MINISTERUL EDUCAȚIEI ŞI CERCETĂRII ŞTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

SISTEM MOBIL PENTRU LOCALIZARE SI CALCULARE RUTE DE TRANSPORT URBAN

LUCRARE DE LICENȚĂ

Absolvent: **Ciprian Adrian ANTAL**

Coordonator ştiinţific: **asis. ing. Cosmina IVAN**

2016

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Ciprian Adrian ANTAL**

TITLUL LUCRĂRII DE LICENŢĂ

1. **Enunţul temei:** *Scopul acestui proiect este de a defini, proiecta şi implementa un sistem mobil care va permite calcularea de rute folosind mijloacele de transport în comun dintr-un oraş si in acelaşi timp să ofere toate informaţiile şi serviciile necesare astfel încat deplasarea cu aceste mijloace să fie cât mai uşoară.*

2. **Conţinutul lucrării:** *Introducere, Obiectivele proiectului, Studiu bibliografic, Analiză şi Fundamentare teoretică, Proiectare de detaliu şi implementare, Testare şi Validare, Manual de instalare şi utilizare, Concluzii, Bibliografie, Anexa*

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**:

5. **Data emiterii temei:** 1 noiembrie 2015

6. **Data predării:** 30 Iunie 2016

Absolvent: _____

Coordonator ştiinţific: _____

Cuprins

Capitolul 1. Introducere

1.1. Contextul proiectului

În ultima perioadă, de când oamenii au tot mai puţin timp la dizpoziţie şi sunt într-o alergare continuă, dorind ca orice acţiune care o fac să fie finalizată cât mai repede cu putință, aplicaţiile mobile au apărut ca o soluţie aşteptată de toată lumea spre rezolvarea acestei probleme şi a fost rapid îmbrăţişată de un mare număr al populaţiei ceea ce a dus la o dezvoltare uluitoare a acestor tehnologii în ultmii zece ani. Datorită acestei evoluţii, acum, în zilele noastre, orice fel de aplicaţie poate fi proiectată şi dezvoltată pe aceste platforme.

Evoluția aplicaţiilor mobile s-a intensificat semnificativ în ultimii ani, rezultatul fiind acesta care este cât se poate de vizibil: mii şi zeci de mii de aplicaţii disponibile pentru toate sistemele de operare mobile, în diferite domenii: sport, finanţe, cultural, muzică, filme etc.

Principalele tehnologii mobile cunoscute azi şi unde numărul de aplicaţii cresc în fiecare zi sunt binecunoscutele: iOS şi Android. Sistemul de operare iOS este cel care este prezent pe toate dizpozitivele produse de Apple, iar Android pe diferite telefoane (ex Samsung, HTC etc).

De asemenea, împreună cu evoluţia tehnologiilor mobile tot în aceşti ultimi ani au evoluat extrem de mult şi sistemele de navigaţie. Acestea sunt prezente peste tot: maşinile au sisteme de navigaţie, există dispozitive speciale pentru acest lucru, dar până şi telefoanele mobile oferă posibilitatea de instalare a asftel de aplicaţii. Acum se par normale oricărui om şi ne gândim cum am putut trăi fără ele în trecut, lucru care încet începe să se aplice şi în cazul telefoanelor mobile, existând deja tot mai multe aplicaţii care permit utilizatorului să facă aproape orice direct de pe telefonul mobil: plata facturi, cumpărare de pe diverse site-uri, controlarea dispozitivelor electrice din casă şi multe alte lucruri care cândva se credeau a fi doar ficţiune, astăzi se dezvoltă sub ochii noştri şi ajung să ni se pară aşa de normal încât parcă au existat întotdeauna.

Astăzi, trăind în acest context, s-a considerat că cel mai potrivit sistem care să rezolve tema aleasă ar fi unul în domeniul acesta, a tehnologiilor mobile şi a sistemelor de navigaţie. Astfel îmbinând aceste sfere ale tehnologiei s-a ajuns la ideea de a crea un sistem pentru eficientizarea transportului în comun. În ultima perioada tot mai multe companii s-au axat dezvoltarea proiectelor în aceasta arie, însă nefiind acoperită în toate aspectele de sistemele actuale, dar în scurt timp şi acest domeniu va suferi impactul tehnologiei şi vor fi oferite sisteme care să funcţioneze în orice oraş al lumii.

1.2. Motivarea temei alese

Evoluţia tehnologiei şi a GPS-ul din ultimii ani ne duce cu gândul la ideea de a implementa si folosi sisteme de navigaţie şi indicaţii de orientare capabile sa ofere asistență în orice fel de locații, chiar și în interiorul clădirilor.

După cum s-a precizat şi mai sus, în domeniul navigaţiei folosind mijloacele de transport încă mai există un există o lipsa destul de mare în ceea ce priveşte sistemele de navigaţie, deoarece până acum pricipala ramură pe care s-au axat sistemele de navigaţie sunt cele rutiere.

Aspectul acesta, împreună cu dorinţa de a fi aprofundate tehnologiile mobile, în special ramura iOS, dar şi părţi de algoritmică pentru a aplică algoritmii teoretici studiaţi în cadrul cursurilor pe ceva practic, a fost principala motivitie înspre a alege implementarea unei soluţii care să rezolve deplasarea din punctul A în punctul B într-un oraş, folosind mijloace de transport în comun dar şi rutare/navigaţie pietonală. Sistemul se bazează pe anumite date, colectate în prealabil din mai multe surse care vor formă un data layer al aplicaţiei, ce va fi mapat într-un graf pentru a se putea obţine rezultate de rutare cât mai eficiente.

După analiza acestor lucruri prezentate mai sus şi din dorinţa de a aduce un sistem nou care să ajute la deplasarea folosind mijloacele de transport in comun, iar cum acum majoritatea populaţiei deţine un telefon inteligent s-a considerat că un astfel de sistem ar fi foarte benefic, în special pentru turişti sau persoane care sunt noi într-un oraş anume, dar chiar şi pentru localnici într-un oraş foarte mare sau cu o reţea de transport foarte extinsă care poate creea probleme sau confuzii.

1.3. Conținutul lucrării

În această secţiune se vor prezenta pe scurt fiecare capitol al acestei lucrări şi ideea principala care este abordată în fiecare dintre ele.

În primul capitol, s-a evidenţiat contextul proiectului, statusul tehnologiilor care sunt folosite în acest proiect, motivaţia şi factorii care au contribuit la decizia începerii proiectării şi implementării acestui sistem, iar în final conţine un scurt rezumat a ceea ce este prezentat mai departe în această lucrare, în fiecare capitol în parte.

În cel de-al doilea capitol se vor prezenta obiectivele proiectului, ceea ce se urmăreşte a se învaţa pe parcursul dezvoltării lui dar şi scopurile finale ale acestuia. Tot în acest capitol se vor evidenţia şi cerinţele proiectului, formate din cerinţe funcţionale şi cele non-funcţionale.

Capitolul al treilea este constă într-un studiu bibliografic, ce reprezintă un rezumat al tuturor documentărilor ce au avut loc înainte de începerea proiectului şi aprofundarea subiectelor care au ajutat la dezvoltarea/înţelegerea mai bună a cerinţelor în vederea implementării ulterioare. Pe lângă aceste subiecte cum ar fi Rutarea, Geocodarea, GPS-ul, tot în acest capitol sun prezentate şi câteva sisteme similare, dintre cele existente (puţine la număr), cu avantajale şi dezavantajele fiecăruia.

Capitolul patru reprezintă rezumatul fundamentării teoretice asupra tehnologiilor, protocoalelor şi algoritmilor utilizaţi/e în rezolvarea problemei.

Al cincilea capitol reprezintă descrierea în detaliu a tutoror etapelor urmaţi în proiectarea şi implementarea sistemului. Capitolul prezintă fiecare pas până la cel mai mic detaliu, cuprizandu-se aici de la metodele prin care s-au colectat datele în prima etapă a proiectului, până la implementarea componenţei de rutare şi integrarea ei în aplicaţie, integrarea hărţilor în aplicaţie şi combinarea celor două componente astfel încât la sfârşit să se obţină un sistem funcţional şi gata pentru a fi folosit.

Capitolul al şaselea prezintă principalele metode de testare care s-au folosit pentru acest sistem, dar şi diferite tool-uri care au fost integrate pentru a ajută la testare. În principal testarea a fost manuală, cuprinzând diferite cazuri de test şi anumite scenarii, încercând să se acopere fiecare domeniu. Tot în cadrul testării manuale s-a realizat şi testarea interfeţei grafice.

Capitolul şapte prezintă metoda de instalare a aplicaţiei pe un dispozitiv, sub formă de tutorial, astfel încât cititorul să poată din acest document să efectueze instalarea fără nicio problema. Sunt prezentaţi fiecare paşi care trebuie efectuaţi pentru această. Tot în acest capitol este prezentată şi interfaţa utilizator/grafică a aplicaţiei şi modul de utilizare a acesteia, ilustrând prin capturi de ecran din cadrul aplicaţiei fiecare funcţionalitate principala.

Capitolul opt reprezintă concluziile trase în urmă documentării, proiectării şi implementării acestui sistem. Sunt prezentate contribuţiile proprii, ce s-a realizat, avantantajele aplicaţiei dar şi câteva din dezvoltările ulterioare posibile, care s-ar putea implementa şi ar face această aplicaţie mult mai puternică şi mai stabilă, oferindu-i şi alte funcţionalităţi în plus.

Capitolul 2. Obiectivele Proiectului

2.1. Obiective generale

Obiectivul principal este realizarea unui sistem care să rezolve problema deplasării folosind mijloacele de transport în comun dintr-un oraş. Sistemul trebuie să ofere informaţii despre toate mijloacele de transport, orare, rute, staţii etc. Aplicaţia trebuie să fie disponibilă pentru cât mai multe categorii de oameni, uşor de folosit şi la îndemână oricui.

Sistemul este menit sa ofere utilizatorilor posibilitatea de a calcula/vizualiza rute între locaţia lor şi o altă locaţie din oraş şi de a primi direcţii spre staţiile de autobuz (când utilizatorul nu este în autobuz) şi totodată informaţii despre schimbarea autobuzelor până la destinaţia finală. Componenta de rutare trebuie să fie implementată utilizând un algoritm de căutare a celei mai bune căi dintr-un punct în altul. (ex Algoritmul lui Dijkstra, A*).

Proiectul constă în proiectarea şi implementarea unui algoritm (componente de rutare) care mai apoi va fi integrat într-o aplicaţie mobilă (iOS,Android) pentru a fi demonstrate funcţionalităţile. În plus faţă de această componentă vor fi şi alte funcţionalităţi care vor face că aplicaţia să fie mai complexă, mai folositoare utilizatorilor şi totodată poziţionată cât mai bine în clasamentul aplicaţiilor de acest gen. Sistemul implementat va conţine şi un data layer ,mai precis o baza de date cu toate informaţiile

Un alt obiectiv este aprofundarea cunoştinţelor despre tehnologiile mobile, în special tehnologia iOS şi a librăriilor aferente. Aplicaţiile iOS sunt dezvoltate independent de orice alte aplicaţii mobile şi necesită scrierea programului în limbajele suportate de această tehnologie (Objective-C sau Swift), un MacBook şi programul Xcode instalat.

2.2. Cerințele proiectului

Cerinţele proiectului sunt prezentate mai jos şi sunt împărţite în cerinţele funcţionale ale sistemului şi cerinţele non-funcţionale. Acestea sunt obiectivele principale care se doresc a fi implementate prin acest sistem.

2.2.1. Cerințe funcționale

- Bus tracking - este un serviciu de simulare a autobuzelor în timp real, pentru a vedea pe harta locul unde un autobuz este la o anumită ora (estimarea este aproximativă, totul fiind calculat matematic).
- Navigaţie- din momentul în care s-a calculat ruta, se poate începe o navigaţie, care va oferii informaţii atât când utilizatorul este pe jos (pedestrian mode) dar şi când este în bus (ex. ce staţie urmează, când trebuie să coboare, care autobuz trebuie luat în următoarea staţie etc)
- Vizualizare orare/rute/staţii şi toate informaţiile care sunt disponibile şi pe site-ul CTP.
- Buy a ticket - direct din aplicaţie / Call a taxi – informaţii despre companiile de taxi din oraş

- Raportare trafic – utilizatorii care sunt într-un autobuz pot să raporteze traficul – toţi ceilalţi utilizatori care se găsesc în zona respectivă fiind notificaţi pentru a şti să evite acele porţiuni de drum şi acele autobuze. (necesită server side)
- Oferă funcţionalitate fără o conexiune la internet (Offline mode).

2.2.2. Cerințe non-functionale

- Utilizabilitate – reprezintă gradul de uşurinţă în utilizarea sistemului. Ţinta este ca sistemul să fie cât mai uşor de folosit, astfel interfaţa aplicaţiei trebuie să fie cât mai intuitivă şi uşor de folosit. Cu cât design-ul aplicaţiei este mai frumos şi în acelaşi timp uşor de folosit fără a fi nevoie de nicio instruire înainte de utilizare , numărul de utilizatori va creşte considerabil.
- Performanţă – reprezintă rapididatea cu care aplicaţia răspunde când un utilizator accesează o funcţionalitate oferită de aplicaţie, dobândită în acest sistem printr-un flux al operaţiilor foarte bine stabilit şi implementat ce oferă rapidate în execuţie.
- Disponibilitate – reprezintă faptul că sistemul poate fi folosit în orice moment. Din moment ce principalele funcţionalităţi vor funcţiona (şi) în mod offline, acesta cerinţă se poate consideră îndeplinită.
- Sistem deschis - sistemul este uşor de îmbunătăţit, arhitectură permite adăugarea de noi componente sau funcţionalităţi, fără a afecta componentele actuale.

Capitolul 3. Studiu Bibliografic

Scopul principal al sistemului dezvoltat este de a permite calcularea de rute în diferite moduri, folosind mijloace de transport în comun, la nivel de oraş/zona/ţară şi să conţină un algoritm de rutare bazat pe teoria grafurilor. Pentru realizarea acestui proiect s-au studiat următoarele domenii prezentate în următoarele secţiuni.

3.1. GPS

GPS-ul (Global Positioning System) este un sistem de navigaţie spaţial care oferă locaţia şi informaţii despre timp în orice condiţii meteorologice oriunde pe Pământ. Sistemul oferă informaţii armatei, civililor şi utilizatorilor comerciali în toată lumea. Dezvoltarea acestui sistem a început în anii 1960, continuat apoi în 1973 de Statele Unite ale Americii ajungându-se abia în anul 1995 la un sistem funcţional în totalitate.[2]

În ultimii ani această tehnologie s-a dezvoltat tot mai mult şi în piaţă telefoanelor mobile, astfel orice telefon din această generaţie are incoporat acest sistem GPS, astfel fiind posibilă localizarea lui, dar totodată şi folosirea acestora pentru a obţine direcţii către destinaţia dorită de utilizator sau pentru a găsi informaţii despre punctele de interes din apropiere.

Un receptor GPS [3] se bazează pe unde radio, la fel că un telefon celular, însă în loc că să folosească antene, relee pe pământ acestea comunica cu sateliţi, care orbitează în jurul planetei noastre. Pentru a fi determinată locaţia curentă a unui utilizator, un receptor GPS trebuie să determine: locaţiile a cel puţin trei sateliţi apropiaţi de utilizator şi în acelaşi timp unde este utilizator în relaţie cu aceşti sateliţi. După această receptorul foloseşte un proces numit “trilateration” (eng.) că să poată determina poziţia exactă, practic, se “desenează” o sfera în jurul fiecărui dintre cei trei sateliţi, aceste trei sfere se vor interesecta în două puncte – unul în spaţiu, iar unul pe pământ – punctul de pe pământ unde cele trei sfere se întâlnesc este locaţia utilizatorului.

Exemple în privinţa folosirii sistemului GPS în aplicaţia creată:

1) Urmărirea locaţiei: în fiecare moment în aplicaţie utilizatorul poate să vizualizeze pe harta locaţia curentă şi toate punctele de interes din zona respectivă şi nu numai.

2) Direcţii “Turn-by-turn”: aplicaţia oferă cu ajutorul acestui sistem direcţii pentru utilizatori şi chiar şi înştiinţări audio pentru că utilizatorul să poată naviga spre destinaţia dorită.

3.2. Rutarea

Este un proces pentru a selecta cea mai bună cale într-o reţea de noduri. De obicei acest termen se foloseşte în reţelele de calculatoare, dar pe acelaşi principiu îl găsim prezent şi în teoria grafurilor, scopul fiind de a găsi cea mai bună cale de la un nod X la un nod Y. În dezvoltarea acestei lucrări s-a studiat mult acest concept de “rutare” ajungându-se la o soluţie cât mai optimă de a rezolva această problema. Lucrarea prezintă

creearea unui algoritm care va permite calcularea mai multor rute între două locaţii dintr-un oraş, folosind mai multe mijloace de deplasare.

3.3. Geocodarea

Geocodarea constă în procesul de a transforma descrierea unei locaţii (ex. Adresa, numele unei locaţii etc) într-o locaţie pe suprafaţa pământului, mai exact în coordonate GPS (latitudine, longitudine).Putem face geocodarea prin introducerea unei singure descrieri de locaţie sau prin mai multe descrieri în cadrul unui tabel. Locaţiile rezultate sunt returnate că şi trăsături cu anumite atribute, care pot fi folosite pentru mapare sau analiză spaţială.[8]

Geocodarea este folosită pentru simpla analizare a datelor, pentru afaceri, pentru gestionarea populaţiei, dar şi în tehnici de distribuire. Cu ajutorul adreselor geocodate se pot vizualiza spaţial acele adrese şi putem recunoaşte asemănări între informaţie.

Adresa pentru geocodare

O adresă este o simplă metodă folosită pentru a descrie o locație. Pentru geocodare, aceasta poate fi o adresă de stradă, un nume a unui loc sau o locație care este identificată de un cod. O adresă descrie cum putem referi o locație bazată pe niște proprietăți în sistemul geografic de informare.

Adresele au câteva caracteristici specifice. O adresă conține anumite elemente și sunt prezentate într-o mulțime de formate. La geocodare formatul adresei este interpretat și elementele acesteia sunt identificate și comparate cu elemente stocate în bază de date.

Reverse geocoding

Acesta este procesul opus geocodarii care transformă un punct geografic (latitudine, longitudine) într-o adresa de normală folosită în viață de zi cu zi. Permite identificarea de adrese apropiate, localuri sau anumite zone (ex. Cartiere, judeţe, oraşe, ţări etc). Combinată cu geocodarea şi rutarea este o componentă critică a serviciilor mobile de localizare, astfel fiind prezentă şi în aplicaţia descrisă în această lucrare.

3.4. Sisteme similare

Până în urmă cu puţini ani, deplasarea folosind mijloacele de transport în comun se bazau doar pe programe care se puteau găsi sau nu în staţiile de autobuz, uneori oamenii doar stand aşteptând un autobuz oare fără a şti când acesta va ajunge. Şi în acest domeniu, la fel că în majoritatea, odată cu această explozie a tehnologiei lucrurile au avansat foarte mult, astfel se găsesc informaţii despre mijloacele de transport în comun începând din staţiile de autobuz până pe diferite website-uri ale companiilor sau chiar aplicaţii pentru telefoane mobile care colectează datele de pe aceste site-uri şi le oferă utilizatorilor într-o formă mai uşor de accesat oriunde şi oricând.

Că rezultat a documentării despre anumite sisteme care oferă informaţii sau chiar rutare cu mijloace de transport în comun s-au identificat următoarele: Transit App, Moovit, Public Transit by Google.

3.4.1. Transit App

Această aplicaţie oferă informaţii despre mijloacele de transport din mai multe oraşe ale lumii. Funcţionează în special în SUA, dar are acoperire şi în câteva oraşe din Europa (Paris, Milano etc).

Transit App [23] permite utilizatorilor să vizualizeze orele la care următoarele autobuze vor pleca din staţia curentă. Permite planificare de călătorii şi notifica utilizatorul pentru orice schimbare. Permite diferite mijloace de transport: Uber, inchiriere maşină sau vizualizare staţii de biciclete care sunt în apropiere.

Funcţionalităţi:

- Calculează când următorul metrou sau aubotuz va ajunge in staţie folosind real-time predictions.
- Planifică trasee A-B cu uşurinţă.
- Vizualizează exact unde mijlocul de transport este pe harta in timp real.
- Vizualizează orare si rute, chiar si offline
- Localizează staţii de biciclete
- Găseşte car2go's in apropiere si închireaza maşină cu uşurinţă.
- Verifică în timp real estimări pentru cele mai aproapiate maşini Uber şi cere o călătorie prin două atingeri.
- Programează alarme înainte că autobuzul sau trenul tău să ajungă.
- Permite folosirea functionalitatilor din alte aplicaţii: Uber, car2go etc.

