

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN  
IAȘI  
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**PublicTransportMonitor**

propusă de  
**Adelin Pămînt**

**Sesiunea: *Februarie, 2020***

**Coordonator științific**  
***Prof. Colab. Florin Olariu***

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN  
IAȘI  
FACULTATEA DE INFORMATICĂ**

# **PublicTransportMonitor**

**Adelin Pămînt**

**Sesiunea: *Februarie, 2020***

**Coordonator științific  
*Prof. Colab. Florin Olariu***

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul PublicTransportMonitor, codul sursă al programelor și celelalte 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.

Iași, data

Absolvent Prenume Nume

---

(semnătura în original)

# CUPRINS

<b>INTRODUCERE</b>	<b>5</b>
1. Motivație	5
2. Introducere în temă	6
3. Structura lucrării	7
<b>CONTRIBUȚII</b>	<b>8</b>
<b>CAPITOLUL 1 - DESCRIEREA PROBLEMEI</b>	<b>9</b>
<b>CAPITOLUL 2 - ABORDĂRI ANTERIOARE</b>	<b>10</b>
2.1 Moovit	10
2.2 Google Maps	11
3.3 HEREITIS	13
<b>CAPITOLUL 3 - DESCRIEREA SOLUȚIEI</b>	<b>15</b>
3.1 Descriere proiect	15
3.2 Arhitectura proiectului	16
3.2.1 Arhitectură frontend	17
3.2.2 Arhitectură backend	19
3.3 Implementarea proiectului	20
3.3.1 Înregistrarea unui mijloc de transport	20
3.3.2 Căutarea unui mijloc de transport	23
3.3.3 Vizualizare detalii linie	27
3.3.4 Vizualizarea liniilor pe hartă	29
3.3.5 Stocarea datelor folosind SQL și Entity Framework Core	30
<b>CONCLUZIILE LUCRĂRII</b>	<b>34</b>
Concluzii generale	34
Direcții de viitor	35
<b>BIBLIOGRAFIE</b>	<b>36</b>
Link-uri poze:	36
<b>ANEXE</b>	<b>37</b>

# INTRODUCERE

## 1. Motivație

Ideea aplicației a pornit din dorința de a face mai plăcută și de a ușura folosirea mijloacelor de transport public din Iași. Totodată aplicația ajută la optimizarea timpului necesar pentru a folosi mijloacele de transport public, aceasta minimizând timpul petrecut de așteptare al unui autobuz sau al unui tramvai.

Transportul public este crucial într-un oraș, acesta aducând numeroase beneficii, de la accesibilitate din punct de vedere financiar, până la reducerea considerabilă a traficului și implicit, a poluării aerului.

În timp ce transportul public aduce numeroase beneficii, serviciile oferite trebuie să se afle la un anumit nivel pentru a fi folosit de către publicul larg. Unul din cele mai importante criterii pe care serviciile de transport public trebuie să îl îndeplinească îl reprezintă accesul la informații legate de locația mijloacelor de transport public, astfel încât un utilizator să își poată organiza activitățile în funcție de timpul necesar unui mijloc de transport să ajungă într-o stație de interes pentru utilizator. Acest criteriu nu este îndeplinit de către serviciile de transport public din Iași, și această aplicație a fost gândită și construită cu scopul de a remedia acest neajuns.

## **2. Introducere în temă**

Lucrarea realizată constă într-o aplicație din care un utilizator poate beneficia de numeroase funcționalități a căror scop este de a îmbunătăți experiența folosirii mijloacelor de transport în comun.

Poate primi informații legate de autobuze sau tramvaie ce îi sunt utile pentru a se deplasa către orice destinație din Iași.

Are posibilitatea de a vizualiza în timp real informații despre călătoria pe care o realizează prin intermediul unui mijloc de transport, utilizatorul știind în mod constant care este următoarea stație în care va ajunge, câte stații mai are de parcurs până la destinație și când a ajuns la destinație.

O altă funcționalitate, și probabil cea mai importantă a acestei aplicații este aceea de a putea vizualiza informații legate de locația curentă a unui autobuz sau a unui tramvai de care un utilizator se poate folosi pentru a se deplasa către destinația dorită.

Aceasta din urmă fiind funcționalitatea de bază a aplicației, deoarece astfel este eliminat neajunsul ce constă în neștiința locației oricărui autobuz sau tramvai, neajuns ce are ca și consecință forțarea utilizatorului serviciilor de transport public să aloce îndeajuns timp de așteptare în stație astfel încât să poată ajunge la timp destinație. Beneficiul aceste funcționalități este cu atât mai vizibil în situații în care timpul de care dispune o persoană este limitat și optimizarea acestuia devine un factor extrem de important, precum un student ce trebuie să ajungă la facultate într-o zi de examen, a unei persoane ce trebuie să ajungă la un interviu și așa mai departe.

### 3. Structura lucrării

Lucrarea conține trei mari capitole generale în care sunt detaliate mai multe informații despre lucrare. Per ansamblu, din acestea reies următoarele: impulsul ce m-a determinat să aleg această temă, ce îmbunătățiri aduce soluția propusă de mine în comparație cu soluții deja existente, cât și funcționalitățile implementate în aplicație și cum funcționează acestea în spate.

În **primul capitol** este descrisă nu doar utilitatea, cât și importanța serviciilor de transport public într-un oraș modern, sunt detaliate beneficiile pe care acestea le aduc, dar și neajunsurile acestora la momentul actual, neajunsuri care îi determină de multe ori pe oameni să aleagă alte variante în favoarea mijloacelor de transport în comun când vine vorba de deplasarea dintr-un loc în altul în cadrul orașului.

În **al doilea capitol** sunt prezentate câteva aplicații deja existente ce au abordat problema descrisă în Capitolul 1 și cu ce rezolvări au venit. Sunt detaliate funcționalitățile implementate în aceste soluții, îmbunătățirile pe care acestea le aduc în ceea ce privește transportul public, cât și funcționalități pe care eu le consider esențiale și lipsesc din aceste soluții.

În **al treilea capitol** este descrisă abordarea mea atât de a rezolva problema descrisă în primul capitol cât și de a umple golurile din aplicațiile descrise în al doilea capitol. Este descrisă ideea generală a proiectului, arhitectura acestuia, alături de funcționalitățile principale și cum sunt implementate acestea.

În final, se pot regăsi concluziile trase de mine în legătură cu această soluție și implementarea acesteia.

## CONTRIBUȚII

Numeroase modalități au fost încercate de-a lungul timpului pentru rezolvarea problemelor transportului public. Majoritatea oferă utilizatorilor detalii privind mijloacele de transport pe care le pot folosi pentru a ajunge la destinația dorită, și totodată oferă o aproximare a momentului în care acele mijloace de transport vor ajunge într-un loc de interes pentru utilizator. Și în timp ce această aproximare poate fi o adiție foarte utilă, din păcate este de cele mai multe ori departe de a fi precisă, rezultând într-o funcționalitate care în cele din urmă ajunge să încurce mai mult decât să ajute și în final să nu mai fie folosită deloc.

Soluția propusă de PublicTransportMonitor este de a nu se axa pe acest tip de informație, ci pe informații concrete astfel încât utilizatorul să poată deduce când va ajunge un mijloc de transport într-un loc de interes pentru acesta. Utilizatorul nu va putea deduce când va ajunge mijlocul de transport cu o precizie la nivel de minut, însă va ști stațiile deja parcurse de acesta, și astfel va putea deduce cu ușurință când și unde va ajunge un mijloc de transport.



## **CAPITOLUL 1 - DESCRIEREA PROBLEMEI**

Deplasarea folosind mijloacele de transport în comun aduce numeroase beneficii, atât locuitorilor orașelor cât și mediului înconjurător. Pentru noi, în calitate de utilizatori beneficiile sunt numeroase. Din punct de vedere al cheltuielilor, folosirea mijloacelor de transport în comun este de departe cea mai accesibilă. Un beneficiu pentru mediul înconjurător, și implicit și pentru noi, este faptul că folosirea cât mai mult a mijloacelor de transport în comun va rezulta în reducerea drastică a traficului în orașe; astfel, oamenii nu vor mai fi blocați cu orele pe drum, nivelul de poluare se va reduce considerabil, fapt ce în final tot pentru utilizatori reprezintă un mare avantaj.

Enumerarea de avantaje poate continua, însă ce consider mai important este ce anume lipsește când vine vorba de servicii de transport în comun dacă în timp ce aduc numeroase avantaje de luat în considerare, majoritatea oamenilor sunt reticenți în a le folosi.

Problema principală când vine vorba de deplasarea folosind mijloacele de transport în comun este faptul că publicul nu are acces la locația acestora și astfel folosirea lor presupune luarea în calcul a timpului de așteptare în stații, timp ce poate fi insignifiant, cum ar fi 2 minute, dar poate deveni și o mare problemă atunci când ajunge și la 30 de minute. Și mai problematică decât așteptarea timp de 30 de minute pentru un mijloc de transport este variația timpului de așteptare, pentru că din cauza acestei probleme oamenii nu pot să își planifice într-un mod cât mai eficient posibil timpul și sunt nevoiți să-și ia o marjă de eroare uneori chiar mare în funcție de cât de important este pentru aceștia să ajungă la timp la destinație.

