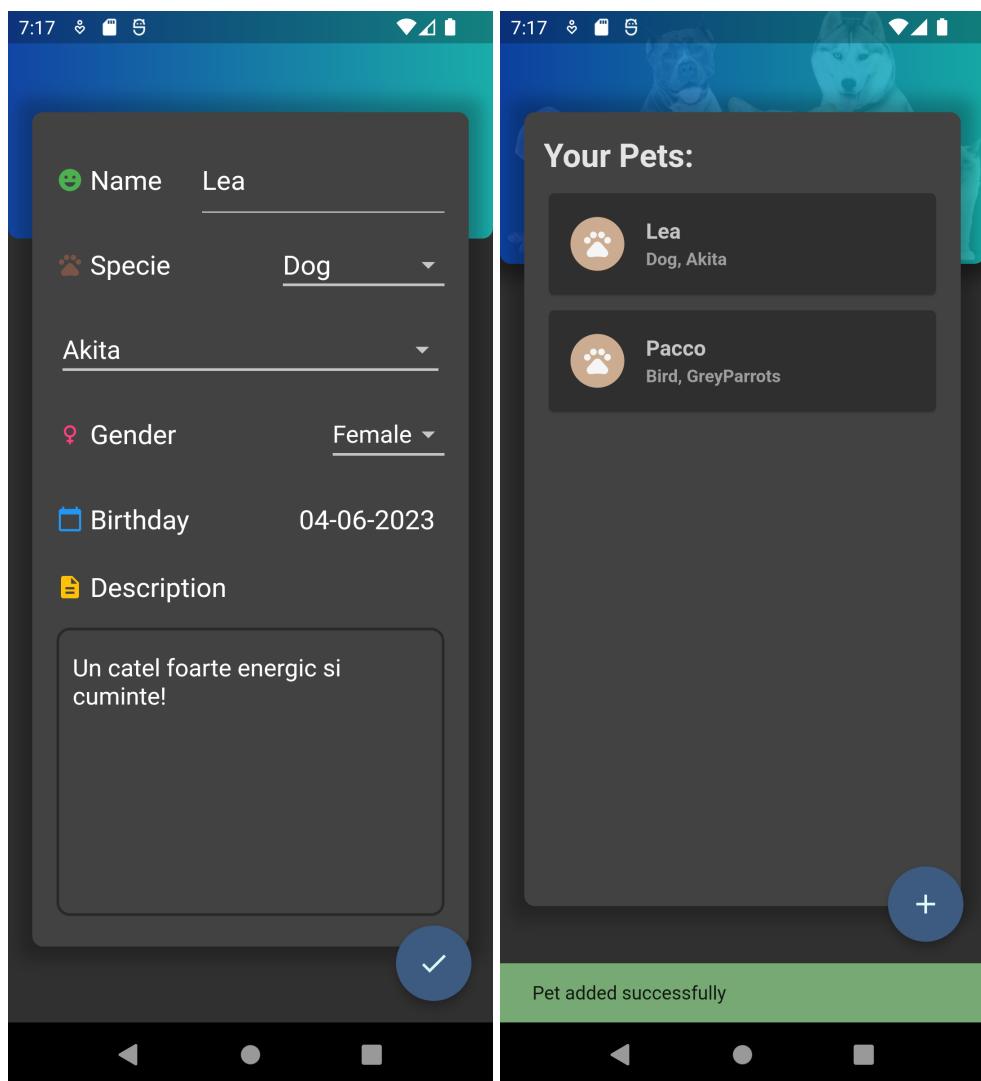


Figura 4.4: Meniul „Manage Your Pets”

## 4.1.2 Appointments

În interiorul acestui meniu utilizatorului are posibilitatea de a vizualiza programările curente poste de acesta și creeze noi programări.

Fiecare programare care apare în acest meniu va fi insotita de un mesaj care reprezinta statusul acesteia. Statusul unei programări poate fi unul din următoarele:

- **Pending** - programarea a fost postată cu succes de utilizator și se află în aşteptarea unui „Walker” care să o accepte.
- **Assigned** - programarea a fost acceptată de un „Walker”.
- **In Progress** - programarea a început și este în curs de desfășurare. Utilizatorului poate vizualiza locația curentă a „Walker-ului” pe hartă.