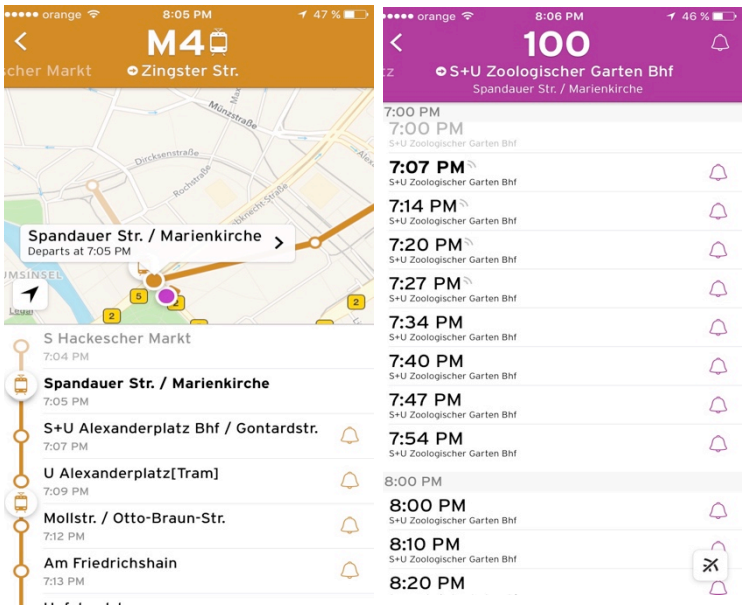

Fig 3.1 - Interfaţa grafică TransitApp

Avantaje:

- Cuprinde toate mijloacele de transport, metro autobuze trenuri etc
- Accesul la aplicaţia Uber, car2go etc
- Localizare staţii de biciclete
- Funcţionalitate foarte bună în UŞA.
- Oferă suport şi pentru Apple Watch!

Dezavantaje:

- Număr limitat de orașe in Europa / la nivel mondial.

3.4.2. Moovit

Moovit [17] este aplicaţia numărul unu în acest domeniu care există pe piaţă, cu peste 30 de milioane de utilizatori în peste 800 de oraşe ale lumii. Cercetările spun că numărul utilizatorilor este în creştere şi pe lângă această Moovit adaugă câte un oraş nou în aplicaţie la fiecare 24 de ore, ceea ce este un progres imens – datele despre companiile de transport din oraşe fiind cele mai importante şi greu de obţinut pentru astfel de aplicaţii. Aplicaţia combină toate mijloacele de transport de care un oraş dispune de la tramvaie, autobuze, taxi-uri, bike share etc

Pentru că transportul în comun este întotdeauna previzibil, Moovit actualizează constant datele în momentul în care operatorii de transit modifică ceva într-un anumit oraş, utilizatorul fiind astfel ferit de a ajunge într-o staţie închisă sau unde un autobuz nu va mai ajunge.

Funcționalități:

- Direcţii oferite în timp real prin notificări la fiecare 'pas' în timpul călătoriei. Aplicaţia ştie exact locaţia utilizatorolui, cât timp acesta are de aşteptat şi câte opriri mai sunt până la destinaţie.
- Bike share – integrează serviciile de bike sharing în oraşele unde acestea sunt disponibile.
- Planificarea sosirii autobuzelor în staţiile de autobuz, astfel utilizatorul ştie când să se îndrepte spre această, fără a fi nevoit să aştepte prea mult în staţie.
- Acoperire în peste 800 de oraşe
- Permite salvarea locaţiilor favorite sau care sunt folosite frecvent pentru o mai rapidă găsire acestora la o altă utilizare a aplicaţiei.

Avantaje:

- Oferă suport pentru Apple Watch.
- Oferă functionalitaţi si in Cluj-Napoca.

Dezavantaje:

- nu oferă functionalitaţi offline

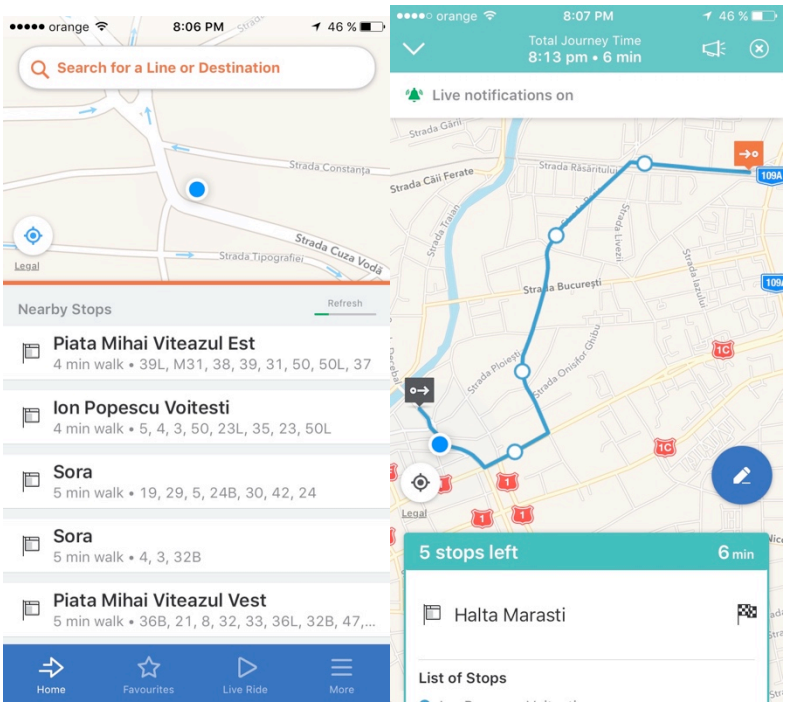

Fig 3.2 - Interfața grafică Moovit

3.4.3. Public Transit by Google

Bineînţeles, un sistem similar oferă deja şi Google. Public Transit are menirea de a oferi celor care utilizează aplicaţia Google Maps posibilitatea de a calcula o ruta sau a vedea informaţii şi despre mijloacele de transport în comun din oraşul respectiv(şi nu numai). Acest sistem este unul care în ultimul an a luat o amploare deosebită, dezvoltându-se în şi România şi oferind servicii de rutare cu mijloacele de transport în comun şi în oraşul Cluj-Napoca.

Pentru colectarea datelor şi stocarea datelor Google foloseşte fişierele de tip GTFS, în funcţie de care oferă toate informaţiile necesare utilizatorului.

Funcționalități:

- ofera informații despre mijloacele de transport in comun: orare, statii, trasee etc
- ofera rutare folosind mijloacele de transport in comun
- combina impreuna cu mijloacele de transport din orase si cele care sunt extra-urbane sau care fac legatura dintre doua sau mai multe orase

Avantaje:

- suport in foarte multe localitati
- rutarea tine cont de orele de circulatie ale autobuzelor

Dezavantaje:

- nu ofera functionalitati offline
- nu ofera navigatie si informatii in tipul navigatiei despre pasii urmatori

3.5. Concluzii

În urma studiului realizat pe sistemele similare se poate observa că aplicaţiile descrise oferă, printre cele mai importante funcţionalităţi, posibilitatea de localizare a utilizatorlui, urmărirea poziţiei, informaţii despre mijloacele de transport în comun, planificare de călătorii, salvare locaţii favorite, calculare de rute, oferă estimări ale timpului de pornire/sosire etc. În opinia noastră, printre cele mai importante şi folositoare funcţionalităţi pentru o asftfel de aplicaţie ar fi: vizualizare orare, staţii, trasee, calculare rute din poziţia A către poziţia B (folosind până şi rutare pietonală), dar şi navigare pe aceste rute, utilizatorul primind notificări când trebuie să schimbe un autobuz.

Dintre sistemele prezentate mai sus singurul care funcţionează pentru ţară noastră este Public Transit by Google şi este într-o creştere continuă în ultima perioadă, acoperind multe deja destul de multe orașe și din România, inclusiv Cluj-Napoca.

În concluzie, în acest capitol s-a ales descrierea tuturor exemplelor de mai sus, deoarece reprezintă aplicaţii asemănătoare cu aplicaţia dezvoltată şi prezentată în această lucrare şi au fost un exemplu atât pentru interfaţă grafică cât şi pentru funcţionalităţile care au fost implementate.

Capitolul 4. Analiză şi Fundamentare Teoretică

După cum deja s-a stabilit, această lucrare se bazează foarte mult pe grafuri, parcurgeri şi găsirea celor mai scurte cai între două noduri, de aceea în continuare se vor prezenta câteva definiţii şi elemente de baza din cadrul grafurilor, împreună cu prezentarea algorimtilor studiaţi şi motivaţia alegerii algoritmului implementat în final.

4.1. Grafuri

4.1.1. Elemente de teorie a grafurilor

Înainte de a fi prezentată teoria şi conceptele de baza ale grafurilor se vor evidenţia câteva definiţii a unor termeni des întâlniţi în restul lucrării.

Noduri (N): sunt un set de puncte unde două sau mai multe linii se întâlnesc. Formează nodurile grafului care vor fi conectate între ele prin muchii. În reprezentarea folosită pentru implementarea algoritmul de baza al sistemului de rutare folosind mijloace de transport în comun, noduri vor reprezenta staţiile de autobuz din oraş.

Muchie (M): este un segment care conectează două noduri. În această lucrare muchiile sunt reprezentate de distanţa între două staţii de autobuz de pe aceeaşi ruta.

Cale / **Drum**: într-un graf, un drum traversează o secvenţă de noduri conectate între ele prin muchii. În secvenţa de muchii nu sunt elemente duplicate.

Ciclu: este un drum care începe dintr-un nod şi termină în acelaşi nod. Lungimea unui drum sau a unui ciclu este determinată de numărul de muchii care îl alcătuiesc.

Cost: o valoare atribuită unei muchii a grafului, reprezentând costul deplasării pe muchia respectivă.

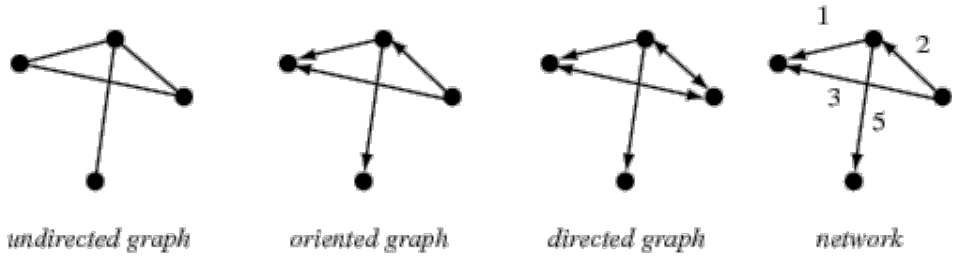

Fig 4.1 - Tipuri de grafuri

Teoria grafurilor este o teorie matematică, folosită în domeniul matematicii şi al ştiinţei calculatoarelor, care conţine structuri matematici folosite pentru a modela şi evidenţia relaţii între obiecte.

Spre exemplu, o reţea de autobuze poate fi reprezentată ca un graf, unde staţiile de autobuz sunt nodurile acestuia, iar legăturile dintrele sunt muchiile. Această reprezentare va fi folosită în implementarea sistemului prezentat în această lucrare.

Grafurile se împart în două mari categorii:

- orientate - muchiile dintre două noduri au o singură direcţie
- neorientate - muchiile dintre două noduri au ambele direcţii

4.1.2. Algoritmi de parcurgere a grafurilor

Algoritmii de parcurgere şi găsirea celor mai scurte drumuri între două noduri ale grafului fac parte din problemele fundamentale din teoria grafurilor. Printre algorimtii de parcurgere şi de ccautare se pot evidenţia Căutarea în adâncime, căutare în lăţime, algoritmul lui Dijkstra s.a.m.d.

Construirea unui graf şi aplicarea algoritmilor asupra lui ajută la rezolvarea destul de uşoară a uneia dintre cele mai des întâlnite şi importante probleme, şi anume: găsirea celui mai scurt drum dintr-un punct A într-un punct B.

De exemplu, această soluţie se poate aplica în rezolvarea problemei de a găsi cel mai scurt drum dintre două oraşe (oraşele fiind reprezentate de noduri).

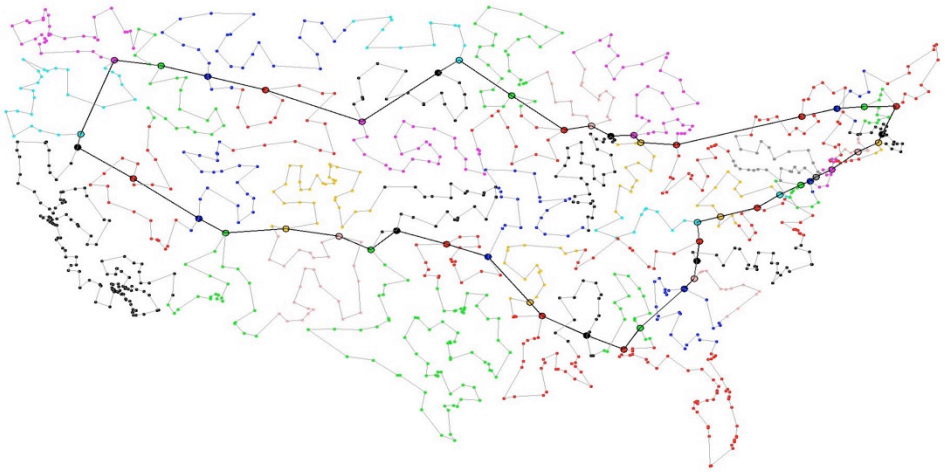

Fig 4.2 - O reprezentare sub forma de graf a oraselor din SUA[1]

Această problema se poate rezolva folosind diferiţi algoritmi, în functe de structura grafului rezultat. În secţiunea următoare algorimtii de căutare a celui mai scurt drum vor fi clasificaţi în[18]:

Algoritmi cu un singur nod sursă (en. Single source shortest path 'SSSP'): în acest tip de problema, se doreşte determinarea celui mai scurt drum de la un nod sursă către un alt nod sau către toate celelalte noduri ale grafului. Acest tip de problema este rezolvat cel mai eficient folosind Algoritmul lui Dijkstra, în special când în graf nu există costuri negative, însă dacă ele apar algoritmul Bellman-Ford este mult mai eficient decât cel menţionat anterior.

Cel mai scurt drum între toate perechile de noduri (en. All pair's shortest path (APSP): dacă se doreşte determinarea celui mai scurt drum dintre fiecare nod cu fiecare nod în parte. Dintre algoritmii folosiţi pentru această problema fac parte: Algoritmul Floyd-Warshall, Algoritmul lui Jonshon (Johnson's Algorithm) - este mai rapid pentru grafuri dense.

[1] GLKH, http://www.akira.ruc.dk/~keld/research/GLKH/

Pentru rezolvarea acestei probleme sunt şi alţi algoritmi în afară de cei amintiţi, cum ar fi: Algoritmul A*, Algoritmul Viterbi etc. Dijkstra's Algorithm este un algoritm care rezolva problema căutării celei mai scurte cai de la un nod la un alt nod într-un graf. Algoritmul returnează în final cea mai bună cale de la un nod sursă la toate celelalte noduri din graf. Presupune vizitarea fiecărui nod al grafului şi nu se opreşte decât după ce termină vizitarea lor.

A* - este un algoritm bazat pe Dijkstra's Algorithm dar în acelaşi timp şi algoritmul de parcurgere BFS a unui graf. [21] Avantajul acestui algoritm este că permite încărcarea unei părţi mai mici din graf, deoarece foloseşte o euristică, care ajută în luarea deciziilor dacă graful ar trebui să continue construirea în acea direcţie sau ar trebui să se întoarcă şi să continue pe altă cale. Este folositor unde grafurile ajung la dimensiuni foarte mari iar asta duce la probleme de performanţă şi eficientă. În acest caz A* este algoritmul potrivit.

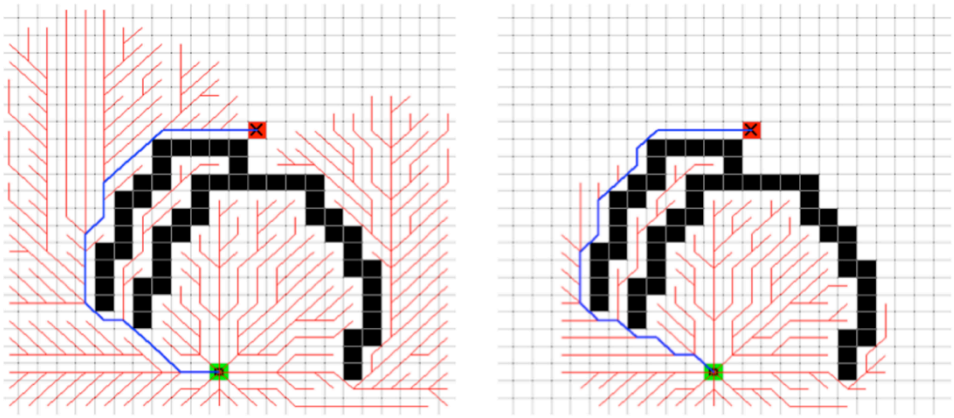

Fig 4.3 - Comparație intre expandarea grafului folosind algoritmii Dijkstra(stânga) și A*(dreapta)

Analizând figura de mai sus, cu siguranţă prima părere ar fi că A* este mai eficient, ceea ce este şi adevărat, dar trebuie avut în vedere şi faptul că necesită o implementare mult mai complexă.[21] Privind la problema abordată în această lucrare şi la ceilalţi factori, s-a hotărât că algoritmul folosit va fi bazat pe Algoritmul lui Dijkstra, fiind îmbunătăţit cu o funcţie de decizie mai complexă(nu doar din punct de vedere a costului) care va determina algorimtul să aleagă muchiile păstrând dacă se poate acelaşi autobuz de la început până la sfârşit sau schimbarea lui de cât mai puţine ori este posibil. Totodată pentru nivelul sistemului acest algoritm se va comporta destul de bine, nefiind nevoie de A*, deoarece graful construit pe care se va aplica algoritmul nu va fi aşa de mare/dens/complex.

4.1.3. Algoritmul lui Dijkstra

Acest algoritm a fost proiectat în anul 1956 de către Edsger Dijkstra, de unde vine şi numele algoritmului şi este unul dintre cei mai populari algoritmu care rezolva problema drumului celui mai scurt dintre două noduri ale unui graf.

Algoritmul se poate aplica numai pentru grafuri care nu au costuri negative pe niciuna dintre muchiile lui şi se poate obţine astfel rezolvarea pentru a obţine cel mai scurt drum între un nod şi toate celelalte noduri ale grafului (aşadar este de tipul SSSP [7] conform clasificării anterioare).

Spre exemplu dacă staţiile şi traseele autobuzelor dintr-un oraş ar fi mapate pe un graf, prin noduri şi muchii, putem să găsim cel mai scurt drum dintr-o staţie în oricare altă staţie de autobuz din acel oraş.

Algoritmul are urmatoarele etape[10]:

1. Se alege nodul sursă şi se marchează ca permanent, iar costul se inițializeaza cu 0.
2. Se verifică fiecare nod vecin din nodul curent.
3. Calculează costul cumulat pentru fiecare nod vecin si se aleg temporar ca nod curent.
4. Nodurile temporare se verifică astfel:
    a. Se alege un nod cu cel mai mic cost acumulat și se marcheaza ca nod permanent.
    b. Daca sunt mai multe căi pentru a ajunge la un nod, atunci cea mai scurta este alesă.
5. Se repeta pașii 2-4 până când toate nodurile sunt marcate permanente.

Exemplu: În figura 4.4, prezentată mai jos, este reprezentat un graf cu 8 noduri conectate între ele prin muchii care au asignate costuri şi se doreşte calcularea celui mai scurt drum din punctul A către toate celelalte noduri.[9]

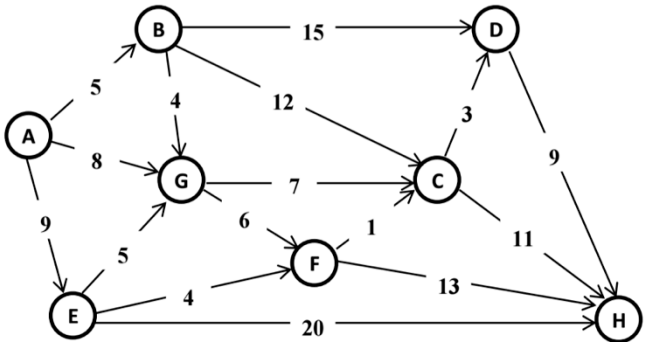

Fig 4.4 - Exemplu de graf orientat

Observații:

• costul drumului dintre un nod şi el însăşi este notat cu (-----), deoarece graful nu conţine bucle

• se înregistrează costul fiecărui nod în raport cu celelalte

• se foloseşte semnul infinit (∞) dacă nu există cale directă

• pătratul negru marchează nodul ca vizitat

Figura de mai jos se va folosi pentru implementarea algoritmului şi ilustrarea acestuia: coloana "Visited nodes" se referă la nodurile care au fost vizitate, "Current node" se referă la nodul care se foloseşte într-un anumit pas, iar celelalte coloane se referă la restul nodurilor:

Tabel 4.1 - Implementarea algoritmului lui Dijkstra

Visited node	Current node	A	B	C	D	E	F	G	H
A	A	-----	5,A	∞	∞	9,A	∞	8,A	∞
A,B	B		-----	17,B	20,B	9,A	∞	8,A	∞
A,B,G	G			15,G	20,B	9,A	14,G	-----	∞
A,B,G,E	E			15,G	20,B	-----	13,E		∞
A,B,G,E,F	F			14,F	20,B		-----		26,F
A,B,G,E,F,C	C			-----	17,C				25,C
A,B,G,E,F,C,D	D				-----				25,C
A,B,G,E,F,C,D,H	H								-----

Folosind tabelul de mai sus se poate găsi drumul cel mai scurt alegând ultima valoare pentru coloana nodului şi urmărind valorile cele mai mici şi nodurile care sunt în legătură cu acestea.