## CAPITOLUL 2 - ABORDĂRI ANTERIOARE

### 2.1 Moovit

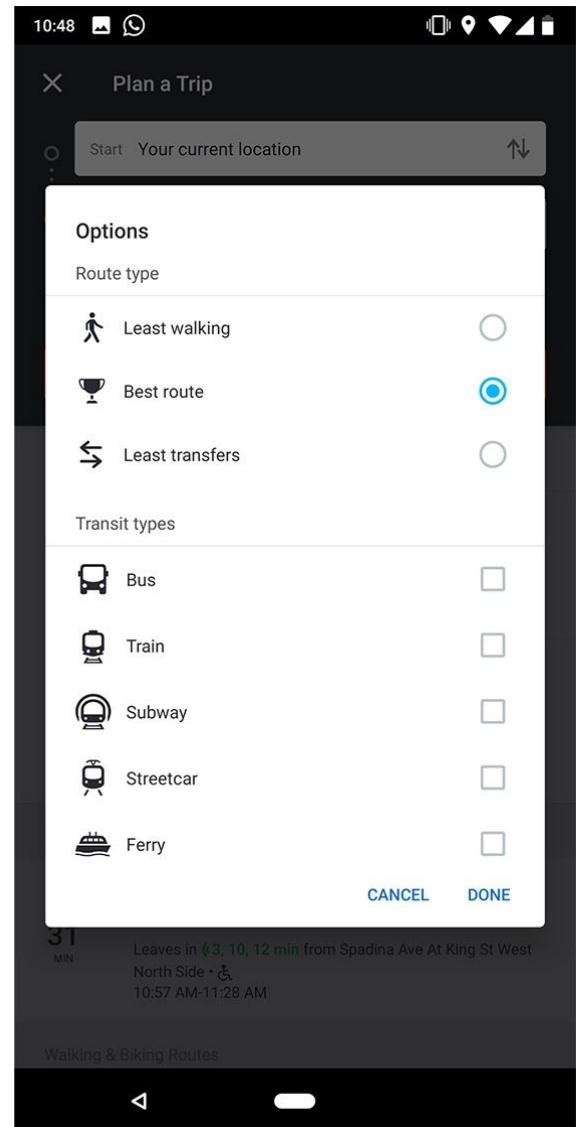
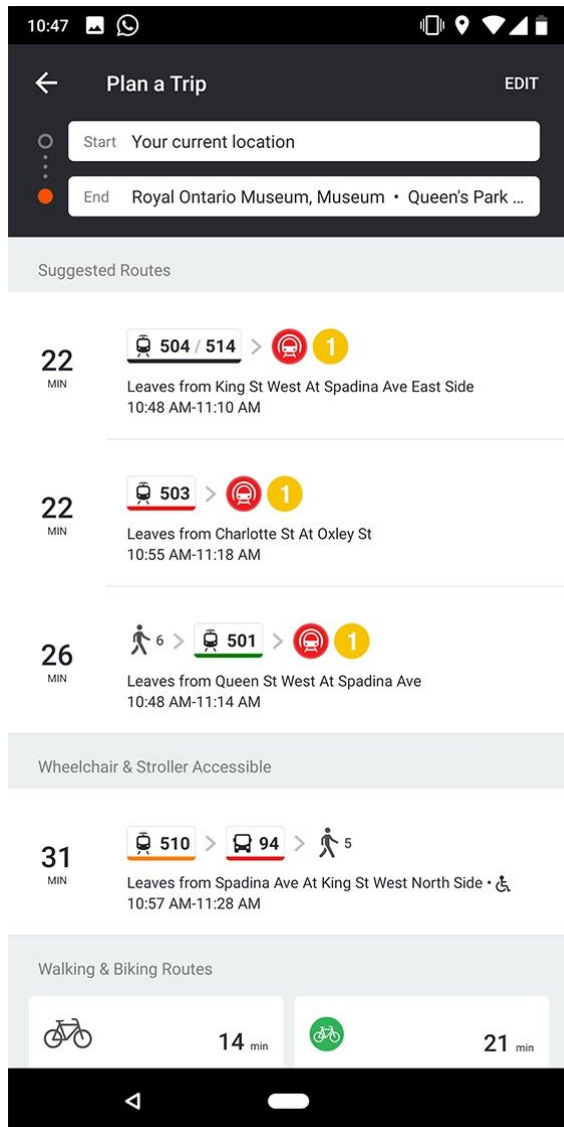


Figura 1: Aplicația Moovit [1]

Moovit este una din cele mai populare soluții de îmbunătățire a serviciilor de transport public. Această aplicație oferă utilizatorilor informații despre mijloacelor de transport pe care le pot folosi pentru a se deplasa. Aceasta are ca și avantaje faptul că oferă informații legate de trenuri, metrouri, etc. De asemenea oferă informații despre timpul aproximativ în care va ajunge un mijloc de transport într-o anumită stație.

Problema cu această aproximare o reprezintă posibilitatea apariției de accidente, trafic, și multe altele care dau peste cap această aproximare, și astfel utilizatorul nu are la dispoziție informații sigure legate de mijloacele de transport.

## 2.2 Google Maps

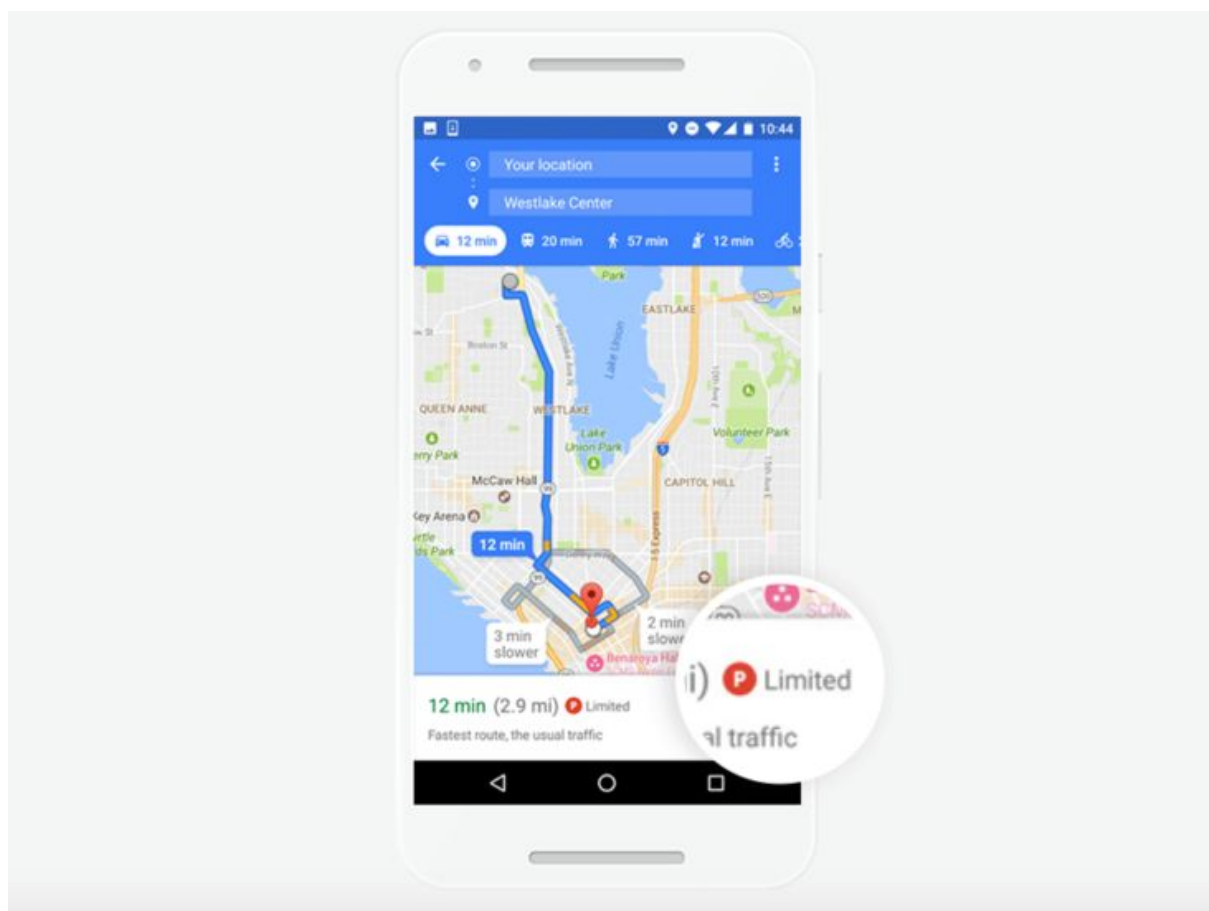


Figura 2: Aplicația Google Maps [2]

Google Maps este probabil cea mai folosită aplicație pentru deplasarea folosind mijloacele de transport în comun. Beneficiile pe care această aplicație le aduce sunt: posibilitatea de a vedea ce mijloace de transport putem folosi pentru a ne deplasa dintr-un loc în altul, inclusiv pe care le putem împreună pentru a merge pe cea mai eficientă rută.

Probabil factorul care face această aplicație să fie atât de folosită este că pe lângă faptul că livrează la o calitate de nivel înalt serviciile pe care le oferă, vine preinstalată pe majoritatea telefoanelor, tabletelor și este disponibilă și în browser.

Problema însă cu această aplicație este aceeași cu cea de la Moovit; și anume că deși oferă și aproximații legate de când va ajunge un mijloc de transport în stație, tot nu oferă informații ce sunt 100% valide cu privire la acest aspect.