Tot în interiorul acestui meniu utilizatorul are posibilitatea de a posta noi anunțuri pentru una din cele 5 categorii: „Walk”, „Salon”, „Sitting”, „Vet” sau „Shopping”. Fiecare categorie va fi însotită de un button separat care va deschide un meniu în care utilizatorul va putea completa informațiile necesare pentru postarea anunțului. O dată postat anunțul, acesta va apărea în lista de programări curente și va putea fi acceptat de un „Walker”.

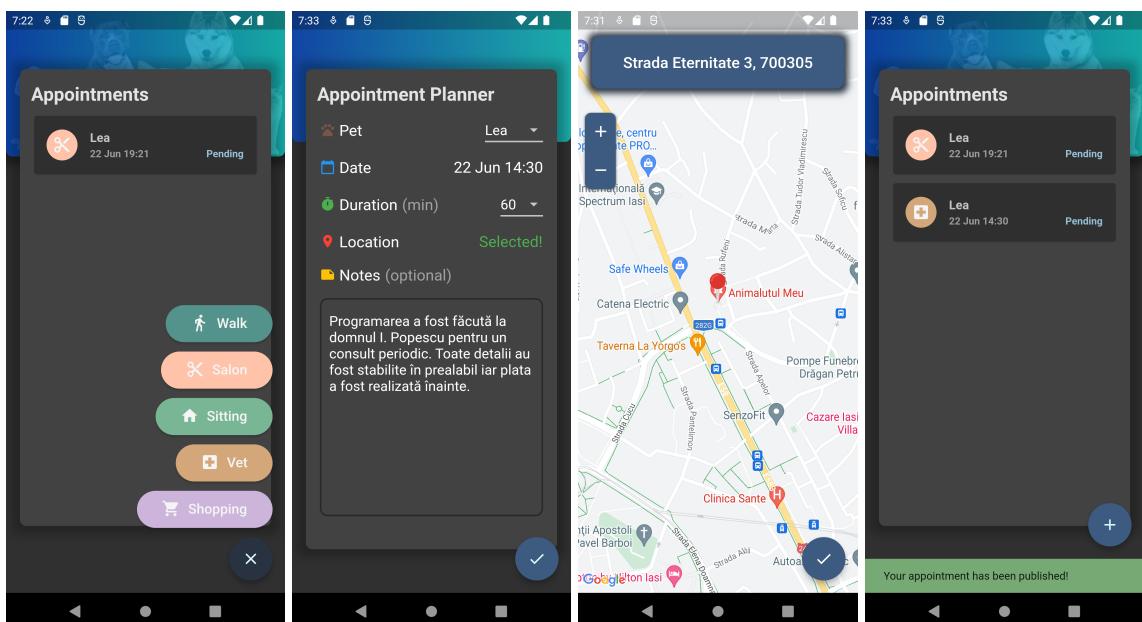


Figura 4.5: Meniul „Appointments”

În figura 4.5 se poate observă atât meniul „Appointments” care conține lista de programări curente, cât și meniul pentru crearea unei noi programări alături de pașii

necesari pentru crearea acesteia. Tot aici se poate observă utilizarea a trei dintre servicii externe amintite anterior, Google Maps, Geolocator și Geocoding. Primul folosit pentru a afișa hărții, al doilea pentru a determina locația curentă a utilizatorului și al cel din urmă pentru a determina adresa și informațiile locației selectate de utilizator pe harta.

#### 4.1.3 History & Reports

Acest meniu oferă utilizatorului posibilitatea de a vizualiza istoricul programărilor poste de acesta și indeplinite de un „Walker”.

### 4.2 Meniurile Walker-ului

Utilizatorul poate accesa meniu-urile „Walker-ului” și meniurile sale prin intermediul butoanelor din meniul „Profile Types” din partea inferioară a ecranului principal. O dată ce a fost selectat unul din profile, utilizatorul va fi redirectionat către ecranul principal al profilului selectat. Aici se află 3 meniuri diferite, meniul „Your Agenda”, „Search Appointment” și „Your Preferences”. În continuare voi prezenta fiecare meniu în parte.