Folosind aceste rezultate se pot găsi rutele pentru a acoperi toate nodurile, aşa cum este prezentat în figura următoare:

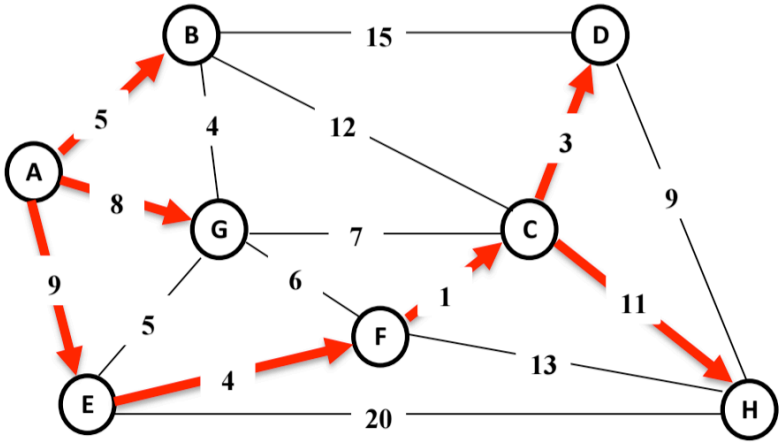

Fig 4.5 - Cel mai scurt drum din A către toate celelalte noduri

4.2. Tehnologii si protocoale utilizate

În această secţiune se vor prezenta tehnologiile utilizate împreună cu avantajele şi motivaţia de ce acestea au fost alese.

4.2.1. Sistemul de operare iOS (iPhone OS)

Sistemul de operare este dedicat telefoanelor mobile şi a fost dezvoltat de Apple Inc. şi distribuit exclusiv pentru dispozitivle Apple. Este sistemul de operare care rulează pe toate dispozitivile mobile ale companiei, iPhone, iPad şi iPod touch şi este al doilea cel mai popular sistem de operare din lume, după Android.

Acest sistem de operare, de altfel la fel ca celelalte dezvoltate de Apple, nu permite utilizatorilor să personalizeze foarte multe lucruri însă acest lucru duce la un avantaj foarte important al acestui sistem de operare, și anume: Stabilitate [15]. Deşi alte sisteme de operare (Android, Windows phone, Symbian) sunt considerate mai performante decât iOS, niciunul măcar nu se apropie de stabilitatea pe care acest sistem o oferă.

Un alt avantaj al acestui sistem este viteza. Deoarece sistemul de operare are şi o stabilitate bună, memoria şi procesorul pot acorda multă atenţie şi vitezei de procesesare și totodată timpului de răspuns la accesarea unei funcționalități. Jocurile, încărcarea pozelor, videourilor, sunt doar la o atingere distanţă. Bineînţeles, această este în versiunea curentă iOS 9.0, până aici au fost multe schimbări şi multe nereguli. Spre exemplu, un device care rulează iOS poate obţine aceeaşi viteză şi poate rula jocuri cu grafică intensivă la fel de bine că un procesor dual core la viteză de 1.4 GHz şi toate acesta prin faptul că toate resursele sunt controlate într-un mod eficient şi corect. Un alt motiv al vitezei este şi faptul că aplicația rulează în cod nativ, nu printr-o maşină virtuală.

Interfaţa iOS este uşor de folosit, totul fiind foarte explicit şi simplu (pentru a avea o piaţă cât mai largă de desfacere).

4.2.2. OpenStreetMap (OSM)

OpenStreetMap [20] fost lansat în 2004, în Marea Britaniei, în timpul când informaţiile despre hărţi şi datele de acest gen erau controlate cel mai mult de asociaţii guvernamentale şi alte companii private, astfel accesul la acest gen de date fiind foarte greu şi totodată, foarte scump. Idea din spatele OpenStreetMap a pornit din dorinţa de a rezolva această problema, ba chiar mai mult, de a face disponibil cât mai multe date geografice oamenilor de rând, într-un format electronic, astfel s-a născut ideea de a dezvolta o harta gratuită, uşor de modificat, a întregii lumi, prin costuri cât mai puţine, pornindu-se pe modelul Wikipedia, unde oricine poate adauga informaţii, edita, valida etc, având nevoie doar de un simplu cont şi de bună voinţă. Datorită uşurinţei de a fi folosit şi a dorinţei voluntarilor, numărul acestora ajungând azi la peste 2,2 milioane de utilizatori înregistraţi care îşi aduc aportul în îmbunătăţirea şi actualizarea datelor.

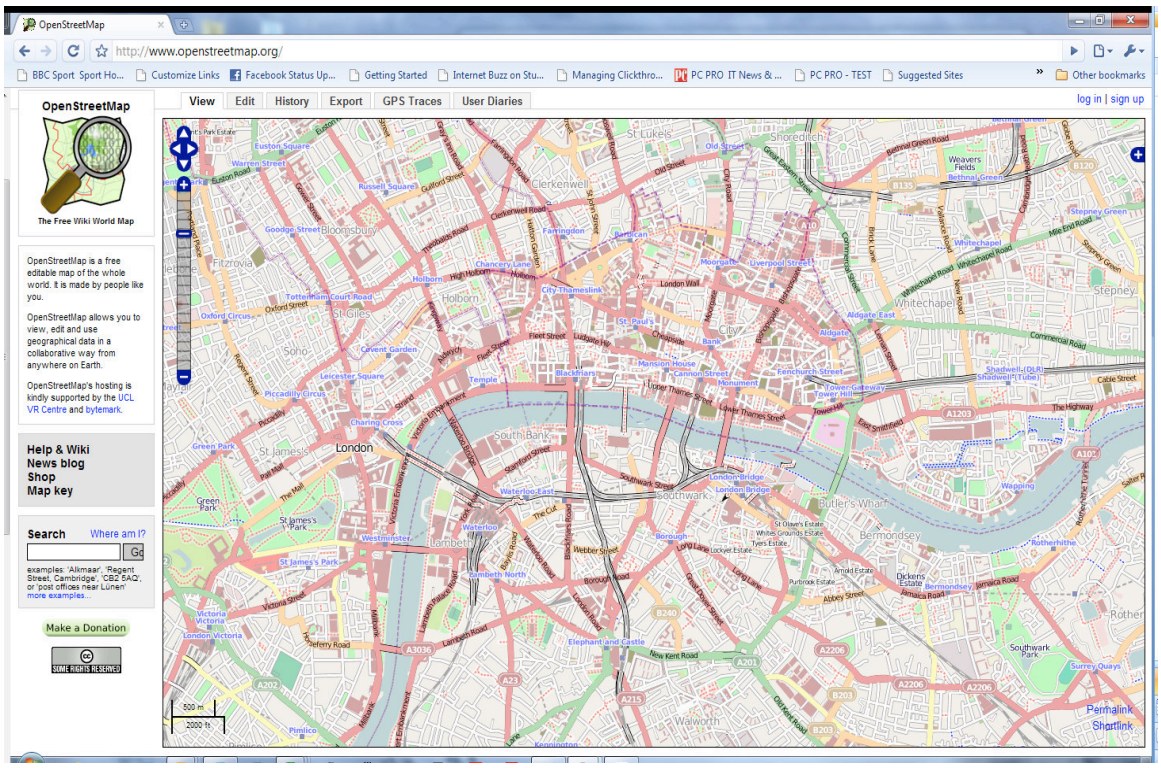

Fig 4.6 - OSM web

4.2.2.1. Elemente OSM

Elementele cu care se lucreaza în OSM sunt următoarele[18]:

Nodurile

Definesc o locaţie geografică specifică, reprezentată prin latitudine şi longitudine. Se pot folosi pentru a reprezenta diferite locaţii pe harta, puncte de interes (POI), semne de circulaţie. Orice element static de pe harta poate fi reprezentat printr-un nod, iar între acestea pot există şi anumite relaţii.

Căile

O cale este o lista ordonată de noduri, care poate reprezenta fie un segment liniar de pe harta (drumuri, cursul unui râu) fie marginile unei zone (clădiri, parcuri, păduri, spaţii verzi etc). Orice element din lumea reală care poate fi reprezentată printr-o linie sau un poligon poate fi reprezent pe harta cu ajutorul acestor liste de noduri.

Relaţiile

Reprezintă o relaţie între noduri, cai sau chiar relaţii. Spre exemplu, o ruta a unui autobuz este reprezentată printr-un lanţ de drumuri şi staţii.

4.2.2.2. Definirea atributelor

Datele fiind stocate în fisere XML, atributele unui element sunt definite şi salvate prin folosirea diverselor tag-uri. Un tag constă într-o pereche cheie-valoare, de exemplu (amenity=library). Specificaţiile tuturor tag-urilor şi ale standardelor sunt specificate şi disponibile pe site-ul OSM.

Orice utilizator înregistrat are acces să modifice orice astfel de tag care este stocat în OSM. Pentru aceasta există mai multe variante şi programe dezvoltate. Cel mai simplu mod este direct de pe platforma oferită de OSM, însă pot apărea dificultăţi datorită volumului imens de date disponibil într-o porţiune de harta. O alternativă mult mai prietenoasă şi mai uşor de folosit pentru modificarea hărţilor este utilitiarul JOSM, o aplicaţie desktop, care permite mult mai uşor vizualizarea nodurilor, relaţiilor, tag-urilor şi multe altele, oferind o interfaţă prietenoasă şi uşor de folosit pentru orice utilizator.

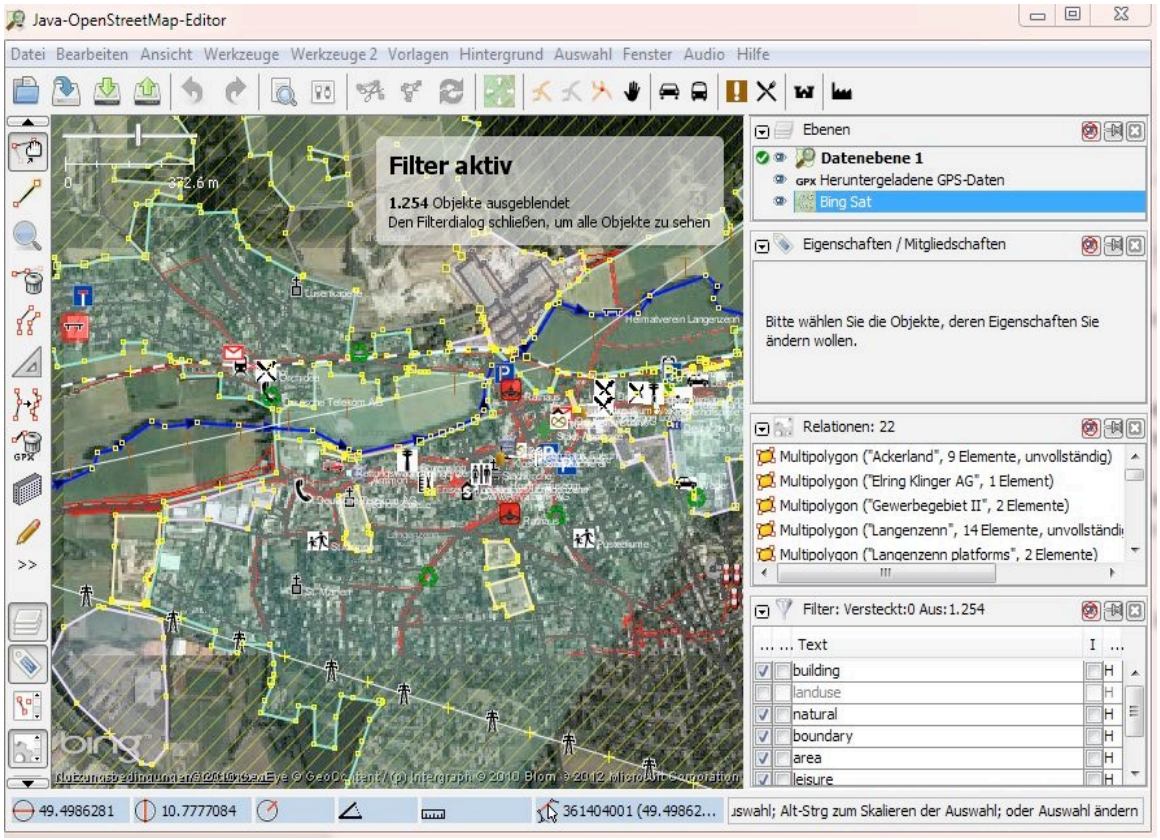

Fig 4.7 - Interfata grafică JOSM

4.2.2.3. De ce OSM?

Cel mai probabil, prima reacţie după ce am auzit despre această platforma ar fi "De ce OSM când avem Google Maps ?", răspunsul scurt este, deoarece dorim o harta care să fie accesibilă tuturor, gratuită, uşor de editat fără a fi controlată de cineva şi a depinde de cineva la orice mişcare care se face.

OpenStreetMap nu e deţinut de o companie anume, ci e al nostru, al tuturor, astfel investiile sunt minime, dar totuşi cu impact şi un rezultat foarte important. Google investeşte peste un miliard de dolari anual pentru întreţinerea hărţilor, editarea lor şi adăugarea de noi informaţii.[3] Bineînţeles ca urmare a acestor investiţii hărţile lor sunt mult mai exacte, actualizate şi includ multe puncte de interes, informaţii despre trafic (chiar şi live trafic în majoritatea oraşelor mari) şi bineînţeles, nu uităm de binecunoscutul Google StreetView. Totuşi, având toate acestea, când dorim să alegem o tehnologie pentru un produs se analizeaza bine piaţa şi toate avantajele, dar cel mai mult o să privim la costuri, astfel OpenStreetMap oferă o alternativă foarte flexibilă, evitând astfel un monopol pe această piaţă şi lansând un singur produs să conducă întreagă industrie. Oricine poate folosi informaţiile adunate, deoarece nicio companie nu le deţine astfel încât să poată să ducă tot acest efort într-o direcţie rea.

4.2.3. Tipuri de fişiere utilizate

În secţiunea următoare vor fi prezentate tipurile de fişiere utilizate în diferite etape ale dezovltarii acestui sistem.

4.2.3.1. XML

Fişierele XML [1] sunt fişiere care pot conţine date structurate după anumite tag-uri. Spre deosebire de fişiere HTML, care au un set de tag-uri predefinite, fişierele XML acordă posibilitatea aplicaţiilor/utilizatorilor de a-şi defini propriile seturi de tag-uri. Folosirea fişierelor XML poate fi foarte simplă sau în acealsi timp foarte complexă, depinzând de cât de mult utilizatorul dezvoltă structura fişierului.

Câteva dintre regulile generale despre fişierele XML:

- tag-urile sunt case-sensitive: ex. , , sunt toate diferite
- fiecare tag trebuie închis, iar la sfârşit trebuie neapărat închis. ex:
- structura fişierelor trebuie să aibă o structura de arbore

Un fişier XML este considerat "bine format" dacă acesta se conformă fiecărui criteriu specific acestui tip de fişier. Aceste fişiere de asemenea au abilitatea de a defini atribute şi de a descrie caracteristici [13] pentru fiecare element al sau în tag-ul de început al unui element.

Principalul avantaj îl reprezintă mărimea datelor ce pot fi stocate în acest tip de fişiere.

Dezavantajul acestui tip ar fi complexitatea la care se poate ajunge, iar astfel structura fişierului poate deveni foarte greu de descifrat.

4.2.3.2. CSV

Fisierele CSV [5] ("Comma Separated Values") sunt de obicei folosite pentru a se schimba date intre diferite aplicatii. Datele sunt foarte usor de parcurs si de extras.

Formatul fişierelor CSV:

• fiecare înregistrare este reprezentată pe o linie - separatorul de linii trebuie să fie LF (0x0A) sau CRLF (0x0D0A), dar totodată un separator poate fi inclus deja în date

• câmpurile sunt separate prin virgule

• spaţiul liber din stânga şi dreapta datelor de pe o linie este ignorat - excepţie făcând cazul când acest spaţiu este delimitat de ghilimele ("")

• este necesară delimitarea câmpurilor

Principalul avantaj ale acestor fişiere este faptul că oferă posibilitatea de a se lucra cu valori numerice foarte uşor, atât stocarea lor cât şi parcurgerea lor.

4.2.3.3. JSON

Se încadreaza în aceeaşi categorie cu fişierele prezentate mai sus, având astfel ca scop tot comunicarea între mai multe aplicaţii. Poate fi folosit ca o alternatiă mai uşoară a fişierelor XML. Conţine un set de reguli minim, şi în acelaşi timp simplu, însa JSON poate reprezenta structuri complexe de date si poate descrie obiecte, şiruri de obiecte etc. Sintaxa JSON este oferita de limbajul JavaScript, ceea ce face folosirea lui foarte usoara in acest limbaj, fiind deja definite functii speciale de parcurgere si scriere, dar si altele. Datorită uşurinţei în folosire JSON a devenit din ce in ce mai folosit si mai preferat de catre tot mai multe aplicatii, inlocuind adesea formatul XML.

Exista diverse librarii pentru aproape orice limbaj de programare care ofera posibilitatea de a face conversie de la fisiere XML la fisiere JSON si alte optiuni.

Un scurt exemplu de cod JSON:

```
{
"studenţi": [
                { "prenume":"Ion" ,
                  "nume":"Davidescu" },
                { "prenume":"Maria" ,
                   "nume":"Stoian" },
                { "prenume":"Petre" ,
                  "nume":"Ionescu" }
        ]
}
```

Această secvenţă ilustrează uşurinţa de a reprezenta un şir de obiecte în acest format. Secvenţa descrie şirul "studenţi" care conţine trei obiecte, fiecare având alte două câmpuri ("prenume", "nume").

Avantajul principal al fiserelor JSON îl reprezintă uşurinţă, simplitatea şi rapiditatea de parcurgere.

4.2.4. Objective-C

Objective-C [15] este principalul limbaj de programare folosit de Apple pentru sistemele de operare OS X şi iOS. Limbajul de programare a fost originar dezvoltat la începutul anilor 1980. A fost selectat că principalul limbaj de programare folost de sistemele de operare NeXT şi NeXTSTEP, din care OS X şi iOS sunt derivate.

Ca sintaxă, Objective-C este un layer peste C, ceea ce înseamnă că se poate compila orice program C cu un compilator Objective-C şi integrare de cod C în orice clasa Objective-C fără nicio problema. - prezentare mai detaliată, sintaxa, structura etc. Codul Objective-C de obicei au fişiere de implementare cu extensia .m şi fişiere header cu extensia .h, la fel că fişierele din limbajul C. Fişierele care conţin Objective-C-C++ au extensia .mm. Orice clasa descrisă în acest limbaj trebuie să aibă cele două fişiere, în fişierul "header" vor fi declarate toate atributele şi metodele clasei respective.

Atributele unei clase sunt prezentate sub formă de proprietăţi, care sunt declare în interfaţa clasei. Doar atributele şi metodele publice vor fi declarate în interfaţă, cele private vor fi declarate doar în fişierul de implimentare. Compilatorul va genera automat "getters & setters" pentru fiecare proprietate, care vor fi apelate prin operatorul '.' (punct) după cum urmează: obj.proprietate = value (set) sau value = obj.proprietate (get). Deasemenea, o proprietate poate fi declara ca 'weak' sau 'strong', ceea ce implică management-ul memoriei. O proprietate care va fi declarată 'strong' nu va fi dealocata niciodată cât timp obiectul care o conţine nu va fi dealocat.

Pe parcursul dezvoltării limbajului Objective-C, una dintre cele mai mari probleme a fost menţinerea codului într-o manieră cât mai generică, pentru a acoperi cât mai multe din cazurile posibile. Din programarea structurală s-a putut observa un avantaj mare, şi anume, una dintre cele mai bune metode de a îmbunătăţii codul este de a-l împărţi în bucăţi cât mai mici. Astfel Objective-C a împrumutat şi a extins încă un concept creat de Smalltalk, numit Categories. Metodele definte într-o categorie, care este adăugată peste o clasa, în timpul rulării va oferi instanţei acelei clase posibilitate de a apela metode definite în categoria respectivă, implementarea acesteia fiind ascunsă de clasa care o apelează. O categorie poate accesa toate proprietăţile publice din clasa respectivă.

Este un limbaj mai diferit decât cele folosite la ora actuală (şi mai dificil în anumite cazuri), însă principalul avantaj îl oferă rapiditatea şi stabilitatea oferite de către C/C++.

4.2.5. Design patterns

Un design pattern reprezintă o soluţie de implementare pentru o problema des întâlnită în cadrul dezvoltării aplicaţiilor software, menită să rezolve această problemă într-un context particular.

4.2.5.1. Model View Controller

Principalul desing pattern folosit în orice aplicaţie iOS este unul arhitectural, şi anume, binecunoscutul MVC [4]. Acesta este fundamental pentru multe mecanisme şi tehnologii în Cocoa Touch (Apple Technologies). Obiectele în MVC au fiecare unul dintre următoarele roluri: model, view sau controler. Acest pattern defineşte deasemenea şi calea prin care obiectele comunica între ele peste "barierele" imaginare definite de această împărţire.