### 3.3 HEREITIS

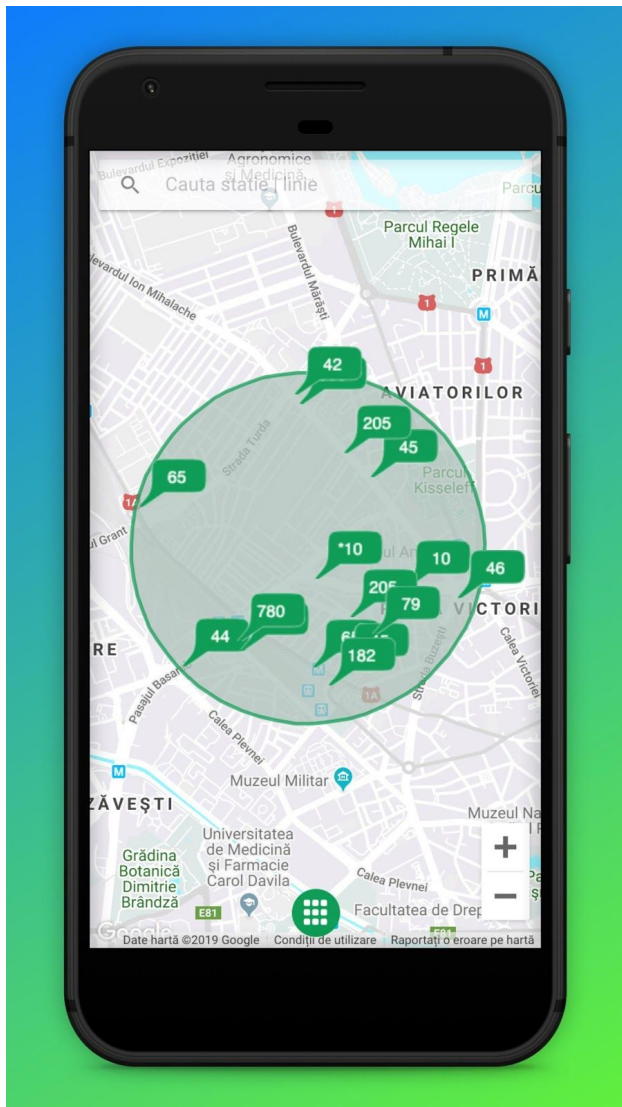


Figura 3: Aplicația HEREITIS [3]

HEREITIS este una din cele mai noi soluții pentru rezolvarea neajunsurilor serviciilor de transport public. Aceasta a pornit cu o idee similară cu PublicTransportMonitor, având la bază principiul de a lăsa la latitudinea utilizatorilor să înregistreze un mijloc de transport. Ulterior, aplicația a trecut de la această funcționalitate la folosirea unor date expuse de CTP ce țin de locația curentă a mijloacelor de transport.

Aplicația este una reușită, însă prezintă unele aspecte ce pot fi îmbunătățite, cum ar fi:

- Interfața: nu mă refer aici la aspectul acesteia ci la ușurința de a o folosi, de a găsi ceea ce caut
- Filtrare: HEREITIS afișează la un moment dat toate mijloacele de transport pe o rază de 10 km., ceea ce face îngreunează utilizatorul din a urmări doar informațiile de interes pentru acesta

Este de discutat dacă folosirea datelor expuse de CTP este modalitatea mai bună de a expune date în aplicație deoarece acele date vin cu anumite limitări, cum ar fi: posibilitatea de a actualiza datele este limitată la posibilitatea de a expune date de către CTP, iar acuratețea datelor nu poate fi constantă, acest fapt fiind o consecință a felului în care sunt expuse datele legate de mijloacele de transport.

PublicTransportMonitor acoperă aceste neajunsuri prin a oferi utilizatorilor posibilitatea de a filtra informațiile afișate, iar datele legate de mijloacele de transport sunt calculate și afișate în funcție locația utilizatorului și datele introduse de acesta.

## CAPITOLUL 3 - DESCRIEREA SOLUȚIEI

### 3.1 Descriere proiect

Proiectul realizat este o aplicație în care orice utilizator are posibilitatea fie de a căuta mijloace de transport în comun a căror locație este vizibilă pentru utilizator, fie să contribuie la aplicație prin a înregistra un mijloc de transport în comun pentru a îi face vizibilă locația pentru ceilalți utilizatori ai acesteia.

Funcționarea aplicației se bazează pe implicarea comunității. Din acest motiv a fost concepută într-un mod cât mai simplist din punct de vedere al interfeței. În același timp au fost luate în calcul ușurința de a fi folosită și oferirea de informații cât mai concise dar utile în același timp, utilizatorul având posibilitatea de a accesa informații mai detaliate dacă dorește. Interfața a fost gândită să fie cât mai simplă deoarece cum aplicația se bazează pe utilizatorii acesteia utilizarea ei trebuie să fie cât mai ușoară și intuitivă.

În calitate de utilizator al aplicației, două roluri sunt posibile.

Unul dintre acestea este cel de utilizator care înregistrează un mijloc de transport în comun. Tot ce trebuie acesta să facă este să introducă în aplicație stația din care pornește, stația în care vrea să ajungă și numele mijlocului de transport cu care dorește să meargă. Pentru a facilita utilizarea aplicației, o listă cu numele fiecărei stații este afișată când utilizatorul începe să scrie numele unei stații, lista respectivă conținând din ce în ce mai puține stații pe parcurs ce utilizatorul scrie, și orice element din listă poate fi selectat pentru a fi scris automat. Apoi, cu acordul acestuia, locația utilizatorului va fi persistată și implicit, locația mijlocului de transport în care se află. După ce a înregistrat călătoria, și implicit, locația mijlocului de transport, utilizatorul va putea urmări progresul călătoriei acestuia până va ajunge la destinație.

Celălalt rol posibil este acela de utilizator care folosește informațiile expuse de către celălalt. Acest utilizator își va introduce stația de plecare și stația în care dorește să ajungă. Rezultatul căutării fiind toate mijloacele de transport a căror locație este expusă print alți utilizatori și a căror traseu se potrivește cu cel al utilizatorului. De asemenea stațiile pe care utilizatorul trebuie să le introducă pot fi completate în mod automat urmărind aceeași procedură cu cea descrisă ulterior pentru utilizatorul care expune date despre un mijloc de transport. Dacă dorește, utilizatorul are de asemenea posibilitatea de a deschide o fereastră cu mai multe detalii legate de un anumite mijloc de transport de care este interesat în mod special.

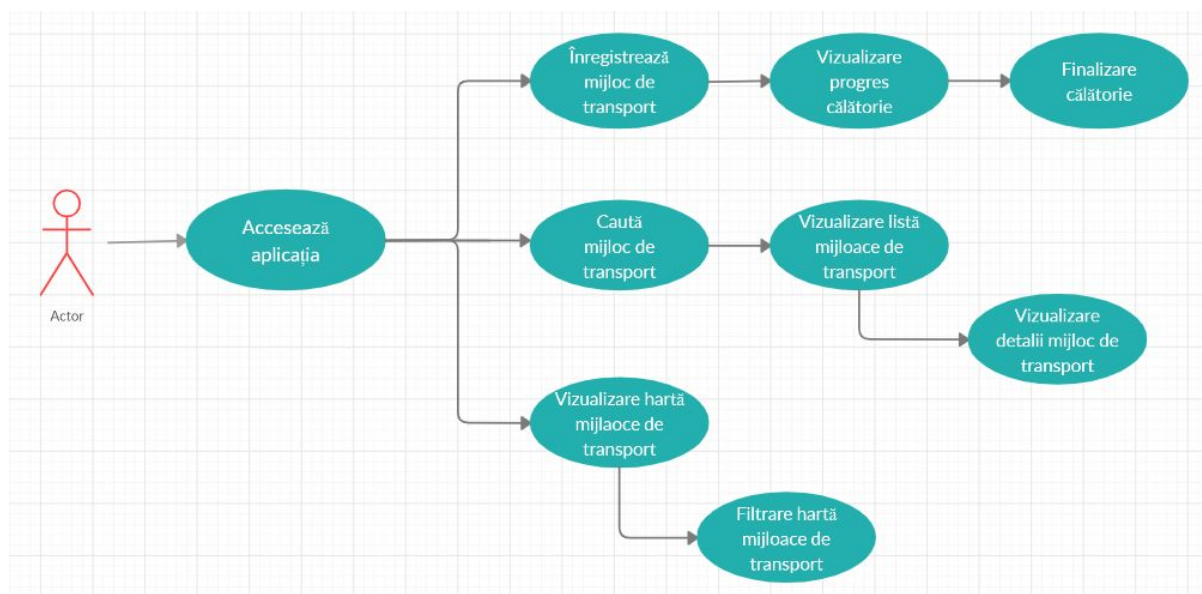


Figura 4: Scenariile de utilizare ale aplicației

### 3.2 Arhitectura proiectului

În diagrama de mai jos este prezentat felul în care a fost proiectată aplicația.

Elementele principale ale aplicației sunt:

- **Utilizatorul:** ce poate fi oricine are acces la internet și este de acord ca aplicația să îi acceseze locația



- **Aplicația web:** expune funcționalitățile de a înregistra sau căuta mijloace de transport
- **Serverul:** orchestrează datele primite de la aplicația web pentru a calcula locațiile tuturor mijloacelor de transport curente și de a le expune
- **HTML Geolocation API:** API-ul folosit la nivel de client pentru a cere date legate de locația curentă a unui utilizator