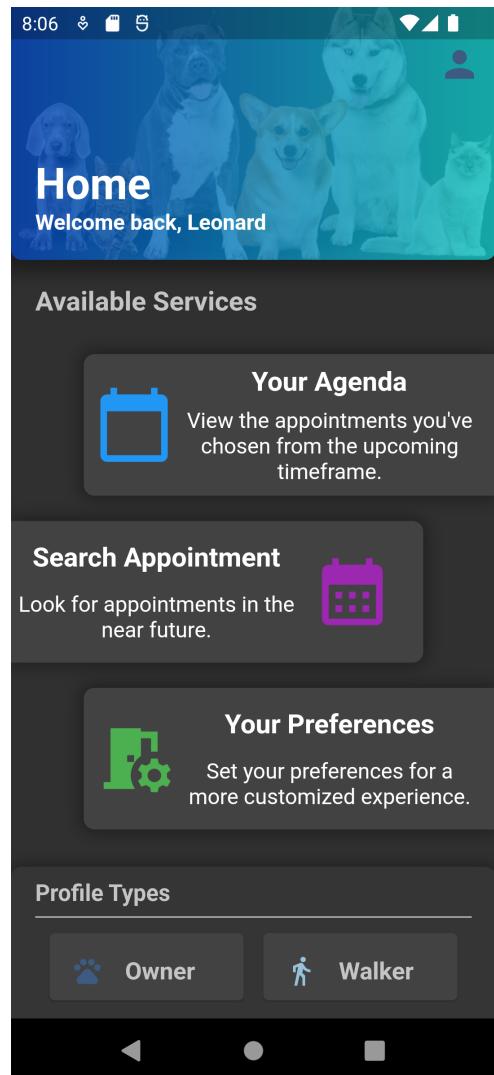


Figura 4.6: Meniul principal al „Walker-ului”

#### 4.2.1 Search Appointment

În acest meniu vor apărea programările poste de utilizatorii și pe care „Walker-ul” le poate acceptă. Acestea sunt sortate după preferințe și apar în ordinea în care acestea ar trebui îndeplinite. De aici „Walker-ul” le poate sorta manual și să își aleagă programările care îi sunt pe plac și i-ar face plăcere să le îndeplinească.

Tot aici se află și butonul pentru generarea unei liste de programări personalizate. Acesta va deschide un meniu în care utilizatorul va trebui să specifică modul în care se va deplasa între programări, cu mașina sau pe jos, și ziua pentru care se dorește generarea. După ce a fost generată lista, aceasta va trebui să fie acceptată de utilizator pentru ca ele să fie adăugate în lista sa de programări. Toate programările acceptate de utilizator vor fi vizibile în meniul „Your Agenda”.

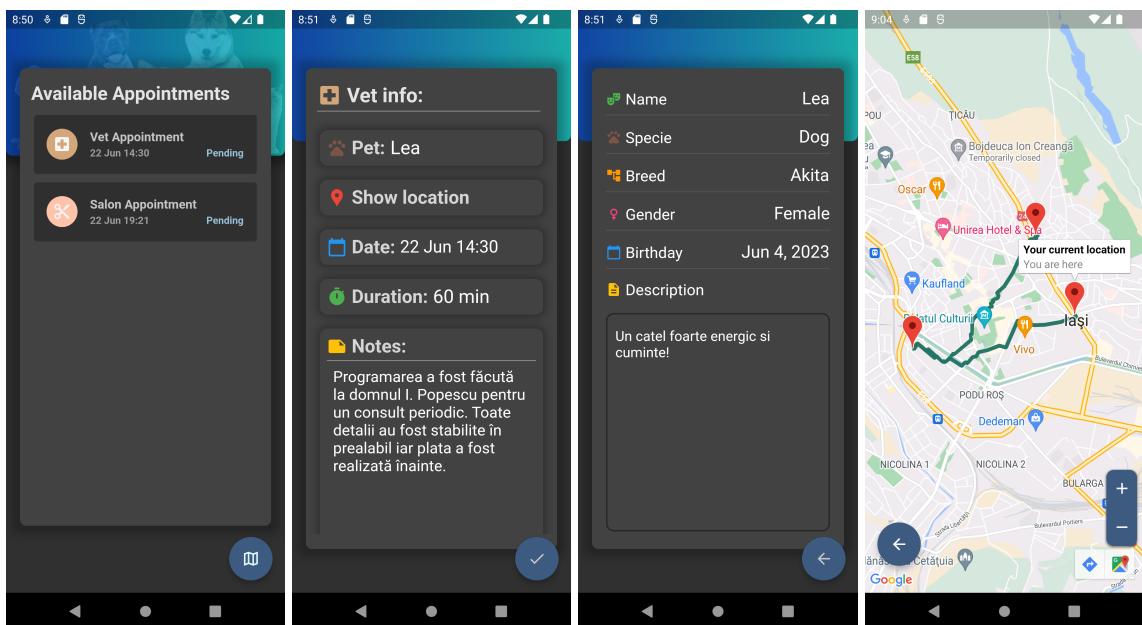


Figura 4.7: Meniul „Search Appointment”