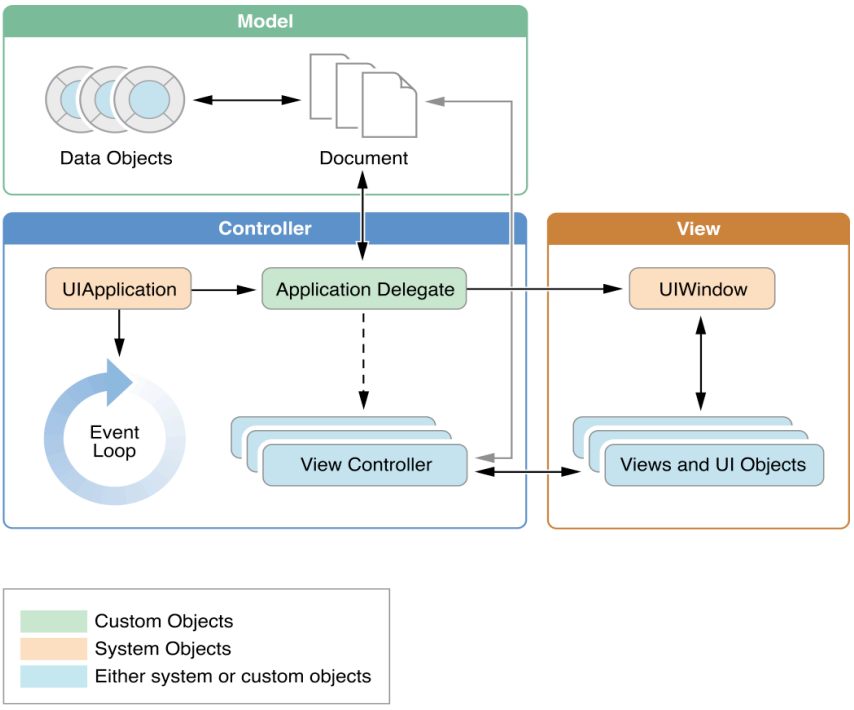

Fig 4.8 - Ilustrarea comunicării MVC in aplicațiile iOS

Obiectele din model sunt folosite pentru a stoca datele din aplicaţie şi pentru a putea fi manipulate. Aceste obiecte nu ar trebui nici într-un fel să aibă legătură cu obiectele din categoria View, mai exact cu obiectele care alcătuiesc interfaţa grafică.

Pachetul View şi obiectele sale pot numai să răspundă acţiunilor utilizatorului şi să fie prezentate pe ecran. Un view este de obicei afişat pe ecran conţinând date din obiectele de model. La o primă vedere s-ar spune că cele două module comunica, însă între acestea nu există nicio legătură directă.

Legătura între model şi view este rolul controller-ului, care trebuie să asculte constant după acţiunile care sunt făcute de către utilizatorul prin intermediul interfeţei grafice şi să ofere informaţii din datele aplicaţiei. Pe lângă rolul de intermediar dintre View şi Model , un controller poate efectua şi alte operaţii pentru aplicaţie, cum ar fi menţinerea ciclului de viaţă a unor obiecte, efectuarea de setări asupra altor obiecte sau coordonarea task-urilor aplicaţiei.

4.2.5.2. Comunicarea intre obiecte

Cu siguranţă în orice aplicaţie este necesară comunicarea între obiecte [4], astfel fiecare limbaj de programare a dezolvtat una sau mai multe metode pentru ca acest lucru să se poată realiza. Chiar mai mult, fiind o problema des întâlnită a apărut şi unul dintre cele mai cunoscute desing pattern-uri, şi anume Observer, cu rol de a permite trimiterea de notificări de la un obiect la altul în anumite circumstanţe. Objective-C oferă mai multe astfel de posibiltati, care sunt de baza pentru acest limbaj, şi vor fi prezentate mai jos.

Notification Center

Notificariile funcţionează exact ca şi o staţie radio. O staţie radio poate avea mai mai mulţi ascultători, însă aceasta nu ştie nimic despre ei. Cu alte cuvinte este exact ceea ce design pattern-ul Observ presupune. Când un mesaj este trimis oricine care ascultă către acea 'staţie' îl va primi, sub formă de notificare. Obiectul care trimite mesajele nu ştie nimic despre cele care ascultă, sau dacă este cel puţin una care ascultă. De asemenea nu ştie nimic nici despre ce va face obiectul care primeşte notificarea.

În iOS acest concept este foarte uşor de folosit, mai exact, sunt necesare numai două linii de cod. De exemplu, dacă se doreşte o clasa să asculte după notificări trimise de o altă clasa, sau după o notificare cu un anumit nume, tot ce trebuie făcut este scrierea unei linii: [NSNotificationCenter defaultCenter] addObserver:selector:name:object:]. Parametru 'selector' reprezintă metoda care se va apela în momentul în care o notificare este primită.

Pentru a trimite notificări, la fel de uşor, se apelează metoda: [NSNotificationCenter defaultCenter] postNotification:], iar astfel fiecare ascultător va fi notificat.

Key-Value Observing

Este un mecanism care permite obiectelor să fie notificate de fiecare dată când o anume proprietate a unui obiect a fost modificată. Obiectul care doreşte să fie notificat trebuie să se înregistreze la obiectul respectiv folosind metoda:

-addObserver:ForKeyPath:options:context: , astfel se va crea legătura între cele două obiecte.

Delegation pattern

Notificariile ştim că pot realiza comunicarea între unul şi mai multe obiecte, mai exact o relaţie de tip 1-n. În cazul în care dorim să avem comunicare de tip 1-1 putem aplica acest Delegation pattern. Acest şablon foloseşte Protocoale (asemănătoare interfetelor din Java). O clasa care implementează acest şablon are o proprietate de tipul ' id' numită 'delegate', ceea ce înseamnă că acea clasa va implementa metodele definite în protocolul 'MyProtocol'. Metodele definite într-un protocol pot fi obligatorii sau opţionale.

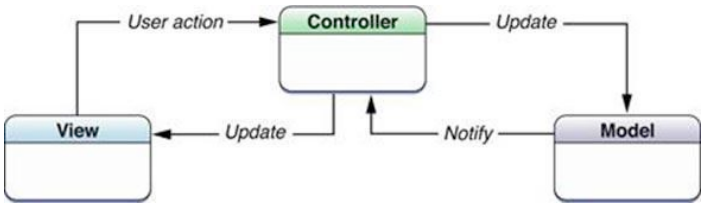

Fig 4.9 - Delegation pattern în iOS

Funcţionalitatea este simplă, în momentul în care clasa care are ca proprietate protocolul transmite mesajul 'delegatului', clasa care implementează acel protocol va apela instant metodele trimise de clasa anterioară.

4.2.6. Core Data

Core Data este metoda de păstrare a datelor oferită de Apple în OS X şi iOS. A fost introdusă în MAC OX 10.4 Tiger şi iOS SDK 3.0. Permite organizarea datelor după modelul relaţional entitate-atribut, ca să fie stocate în fişiere XML, binare sau folosind SQLite. Datele pot fi manipulate folosind obiecte de nivel mai înalt reprezentat obiectele şi relaţiile dintre ele. Nu există chei primare sau străine, relaţiile se fac în funcţie de atribute. Core Data comunica direct cu SQLite fără că cel care proiectează baza de date să aibă vreo legătură cu limbajul SQL.[6]

După o analiza amănunţită a acestui domeniu, se poate observa că acesta este un manager a unui graf care conţine obiecte cu un ciclu de viaţă, care permite căutarea şi păstrarea datelor. Printe funcţionalităţi se încadrează următoarele:

• se pot realiza conexiuni între obiecte

• dacă se schimbă conexiunea la obiectul A atunci şi la capătul B se va actualiza în mod automat

• ştergerea obiectelor poate genera efectul de cascadare şi toatele celelalte obiecte legate de obiectul şters vor fi şterse sau anulate

Toate obiectele Core Data sunt instantiate în obiecte Objective-C care îşi vor controla relaţiile, atributele şi ciclul de viaţă. Acest lucru înseamnă de asemenea că proprietăţile şi comportamentul este implementat prin metode, făcându-le astfel observabile pe ambele.[12]

Core Data este alcătuită din puţine elemente. Este o tehnologie foarte flexibilă, şi de cele mai multe ori partea de setare a configurărilor este foarte uşoară.

Când toate componentele sunt legate împreună, ne putem referi la ele că şi "Core Data Stack". Această este alcătuită din două părţi principale. Prima parte este despre controlarea grafului de obiecte, iar a doua parte despre persistenţa (ex. salvarea stării modelului curent şi încărcarea înapoi din memorie).

Între aceste două părţi, în mijlocul stivei [14], stă componenta de persistenţa şi anume "Persistent store coordinator" (PSC). Acesta face legătură între cele două componente enunţate mai sus, ori de câte ori acestea două vor să comunice între ele. Figura următoare prezintă exact cele prezentate mai sus, într-o imagine uşor de înţeles

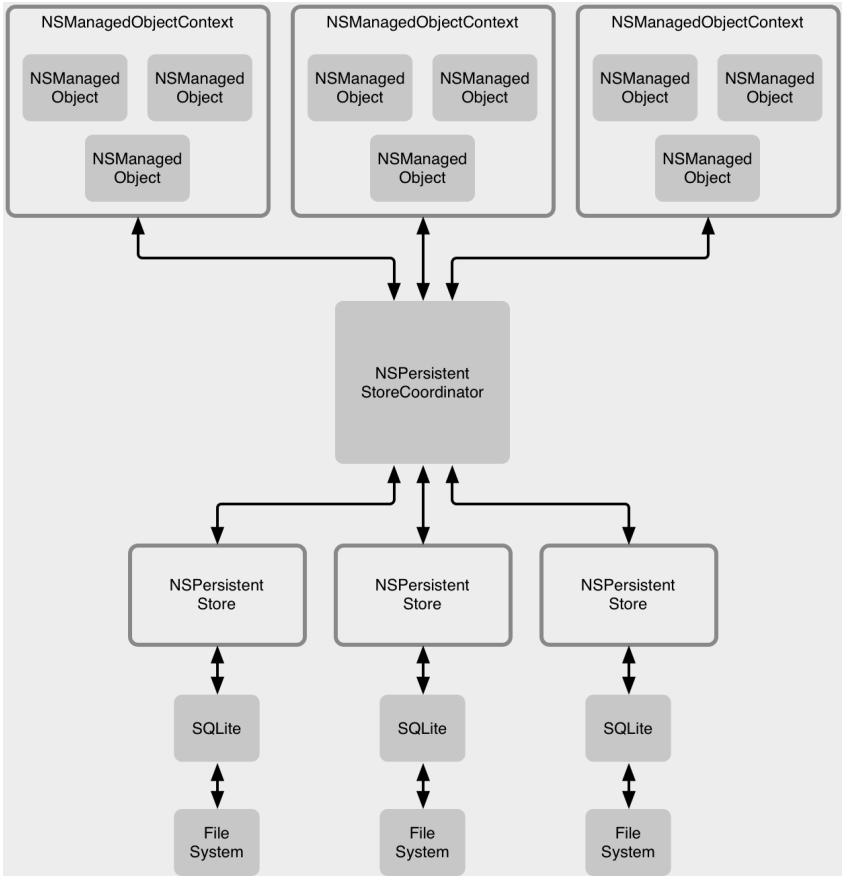

Fig 4.10 - Stiva Core data[14]

Principalul avantaj al acestei tehnologii este integrarea in aplicațiile iOS și din moment ce aceasta este singura metodă la nivelul persistenței în acest gen de aplicații a fost necesară implementarea unei soluții folosind Core Data, deși la implementare de multe ori comunicarea între aplicație și baza de date poate crea dificultăți.

4.2.7. SKMaps

SKMaps este un framework creat şi dezvoltat de Skobbler (by Telenav), companie care acum face parte din Telenav Inc. Acest SDK este construit peste un un layer de C++ care comunica direct cu OSM, de unde se adună toate datele, pentru căutări, creare de rute etc. [21]

Este disponibil în mai multe tehnologii, cum ar fi, iOS, Android şi Web. iOS SDK oferă suport pentru orice device Apple: iPhone, iPad, iPod. În acelaşi timp oferă suport offline pentru toate funcţionalităţile sale. Afişarea hărţilor, calcularea rutelor, navigare "turn-by-turn", afişare puncte de interes, toate sunt posibile fără acces la internet, dar în acelaşi timp pentru a mări viteza în timpul utilizării există şi funcţionalitatea hibridă, online/offline.

Avantajul acestei tehnologii este acela că ofera suport pentru calcularea de rute având un șir de locații transmise prin parametru, dar si ușurința de integrare in aplicație, documentația bogată și modul de implementare care este foarte ușor.

4.2.8. Xcode

Xcode este un pachet software folosit de programatori/ingineri pentru a scrie software pt MAC OS X, iOS devices, Apple Watch şi Apple TV. Este un IDE care conţine editaore, compilatoare şi alte tool-uri care lucrează împreună în a ajuta programatorul să scrie software, să compileze, să îl încarce pe un device, debug, şi pentru a fi trimisă aplicaţia către AppStore. Xcode limbajele: C, C++, Objective-C, Java, AppleScript, Python,Ruby, Rez şi Swift. În plus, include documentaţia Apple pentru programatori şi Interface Builder – tool care permite construcţia interfetelor grafice. De asemenea este singura posibiltate de scrie aplicații de acest gen.

4.3. Cazuri de utilizare

În continuare vor fi prezentate în detaliu principalele cazuri de utilizare, precizându-se precondiţiile, postcondiitle şi fluxul principal, iar în final o diagramă care prezintă toate cazurile de utilizare ale aplicaţiei mobile.

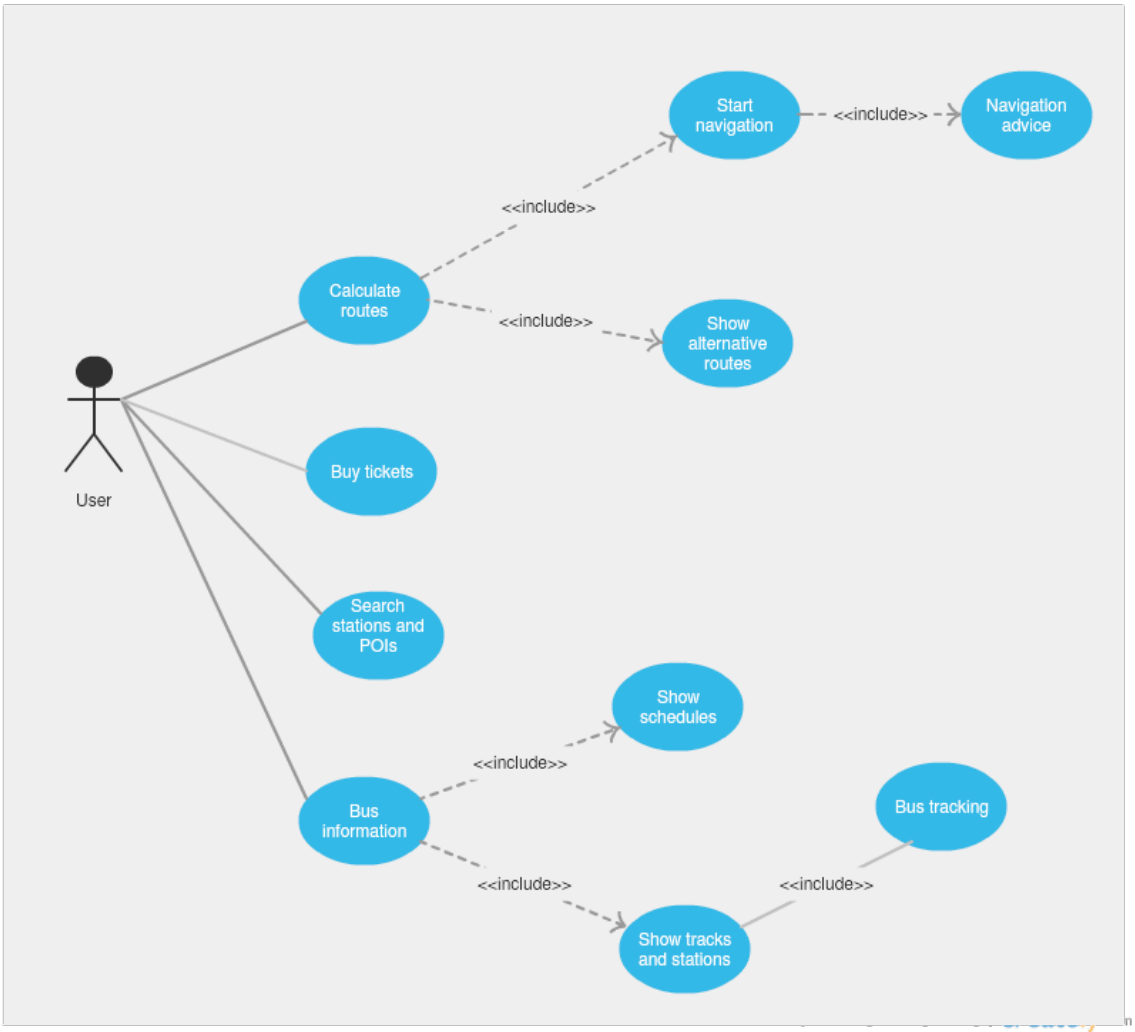

Fig 4.11 - Cazuri de utilizare

Caz de utilizare CU1: Vizualizare informații autobuze (Bus information)

Actor principal: Utilizatorul

Aplicatia trebuie sa fie instalata cu succes pana la final.
Precondiții:
- Actorul trebuie sa aiba acces la harta.

Postcondiții:
- Actorul poate vizualiza orarele, statiile si traseul exact al fiecarui autobuz

Scenariu principal:
1. Actorul acceseaza harta principala.
2. Selecteaza o statie de pe harta.
3. Este prezentat un tabel cu mijloacele de transport care au statia selectata in traseu.
4. Prin selectarea unei linii de autobuz, traseul acestuia va fi desenat pe harta.

Scenariu alternativ:
2a. Utilizatorul acceseaza bara de cautare si cauta dupa o anumita statie.

Caz de utilizare CU2: Urmarire progres autobuz (Bus tracking)

Actor principal: Utilizatorul

Aplicația trebuie sa fie instalata cu succes pana la final.

Precondiții:
- Actorul trebuie sa aiba acces la harta si baza de date sa fie creata in totalitate.

Precondiții:
- Actorul vizualizeaza traseul selectat, cat si autobuzele care se estimeaza a fi pe traseu la ora respectiva.

Scenariu :
1. Actorul acceseaza harta principala.
2. Selecteaza o statie de pe harta.
3. Este prezentat un tabel cu mijloacele de transport care au statia selectata in traseu.
4. Prin selectarea unei linii de autobuz, traseul acestuia va fi desenat pe harta impreuna cu mijloacele de transport care sunt pe traseu la momentul respectiv.

Caz de utilizare CU3: Inceperea unei navigatii (Start navigation)

Actor principal: Utilizatorul

Aplicatia trebuie sa fie instalata in mod corect, iar baza de date locala creata.

Precondiții:
- Actorul trebuie sa aiba acces la harta, iar datele sa poata fi accesate.
- O ruta trebuie sa fie calculata cu succes inainte de a incepe navigatia.

Precondiții:
- Actorul poate vizualiza ruta, opri navigatia, vizualiza statiile urmatoare si primii indicatii despre urmatoarele actiuni.

Scenariu principal:
1. Actorul acceseaza harta principala.
2. Selecteaza o statie de pe harta.
3. Alegera optiunii "Route" prezentata langa numele statiei.
4. Selectare ruta.
5. Start navigatie.
6. Stop navigatie.

Alternative Flow:
2a. Utilizatorul acceseaza bara de cautare si cauta dupa o anumita statie, adresa, punct de interes.

Capitolul 5. Proiectare de Detaliu si Implementare

În acest capitol se va prezenta detaliat fiecare faza a implementării proiectului. Mai întâi sistemul va fi prezentat ca un întreg şi va fi evidenţiată arhitectura conceptuală şi modul de comunicare dintre cele mai mari componente ale sale, iar apoi fiecare din aceste componente se vor trata pe rând pentru a fi descrise, a se evidenţia metoda aleasă pentru implementare şi cum acestea îndeplinesc fiecare cerinţă.

5.1. Structura generala a sistemului

Înainte de a intra în detalii şi a prezenta amănunţit fiecare componentă, trebuie să fie înţeles modul de funcţionare a aplicaţiei şi rolul fiecărei modul implementat. În diagrama următoare este prezentată o arhitectură conceptuală a aplicaţiei, mai exact modul cum fiecare componentă comunica cu celelalte.