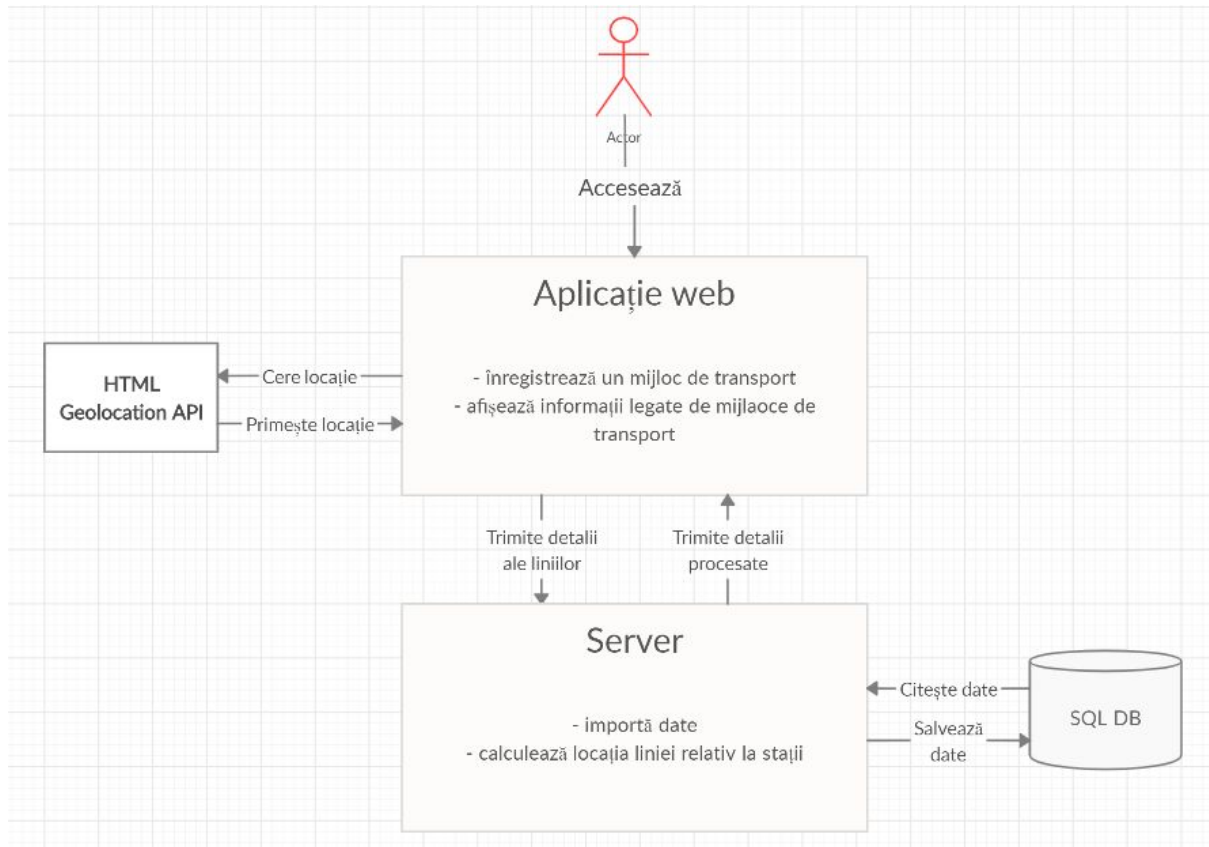


Figura 5: Diagramă contextuală

### 3.2.1 Arhitectură frontend

Pentru client am ales să construiesc o aplicație web folosind Angular, deoarece astfel elimin problema compatibilității pe diferite platforme. Aplicația la nivel de client este construită în concordanță cu Angular Styleguide. Aceasta este împărțită în componente, în cadrul cărora au fost respectate standardele pentru construirea unei aplicații Angular.

Logica aplicației este încapsulată în servicii, fiecare serviciu respectând principiul Single Responsibility. Aceste servicii sunt apoi injectate în componente folosind mecanismul de Dependency Injection încorporat în Angular.

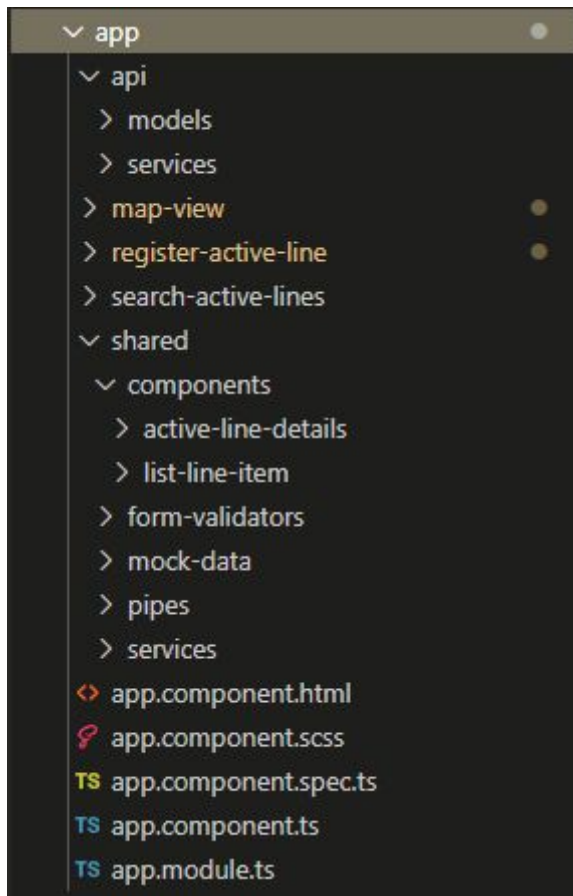


Figura 6: Soluția de frontend

Serviciile sunt de asemenea împărțite în 2 categorii: servicii ce țin de interacționarea cu API-ul construit; tot ce înseamnă citire și trimitere de date către server, și servicii ce țin de logica implementată în frontend, cum ar fi serviciul pentru orchestrarea funcționalităților legate de locația unui utilizator.

De asemenea au fost respectate principii și practici folosite pentru a scrie cod Typescript, cum ar fi: folosirea de tipuri primitive, definirea concretă a tipurilor folosite fie tipuri de parametri, fie tipul de returnare așteptat al unei metode, folosirea de tipuri Union, etc.

### 3.2.2 Arhitectură backend

Soluția de backend este împărțită în 2 mari componente:

- ImportModule: ce ține de partea de importare a datelor de pe site-ul CTP
- MainModule: ce ține de partea de logică a aplicației în sine

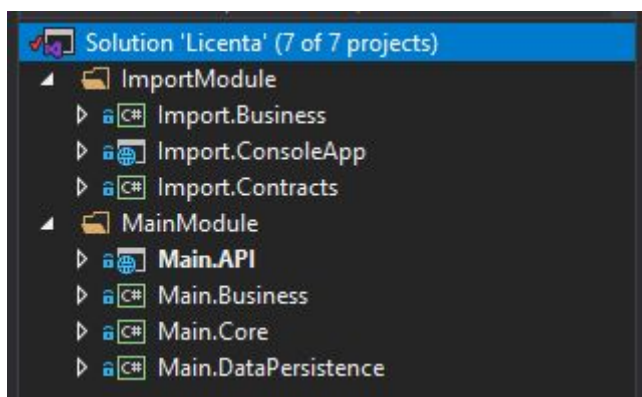


Figura 7: Soluția de backend

În componenta ImportModule se află toată logica pentru aducerea datelor de pe site-ul CTP, maparea acestora la obiecte definite în proiectul Import.Contracts pentru a putea fi persistate în baza de date, și introducerea acestor date în baza de date. Această componentă poate fi extinsă pe viitor pentru a importa date din diferite endpoint-uri, singurele lucruri necesare fiind respectivele endpoint-uri și contractele care trebuie respectate.

În componenta MainModule se află logica efectivă a aplicației. Modulul este structurat astfel: Main.API este API-ul expus cu funcționalitățile necesare, acesta folosește servicii implementate în Main.Business, proiect care la rândul lui respectă contracte pentru repository-urile definite în Main.Core. Serviciile folosite în Main.Core sunt wrappere pentru repository-urile implementate pentru persistarea datelor în baza de date, acestea executând diferite metode ce țin de procesarea datelor înainte de a le scrie în baza de date sau de a le trimite către client.

Pentru operațiunile de scriere în baza de date sau citire din aceasta, nu au fost folosite entitățile definite pentru fiecare obiect, ci DTO-uri specifice fiecărei acțiuni ale unei entități. Astfel, este separată logica pentru persistarea datelor de cea pentru procesarea acestora, și viitoare schimbări necesare pentru operații de scriere/citire din baza de date vor presupune schimbări la nivelul DTO-urilor, dar nu și la nivelul entităților acestora, maparea dintre entități și DTO-uri fiind procesată folosind AutoMapper.

### 3.3 Implementarea proiectului

#### 3.3.1 Înregistrarea unui mijloc de transport

Orice utilizator al aplicației poate înregistra un mijloc de transport. Aceasta este funcționalitatea pe care se bazează această aplicație. Pentru a înregistra un mijloc de transport, utilizatorul trebuie să navigheze către pagina **Register Line** și să introducă în aceasta stația din care pornește, stația în care dorește să ajungă, și numele mijlocului de transport în care s-a urcat/se urcă. Pentru a ușura sarcinile utilizatorului, atunci când începe să scrie numele unei stații, o listă cu toate opțiunile acestuia vor fi afișate. Această listă se scurtează pe parcurs ce utilizatorul completează câmpul respectiv, și orice stație din listă poate fi selectată și ca rezultat numele acesteia este completat în mod automat în câmpul respectiv.

The illustration shows a user interface for registering a transport line. At the top, there are three input fields: 'Start station' with the value 'Sala Polivalenta', 'Destination station' with the value 'Universitate', and 'Line name' with the value '28'. Below these fields is a 'Submit' button. A large, light blue location pin icon is positioned behind the form. Below the 'Submit' button, a message reads 'Successfully registered line!'. Underneath this message is an illustration of a person sitting on a large sphere, interacting with a computer monitor that displays a blue checkmark. Below the illustration is a button that says 'Navigate to the created line'. The entire scene is set against a background of stylized buildings and trees.

Figura 8: Formular de înregistrare al unei linii

Start station  
Sala Polivalenta

Destination station  
uni

Universitate  
Piata Unirii

Submit

Figura 9: Funcționalitatea de auto completare

În frontend, aceste informații sunt centralizate și procesate astfel:

```
private handleLineRegister(): void {
  this._locationService.startTracking((position: Position) => {
    let activeLineForCreation = this.buildActiveLine(position);
    let activeLineForPatch = this.buildActiveLineForPatch(position);

    this._subscription = this._activeLineService.getActiveLineObservableByUri(
      this._registeredLineUri,
      activeLineForCreation,
      activeLineForPatch
    )
    .subscribe((response: HttpResponse<ActiveLineForCreation> | ActiveLine) => {
      // null check because PATCH returns NoContent() and thus, null ?
      if (response !== null &&
        'headers' in response) {
        this._registeredLineUri = response.headers.get('location');
        this._subscription.unsubscribe();
      }
    });
  });
}
```

Figura 10: Implementarea pentru înregistrare și urmărire a unei linii

1. În primă instanță, este apelată metoda **startTracking** ce aparține de **LocationService**. În cadrul acesteia este apelată metoda **watchPosition** din Geolocation API. Funcția apelată în cazul în care locația utilizatorului este identificată cu succes este cea definită în **handleLineRegister**, după cum putem observa în figura de mai sus.

```

public startTracking(successFunction: PositionCallback): void {
  if (navigator.geolocation) {
    this._watchId = navigator.geolocation.watchPosition(successFunction, this.errorFunction, this._trackingOptions);
  }
}

```

Figura 11: Implementarea metodei de urmărire a API-ului

2. Pasul următor este reprezentat de crearea a două obiecte diferite, unul pentru a crea o nouă linie, celălalt pentru a modifica linia deja creată. Ambele obiecte sunt create în cadrul acestei metode deoarece pe tot parcursul călătoriei utilizatorului locația acestuia este urmărită prin metoda **startTracking**, în primă instanță linia va fi creată, URI-ul acestei linii va fi salvat local și la acel URI se vor face modificările cu obiectul creat special pentru această operație
3. Logica ce ține de decizia de a adăuga o nouă linie sau de a modifica una existentă este încapsulată în metoda **getActiveLineObservableByUri** din **ActiveLineService**.

```

public getActiveLineObservableByUri(
  activeLineUri: string,
  activeLineForCreation: ActiveLineForCreation,
  activeLineForPatch: ActiveLineForPatch): Observable<HttpResponse<ActiveLineForCreation> | ActiveLine> {
  return of(activeLineUri)
    .pipe(
      mergeMap(uri =>
        iif(() => uri === '',
          this.registerActiveLine(
            activeLineForCreation),
          this.patchActiveLine(
            activeLineUri,
            activeLineForPatch))
      )
    );
}

```

Figura 12: Implementare alegere dinamică a acțiunii pe linie

Aceasta se folosește de operatorul **RxJS iif** pentru a apela ori metoda de creare definită în serviciu, ori pe cea de modificare.