În figura 4.7 se poate observă meniul „Search Appointment” care conține lista de programări disponibile. Dând click pe o programare va apărea un meniu pentru vizualizarea informațiilor despre programarea selectată. Din acest meniu putem vizualiza informațiile despre animalul de companie pentru care se face programarea, data și durata programării, unde data reprezintă momentul în care animalul trebuie să fie prezent la medic iar durata reprezintă timpul necesar consultației. Tot aici putem vizualiza și o hartă care include locația unde se află animalul de companie și de unde trebuie ridicat alături de locația unde acesta trebuie să ajungă. Pentru a ușura muncă „Walker-ului” am generat și drumul minim pe care acesta trebuie să îl parcurgă de la locația sa ac-

tuala până la cele două locații menționate anterior. Drumul generat în poză a patra a fost generat pentru a fi parcurs pe jos. Pentru fiecare punct de pe harta a fost calcula și distanță alături timpul necesar pentru a ajunge la el de la punctul anterior.

#### 4.2.2 Your Preferences

Acest meniu oferă utilizatorului posibilitatea de a vizualiza preferințele și de a le edita. Preferințele sunt împărțite în 2 categorii: preferințe pentru animalele de companie și preferințe pentru programări. Pentru fiecare element din aceste categorii utilizatorul poate alege unul din cele 3 niveluri de preferințe:

- **Low** - utilizatorul nu ar prefera să aibă de a face cu acest element.
- **Medium** - utilizatorul este indiferent în ceea ce privește acest element.
- **High** - utilizatorul ar dori să aibă de a face cu acest element.

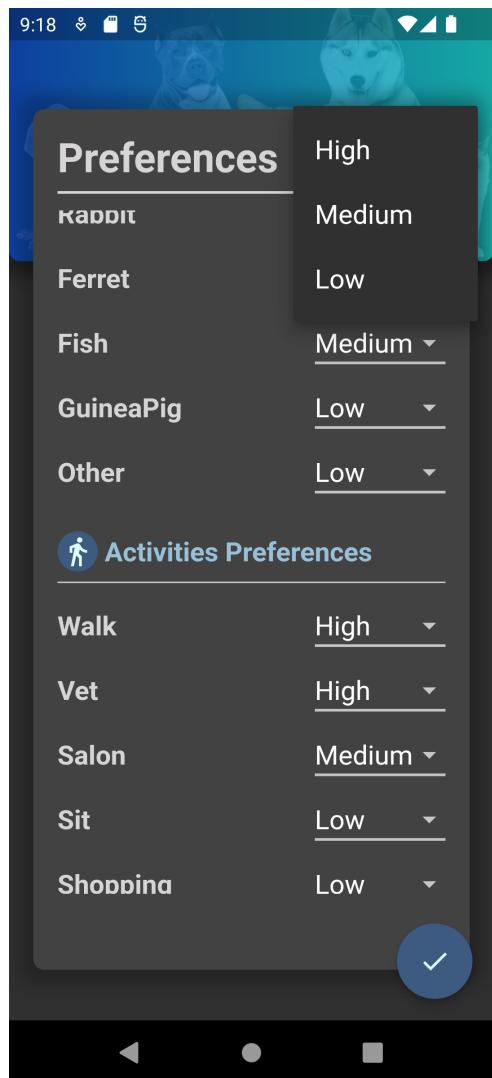


Figura 4.8: Meniul „Your Preferences”

### 4.2.3 Your Agenda

Acest meniu oferă utilizatorului posibilitatea de a vizualiza agenda sa. Aici se află programările acceptate de acesta și care se află în curs de așteptare. De aici utilizatorul poate începe o programare când click pe această iar în acel moment harta va fi încărcată iar locația sa va fi urmărită constant. De acum sarcina „Walker-ul” este de a ajunge la locația indicată pe hartă unde se află animalul de companie și după în după în funcție de fiecare tip de programare va avea o sărină diferită de înplinit.