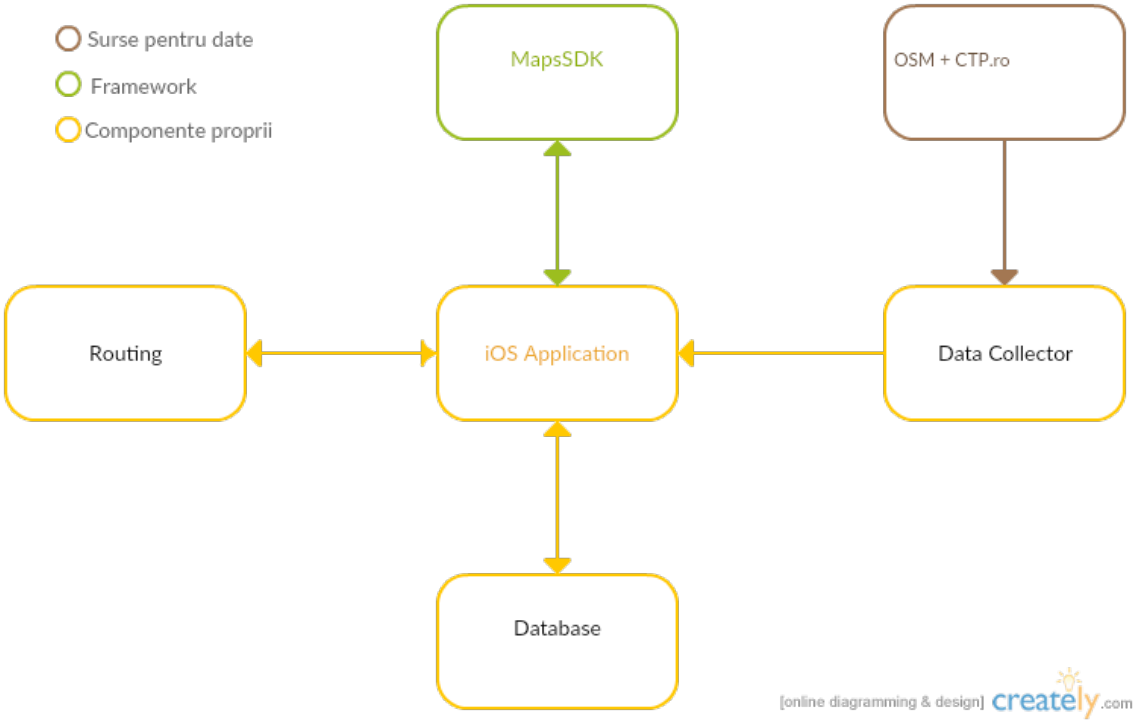

Fig 5.1 - Arhitectura conceptuală a sistemului

După cum se poate observa din diagrama de mai sus, sistemul poate fi împărţit în trei mari părţi, după cum urmează: diferite surse de date - folosite pentru a fi colectate informaţiile necesare, o librărie (framework) care să ofere suport în crearea aplicaţiilor bazate pe hărţi (sau care folosesc hărţi pentru anumite scopuri) şi nu în ultimul rând, mai multe componente proprii create pentru ca aplicaţia să funcţioneze şi să îndeplinească cerinţele cerute.

În primul rând, pentru că acest sistem să poată fi folosit în mod real a fost necesară colectarea mai multor informaţii valide, şi de folos în ceea ce priveşte transportul în comun, cum ar fi:

• staţii de autobuz - nume şi coordonate geografice (pentru a putea fi evidenţiate pe harta)

• trasee/rute ale autobuzelor - reprezentate de un şir de staţii

• orare şi informaţii despre orele de plecare

• bus tracks - (traseu de autobuz) reprezentat printr-un şir de locaţii geografice care urmează exact traseul autobuzului pe harta (aceasta pentru a trata special cazurile unde autobuzele au trasee prin zone unde traficul rutier este interzis)

• tarife pentru călătoriile cu autobuzele

• informaţii despre companiile de taxi

În acest context trebuie să fie menţionat că toate informaţiile amintite care au fost colectate, au fost doar cele pentru a oferi suport în oraşul Cluj-Napoca. Pentru această s-au folosit ca surse principale (aşa cum se observă şi în diagramă) OSM (OpenStreetMap) şi site-ul companiei de transport public din Cluj-Napoca: www.ctp.ro . Toate aceste date au fost colectate în mod automat, prin implementarea mai multor componente şi a unor metode de evaluare care vor fi prezentate mai târziu în această lucrare, în secţiunea corespunzătoare fiecărei componente.

În al doilea rând, librăria care oferă suport pentru folosirea hărţilor în aplicaţie este SKMaps, produs oferit gratuit de Skobbler (by Telenav) aşa cum s-a amintit deja şi în capitolul de fundamentare teoretică.

În al treilea rând, al treilea (şi cel mai consistent) pachet de componente şi module este cel implementat în totalitate pentru această lucrare, începând de la componenta de rutare, care e folosită pentru a calcula rute folosind mijloace de transport în comun, dar în acelaşi timp este şi baza implementării funcţionalităţii care oferă navigaţie dar şi a serviciului de urmărire a autobuzelor. O altă componentă este cea mai sus amintită, folosită în contextul colectării datelor necesare pentru sistem, care constă din mai multe subprograme folosite pentru diferite sarcini pe parcusul colectării datelor. Cea de-a treia componentă proiectată pentru acest sistem este baza de date, care constă într-o arhitectură Core Data şi este locală, pe fiecare client, populandu-se la instalarea aplicaţiei, iar astfel se obţine suportul pentru funcţionalitatea offline.

Pe lângă aceste componente, aplicaţia client, este bazată pe tehnologia iOS şi menită să funcţioneze pe toate dispozitivle mobile care rulează sistemul de operare oferit de Apple. Aplicaţia permite utilizatorului să beneficieze de toate funcţionalităţile specificate în descrierea problemei. Structura aplicaţiei client va fi prezentată în subcapitolele următoare împreună cu detaliile specifice de implementare a aplicaţiilor iOS.

După cum se observă în Fig 5.1, toate componentele implementate sau folosite sunt în strânsă legătură de comunicare cu aplicaţia client. Datele colectate sunt încărcate în aplicaţia client la prima instalare şi se populează baza de date. Când baza de date este consistentă şi creată în totalitate cu ajutorul librăriei de hărţi se pot evidenţia pe harta staţiile de autobuz şi informaţiile necesare. Totodată după crearea bazei de date şi componenta de rutare poate fi folosită, aceasta având nevoie de toate informaţiile despre

autobuze pentru că să poată funcţiona în mod corespunzător şi a împlini cerinţele cerute a fi implementate.

5.2. Componenta de rutare

Acest subcapitol prezintă în detaliu în prima faza proiectarea, iar apoi implementarea acestei componente. În plus va fi prezentată şi funcţionalitatea acesteia şi modul în care datele sunt mapate într-un graf pentru a se putea aplica algoritmul de rutare implementat.

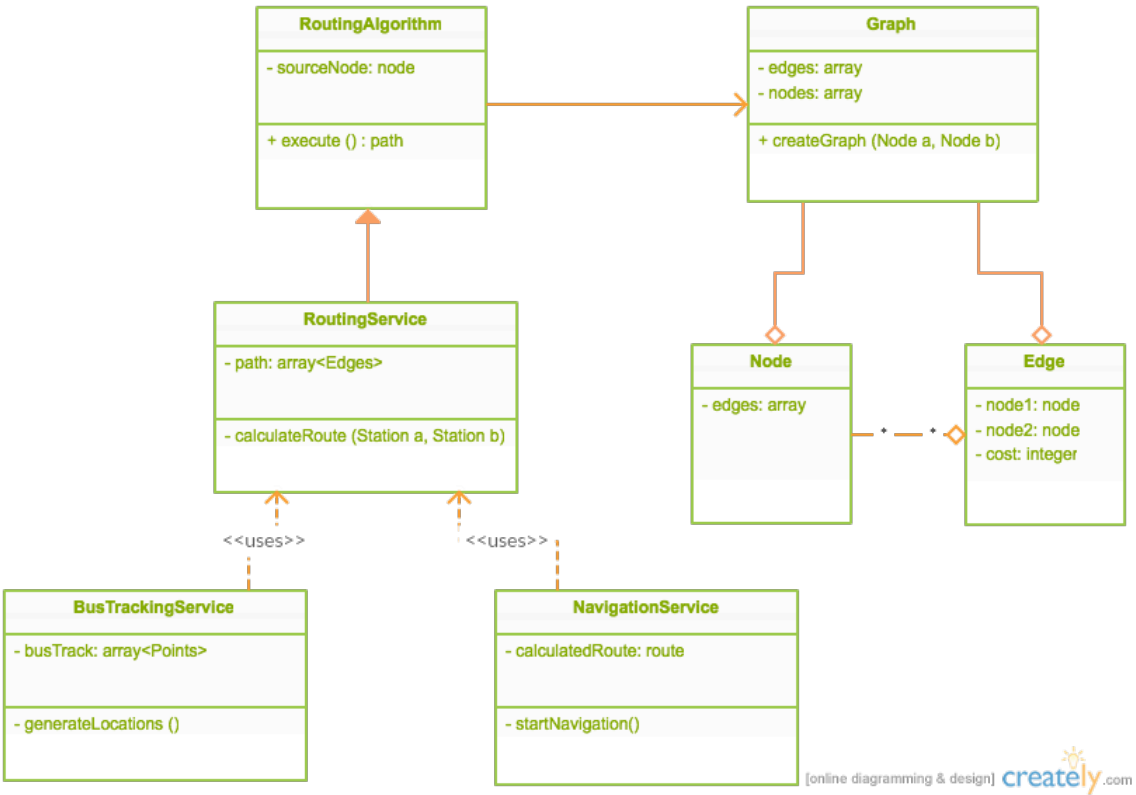

Fig 5.2 - Structura componentei de rutare

Figura de mai sus ilusteaza într-un mod cât mai uşor de înţeles structura componenţei de rutare implementate. După cum se poate observa această componentă are la baza teoria grafurilor. Cel mai de jos nivel al componenţei este reprezentat de bazele unui graf, şi anume: noduri şi muchii. Fiecare nod are în evidenţă muchiile din care acesta face parte dar în acelaşi timp şi un identificator pentru a permite lucrul cu acesta. Două noduri care au o legătură între ele alcătuiesc o muchie, astfel în modelul proiectat fiecare muchie are asociate două noduri. Graful create va fi unul ordonat, de aceea contează ordinea asignării nodurilor unei muchii. Totodată o muchie conţine şi un cost , care va fi atribuit la crearea grafului pentru fiecare muchi în parte şi folosit în algoritmul de rutare pentru a se putea obţine cel mai scurt drum între două noduri.

Graful în modelul dat este alcătuit în principal de mai multe muchii (un set de muchii) dar ar putea conţine şi noduri izolate (mai rar întâlnite în acest sistem) de aceea se păstrează o referinţă şi către nodurile acestea. După cum se observă , acest obiect are o singură metodă, şi anume cea de creare. Pentru a putea aplica algoritmul de calculare a celui mai scurt drum mai întâi a fost nevoie de crearea unui graf pe care să fie aplicat algoritmul. Astfel metoda de construire a grafului pentru calcularea unei rute din punctul staţia A în staţia B este prezentată în continuare.

5.2.1. Construcţia grafului

Înainte de a începe construcţia este necesar a specificarea a două staţii între care se doreşte calcularea unei rute. Deoarece se doreşte o implementare cât mai eficientă a acestei componente fiecare aspect şi detaliu trebuie luat în considerare, cum ar fi în cazul acesta timpul de construcţie a grafului. Dacă la fiecare calculare de ruta graful s-ar construi în totalitate, adică să cuprindă toate staţiile de autobuz din oraş şi toate rutele, efiecienta acestei componente ar fi aproape nulă. O altă abordare în acest sens ar fi crearea grafului în momentul în care aplicaţia este accesată şi ţinerea acestuia în memorie. Această metodă are două mari dezavantaje, primul fiind dat de faptul că aplicaţia ar consuma mult mai multă memorie pe toată durata rulării, iar cel de-al doilea dezavantaj ar fi probabilitatea destul de mare ca să fie necesară doar calcularea unei singure rute la o rulare a aplicaţiei, fapt ce ar duce la ocuparea unui spaţiu destul de mare din memorie fără niciun folos. Singurul avantaj al acestor două metode prezentate până acum este acela că rutele calculate ar fi cu siguranţă cele mai eficiente, având tot graful creat.

Acest avantaj nu a fost considerat un motiv suficient de întemeiat astfel încât să se aleagă una dintre cele două metode, deoarece există algoritmi complecşi care permit obţinerea celei mai eficiente rute fără ca să fie nevoie de încărcarea completă a grafului. Însă scopul acestui lucrări fiind de a oferi posibilitatea de a calcula rute din orice poziţie, nu neapărat doar dintr-o staţie de autobuz către o altă staţie, s-a ales următoarea abordare pentru construcţia grafului.

În primul rând, în graful construit staţiile vor fi reprezentate prin noduri, iar rutele dintre două staţii vor fi reprezentate prin muchii care au asignate un cost , reprezentând timpul necesar deplasării între acele două noduri (staţii). Pentru asignarea acestui cost au fost implementat două metode: Calcularea unei rute între fiecare două noduri şi Calcularea matematică a timpului de parcurgere a distanţei dintre două noduri.

Metoda 1: Calcularea unei rute între fiecare două noduri

În implementarea acestei metode sunt implicate două etape, şi anume: prima etapă este reprezentată de construcţia unei structuri de date care să memoreze toate operaţiile necesare (rutele care trebuie calculate), iar în a două parte calcularea rutelor şi asignarea pentru costului ficarei muchii în parte.

După cum se ştie, într-un oraş unele autobuze pot avea un traseu special, pe unde o librărie folosită pentru a obţine rute nu ar alege varianta de a trece şi prin acele zone, astfel a fost necesar căutarea unei metode speciale, care să permită calcularea rutelor exact printr-un anumit şir de locaţii geografice transmise ca parametru pentru a se putea

obţine exact timpul de parcurgere al traseului de către fiecare autobuz. Astfel librăria aleasă oferă o metodă publică prin care se poate calcula o ruta de acest gen. Metoda este evidenţiată în figura următoare:

```
[[SKRoutingService sharedInstance] calculateRouteWithSettings:self.routeSettings
    customLocations:tempViaPoints];
```

Fig 5.3 - Secvenţă cod calculare rute

În acest apel de funcţie se observă cum o ruta poate fi calculată folosind un şir de locaţii transmise ca paramentru. Dat fiind faptul că răspunsul venit de la această metodă este unul asincron şi vine pe un alt fir de execuţie decât cel principal a fost necesară crearea acelei structuri de date care să permite executarea în ordine a fiecărei operaţii. Astfel această structura este defapt o coadă (structura FIFO), permiţând ca fiecare operaţie să fie executată în ordine. În momentul când se creează o legătură între două noduri un element este plasat în coadă, iar când va fi executat şi se obţine ruta între nodurile respective muchia va fi adăugată în graf având costul asignat.

Această metodă este folositoare şi se poate executa destul de repede mai ales dacă algoritmul de alegere a nodurilor şi muchiilor care trebuie adăugate în graf este unul complex şi se vor alege puţine muchii şi noduri astfel ajungându-se uşor la o soluţie.

Deoarece în această lucrare accentul s-a pus mai mult pe algoritmul de rutare, în ceea ce priveşte construcţia grafului s-a ales o varianta mai simplă, care va fi prezentat mai jos în lucrare), iar de aceea această metodă deşi a fost implementată a fost lăsată deoparte deocamdată în favoarea următoarei metode care va fi prezentată.

Metoda 2: Calcularea matematică a timpului de parcurgere a distanţei dintre două noduri

După cum deja s-a amintit această metodă a fost cea păstrată în soluţia finală a sistemului, din motive stricte de eficientă, astfel durata de calcul a unei rute scăzând în mod considerabil. Metoda se bazează tot pe organizarea operaţiilor într-o coadă însă de dată această, în locul calculării rutelor cu serviciile oferite de SKMaps, s-a ales calcularea lor după un model matematic simplu.

Pentru fiecare muchie care trebuie calculată se ştiu punctele geografice prin care această trece, astfel este uşor să calculăm distanţă între fiecare două puncte, obţinând astfel prin însumarea acestora distanţa finală a segmentului de drum.

Distanţa dintre două locaţii geografice se poate calcula uşor folosind formula de calcul pentru distanţă dintre două puncte:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Această formulă a fost folosită pentru a se calcula distanţele dintre fiecare două locaţii consecutive ale segmentului din traseu. Astfel s-a folosit o metoda din pachetul CLLocation, oferit de limbajul Objective-C, care calculează distanţa aeriană dintre două puncte exact dupa formula prezentată.

Această clasa defineşte o metodă care calculează distanţa dintre două locaţii după cum este ilustrat în figura următoare:

```
CLLocationDistance dist = [start distanceFromLocation:location];

double timeInHours = (dist/1000.0) / 40.0;
distance += dist;
finalTime += timeInHours*60;
start = tempViaPoints[i];
```

Fig 5.4 - Calcularea timpului de parcurgere dintre două puncte

Obiectul "start" cât şi obiectului "location" sunt de tipul CLLocation, care permite prin apelul metodei "distanceFromLocation:" să calculeze distanţa dintre două locaţii, returnând un obiect de tipul "CLLocationDistance" care are defapt la baza tipul 'float'.

După execuţia fiecărei operaţii din coadă costul calculat este asignat muchiei corespunzătoare şi astfel se obţin informaţiile necesare pentru a se putea calcula o ruta.

Aceastea sunt cele două metode implementate pentru calcularea costurilor şi asignarea loc muchiilor corespunzătoare.

Un ultim detaliu în acest subcapitol, care ţine de construcţia grafului este alegerea staţiilor şi muchiilor care vor face parte din graful în care se va cauta drumul ce mai scurt către destinaţie. După cum s-a amintit, accentul nu este pus în această lucrare aşa de mult pe construcţia grafului ci mai mult pe cel de alegere a drumului cel mai scurt. Totuşi, nu s-a ales una dintre metodele prezentate deja posibile pentru a reprezenta staţiile şi rutele lor, ci s-a procedat în felul următor. Presupunem că utilizatorul este la poziţia A şi doreşte să ajungă la poziţia B. Pentru această se caută cea mai apropiată staţie de la poziţia curentă a utilizatorului, dacă această distanţă este mai mică de 1000m, atunci căutarea continuă până se găsesc maxim patru staţii, cele mai apropiate. Acelaşi algoritm se aplică şi pentru poziţia finală a utilizatorului, iar astfel se găsesc un număr de staţii care constau în baza construirii grafulu. După ce acestea au fost alese se creează în primul rând muchiile de la nodul care reprezeinta poziţia curentă şi cea finală către staţiile apropiate fiecăreia. Aceste rute vor fi marcate special cu cu marcajul "pedestrian" deoarece costurile lor sunt calculate folosind serviciile SKMaps (deoarece sunt puţine la număr). În pasul următor se consideră fiecare staţie în parte şi se adaugă toate rutele care trec prin staţia respectivă.

După ce graful este creat astfel, în ultimul pas, pentru fiecare două staţii care sunt situate la o distanţă mai mică de 300m se adaugă o ruta de tipul "pedestrian" considerată folositoare în anumite cazuri când autobuzul ar trebui schimbat şi este necesară deplasarea dintr-o staţie în altă.

Aceasta este o metoda de construcţie a grafului abordată în această lucrare, stand la baza componentei de rutare. În continuare va fi prezentat şi algoritmul de rutare care permite calcularea rutelor hibride, atât cu mijloacele de transport cât şi prin rutare pietonală.

5.2.2. Algoritmul de rutare

Principala funcţionalitate pe care s-a axat proiectarea acestei componenete de rutare este algoritmul prin care se decide cea mai bună soluţie. Principalul scop este de a obţine drumul cel mai scurt, unde costul este reprezentat de timp, astfel se doreşte obţinerea unei rute care necesită cel mai scurt timp pentru a ajunge la destinaţie.

În capitolul referitor la fundamentarea teoretică a fost prezentată în detaliu funcţionarea şi logică algoritmului lui Dijkstra, iar din acest motiv nu va mai fi reluată aici, ci în această secţiune se vor prezenta doar îmbunătăţirile aduse acestui algoritmului pentru a se mapa cât mai bine pentru problema abordată şi pentru a obţine rezultate cât mai eficiente. Algoritmul permite calcularea rutelor dintr-un punct către toate celelalte puncte ale grafului, precum a fost deja precizat în capitolul anterior. Astfel după rularea algoritmului se obţin toate drumurile către fiecare nod al grafului, însă se va alege şi se va păstra doar drumul către staţia destinaţie aleasă de utilizator.

După cum fiecare nod are asignată o staţie de autobuz şi un identificator, pentru a lucra mai uşor cu acestea, rezultatul în urmă rulării va fi reprezentat printr-un şir de muchii, astfel având şi nodurile/staţiile prin care trebuie trecut dar şi autobuzul cu care se poate face deplasarea între cele două staţii.

Mai întâi s-a realizat implementarea algoritmului lui Dijkstra aşa cum a fost prezentat, şi aşa cum funcţionează el în mod normal. După rularea mai multor teste şi calcularea mai multor rute s-a observat că autobuzele care sunt alese pentru deplasare sunt schimbate frecvent (uneori chiar de la o staţie la altă, deşi s-ar fi putut continuă mai departe cu autobuzul curent). S-a înţeles că această este o problema şi soluţia prin care se poate rezolva este adăugarea unui cost mai ridicat în momentul când autobuzul trebuie schimbat. Această abordare a fost implementată prin adăugarea de muchii de la nodul de unde porneşte autobuzul către toate nodurile care sunt cuprinse pe traseul acestuia, astfel s-a poate hotărî mutl mai repede dacă ar există o cursa directă şi dacă un autobuz se poate păstra pentru o perioada mai lungă de timp. Scopul fiind ca să se ajungă la destinaţie în timpul cel mai scurt folosind când mai puţine autobuze.