În backend, informațiile primite de la frontend sunt procesate astfel:

1. În cazul creării unei noi linii, informațiile primite de la frontend sunt folosite pentru a identifica toate detaliile liniei introduse de

către utilizator. Totodată sunt adăugate detalii calculate specifice pentru linia adăugată

```
2 references | Adelin Pamint, 4 days ago | 1 author, 4 changes
public Guid CreateActiveLine(ActiveLineForInsertDto activeLineForInsertDto)
{
    var activeLine = _mapper.Map<ActiveLine>(activeLineForInsertDto);
    activeLine.NextStation = _activeStationService.GetNextStationOf(activeLine).Name;
    activeLine.NextStationStatus = NextStationStatus.JustLeftStation;

    _activeLineRepository.AddActiveLine(activeLine);
    _activeLineRepository.SaveChanges();

    return activeLine.Id;
}
```

Figura 13: Crearea liniei pe server

În cazul unei modificări a unei linii existente, distanța parcursă de aceasta se calculează folosind formula Haversine. În funcție de rezultatul calculelor, detaliile liniei sunt modificate.

```
2 references | Adelin Pamint, 4 days ago | 1 author, 1 change
private decimal HaversineDistanceInKm(decimal currentLatitude, decimal currentLongitude, decimal latitude, decimal longitude)
{
    double radius = 6371;
    var lat = (double)(latitude - currentLatitude) * (Math.PI * 180);
    var lng = (double)(longitude - currentLongitude) * (Math.PI * 180);
    var h1 = Math.Sin(lat / 2) * Math.Sin(lat / 2) +
        Math.Cos((double)currentLatitude * (Math.PI * 180)) *
        Math.Cos((double)latitude * (Math.PI * 180)) *
        Math.Sin(lng / 2) * Math.Sin(lng / 2);
    var h2 = 2 * Math.Asin(Math.Min(1, Math.Sqrt(h1)));
    return (decimal)(radius * h2);
}
```

Figura 14: Metoda pentru calcularea distanței între puncte

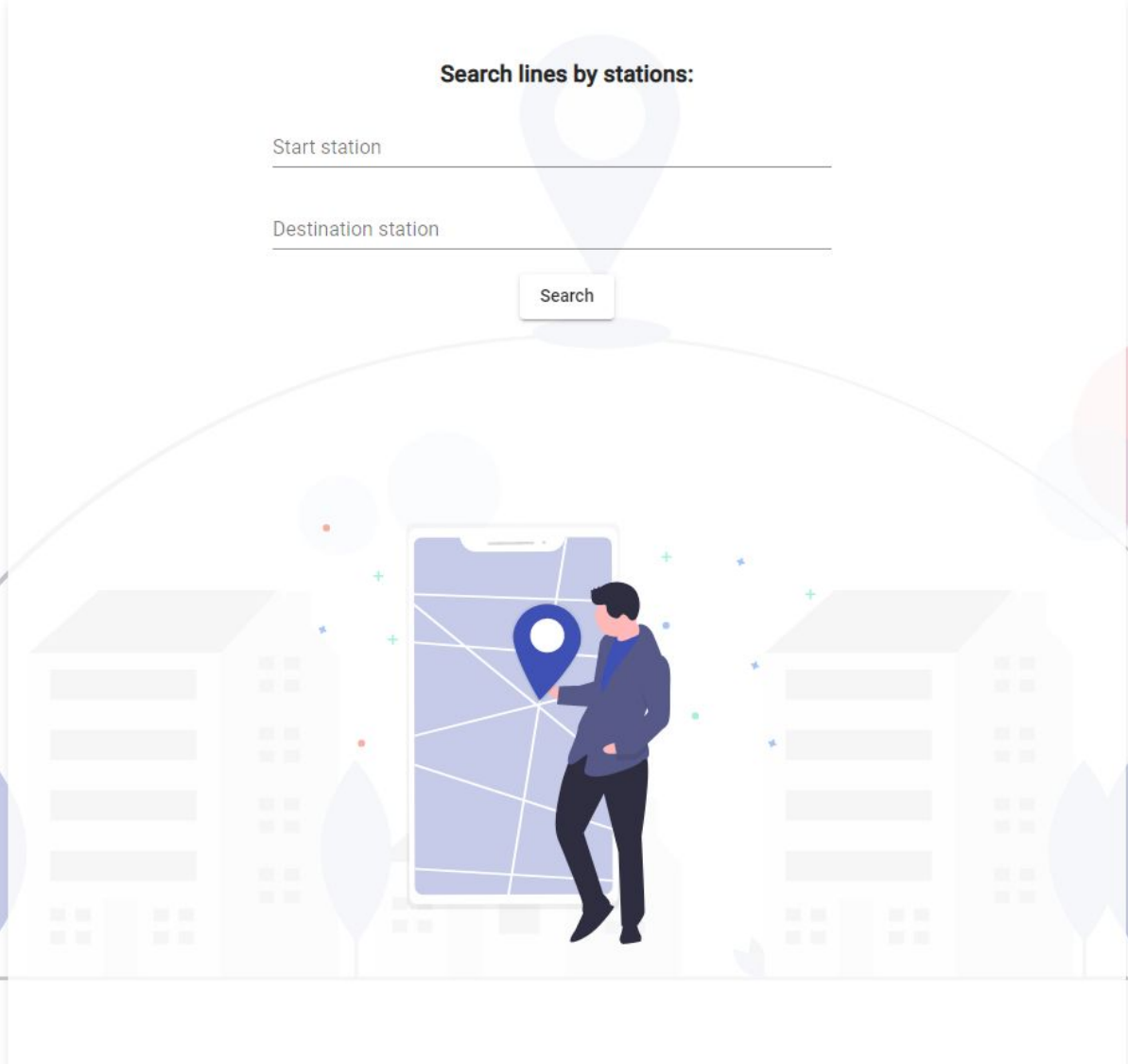
### 3.3.2 Căutarea unui mijloc de transport

Orice utilizator poate căuta mijloace de transport de interes pentru acesta. Pentru a putea efectua o căutare, utilizatorul trebuie să introducă numele stației din care dorește să pornească și numele stației în care dorește să ajungă. Pentru a facilita sarcinile pe care acesta trebuie să le îndeplinească, atunci când utilizatorul începe să scrie numele unei stații, o listă cu toate stațiile disponibile îi este afișată.



Această listă este filtrată în mod constant în funcție de caracterele introduse de către utilizator, și orice stație din listă poate fi selectată pentru a completa în mod automat un câmp.

În funcție de rezultatele căutării, o listă de informații legate de liniile ale căror locație este expusă de către alți utilizatori este afișată.



**Search lines by stations:**

Start station \_\_\_\_\_

Destination station \_\_\_\_\_

Search

Figura 15: Formular pentru căutarea unei stații



În frontend, informațiile introduse de către utilizator sunt centralizate și procesate astfel:

```
protected submit() {  
  this._activeLineService.getActiveLines(  
    this.stationsForm.get('startStation').value,  
    this.stationsForm.get('destinationStation').value  
  ).subscribe(  
    (data: ActiveLine[]) => this.activeLines = data  
  );  
}
```

Figura 16: Implementarea în frontend a căutării de linii

1. În primă instanță este apelată metoda **getActiveLines** din serviciul **ActiveLineService** și primește drept parametri stația de pornire introdusă de utilizator, și pe cea în care acesta dorește să ajungă
2. Pasul al doilea constă în salvarea într-o variabilă locală a datelor recepționate de la server, date ce vor fi ulterior folosite pentru a afișa utilizatorului informații legate de căutarea acestuia

Informațiile afișate utilizatorului reprezintă linii ale căror locație este cunoscută în aplicație. Acestea sunt afișate în modul următor:

**Search lines by stations:**

Start station  
Elena Doamna

Destination station  
Targu Cucu

Line: <b>28</b>	Next station: Palatul Culturii Line status: Running Last updated at: 10:39 PM
Line: <b>42</b>	Next station: Tesatura Line status: Running Last updated at: 10:39 PM
Line: <b>43</b>	Next station: Moara 1 Mai Line status: Running Last updated at: 10:39 PM

Figura 17: Detaliile afișate utilizatorului în urma căutării

În backend, informațiile primite de la frontend sunt procesate astfel:

1. Sunt căutate în baza de date toate liniile ce corespund cu datele introduse de utilizator. Liniile rezultate sunt mapate la **ActiveLineForDisplayDto**, astfel, doar informațiile necesare afișării în frontend sunt expuse.

```

2 references | Adelin Pamint, 14 days ago | 1 author, 2 changes
public IEnumerable<ActiveLine> GetActiveLines(ActiveLineResourceParameters activeLineResourceParameters)
{
    if (string.IsNullOrEmpty(activeLineResourceParameters.StartStation) ||
        string.IsNullOrEmpty(activeLineResourceParameters.DestinationStation))
    {
        return GetActiveLines();
    }

    return GetActiveLines()
        .Where(activeLine => HasStationsInGivenOrder(activeLine, activeLineResourceParameters)) // IEnumerable<ActiveLine>
        .ToList(); // List<ActiveLine>
}

```

Figura 18: Căutarea liniilor în baza de date

### 3.3.3 Vizualizare detalii linie

Componenta de detalii a unei linii presupune, în acest caz o secțiune în care sunt afișate următoarele: numele liniei, traseul pe care aceasta îl urmează, astfel încât utilizatorul să poată deduce direcția în care merge linia respectivă, stațiile acesteia, fiecare stație fiind marcată în 4 modalități posibile:

- **Passed:** semnifică faptul că linia a cărei detalii sunt urmărite a trecut deja prin stația respectivă
- **Current:** semnifică faptul că linia este staționată în respectiva stație
- **Just passed:** acest marcaj se referă la faptul că linia a plecat de curând din acea stație. Distanța parcursă este luată în calcul pentru definirea acestui marcaj
- **Within line range:** acest marcaj este predestinat situațiilor în care linia este în apropierea următoarei stații

Fiecare stație are, de asemenea, ora la care i-a fost atribuită un anumit marcaj.

Detaliile unei linii pot fi vizualizate atât de un utilizator care a înregistrat o anumită linie, cât și de un utilizator care caută anumite linii în aplicație. Scopul acestei funcționalități are, însă, diferite beneficii în funcție de tipul de utilizator care o folosește.

Un utilizator care a înregistrat o linie are ca beneficii rezultate din această funcționalitate faptul că poate vedea un progres live al călătoriei acestuia. Poate vedea atât stațiile pe care le-a parcurs, cât și pe cele pe care i-au mai rămas. Astfel, utilizatorul poate face o aproximare a timpului pe care îl mai are de parcurs.

Un utilizator care caută linii, beneficiază de această funcționalitate prin prisma faptului că poate vizualiza detaliile unei singure linii care îi este folositoare, eliminând, astfel,

efortul suplimentar pe care trebuie să îl depună pentru a urmări o singură linie într-o colecție cu număr variabil de linii. De asemenea, în cazul în care utilizatorul nu își poate da seama de locația liniei doar din stația în care se află sau care este următoarea stație în care urmează să ajungă linia, utilizatorul se poate folosi de modul detaliat de vizualizare a liniei pentru a vedea câte stații sunt între cele din care acesta dorește să plece, și cea în care sau spre care se îndreaptă linia.

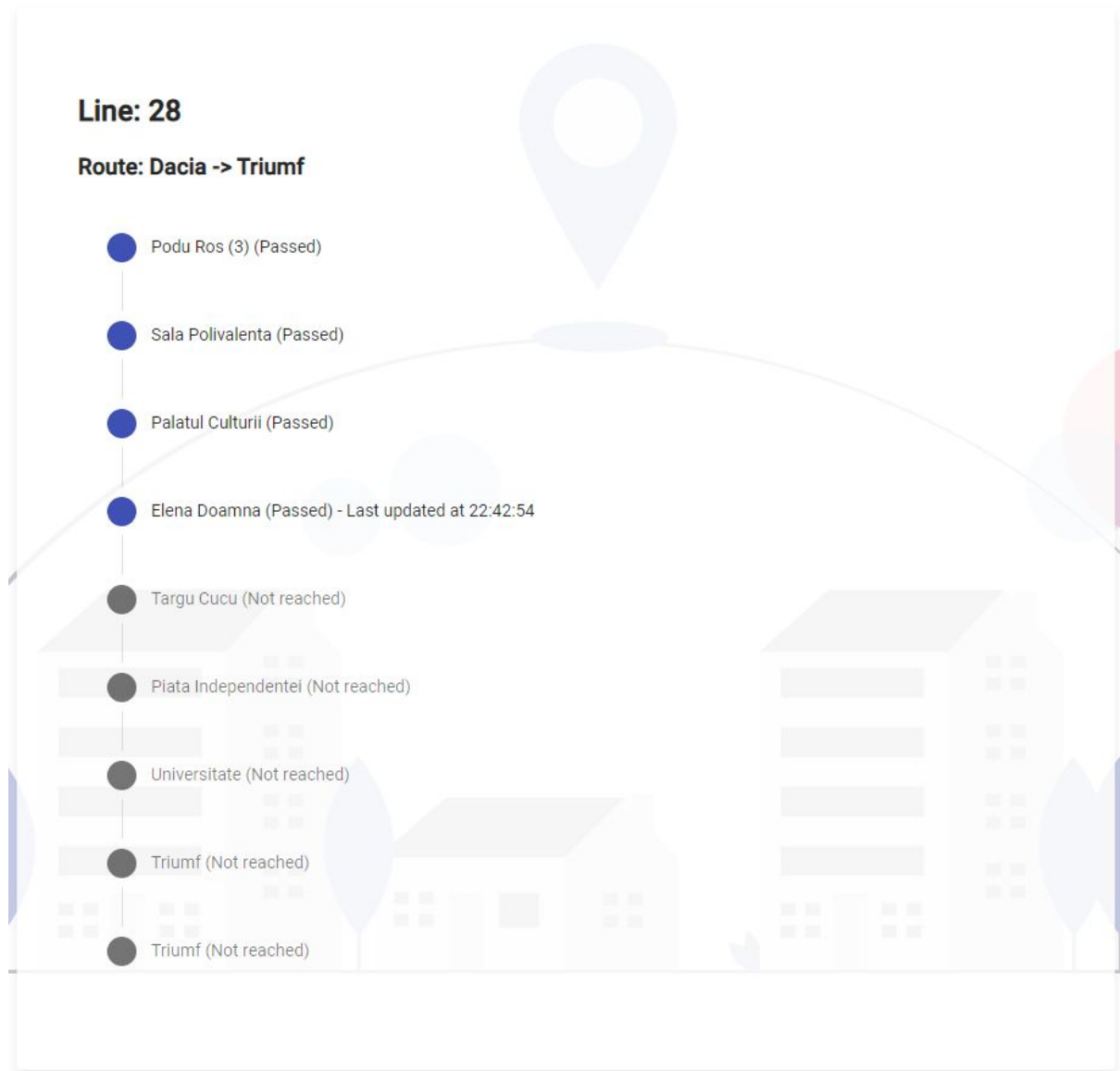


Figura 19: Detaliile unei linii

### 3.3.4 Vizualizarea liniilor pe hartă

Această componentă presupune vizualizarea pe o hartă a tuturor liniilor a căror locație este expusă prin intermediul locației utilizatorilor. Acest mod de vizualizare vine în întâmpinarea utilizatorilor ce sunt obișnuiți cu o vizualizare mai amplă a detaliilor expuse în aplicație.

Detaliile puse la dispoziția utilizatorilor în acest mod de vizualizare sunt următoarele:

- Harta cu împrejurimile, astfel utilizatorul poate vedea direct unde se află o anumită linie
- Liniile active și poziția lor exactă

Liniile vizibile pe hartă pot fi filtrate după nume, astfel în cazul în care harta devine prea aglomerată pentru a putea fi folosită eficient, utilizatorul poate reduce informațiile vizibile doar la cele de interes pentru acesta.

Informațiile afișate pentru fiecare linie pe hartă sunt următoarele: numele acesteia și traseul pe care îl parcurge, astfel utilizatorul poate vedea dacă linia respectivă merge pe un traseu de interes pentru acesta sau nu.



Figura 20: Vizualizarea liniilor pe hartă

### 3.3.5 Stocarea datelor folosind SQL și Entity Framework Core

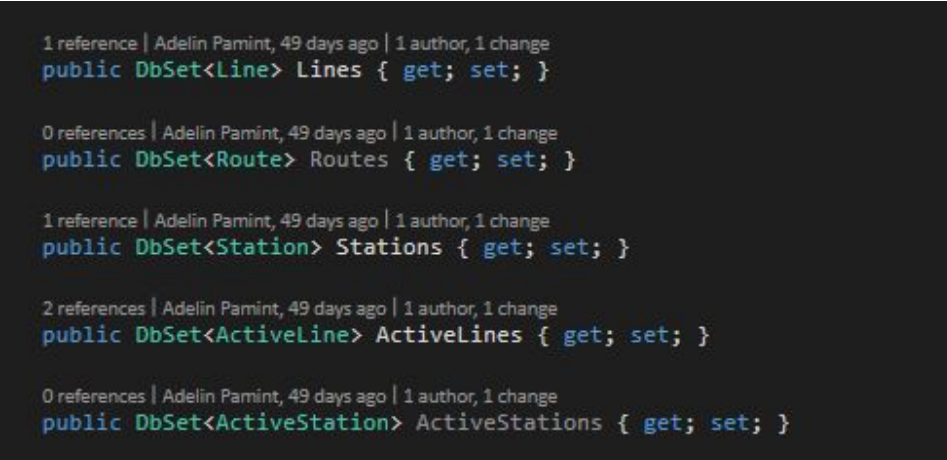
Pentru stocarea datelor am ales să folosesc ORM-ul Entity Framework Core împreună cu o bază de date SQL. Am ales să folosesc EF Core deoarece acest framework vine cu toate funcționalitățile necesare pentru a manageria persistența datelor într-o bază de date.

Logica folosită împărțirea tabelelor și a entităților este asemănătoare cu cea aplicată la împărțirea logicii aplicației astfel încât au rezultat 2 categorii:

- tabelele și entitățile folosite pentru a importa datele de pe site-ul CTP
- tabelele și entitățile folosite pentru a ține evidența liniilor a căror locație este expusă de către utilizatori

Tabelele și entitățile din cele 2 categorii nu sunt conectate în vreun fel, acestea fiind complet separate.