- **Walk** - „Walker-ul” trebuie să plimbe animalul de companie pentru perioada de timp specificată în programare și să îl aducă înapoi la locația de unde l-a preluat după ce a expirat timpul.
- **Salon — Vet** - „Walker-ul” trebuie să ducă animalul de companie la locația indicată pe hartă și să îl aștepte până când programarea este finalizată. După ce programarea este finalizată, „Walker-ul” trebuie să îl aducă înapoi la locația de unde l-a preluat.
- **Sitting** - „Walker-ul” trebuie să aducă animalul de companie la el acasă și să aibă grija de acesta până când programarea este finalizată. După ce programarea este finalizată, „Walker-ul” trebuie să îl aducă înapoi la locația de unde l-a preluat.
- **Shopping** - „Walker-ul” trebuie să meargă la locația indicată pe hartă și să prede produsele comandate de utilizator. Produse trebuie cumpărate de „Walker” înainte iar costul acestora va fi rambursat de către utilizator.

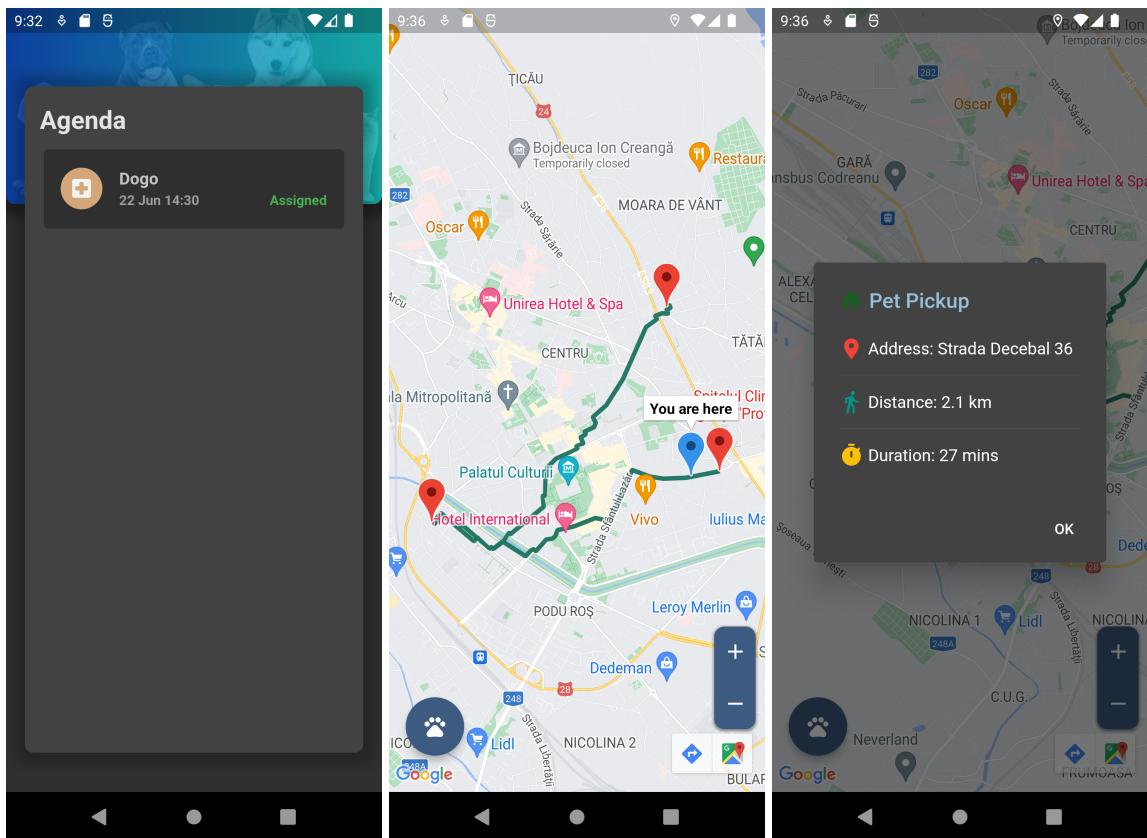


Figura 4.9: Meniul „Your Agenda”

În figura 4.9 se poate observă meniul „Your Agenda” care conține lista de programări acceptate de utilizator. Dând click pe o programare va apărea harta care conține traseul generat al programării. În acest moment locația „Walker-ului” este urmărită constant, aceasta fiind reprezentată de bulina albastră. Dacă dăm click pe una din locațiile de pe harta, va apărea un meniu care ne prezintă informații despre distanță și timpul până la acea locație calculate folosind locația noastră actuală. Pentru a trece prin etapele unei programări în aplicația „Demo” ne vom folosi de butonul din partea stângă care ne va trece pe rând prin toate etapele până la finalizarea programării. Această funcționalitate nu va fi prezentă în aplicația finală deoarece „Walker-ul” va trebui să finalizeze programarea în momentul în care această este finalizată în realitate.