Pentru rezolvarea acestei probleme s-a hotărât modificarea funcţiei de evaluare a costurilor în momentul când trebuie să fie luată o decizie nouă în alegerea muchiei următoare. Asfel funcţia de evaluare nu mai constă doar în evaluarea şi alegerea muchiei cu cost mai mic, ci şi în încercarea de a păstra autobuzul curent dacă este posibil. Astfel s-a decis implementarea unei structuri de date cu ajutorul căreia să se ţină evidenţă numărului de autobuze schimbate până în acel moment şi mărirea costului muchiei respective dacă necesită schimbare de autobuz.

În această implementare, funcţia construită este bazată pe mai mulţi factori, care influenţează decizia în anumite proporţii. Primul factor este cel care a fost de la început, şi anume costul, care este repezent de timpul dintre cele două noduri şi care s-a considerat că nu este la fel de costisitor decât schimbarea autobuzului. Spre exemplu, în majoritatea cazurilor un utilizator ar preferă să stea câteva minute în plus pe drum, decât să fie nevoit să mai cumpere un bilet pentru a schimba autobuzul. În plus pe lângă biletul de călătorie folosit în plus se va mai pierde timp şi pentru aşteptarea noului autobuz. De aceea în abordarea prezentată se consideră mai de folos continuarea drumului cu acelaşi autobuz acolo unde este posibilă, chiar dacă aceast constă într-un mic ocol ceea ce îl va determina pe utilizator să petreacă mai multe momente pe drum, deşi această este o abordare doare teoretică, dat fiind faptul că deşi algoritmul ar spune că schimbarea autobuzului va scurta

timpul, nu se ştie niciodată când autobuzul va ajunge în staţie şi cât de repede se va deplasa spre destinaţie, aici fiind implicat şi un factor foarte important care nu este luat în cosiderare în această soluţie, şi anume: traficul.

Funcţia de evaluare amintită mai sus este prezentată în cele ce urmează, fiind ilustrată matematic, iar mai târziu va fi prezentată şi în implementare.

$$factor = \mathrm{R} * \frac{80}{100} + \mathrm{C} * \frac{20}{100}$$

R - numărul de autobuze care vor fi schimbate dacă se alege aceasta muchie
C- costul muchiei

După cum se poate observa foarte uşor, proporţiile alese în hotărârea cărei muchii se va alege în pasul următor sunt astfel: 80% contează în acest factor numărul de autobuze care sunt schimbate, iar 20% costul efectiv reprezentat de timp. Astfel acest factor este calculat pentru fiecare muchie, iar la final muchia următoare va fi muchia cu cel mai mic factor. Implementarea acestei metode este ilustrată în figura următoare:

```
for (PTEdge *edge in node.edges) {
    if ([edge.node2 isEqual:target]) {
        double routes = 0;
        if (self.routes[node.identifier]) {
            routes = self.routes[node.identifier].count;
        }
        double factor = routes * (80.0/100.0) + (edge.cost * (20.0/100.0));

        if ([minimumEdge.node1 isEqual:edge.node1] && [minimumEdge.node2 isEqual:edge.node2]) {
            if (minimumEdge.otherRoutes == nil) {
                minimumEdge.otherRoutes = [NSMutableSet new];
            }
            [minimumEdge.otherRoutes addObject:edge.routeInfo.name];
        } else if (factor < minFactor) {
            minFactor = factor;
            minimumEdge = edge;
        }
    }
}
```

Fig 5.5 - Implementarea funcţiei de decizie

Pentru implementarea acestei funcţii se poate observa uşor folosirea metodei matematice mai sus amintită, care funcţionează astfel: metoda este menită să returneze următoarea muchie a drumului din nodul curent, unde s-a ajuns până acum. În primul rând se parcurg toate muchiile care au ca nod sursă nodul respectiv, iar acum inloc să tratăm doar cazul costului să fie verificat doar acesta,mai întâi se calculează factorul aminit. Obiectul "self.routes" este un dicţionar care conţine ca şi chei nodurile şi ca valoare numărul de autobuze care sunt schimbate dacă se trece prin acel nod, astfel obţinem R din formulă amintită. Costul C este uşor de obţinut, fiind asignat fiecărei muchii. După calcularea factorului, dacă valoarea nouă obţinută este mai mică decât cea mai bună de până în acest moment, se va păstra muchia curentă ca cea mai bună alegere şi se va actualiza factorul minim. Această filtrare se aplică pentru toate celelalte muchii ale care trec prin nodul curent şi mai departe pentru fiecare nod până se ajunge la destinaţie.

5.2.3. Serviciul de navigatie

Un serviciu de navigaţie este poate fi extrem de folositor mai ales în zilele acestea când reţelele de transport dintr-un oraş au crescut şi s-au existins într-un mod considerabil. Chiar şi în oraşul Cluj-Napoca poate reprezenta o problema, mai ales pentru turişti sau pentru cei care locuiesc doar de puţină vreme în oraş. De aceea, sistemul construit este axat în totatlitate pe simplicafarea transportului în comun pentru utilizatori.

După cum a fost prezentat în secţiunile anterioare, sistemul deja oferă o ruta între oricare două locaţii din oraş folosind mijloacele de transport în comun şi deplasarea pietonală. După implementarea acestei componente s-a hotărât să se meargă mai departe, şi de ce nu, dacă deja o ruta deja este calculată, să fie disponibil şi un sistem de navigaţie, care să ofere informaţii minime despre staţiile următoare, autobuzele care trebuie aşteptate etc.

Pentru aceasta a fost implementat un serviciu separat, la fel ca şi cel de rutare, însă care este bazat pe acesta. Acest serviciu este foarte uşor de folosit, după ce o ruta este calculată tot ceea ce trebuie făcut este de a apela metoda 'startNavigation:', şi selectată clasa care implementează delegatul serviciul de navigaţie, pentru a primi notificările de fiecare dată când un eveniment se apropie sau diferite informaţii sunt disponibile pentru utilizator.

Interfaţa serviciul de navigaţie conţine următoarele funcţionalităţi (Figura 5.6), care vor fi prezentate în cele ce urmează.

```
/** Navigation protocol
 */
@protocol PTNavigationDelegate <NSObject>

- (void)routingService:(PTRoutingService *)routingService didChangeDistanceToDestination:(int)distance
    withFormattedDistance:(NSString*)formattedDistance andOtherRoutes:(NSString *)otherRoutes;

- (void)routingService:(PTRoutingService *)routingService didChangeCurrentStreetName:(NSString *)currentStreetName;

- (void)routingService:(PTRoutingService *)routingService didChangeNextStreetName:(NSString *)nextStreetName;

- (void)routingService:(PTRoutingService *)routingService didChangeCurrentAdviceImage:(UIImage *)adviceImage
    withLastAdvice:(BOOL)isLastAdvice;

- (void)routingService:(PTRoutingService *)routingService didChangeCurrentVisualAdviceDistance:(int)distance
    withFormattedDistance:(NSString *)formattedDistance;

- (void)routingService:(PTRoutingService *)routingService didChangeDistance:(int)distance toNextStation:(PTStation *)
    station andOtherRoutes:(NSString *)routesInfo;

- (void)routingService:(PTRoutingService *)routingService didChangeNextStationName:(NSString *)stationName;

- (void)routingService:(PTRoutingService *)routingService didReachStation:(PTStation *)station;

- (void)routingService:(PTRoutingService *)routingService didChangeDistanceToFinalDestination:(int)distance;

- (void)routingServiceDidReachFinalDestination:(PTRoutingService *)routingService;


@end
```

Fig 5.6 - Interfaţa serviciului de navigaţie

Figura de mai sus ilustrează foarte clar fiecare metodă ce face şi la ce poate fi folosită. Numele serviciilor sunt foart explicite, indiferent dacă este necesar un num lung pentru o metodă, acestea trebuie să fie clare şi prin simpla citire a numelui trebuie să se înţeleagă funcţionalitatea acesteia, fără a fi nevoie de citirea documentaţie (un lucru întâlnit în limbajul Objective-C foarte des, care poate fi numit chiar o caracteristică a acestui limbaj).

Notificările care se obţin de la serviciul de navigaţie pot fi pentru: schimbarea distanţei finale până la destinaţie, schimbarea străzii curente - care returnează şi numele ei-, schimbarea numelui străzii următoare, oferă informaţii despre următoarele manevre care trebuie făcute sau direcţii care trebuie urmate de către utilizator, schimbarea distanţei până la staţia următoare, notificarea utilizatorului când a ajuns într-o staţie anume sau notificarea acestuia când a ajuns la destinaţie.

Toate aceste servicii sunt folosite în aplicaţia iOS pentru a fi implementat sistemnul de navigaţie disponibil, împreună cu interfaţă grafică folosită.

5.3. Colectarea datelor

După cum s-a amintit de mai multe ori deja în această lucrare, o etapă importantă şi destul de dificilă din acest proiect a fost colectarea tuturor datelor necesare pentru a putea creea componentă de rutare şi aplicaţia care o include. În colectarea datelor au fost mai multe etape: prima etapă a constat în colectarea datelor aşa cum sunt disponibile pe internet, după care a urmat în a două etapă selecţia datelor necesare, apoi validarea acestor date, iar în ultima etapă fost construit layer-ul de date care conţine toate informaţiile necesare sistemului propus, fiind cuprinse într-un fişier JSON, care va stă la baza creeari bazei de date în momentul instalării aplicaţiei.

5.3.1. Colectare date CTP.ro

Prima sursă de date, a fost bineînţeles site-ul companiei de transport public din Cluj-Napoca. În prima fază s-au inspectat informaţiile care sunt oferite, şi s-au identificat a fi de folos, iar apoi a început colectarea lor.

În prima faza a colectării s-a identificat formă link-ului care conţine informaţiile necesare, în urmă inspectării codului sursă a paginilor de pe site. După ce a fost identificat link-ul a început crearea unui web scraper care să ajute la descărcarea acestor fişiere. În implementarea acestei componete s-a folosit o conexiune HTTP de tipul GET care a fost setată să facă un request o dată la 2-3 secunde pentru a descarca un fişier (spre exemplu fişierul care conţine orarele autobuzului 25). Componenta a fost lăsat să ruleze mai mult timp, iar încet toate fişierele au fost descărcate.

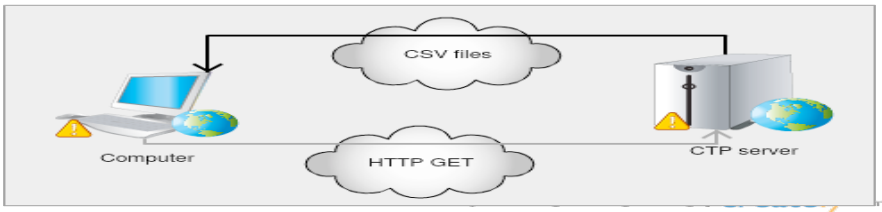

Fig 5.7 - Conexiune client - server

Fiecare răspuns de la server este sub formă unui fişier CSV care a fost pe urmă parcurs cu ajutorului unui program implemenetat pentru această şi s-au extras informaţiile necesare din acesta. Datele care nu a fost dipsonibile pe site-ul companiei dar totuşi au fost necesare pentru aplicaţie, cum ar fi locaţiile staţiilor sau track-urile fiecărui autobuz au fost colectat din OSM, iar metoda de lucru va fi prezentată în secţiunea următoare.

5.3.2. Colectare date OSM

OpenStreetMap după cum deja se ştie este o sursă foarte puternică de date în ceea ce priveşte hărţile, puncte de interes, locaţii, adrese şi multe alte lucruri de acest gen. În sistemul implementat a fost nevoie de anumite date care le-am putut găsi aici mult mai uşor decât în alte părţi. Mai exact, locaţiile staţiilor din oraş dar şi traseele autobuzelor au reuşit a fie colectate de aici. Deoarece OSM este bazat pe comunitate şi elementele sale sunt adăugate în mod voluntar, evident nu se poate găsit chiar tot ce se caută şi mai ales să fie şi actualizate. Însă, în pofida acestui dezavantaj s-au găsit aproape toate staţiile de autobuz din oraş (cu excepţia celor mai noi), dar şi traseele acestora, punct cu punct.

Accesul la datele de pe OSM este grauit pentru oricine prin simplu acces al website-ului. Dacă se doreşte exportarea unei porţiuni din harta, se aproprie până la nivelul minim permis (deoarece dacă se selectează o porţiune prea mare este o cantitate imensă de informaţii care nu poate fi exportată) şi se apasă butonul de pe bara de sus numit "Export" care va genera un fişier cu extensia ".osm" care este defapt un fişier XML, care poate fi uşor prelucrat, iar apoi se pot extrage informaţiile necesare, acestea fiind structurate prin anumite marcaje.

După obţinerea fişierului este foarte uşoară implementarea unui parser cu ajutorul căruia să extragem informaţiile de la anumite marcaje, cele care sunt de folos. După cum se menţiona mai sus, datele nu sunt întodeauna actualizate şi mai conţin şi erori, iar din pricina această a fost necesar şi un proces de validare a date după selecţia acestora, prezentat pe scurt în secţiunea următoare.

5.3.3. Selecţia si validarea datelor

Ca urmare a procesului de colectare prezentat până în acest moment s-au obţinut mai multe fişiere de tipul XML şi CSV. Toate datele necesare în aplicaţie sunt cuprinse în aceste fişiere, dar sunt în acelaşi timp cu multe alte date de care nu este nevoie. Astfel s-au extras din aceste fişiere doar informaţiile necesare, mai exact: din fişierele CSV (adică informaţiile de pe site-ul companiei) s-au extras numele staţiilor, orele de circulaţie ale fiecărui autobuz corespunzătoare fiecare zilei a săptămânii, staţiile corespunztoare fiecărei rute - tur şi retur - şi informaţiile necesare achiziţionării biletelor de autobuz direct din aplicaţia mobilă, iar din fişierul XML s-au selectat track-urile fiiecare linii de autobuz împreună cu locaţiile geografice ale fiecărei staţii. În plus, staţiile care nu au fost identificate în fişierul exportat din OSM, au fost adăugate manual şi astfel şi validate.

După selectarea acestor date şi folosirea lor în aplicaţie, în momentul efectuării unor teste şi afişarea traseelor autobuzelor pe harta s-a constat că punctele care alcătuiau aceste trasee nu erau adauggate în ordine în această structura de date. Pentru a remedia această problema s-a construit un program care a avut ca rol validarea acestor track-uri prin parcurgerea lor punct cu punct şi verificarea distanţei dintre acestea. Dacă două puncte consecutive au distanţă între ele mai mare de 200m înseamnă că e o probabilitate destul de mare ca acel punct să nu fie poziţionat corect (deoarece în mod normal punctele sunt la o distanţă mică unu faţă de celălalt pe harta) şi ca urmare va fi eliminat. După rularea acestui program pe fiecare ruta de autobuz s-a obţinut traseul corect al fiecăruia şi astfel s-a putut continua dezvoltarea aplicaţiei având în layer de date stabil, selectat şi validat.

Pentru a lucra mai uşor cu acestea, toate datele corecte şi selectate au fost scrise într-un fişier de tipul JSON care va sta la baza popularii bazei de date în momentul când aplicaţia va fi rulată pentru prima dată pe telefon.

5.4. Aplicaţia iOS

Aplicaţia iOS are ca rol integrarea tuturor componentelor. Deoarece şi componentă de rutare a fost scris tot în Objective-C s-a hotărât ca în prima versiune acestui sistem totul să se desfăşoare direct pe client, de aceea în această secţiune se va prezena foarte scurt ce s-a implementat în plus faţă de componenta de rutare, modul cum a fost implementat designul aplicaţiei şi o descriere scurtă a arhitecturii aplicaţiei, deoarece este una simplă, la fel ca orice altă aplicaţie iOS şi este construită folosind arhitectură MVC.

În mare parte după ce serviciile au fost implementate a rămas doar crearea unei interfeţe grafice care să poată permite evidenţierea tuturor functionalitatilor cerute şi dezvoltate în acest sistem.

Pentru evidenţierea acestor servicii este bineteles nevoie mai întâi de orice de harta, unde se vor evidenţia majoritatea functionalitatilor. Acest aspect se va trata într-un subcapitol puţin mai târziu şi va fi evidenţiat fiecare pas pentru integrarea hărţilor şi pentru a afişa ceva pe harta.

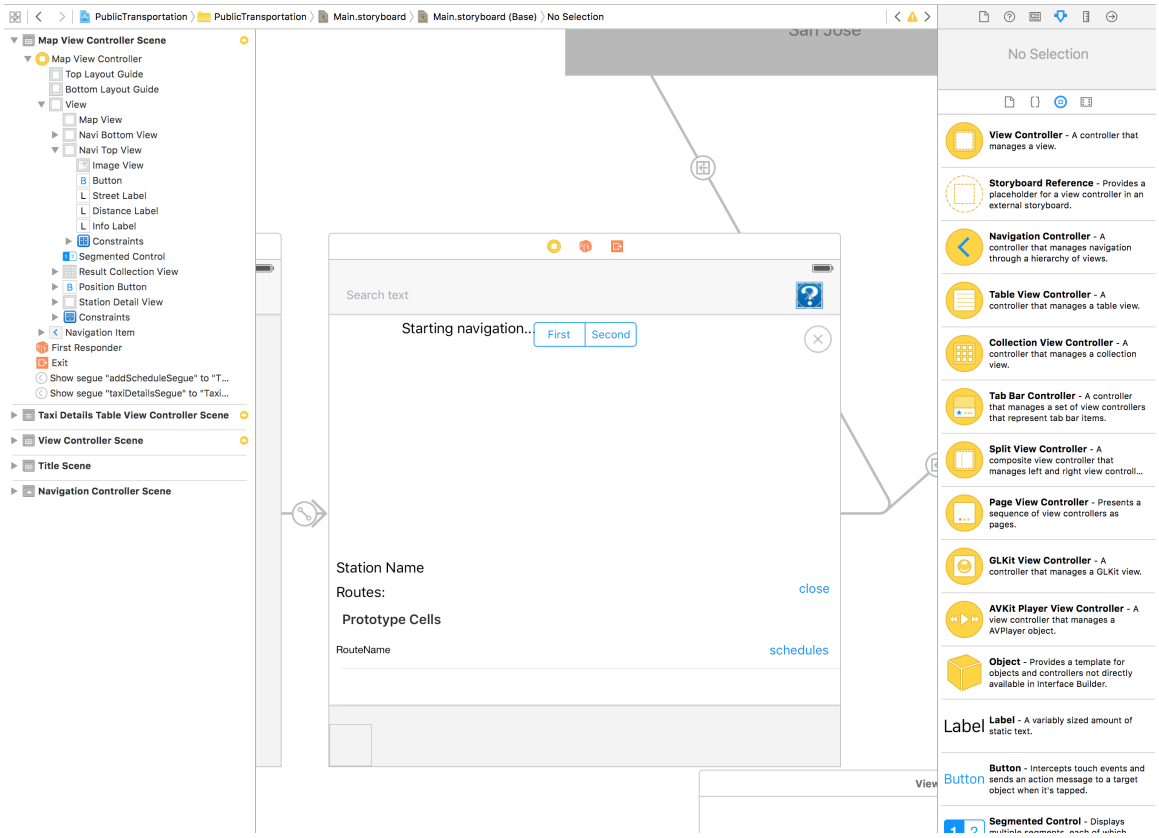

Fig 5.8 - Crearea unei interfeţe grafice

În plus, pentru interfaţă grafică s-au folosit tehnologiile specifice iOS, şi anume:Interface Builder oferit de Xcode. Cu această facilitate Xcode permite oricărui dezvoltator de aplicaţii să creeze interfeţe grafice foarte uşor, folosind atât tehnică "drag-and-drop" cât şi combinarea acestui cu scriere de cod. În continuare se va prezenta un exemplu de implementare a unui View prezentat în parcursul aplicaţiei, cum acesta a fost creat şi cum se face legătură din interfaţa grafică cu obiectele de model folosite.

Folosind acest utilitar ofert de Xcode, se poate adauga un ecran nou foarte uşor aplicaţiei adauganduse din partea dreapta a ecranului. Un ecran nou înseamnă adăugarea unui obiect din lista afişată în partea dreapta, care să fie de tipul "ViewController". Acesta se poate lega de oricare alt ecran printr-o legătură numită "segue" care va avea ataşată o metodă care se va executa în momentul când tranziţia dintre cele două ecrane va avea loc.

Pentru adăugarea oricărui alt element de interfaţă grafică se alege tot din lista amplasată în partea dreapta a figurii şi se face "drag-and-drop" în locul în care se doresta să fie pe ecran. În partea stânga a figurii se observă o ierarhie de view-uri, ceea ce reprezintă ierarhia din ViewController-ul selectat şi cum fiecare view/buton/label etc este aranjat pe acel ecran.