Entitățile folosite pentru logica aplicației sunt bazate pe cele pentru a importa date, însă au diferite modificări pentru a se plia pe nevoile aplicației.



```
1 reference | Adelin Pamint, 49 days ago | 1 author, 1 change
public DbSet<Line> Lines { get; set; }

0 references | Adelin Pamint, 49 days ago | 1 author, 1 change
public DbSet<Route> Routes { get; set; }

1 reference | Adelin Pamint, 49 days ago | 1 author, 1 change
public DbSet<Station> Stations { get; set; }

2 references | Adelin Pamint, 49 days ago | 1 author, 1 change
public DbSet<ActiveLine> ActiveLines { get; set; }

0 references | Adelin Pamint, 49 days ago | 1 author, 1 change
public DbSet<ActiveStation> ActiveStations { get; set; }
```

Figura 21: Entitățile folosite în aplicație

Entitățile create pentru a importa datele sunt create pe baza datelor expuse de către site-ul CTP, modelul acestora fiind asemănător cu JSON-urile expuse pe site, această decizie a fost luată pentru a minimiza nivelul de confuzie în cazul comparării datelor expuse pe site cu cele importate în aplicație. Aceste entități sunt împărțite astfel: Line, Station și respectiv Route, aceasta din urmă reprezentând o tabelă de legătură pentru a realiza o relație între tabelele Lines și respectiv, Stations, după cum se poate vedea în figura următoare:

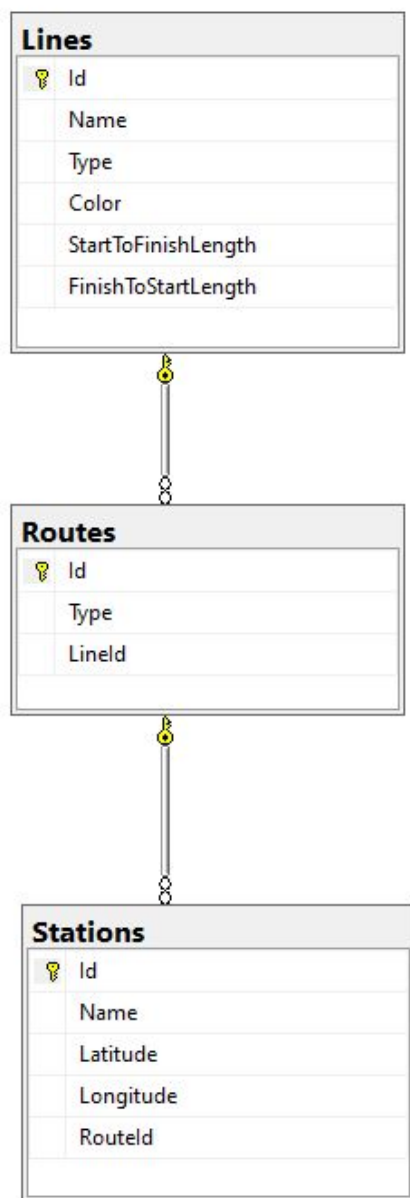


Figura 22: Schema tabelor folosite pentru importare de date

Relațiile între aceste tabele sunt următoarele:

- Între tabela **Lines** și tabela **Routes** este o relație de tip one to many, astfel, o linie poate avea una sau mai multe rute. Această relație are la bază faptul că în timp ce tramvaiele au un singur traseu pe care îl parcurg, autobuzele au 2, de aici nevoia de a avea mai multe rute pentru o linie

- Între tabela Routes și tabela Stations este o relație de tip one to many, deoarece o rută este compusă dintr-o listă de stații parcurse într-o anumită ordine

Entitățile ActiveLine și ActiveStation reprezintă adaptările entităților Line și respectiv, Station. Acestea sunt folosite în cadrul aplicației pentru a ține evidența liniilor a căror locație este expusă prin intermediul locației utilizatorilor.

În cazul acestor entități nu este necesară o tabelă de legătură deoarece o linie activă are la un moment dat un singur traseu, deci o singură rută, caz în care lista de stații ce compun o rută pot fi referențiate direct în cadrul liniei active.

Aceste tabele pot fi vizualizate în figura următoare:

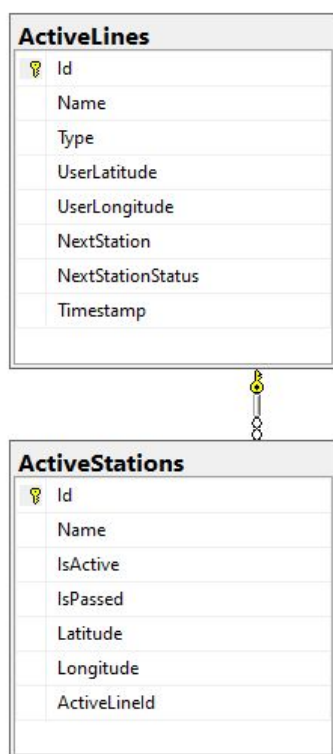


Figura 23: Schema tabelor folosite pentru logica aplicației

Relațiile între aceste tabele sunt următoarele:

- Între tabela ActiveLines și tabela ActiveStations este o relație de tip one to many

Schema completă a bazei de date poate fi vizualizată în figura următoare:



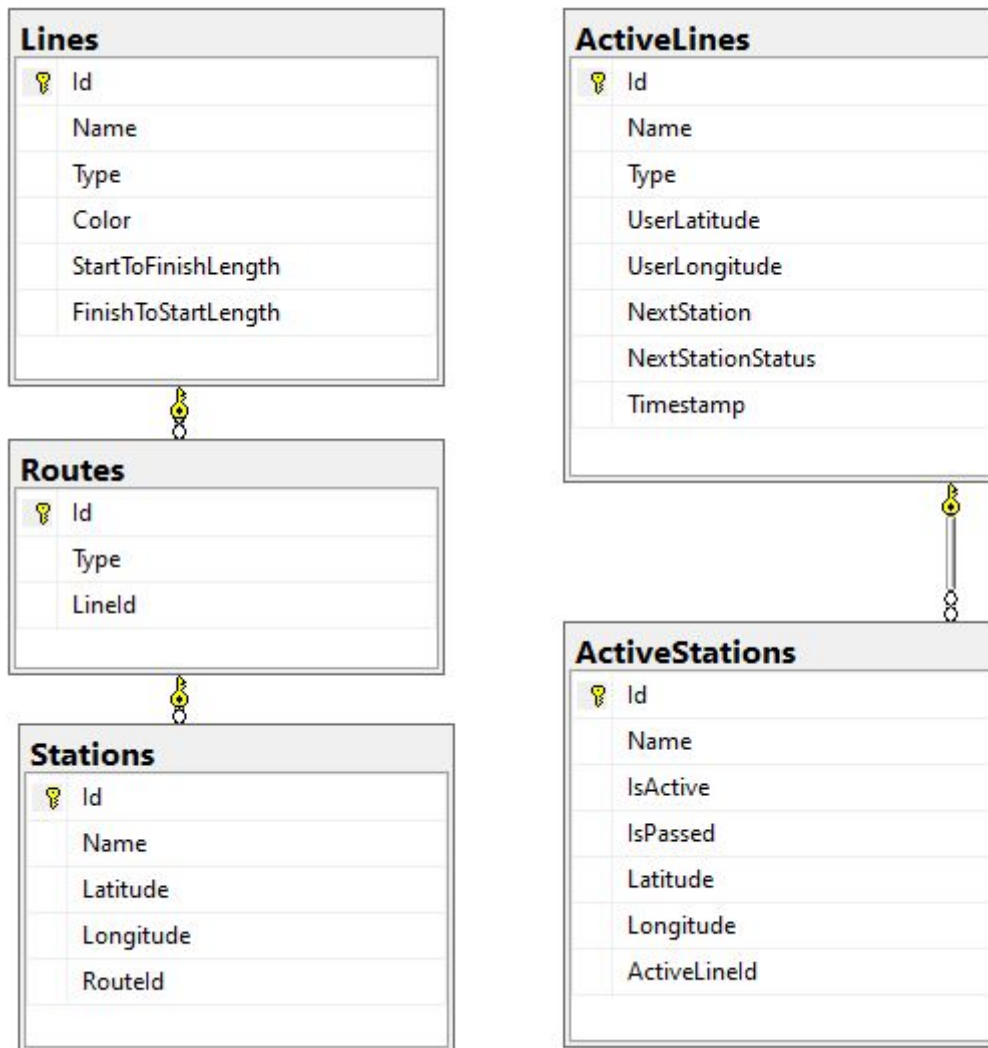


Figura 24: Schema completă a bazei de date

# CONCLUZIILE LUCRĂRII

## 1. Concluzii generale

Scopul aplicației este de a facilita utilizarea serviciilor de transport public, prin a oferi utilizatorilor informații concrete despre locația liniilor ce reprezintă un interes pentru aceștia, și nu aproximări care deși merg spre un nivel de detaliu mai mare, cum ar fi o aproximare atât a orei cât și a minutului în care o linie va ajunge într-o anumită stație, dar a căror acuratețe fluctuează.

Personal, m-am găsit de numeroase ori în postura de a fi nevoit să sacrific o parte din timpul meu pentru a alocă suficient timp de așteptat în stație pentru a putea ajunge la destinație, situație care este cu atât mai neplăcută cu cât este mai limitat timpul pe care îl am la dispoziție și mai important să ajung la destinație la timp, de exemplu într-o zi cu examen. Acest tip de situație putea fi evitată dacă aș fi avut o sursă de încredere ce să ofere informații legate de locația mijloacelor de transport, astfel aș fi putut să-mi organizez timpul luând în calcul aceste informații.

Abordarea folosită în această aplicație este de a folosi locația utilizatorilor pentru a o putea identifica pe cea a mijloacelor de transport public cu care aceștia călătoresc. Astfel, utilizatorii se împart în 2 categorii: cei care expun locația liniilor prin intermediul locației lor, și cei care caută linii printre cele expuse de ceilalți utilizatori.