De asemenea harta curentă poate fi vizualizată și de către „Owner-ul” animalului de companie, putând să urmărească în timp real desfășurarea programării.

# Concluzii

Animalele de companie joacă un rol semnificativ în viațile noastre, aducând bucurie, companie și beneficii pentru sănătatea noastră emoțională și fizică. La rândul lor, ele necesită îngrijire și atenție adecvate pentru a se dezvolta și a fi sănătoase. În acest scop, am dezvoltat o aplicație mobilă care oferă posibilitatea oamenilor de a se ocupa nevoile prietenilor lor necuvântători chiar și atunci când nu le permite acest lucru sau apar situații neprevazute. Aplicația oferă posibilitatea de a găsi o persoană disponibilă să aibă grija de animalul de companie în lipsa proprietarului, fie că este vorba de o plimbare, o plecare de câteva zile din oraș, o vizită la veterinar sau la salonul de înfrumusețare atunci când proprietarul nu disponibil, din păcate. Aceasta aplicație nu înlocuiește grija și dragostea proprietarului, ci doar vine în ajutorul acestuia atunci când nu poate fi prezent.

Aplicația este formată din 2 componente: serverul, realizat în ASP.NET 7.0 folosind o arhitectură "Clean Architecture" și o aplicație mobilă realizată în Flutter, disponibilă atât pentru Android cât și pentru iOS. Aceasta din urmă se folosește și de o multitudine de servicii externe pentru a oferi o mulțime de funcționalități utile și interesante pentru toți utilizatorii. Am ales să folosesc servicii externe pentru a oferi funcții scalabile și complexe, care ar fi necesitată o perioadă mai lungă de timp pentru a fi implementate. Deoarece acestea sunt oferite de companii mari precum Google, care oferă suport și actualizări constante, ceea ce asigură o aplicație robustă și sigură.

Aplicație prezentată oferă funcționalități precum: crearea unui cont, adăugarea, editarea și vizualizarea unui animal de companie, crearea unui anunț și publicarea acestuia, vizualizarea a anunțurilor publice și a celor personale sau vizualizarea unei hărții complexe care cuprinde informații precum: locații, trasee minime, locații curente, distanțe și timpi necesari de parcurgere.

# Bibliografie

- [1] Sara M. Hussein, Ahmed A. Khalifa *Benefits of pets' ownership, a review based on health perspectives*, 2021
- [2] Alessandro Biessek, *Flutter for Beginners*, 2019
- [3] Andrew Troelsen, Philip Japikse, *Introducing ASP.NET MVC*, 2017
- [4] Eric Windmill, *Flutter in Action*, 2019
- [5] Sebastian Faust, *Using Google's Flutter Framework for the Development of a Large-Scale Reference Application*, 2020
- [6] Tung Bui Duy, *Reactive Programming and Clean Architecture in Android Development*, 2017
- [7] Robert C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, 2017
- [8] Pablo Aguilar, Lucas Baggio Figueira, *Clean Architecture is not only about business logic*, Facultatea de Tehnologie a statului São Paulo, Brazilia, 2020
- [9] Payne, R., *Developing in Flutter. In: Beginning App Development with Flutter.*, Apress, Berkeley, CA, 2019
- [10] Itzik Ben-Gan, Adam Machanic, Dejan Sarka, Kevin Farlee, *Performance Tuning and Optimization in SQL Server*, Microsoft Press
- [11] Tian Lou, *A Comparison of Android Native App Architecture - MVC, MVP and MVVM*, Aalto University, School of Science, Master's Programme in ICT Innovation, 2016
- [12] Clean Architecture for ASP.NET Core Solution: A Case Study,  
<https://blog.ndepend.com/clean-architecture-for-asp-net-core-solution/>

[13] Google Maps Platform Documentation,  
<https://developers.google.com/maps/documentation>

[14] Camillo Gentile, Nayef Alsindim, Ronald Raulefs, Carole Teolis, *Geolocation Techniques - Principles and Applications*, 2013

[15] Geolocator Documentation,  
<https://pub.dev/packages/geolocator>

[16] Distance Matrix API Documentation,  
<https://developers.google.com/maps/documentation/distance-matrix/distance-matrix>

[17] ASP.NET documentation,  
<https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0>