Legăturile dintre elementele de UI se pot realiza prin selectarea elementului, apăsarea butonului "Ctrl" de pe tastatură şi apoi crearea unei legături cu mouse-ul de la elementul selectat în fiserul .h sau .m al clasei respective (depinde dacă se doreşte a fi public sau privat). Când se hotărăşte unde să fie plasată proprietate un dialog va apărea pe ecran (Figura 5.9) şi se va cere un nume şi tipul exact al noului obiect care va fi adăugat.

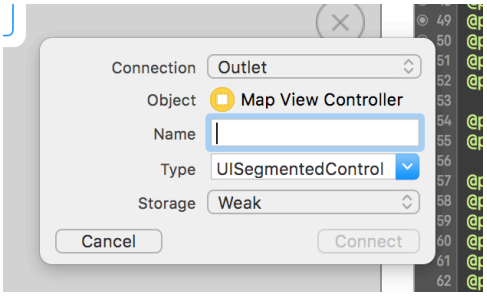

Fig 5.9 - Dialog pentru denumirea elementului de UI

Aici se va trece numele obiectului nou, iar apoi tip-ul obiectului (care de obicei este auto-completat în funcţie de tip-ul obiectului a cărui legătură s-a creat din Interface Builder). După ce proprietatea s-a creat se poate folosi orinde în aplicaţie şi conţinutul acelui view/buton/label poate fi modificat din interiorul codului. Bineînţeles aceste relaţii între elementele de UI şi aplicaţie se realizează numai în interiorul obiectelor de tip ViewController sau View, astfel păstrându-se şi îndeplinindu-se principiile arhitecturii MCV.

5.4.1. Integrarea librăriei SKMaps

Librăria SKMaps s-a folosit în aplicaţia iOS ca suport pentru hărţi şi reprezentarea punctelor de interes pe harta, staţiilor de autobuz, rutelor etc. Pe lângă aceste funcţionalităţi această librărie oferă şi suport pentru căutare de adrese, puncte de interes şi alte locaţii. În continuare se va prezenta modul de integrare a acestei librării într-un

proicet iOS şi totodată se vor prezenta şi anumite secvenţe de cod spre a fi exemplificată usrinta folosirii librăriei.

Cerinţe ale integrării:

• Xcode 5 sau mai nou.
• Arhitectură: armv7, arm64
• iOs 7.0 sau mai nou

După descărcarea de pe site-ul www.skobbler.ro arhiva conţine următoarele :

• demo app - pentru ilustrarea functionalittii
• resurse pentru hărţi si resurse audio

Pentru un proiect Objectiv-C adăugarea librăriei se face după urmatorii pasi:

1) Se adaugă SKMaps.framework in proiect prin 'drag-and-drop'.
2) Prin aceeasi metoda se adauga si fisierul SKMaps.bundle si SKAdvisorResources.bundle.
3) Se selecteaza proiectul si se alege target-ul. Se deschide pagina "Build Phases" si se adauga urmatoare librarii in sectiunea "Link Binary with Libraries":
   a) UIKit.framework
   b) Foundation.framework
   c) CoreLocation.framework
   d) OpenGLES.framework
   e) QuartzCore.framework
   f) CoreGraphics.framework
   g) CoreMotion.framework
   h) libz.dylib
   i) libc++.dylib
4) Se deschide pagina "Build settings" si pentru optiunea "Other linker flags" se intrduce "-ObjC".
5) Se importează header-ul "SKMaps/SKMaps.h" pentru a se putea folosi libraria.

Următorul pas constă în inițializarea librăriei și setarea "API Key" pentru a putea fi folosită. Un astfel de "key" se poate gasi dupa ce s-a creat un cont pe site-ul skobbler si este unic pentru fiecare utilizator al librariei. Initializarea se face in felul urmator:

```
[[SKMapsService sharedInstance] initializeSKMapsWithAPIKey:@"API_KEY"
settings:settings];
```

In următorul exemplu este exemplificată adăugarea unei hărți pe un anumit ecran al aplicației si afișarea acesteia.

```
#import <SKMaps/SKMaps.h>

        @implementation MapDisplayViewController

        - (void)viewDidLoad{

        [super viewDidLoad];
```

```
        SKMapView *mapView = [[SKMapView alloc] initWithFrame:CGRectMake( 0.0f,
0.0f,  CGRectGetWidth(self.view.frame), CGRectGetHeight(self.view.frame) )];

        [self.view addSubview:mapView];

        }
```

Medotda "viewDidLoad" se apelează prima în momentul când acel view va trebui să fie încărcat pentru afişare, de aceea aici este un loc potrivit pentru a se face initializarea hărţii, deoarece când va apărea acel view pe ecran să fie harta deja încărcată. În următorul rând se creează obiectul de tip SKMapView şi se initializeaza cu un frame care în cazul acesta ocupă toată suprafa ecranului. În ultimul rând harta se adaugă pe ecranul prezentat folosind metoda "addSubview:".

De asemenea mai multe setări sunt disponibile pentru a fi setate foarte uşor, pentru diferite ipostaze la hărţii:

```
mapView.settings.rotationEnabled = NO;

mapView.settings.followUserPosition = YES;

mapView.settings.headingMode = SKHeadingModeRotatingMap;

mapView.settings.zoomLevel = SKHeadingModeRotatingMap;
```

Pentru identificarea acțiunilor care au loc pe hartă, spre exemplu ale anumitor gesturi: tap, pinch, long press etc se foloseste pattern-ul Delegate, prezentat in capitolul anterior, după cum urmează:

```
        SKMapView *mapView = [[SKMapView alloc]initWithFrame:[[SKMapView alloc]
initWithFrame:CGRectMake( 0.0f, 0.0f,  CGRectGetWidth(self.view.frame),
CGRectGetHeight(self.view.frame) )];
        mapView.delegate = self;
         [self.view addSubview:mapView];
        - (void)mapView:(SKMapView*)mapView
didChangeToRegion:(SKCoordinateRegion)region{
        }
        - (void)mapView:(SKMapView*)mapView
didTapAtCoordinate:(CLLocationCoordinate2D)coordinate{
        }
        - (void)mapView:(SKMapView*)mapView didRotateWithAngle:(float)angle{
        }
```

După ce o harta va fi adăugată ca subview pe un ecran aceasta trebuie să fie setată ca să asculte după metodele de delegat care vor fi apelate în momentul unei acţiuni a utilizatorului cu harta. După ce acest delegate se setează, se poate observa în exemplul de cod cum metodele care sunt declarate în Protocolul SKMapViewDelegate, trebuie implementate în clasa respectivă. Când o acţiune va avea loc, metoda corespunzătoare se va apela, iar astfel programatorul poate trata acest eveniment şi decide ce secvenţă de cod se va apela la un anumit gest detectat.

Acest principiu este folosit foarte des în aplicaţiile iOS deoarece este foarte uşor de implementat şi este unul dintre mijloacele principale de comunicare între două obiecte Totodată în majoritatea cazurilor în această aplicaţie unde două obiecte trebuie să comunica este folosită această metodă. A fost prezentat acest caz doar pentru a se evidenttia acest aspect şi a se înţelege metoda de funcţionare.

5.5. Proiectarea bazei de date

După cum şi o aplicaţie Android poate avea o bază de date locală, la fel şi aplicaţiile iOS pot să aibă acces la astfel de servicii. Metoda de rezolvare a persistenţei în aplicaţiile specifice produselor Apple este CoreData, tehnologie care a fost prezentată între celelalte tehnologii folosite în aplicaţie.

Baza de date proiectată pentru această aplicaţie are următoarea structura:

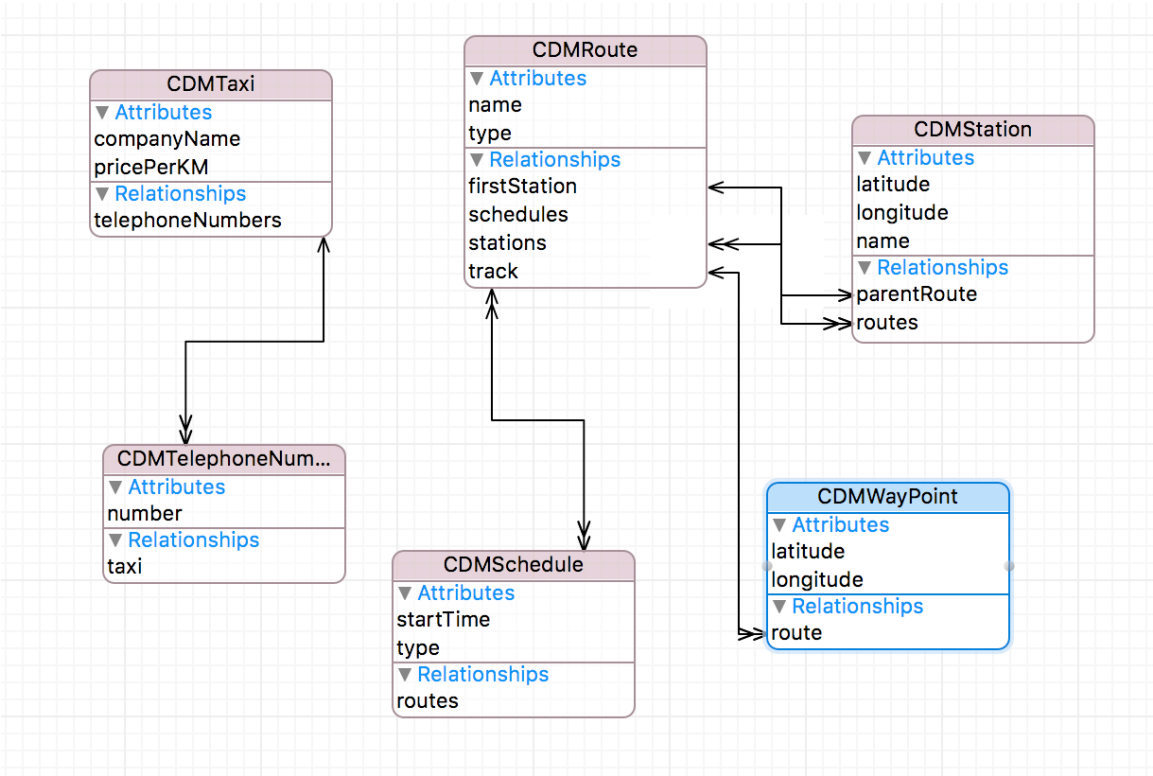

Fig 5.10 - Structura bazei de date

Xcode permite crearea unei astfel de structuri foarte uşor, oferind facilitatea de a modela baza de date atât în modul ilustrat mai sus cât şi la nivel de entităţi. Comunicare cu CoreDate se face prin nişte obiecte speciale, moştenite din clasa NSManagedObject, ceea ce înseamnă că este nevoie de un model special construit doar pentru persistenţa

care oferă metode pentru a fi transformate obiectele în cele construite în modelul aplicaţiei.

5.6. Structura finală a sistemului

În final este prezentată o vedere de ansamblu a întregului sistem creat, o structura puţin mai detaliata decât cea prezentată iniţial, însă în care se poate observa exact comunicarea propusă în arhitectură iniţială. De asemenea o legendă este inclusă în această diagramă pentru a se înţelege şi a se deosebi mai uşor componentele între ele.

După cum se poate observa fiecare nivel de componente este încadrat într-un dreptunghi. Componentele, pachetele şi serviciile implementate specific pentru această lucrare au culoarea albă şi se poate observa cum toate au legătură în final cu aplicaţia client implementată.

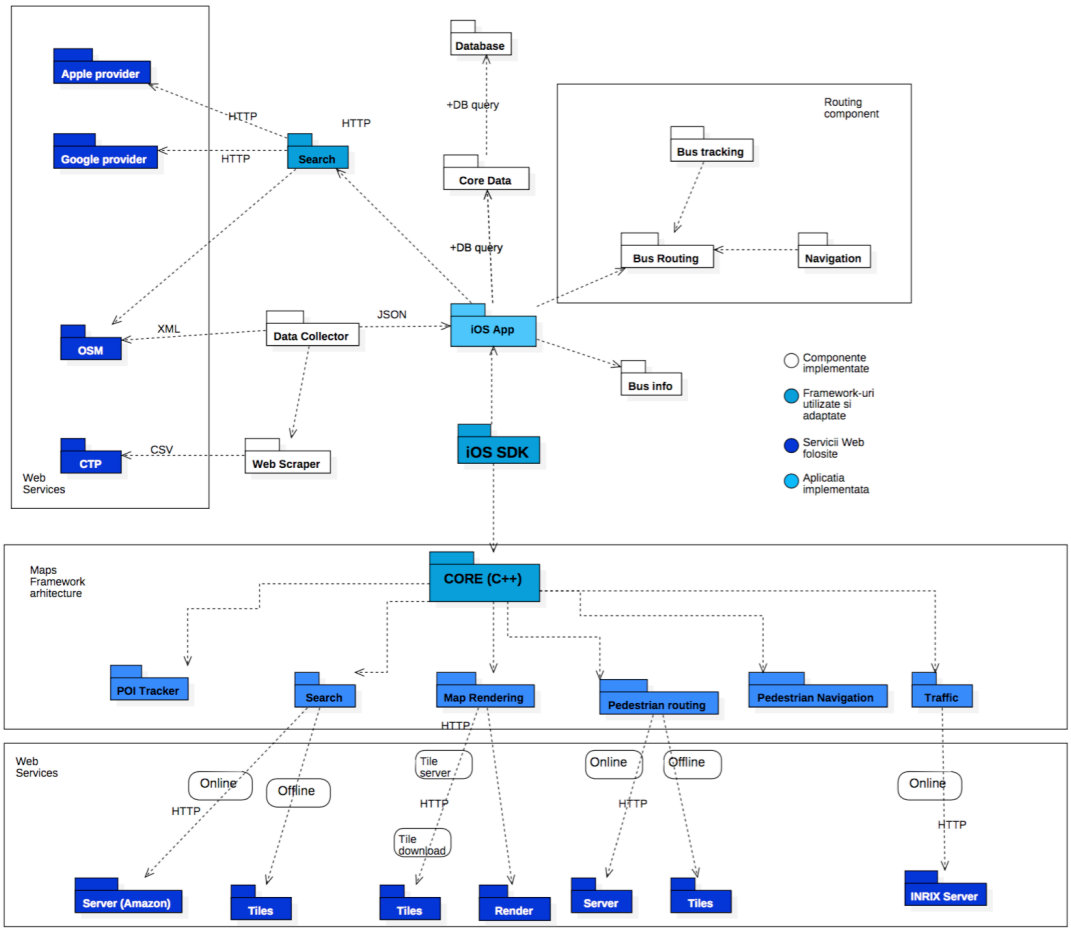

Fig 5.11 - Structura finala a sistemului

De asemenea tot ca şi soluţii proprii în acest sistem sunt şi pachetele colorate mai deschis (ex. iOS App). Prin această structura se doreşte evidenţierea fiecari componente, până la nivel de servicii, cu tot ceea ce a fost folosit în implementarea ei.

Totodată fiecare librărie folosită este ilustrată în această figura, incluzând şi modul ei de comunicare cu exteriorul sau cu serviciile pe care le foloseşte. Se poate

observa că deşi această aplicaţie este doar la nivel de client, întregul proces al dezvoltării acestui sistem a fost destul de complex. În acelaşi timp trebuie amintit, că aceasta este doar o prima abordare a acestei probleme, care la acest nivel a fost mai mult cu scop de a se învaţă lucruri noi şi a se aplicaa unul dintre algoritmii studiaţi în cadrul cursurilor din facultate, însă care se poate duce la un nivel mult mai complex, prin separarea componenţei de rutare şi adăugarea ei unui server, care să permită pe lângă această funcţionalitate şi altele.

Capitolul 6. Testare şi Validare

6.1. Cazuri de testare

Scopul acestui capitol este de a oferi o viziune de ansamblu asupra principalelor metode folosite pentru testarea și validarea aplicației. In ceea ce privește validarea datelor folosite in aplicație deja a fost amintita si prezentată metoda de validare in capitolul anterior, deoarece a fost strans legată de implementarea propriu zisă a unei componente.

Pentru proiectul prezentat în aceasta lucrare un plan de testare a fost stabilit pentru fiecare funcționalitate principală a sistemului. Un plan de testare este alcătuit din mai multe cazuri de testare. Un caz de testare prezintă aspectele functionale de bază despre aplicație și conține în același timp și precondițiile acestuia, spre exemplu, pe ce ecran trebuie să navigheze utilizatorul pentru a avea acces la calcularea unei rute, ce precondiții trebuie sa fie îndeplinite ș.a.m.d.

Foarte important este dezvoltarea cazurilor de testare în fazele de început ale implementării, astfel asigurându-se un proges mai rapid al dezvoltării din princina evitării introducerii de bug-uri din neatentie. De asemenea așa numitele Smoke Tests sunt la fel de folositoare, acestea fiind de altfel o etapă precedentă a testării efective si constă în verificarea dacă programul compileaza sau daca tot mediul este pregătit pentru începerea testării propriu-zise.

În cele ce urmeaza se vor prezenta cele mai importante cazuri de testare ale aplicației, folosite pentru funcționalitățile principale.

Vizualizare informații autobuz - C1

Tabel 6.1 - Caz de testare C1

Pas	Rezultat asteptat
Deschiderea aplicației	Harta prezentată și stațiile afișate
Selectare stație	O fereastră apare în partea de jos conținând numele și opțiunea de calculare rută
Atingerea ferestrei exceptând butonul din dreapta	Extinderea ferestrei si afișarea autobuzelor catre trec prin stația respectivă
Selectarea unui rând din tabel	Se desenează pe hartă traseul respectiv
Apăsarea butonului ”Schedules”	Afișarea orarul pentru autobuzul respectiv

Căutarea unei stații/rute - C2

Tabel 6.2 - Caz de testare C2

Pas	Rezultat asteptat
Deschiderea aplicației	Harta prezentată și stațiile afișate
Selectare ”search text” în partea de sus	Se deschide un ecran nou care permite introducerii unui text sau selectării unor filtre
Introducerea stației/rutei căutate	Afișarea rezultatelor găsite

Urmărire progres autobuz (Bus tracking service) - C3

Tabel 6.3 - Caz de testare C3

Pas	Rezultat asteptat
Deschiderea aplicației	Harta prezentată și stațiile afișate
Selectare stație	O fereastră apare în partea de jos conținând numele și opțiunea de calculare rută
Atingerea ferestrei exceptând butonul din dreapta	Extinderea ferestrei si afișarea autobuzelor catre trec prin stația respectivă
Selectarea unui rând din tabel	Se desenează pe hartă traseul respectiv împreuna cu autobuzele de pe traseu

Începerea unei navigații - C4

Tabel 6.4 - Caz de testare C4

Pas	Rezultat asteptat
Selectarea unei stații sau destinații după metodele prezentate	Afișarea unui dialog care conține butonul ”Route”
Selectarea butonului ”Route”	O fereastră apare în partea de jos conținând un idicator cât timp se calculeaza ruta. Apoi rutele descoperite sunt afișate pe hartă.
Selectarea rutei dorite.	Afișarea pe hartă și activarea butonului de începere a navigației.
Apăsarea butonului ”Start navigation” situat în partea de jos a ecranului.	Un ecran nou care conține informații folositoare unei navigații.

6.2. Sistem pentru simularea locației

Acest sistem are ca scop principal deplasarea cu mijloacele de transport în comun, fapt ce categoric nu se poate testa din birou/acasa/scoala etc. Totuși pentru efectuarea testelor s-a descoperit o soluție care satisface aceste cerințe. Pentru a dobândi acestea a fost necesară integrarea unei librării oferită tot de Skobbler, la fel ca și hărțile care permite simularea locațiilor la fel ca un test care ar fi făcut în realitate.

Pentru integrarea acestui sistem a fost nevoie de a scrie doar două linii de cod, care fac legătura cu sistemul de localizare oferit de librăria de hărți. Locațiile se transmit printr-o conexiune client-server. Pentru aceasta, este necesară implementarea unui ecran în plus in aplicație pentru a permite conectarea cu server-ul care bineînțeles va fi doar pentru modul de ”Debug” al aplicației. După realizarea conexiunii locațiile sunt primite de la server, fie simulate în același timp de pe un iPad, fie prin rularea unui track care a fost memorat în prealabil.

Capitolul 7. Manual de Instalare si Utilizare

Scopul acestui capitol este de a prezenta modul de instalare a aplicației pe telefon si folosirea acesteia.

7.1. Instalare

Pentru a se putea instala o aplicație pe orice dispozitiv Apple trebuie să se utilizeze software-ul iTunes, diponibil pe www.apple.com. Urmatoare secțiune prezintă configurarea acestui software.

Cerințe hardware

Aplicaţtia necesită pentru rulare orice iPhone, iPod, iPad cu o versiune a soft-ul cel putin egală cu iOS 8.0.

Software-ul iTunes se poate instala pe orice calculator care ruleaza sistemul de operare Windows sau macOS.

7.1.1. Configurare iTunes

1. Descărcarea soft-ului disponobil pe site-ul: http://www.apple.com/itunes/download/
2. Se urmează pașii pentru instalare.

7.1.2. Instalare aplicație

Fișierele aplicațiilor iOS au extensia ".ipa" și se instaleaza pe dispozitiv in felul următor:

1. Se deschide iTunes.
2. Se conecteaza telefonul prin cablu la calculator.
3. Se alege optiunea "Add to library" din meniul File.