În calitate de utilizator ce caută linii pentru călătoria sa, detaliile legate de linii pot fi vizualizate în 3 moduri:

- **Minimal:** toate liniile potrivite căutării sunt afișate cu următoarele detalii: numele acesteia, traseul pe care îl parcurge, următoarea stație și statusul liniei relativ la distanța acesteia de stații.
- **Detaliat:** detaliile unei singure linii sunt afișate, acestea conținând: numele acesteia, traseul pe care îl parcurge, un număr mai mare de stații astfel încât utilizatorul poate vedea câte stații mai are linia de parcurs până la acesta, stațiile sunt marcate corespunzător în funcție de distanța acestora de linie, sau dacă deja au trecut prin respectivele stații

- **Hartă:** toate liniile sau liniile filtrate de către utilizator pot fi vizualizate pe o hartă astfel încât utilizatorul să poată avea o privire de ansamblu a opțiunilor curente din punct de vedere al mijloacelor de transport

Dezvoltarea acestei soluții mi-a oferit prilejul de a învăța să construiesc o aplicație end to end, m-a provocat să gândesc o arhitectură ce să se îmbine atât funcționalități pentru importare de date, prelucrarea acestora, cât și extensibilitate, cuplaj mic, astfel încât aplicația să poată fi extinsă pe viitor la numeroase orașe. A fost o provocare la nivel arhitectural, una din care am avut multe de învățat și mi-a extins viziunea din punct de vedere al abordărilor posibile pe partea de structurare a unei aplicații.

## 2. Direcții de viitor

În viitor, ca și funcționalitate ar putea fi adăugată o pagină cu detalii despre linii ce se potrivesc cu cele mai comune căutări ale utilizatorului, cum de multe ori căutările se vor repeta, de exemplu linii pentru trasee acasă - facultate, acasă - loc de muncă, etc., aceasta ar fi o adiție ce ar aduce valoare, deoarece l-ar privi pe utilizator de a repeta aceeași acțiune zilnică de căutare.

O funcționalitate ce ar veni în completarea celei legate de cele mai comune căutări ar putea fi un sistem de notificări, care să trimită notificări utilizatorului legate de linii care de obicei sunt de interes pentru acesta care sunt în apropiere, eventual programate pe un plan pe zile ale săptămânii, de exemplu utilizatorul să poată seta în aplicație să primească notificări în fiecare luni la ora 10:00 legate de trasee pe care să le introducă manual sau legate de traseele cele mai comune ale utilizatorului.

De asemenea, este posibilă pe viitor trecerea de la dependența de utilizatorii aplicației pentru a urmări locația mijloacelor de transport la utilizarea datelor expuse de CTP în cazul în care acestea devin mai stabile și mai concrete.

## BIBLIOGRAFIE

1. Angular: <https://angular.io/>
2. Angular Material: <https://material.angular.io/>
3. HTML5 Geolocation API:  
[https://developer.mozilla.org/en-US/docs/Web/API/Geolocation\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API)
4. Entity Framework Core: <https://docs.microsoft.com/en-us/ef/core/>
5. ASP.NET Core 3.0:  
<https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.0>
6. AutoMapper: <https://docs.automapper.org/en/stable/Getting-started.html>
7. Formula Haversine: [https://en.wikipedia.org/wiki/Haversine\\_formula](https://en.wikipedia.org/wiki/Haversine_formula)
8. SSMS:  
<https://docs.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver15>
9. HEREITIS: <http://hereitis.ro/>
10. Moovit: <https://moovitapp.com/>
11. Google Maps: <https://www.google.ro/maps>
12. Google Maps API: <https://angular-maps.com/api-docs/agm-core/components/agmmmap>
13. unDraw: <https://undraw.co/illustrations>

## Link-uri poze:

1. [1] <https://cdn.mobilesyrup.com/wp-content/uploads/2018/05/moovitTripSearch.jpg>
2. [2]  
<https://techcrunch.com/wp-content/uploads/2017/01/google-maps-parking.png?w=730&crop=1>
3. [3]  
<https://image.winudf.com/v2/image1/Y29tLnJlYWx0aW1lLmJlc2VzX3NjcmVlbl8wXzE1NjIwMTYwMDRfMDQ2/screen-0.jpg?fakeurl=1&type=.jpg>
4. [4] <https://angular.io/generated/images/guide/architecture/overview2.png>

# ANEXE

## 1. Angular

Aplicația a fost dezvoltată la nivel de client folosind framework-ul web Angular, versiunea 8.3.20. Am ales Angular deoarece prezintă numeroase avantaje ce contribuie la productivitate și la dezvoltare, cum ar fi:

- Linia de comandă: aceasta ajută la crearea automată de componente, servicii, testarea și rularea aplicației, etc.
- Suport: multe IDE-uri prezintă numeroase funcționalități pentru Typescript, limbajul folosit în dezvoltarea aplicațiilor Angular, cum ar fi: intellisense, code snippets, etc.

Mai jos poate fi vizualizată arhitectura Angular.

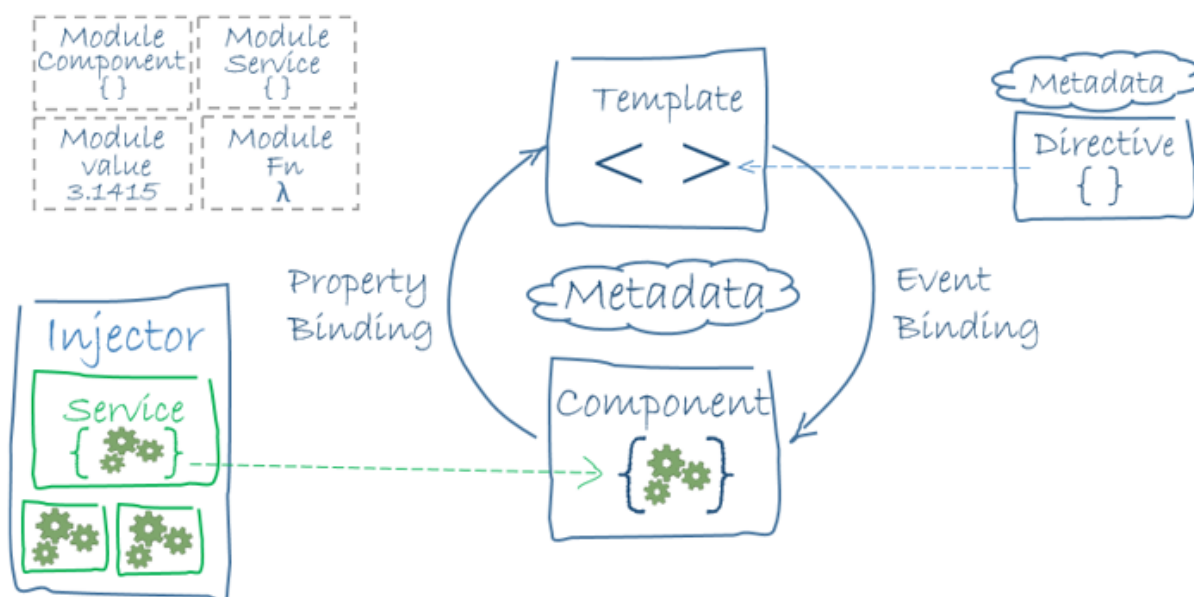


Figura 25: Arhitectura unei aplicații Angular [4]

După cum putem observa, o aplicație Angular se bazează pe componente, în cadrul cărora sunt încapsulate fișierele .html, .css, .ts și .spec.ts pentru fiecare funcționalitate. În aceste componente putem de asemenea injecta servicii definite de noi înșine sau servicii predefinite precum cel de rutare, Angular având suport pentru Dependency Injection.

## **2. Angular Material**

Angular Material este o librărie pentru Angular ce conține componente predefinite.

Motivele pentru care am ales această librărie sunt:

- Consistență în interfață
- Ușurință în utilizare
- Design plăcut
- Performanță

## **3. ASP.NET Core**

Partea de server a fost construită folosind ASP.NET Core. Acesta este un framework open-source, cross-platform, modern, ce facilitează construirea de aplicații web end to end, API-uri, etc. Motivele pentru care am ales ASP.NET Core sunt următoarele:

- Prezintă o integrare ușoară cu framework-uri client-side
- Suport pentru Dependency Injection încorporat
- Performanță

## **4. Entity Framework Core**

Am ales Entity Framework Core pentru a manageria partea aplicației ce ține de interacțiunea cu baza de date. Motivul principal pentru care am ales această tehnologie este faptul că oferă posibilitatea de a lucra cu date prin intermediul unor clase, și nu direct cu baza de date.

## **5. AutoMapper**

AutoMapper este o librărie scrisă cu scopul de a privi dezvoltatorii de scrierea repetitivă a acelorași metode de mapare între obiecte. Această librărie se bazează pe convenții, fapt ce ușurează folosirea acestuia. Am ales să folosesc AutoMapper pentru facilitarea mapărilor de la entități la DTO-uri, și vice versa.

## 6. HTML5 Geolocation API

Am ales să folosesc acest HTML5 Geolocation API deoarece este integrat direct în browser, acest fapt facilitând cu mult construirea logicii în jurul acestuia. Acest API are numeroase avantaje, fapt pentru care am decis să îl folosesc în construirea aplicației. Unele dintre aceste avantaje sunt:

- Compatibilitate: fiecare browser are compatibilitate completă cu acest API
- Securitate: funcționează doar pe HTTPS, permisiunea utilizatorului este cerută înainte de a prelua orice informație legată de locația acestuia
- Este ușor de folosit, prezintă funcționalități relevante în contextul identificării/urmăririi locației.

## 7. Google Maps API

Am folosit Google Maps API pentru a implementa modul de vizualizare pe hartă.

Motivele pentru care am ales acest API sunt următoarele:

- Popularitatea: Google Maps este unul din cele mai populare servicii oferite de Google, prin consecință, majoritatea utilizatorilor vor fi deja familiarizați cu interfața
- Performanța
- Stabilitatea
- Este ușor de folosit