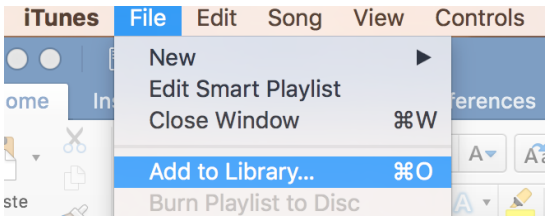

Fig 7.1 - Ilustrarea pasului 3

4. Se cauta fișierul .ipa si se adauga la librarie.
5. Se acceseaza meniul de vizualizare a telefonolui. (Figura 7.1)

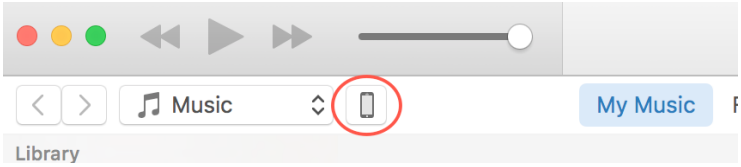

Fig 7.2 - Accesare telefon in iTunes

6. Pasul urmator in partea stanga a ecranului apare sectiune "Apps" - se alege aceasta sectiune si ar trebui sa se ajunga la ecranul urmator:

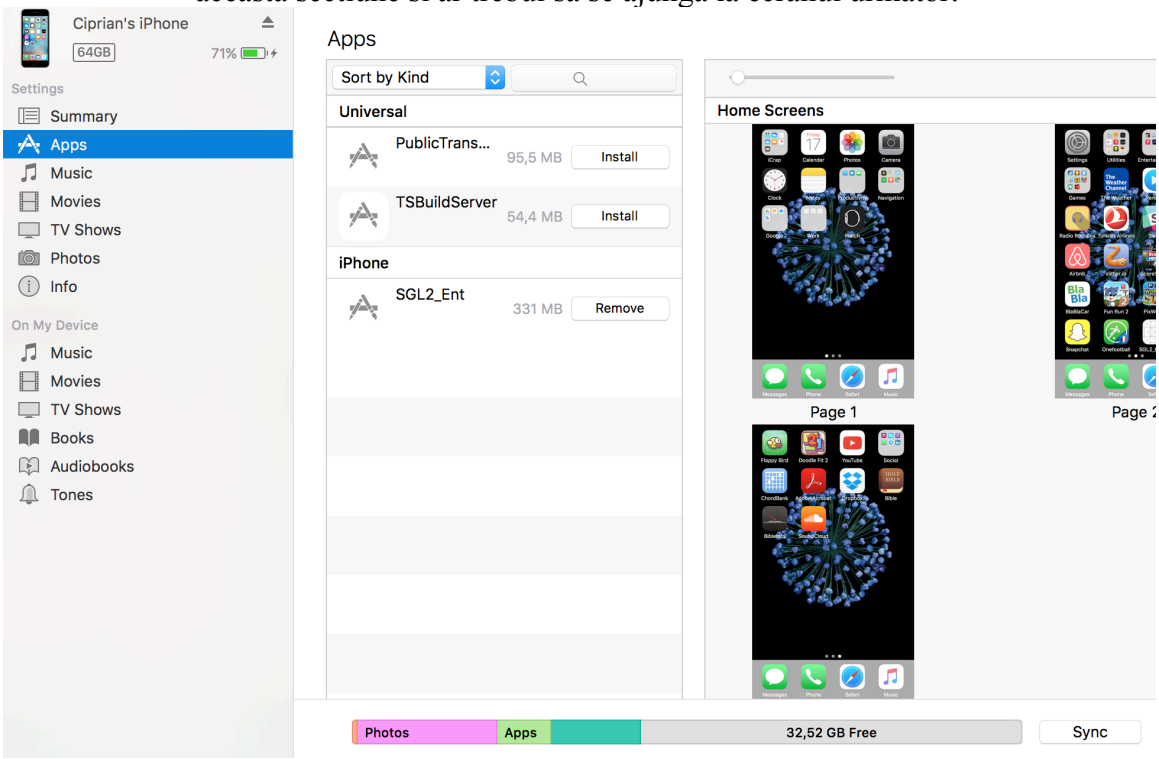

Fig 7.3 - Adaugare si stergere aplicatii iTunes

7. Se vede in lista de aplicații că apare si aplicația adaugată mai devreme "PublicTransportantion" - se selecteaza optiunea "Install", care dupa aceea se va transforma in "Will Install". La fel se poate și sterge o aplicație selectând opțiunea "Remove".
8. Ultimul pas, cand s-au făcut modificările necesare se apasă butonul "Sync" (dreapta jos), iar in urmă toate modificarile se vor sincroniza cu telefonul, iar aplicația va fi instalata.

7.2. Utilizare aplicație

În această secţiune va fi prezentată utilizarea aplicaţiei şi a câtorva funcţionalităţi de baza. Interfaţă utilizator este foarte intiutiva şi nu este deloc greu pentru că o persoană care deschide aplicaţia pentru prima dată să ştie să o folosească.

Ecranul principal al aplicaţiei (Figura 7.3) este reprezentat foarte simplu, de o harta şi o bară de căutare. Pe harta se pot observa nişte puncte gri, care reprezintă toate staţiile de autobuz din Cluj-Napoca. S-a hotărât ca acestea să fie prezente pe ecranul principal în caz că utilizatorul doreşte să vizualizeze toate staţiile care sunt în apropierea lui sau în altă zona a oraşului, fără să ştie numele staţiei, cartierului sau a unui punct de interes din zona respectivă.

După cum se observă, partea din stânga jos există un buton, destul de sugestiv, care dacă este apăsat va centra poziţia utilizatorul pe harta şi va creşte zoom-ul la un nivel mai mare astfel încât să se vadă toate detaliile din zona respectivă.

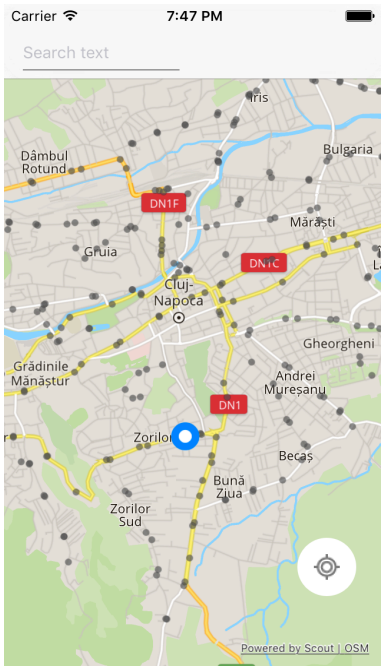

Fig 7.4 - Ecran principal

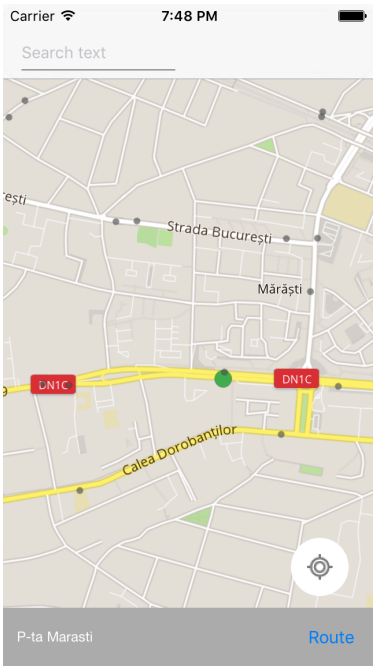

Fig 7.5 - Selectarea unei staţii

Figura 7.4 prezintă selectarea unei staţii de pe harta , care oferă utilizatorului opţiunea de a calcula o ruta direct către acea locaţie , apanasand butonul "Route". Selectarea staţiei (sau a zonei gri din partea de jos a ecranului), va deschide secţiunea cu informaţii despre staţia respectivă: orare, linii etc, la fel ca în Figura 7.5. Orare sunt disponibile după ce o linie de autobuz este selectatat, prin activarea butonului "schedules". De asemenea prin selectarea unei linii, traseul acesteia va apărea desenat pe harta, împreună cu toate staţiile care se află pe acesta şi în acelaşi timpul serviciul de "Bus tracking" este activat (se poate observa iconiţa cu autobuzul albastru), astfel autobuzele care sunt pe traseu în acel moment vor fi simulate, animandu-se pe harta.

Figura 7.6 prezintă ecranul afişat după calcularea unei rute către o destinaţie. Parte de sus arată numărul de rute calculate şi lungimea acestestora. Se poate schimbă ruta, iar când utilizatorul o alege pe cea preferată poate începe o navigaţie pe această apasant buton "Start navigation".

După ce utilizatorul a început o navigaţie, informaţii despre direcţiile următoare vor fi disponibile în partea de sus a ecranului, de asemenea informaţii auditive vor fi transimise utilizatorului până când acesta ajunge în staţia de autobuz.

Selectarea unei adnotaţii pe harta (pin-ul verde) va permite vizualizarea numelui staţiei şi a autobuzelor ce pot fi luate pentru a ajunge la destinaţia dorită.

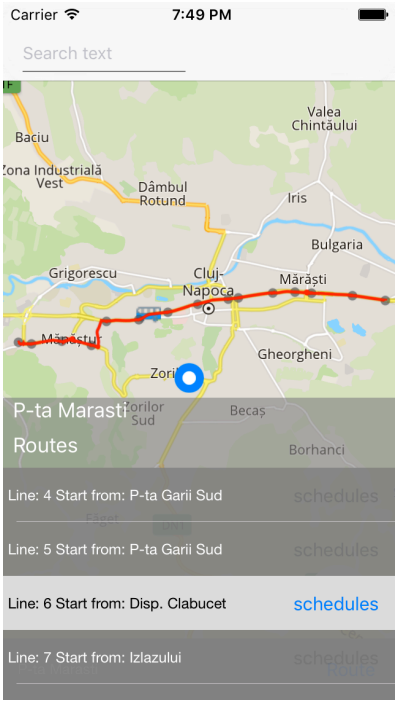

Fig 7.6 - Rute spre destinație

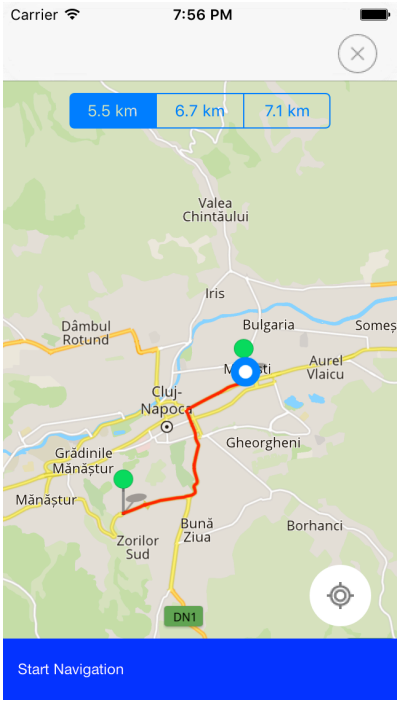

Fig 7.7 - Informații stație

De asemenea, altfe funcţionalităţi ale aplicaţiei sunt ilustatrate în figurile următoare . Printre acestea se numără:căutarea unei adrese, a unui punct de interes, a unei staţii, a unei linii de autobuz sau chiar şi cumpărarea unui bilet de autobuz, direct din aplicaţia aceasta sau contactarea unei firme de Taxi.

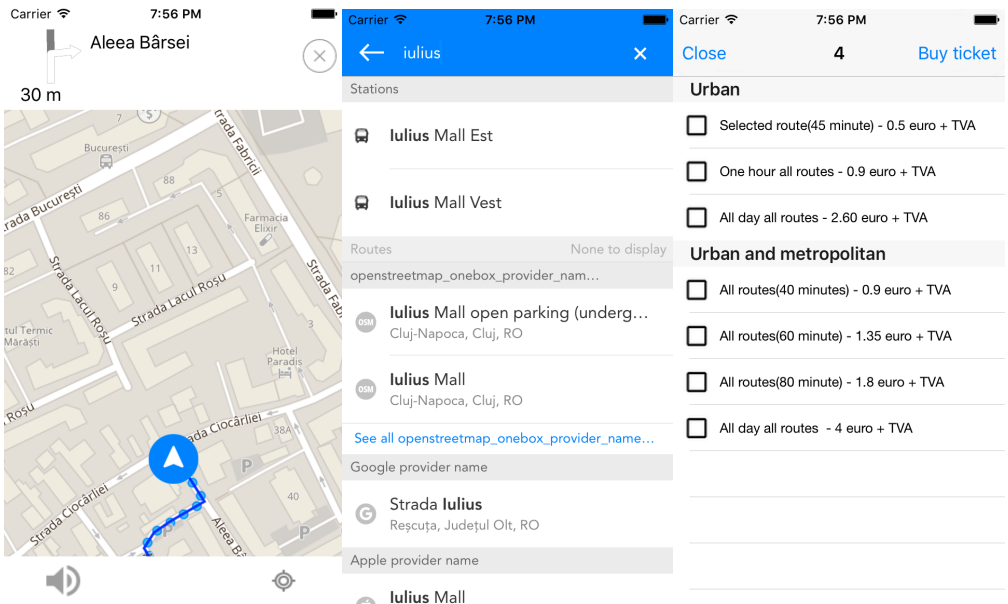

Fig 7.8 - a) Începere navigație; b) Căutare punct de interes; c)Cumpărare bilet

Capitolul 8. Concluzii

Acest capitol descrie principalele aspecte teoretice si practice acumulate prin implementarea si dezvoltarea sistemului prezentat in această lucrare, avantajele acestuia, iar in final cateva posibile dezvoltări ulterioare.

8.1. Realizări

Obiectivul principal dezvoltării acestui sistem este de a pune în practică câteva din cunoştinţele de algoritmică, dobândite în cadrul cursurilor desfăşurate pe parcusul facultăţii. S-a început prin proiectarea algoritmului după modelul învăţat în cursurile care aveau ca scop doboandirea acestor tehnici. S-a ales implementarea algoritmului Dijkstra, pentru început, care apoi a fost dezvoltat şi un pas mai în faţă pentru a indepli cerinţele cerute de problema abordată. Această dezvoltare este prezentat în secţiunea de implementare şi proiectare. Astfel s-au aprofundat cunostiintele despre grafuri, algoritmi de parcurgere, de creare, algortmi de găsire a celor mai scurte drumuri, algoritmul lui Dijkstra. Acest algoritm a fost în totalitate implementat special pentru această lucrare fără a fi deja implementat în altă parte.

Tot pe parcursul proiectării şi implementării acestui sistem, s-au aprofundat cunostiintele despre diferite tipuri de fişiere, crearea şi parsarea acestora, extragerea doar a datelor necesare pentru implementare etc. Tot în legătură cu acest subiect, s-a dezolvatat şi componenta care a descărcat în mod automat toate informaţii despre autobuze şi liniile de autobuze din oraşul Cluj-Napoca, publicate pe site-ul companiei: www.ctp.ro .

Un alt obiectiv a fost acela de a se aprofunda cunostinţele in vederea dezvoltării aplicaţiilor mobile, în special iOS. S-a început prin documentarea despre această tehnologie, iar apoi implementarea propriu-zisă. Acesta este şi un motiv pentru care aplicaţia funcţionează în totalitate la nivel de client, unul dintre scopurile principale ale acestei lucrări fiind învăţarea şi utilizarea a cât mai multor părţi şi librării din iOS.

Un ultim obiectiv principal îndeplinit este folosirea unui framework care oferă suport pentru hărţi (ex. Apple maps, Google maps, skobbler maps) alegându-se varianta Skobbler deoarece este foarte uşor de folosit, de integrat în aplicaţie, şi oferă funcţionalităţi în plus faţă de celelalte: Real reach, Offline maps, toate fiind gratuite. S-au acumulat diferite cunoștințe referitoare integrarea acestei librării într-un proiect şi în acelaşi timp şi folosirea aproape a tuturor funcționalitaților acesteia.

8.2. Avantaje

Printe pricipalele avantaje ale sistemului implementat se găsesc:

- Calcularea celei mai scurte rute dintre două locaţii din Cluj-Napoca, folosind rutare pietonală şi rutare cu mijloacele de transport în comun, disponibile în Cluj Napoca.
- Funcţionalitate în mod offline - după ce aplicaţia este instalată şi sunt downloadate hărţile nu mai este nevoie de acces la internet pentru rutare sau alte servicii, totul fiind la îndemâna utilizatorului indiferent de context.

- Navigarea pe o ruta calculată - oferă informaţii şi alerte despre fiecare pas pe care utilizatorul trebuie să îl realizeze pentru a ajunge la destinaţia dorită.
- Urmărirea autobuzelor pe harta - sistemul implementat de Bus Tracking permite simularea traseelor autobuzelor dar şi a progresului acestora pe harta, în funcţie de ora curentă.

8.3. Dezvoltări ulterioare

Dintre dezvoltările ulterioare gândite pentru acest sistem fac parte:

- Implementarea unui server care să ofere serviciul de rutare - clientul să nu mai conţină logică de rutare (dar să fie cu posibiltate de avea această şi în mod offline dacă doreşte prin descărcarea unei pachet extra de pe server).
- Conectarea aplicaţiei la un back-end, spre exemplu Firebase, pentru creare de conturi pentru utilizatori şi salvarea rutelor favorite, staţii favorite, puncte de interes etc,
- Raportare de trafic - utilizatorii pot raporta trafic într-o anumită zona a oraşului sau pe o anumită ruta, iar ceilalţi utilizatori vor fi notificaţi în timp real.
- Pe aceeaşi idee cu raportarea de trafic, se pot raporta incidente, aglomeraţie în autobuz sau chiar prezența controalelor.
- Aplicarea algoritmului pentru alte seturi de date, de exemplu: reţeaua CFR în România sau reţeaua de autocare, astfel oferind rutare între oricare două locaţii de pe suprafaţa României folosind transportul în comun.

Bibliografie

[1] "A brief introduction to XML", http://tools.arlut.utexas.edu/gash2/doc/xml/introtoxml.html

[2] "What is a GPS?" in Everyday mysteries, Science Reference Researches, http://www.loc.gov/rr/scitech/mysteries/global.html

[3] Aleks Buczkowski, "Why would you use OpenStreeMap if there is Google Maps ?" http://geoawesomeness.com/why-would-you-use-openstreetmap-if- there-is-google-maps/ , 23 Octombrie 2015

[4] Chung, C., Bucanek, J. (2011). Pro Objective-C Design Patterns for iOS. New York:Apress.

[5] Comma Separated Values Standard File Format, http://edoceo.com/utilitas/csv-file- format

[6] Daniel Eggert, Core Data Overview, https://www.objc.io/issues/4-core-data/core-data-overview/, Septembrie 2013

[7] Dijstra's shortest path algorithm, http://www.geeksforgeeks.org/greedy-algorithms-set-6- dijkstras-shortest- path-algorithm/

[8] Geocodarea, https://en.wikipedia.org/wiki/Geocoding

[9] Haitham Latif Hassan Al-Tameemi, "Using Dijsktra Algorithm In Calculating Alternative Shortest Paths For Public Transportation With Transfers And Walking", The Department of Mathematics and Computer Science, Information Technology Program, Cankaya University, June 2014

[10] Jungnickel D., (2005), “Graphs, Networks and Algorithms”, 2nd Edition, Springer-Verlag Berlin Heidelberg, pp. 75-79., Springer-Verlag, 2005

[11] Liu L., (2011), "Data Model and Algorithms for Multimodal Route Planning with Transportation Networks", Technische Universität München, pp. 9-13., 2011

[12] Marcus S. Zarra, "Core Data, 2nd Edition - Data Storage and Management for iOS, OS X, and iCloud, publicata de The Pragmatic Programmers, Dallas, Texas - Raleigh, North Caroline, 2013

[13] Margaret Rouse, Gerhard Hill, Maxine Giza, XML (Extensible Markup Language), http://searchsoa.techtarget.com/definition/XML

[14] Matt Gallagher, Cocoa With Love - Core Data, http://www.cocoawithlove.com/2010/02/differences-between- core-data- and.htm, 16 Februarie 2010

[15] Matt Neuburg, "iOS Programming Fundamentals - Objective C, Xcode and Cocoa Basics", publicata de O'Reilly Media Inc, Sebastopol, CA, Octombrie 2013

[16] Moovit App , http://moovitapp.com/

[17] OpenStreetMap, Map Features, http://wiki.openstreetmap.org/wiki/Map_Features

[18] Sanan S., Jain L., Kappor B., (2013), "Shortest Path Algorithm", International Journal of Application or Innovation in Engineering & Management (IJAIEM), Volume 2, Issue 7, ISSN 2319–4847, pp.316-320, http://www.ijaiem.org/volume2issue7/IJAIEM-2013-07-23-079.pdf

[19] Serge Wroclawski, "Why the world needs OpenStreetMap", https://www.theguardian.com/technology/2014/jan/14/why-the-world-needs-openstreetmap.htm

[20] Skobbler Maps, http://developer.skobbler.com/features#advancedMapStyling

[21] Timothy Merrifield, "Heuristic Route Search in Public Transportation Newtworks", University of Illinois, Chicago, Illinois 2010

[22] TransitApp, https://itunes.apple.com/us/app/transit-app-real-time -tracker/id498151501?mt=8

Anexa 1 - Listă de figuri

Anexa 2 - Listă de tabele